La libreria di riferimento di Python

Versione 2.3.4

Guido van Rossum
Fred L. Drake, Jr., editor

21 maggio 2005

Python Software Foundation
Email: docs@python.org
Traduzione presso
http://www.zonapython.it
Email: zap@zonapython.it

Sommario

Python è un linguaggio di programmazione estensibile, interpretato ed orientato agli oggetti. Supporta un'ampia gamma di applicazioni, dalla semplice elaborazione di testi, ai web browser interattivi.

Mentre il *Manuale di Riferimento Python* descrive esattamente la sintassi e la semantica del linguaggio, non descrive la libreria standard distribuita con il linguaggio, che valorizza appieno la sua immediata usabilità. Questa libreria contiene moduli built-in (scritti in C) che forniscono accesso alle funzionalità di sistema come l'I/O su file, che altrimenti sarebbero inaccessibili ai programmatori Python, come altri moduli scritti in Python che forniscono soluzioni standardizzate per svariati problemi che subentrano nella comune programmazione. Alcuni di questi moduli vengono progettati esplicitamente per incoraggiare e migliorare la portabilità dei programmi Python.

Questo Manuale di Riferimento della Libreria documenta la libreria standard di Python, come molti moduli della libreria facoltativi (che potrebbero essere o meno disponibili, a seconda del supporto offerto dalla piattaforma sottostante e a seconda della configurazione scelta al momento della compilazione). Vengono documentati inoltre i tipi standard del linguaggio e le sue funzioni ed eccezioni built-in, molte delle quali non sono completamente documentate nel Manuale di Riferimento.

Questo manuale assume una conoscenza di base del linguaggio Python. Per una introduzione informale a Python, si legga il *Tutorial Python*; il *Manuale di Riferimento Python* rimane la massima autorità riguardo le questioni sintattiche e semantiche. Infine, il manuale intitolato *Estendere ed Implementare l'interprete Python* descrive il modo in cui aggiungere nuove estensioni in Python e come implementarle in altre applicazioni.

Traduzione in italiano

- 25 maggio 2005

  Finalmente ci siamo, è definitivamente finita anche la revisione della libreria... questa traduzione era stata rilasciata come revisionata “in parte” e quindi il lavoro dietro le quinte è proseguito indefessamente, fino al rilascio odierno del *maggio 2005*.

  Ovviamente un ringraziamento particolare ai tre revisori:

  Antonio Bracaglia antonio.bracaglia[at]teppisti.it, Enrico Morelli morelli[at]cerm.unifi.it e Marco Marconi azazel_arms[at]yahoo.it, in ordine alfabetico.

- 23 dicembre 2004

  La traduzione della documentazione relativa alla libreria del linguaggio, come potrete ben immaginare, è stata molto impegnativa, è costata lacrime e sangue, è durata circa un anno, ma ne siamo soddisfatti!

  Per realizzarla hanno collaborato molti volontari, nel più puro spirito del “free software”, con l’intento di rendere disponibile a tutti la conoscenza di questo manuale nella lingua italiana, con la speranza che anche questo possa aiutare ulteriormente la diffusione di Python, perché non sarebbe per niente male se fosse usato anche nelle scuole d’informatica, come strumento d’insegnamento di base, ne ha tutte le caratteristiche necessarie.

  Traduzione in Italiano della *Libreria di riferimento di Python*, sotto la coordinazione di Ferdinando Ferranti zap[at]zonapython.it, hanno collaborato a tradurre:

  – Marco Marconi azazel_arms[at]yahoo.it,
  – Davide Benini dbenini[at]qubica.net,
  – Paolo Massei paoafr[at]tin.it,
  – Diego Olerni info[at]olerni.it,
  – Pierluigi Fabbris pierluigi.fabbris[at]email.it,
  – Paolo Caldana verbal[at]teppisti.it,
  – Mauro Morichi mauro[at]nonsolocomputer.com,
  – Paolo Mossino paolo.mossino[at]gmail.com,
  – Emanuele Olivetti olivetti[at]itc.it,
  – Nicola Vitale nivit[at]email.it,
  – Bellinetti Antonio geekman[at]aliceposta.it,
  – Michele Leoncelli celli[at]sfera.net,
  – Massimiliano Tiberi max.tbr[at]tiscalinet.it,
  – Yans .H Roll yans[at]puffy077.org,
  – Francesco Bochicchio bockman[at]virgilio.it,
  – Daniele Zambelli daniele.zambelli[at]inwind.it,
  – Beo beo[at]teppisti.it,
  – Alessio G. B. winlinchu[at]yahoo.it,
  – Saba Gianluca gianluca[at]demografiastorica.it,
  – Roberto Distefano robertinux9[at]yahoo.it,
  – Paolo Cecchini spiner[at]paolo-cecchini.org,
  – Davide Bozza davide_bozza[at]hotmail.com.

  Un rigraziamento particolare ad Antonio Bracaglia antonio.bracaglia[at]teppisti.it, Enrico Morelli morelli[at]cerm.unifi.it e Marco Marconi azazel_arms[at]yahoo.it (rispettivamente in ordine alfabetico) che hanno contribuito in modo determinante al rilascio revisionando tutto il documento e vi assicuro che c’era un gran bel lavoro da fare.

  Un ringraziamento anche a tutti coloro che, mediante canali vari hanno dato supporto, consigli ed altro.

  Facciamo le cose serie, ci mettiamo anche una dedica... a Guido Van Rossum no?! ;-)

INDICE

CAPITOLO

UNO

Introduzione

La “libreria Python” contiene alcuni tipi differenti di componenti.

Contiene tipi di dati che normalmente verrebbero considerati parte del “nucleo” di un linguaggio, come numeri e liste. Per questi tipi, il nucleo del linguaggio Python definisce la forma delle costanti manifeste (NdT: literals) e pone alcune costrizioni sulla loro semantica, ma non la definisce appieno. (D’altra parte, il nucleo del linguaggio definisce proprietà sintattiche come l’ortografia e la priorità degli operatori.)

La libreria contiene inoltre funzioni ed eccezioni built-in — oggetti che possono essere usati da tutto il codice Python senza bisogno dell’istruzione `import`. Alcune di queste vengono definite dal nucleo del linguaggio, ma molte non sono essenziali per la semantica di base e vengono descritte solo qui.

La gran parte della libreria, comunque, consiste di una raccolta di moduli. Esistono molti modi per analizzare questa raccolta. Alcuni moduli sono scritti in C e fusi nell’interprete Python; altri sono scritti in Python ed importati in forma sorgente. Alcuni moduli forniscono interfacce che sono fortemente specifiche di Python, come stampare la traccia dello stack; altre forniscono interfacce che sono specifiche di un particolare sistema operativo, come l’accesso ad un hardware specifico; altre forniscono interfacce specifiche ad un particolare dominio di applicazioni, come il World Wide Web. Alcuni moduli sono disponibili in tutte le versioni e port di Python; altri sono disponibili solo quando il sistema sottostante li supporta o li richiede; altri ancora sono disponibili solo quando una particolare opzione di configurazione viene scelta al momento in cui Python viene compilato e installato.

Questo manuale è organizzato “dall’interno all’esterno:” prima descrive i tipi di dati built-in, poi le funzioni ed eccezioni built-in, ed infine i moduli, raggruppati in capitoli di moduli in relazione fra loro. L’ordine dei capitoli e dei moduli all’interno di essi avviene banalmente dal più rilevante al meno importante.

Ciò significa che se iniziate a leggere questo manuale dall’inizio e saltate al prossimo capitolo quando vi sarete annoiati, otterrete un ragionevole riepilogo dei moduli disponibili e delle aree di applicazione che sono supportate dalla libreria di Python. Naturalmente, non *dovete* leggerlo come un principiante — potete anche sfogliare la tabella dei contenuti (all’inizio del manuale), o cercare una specifica funzione, modulo o termine nell’indice (in fondo). Infine, se vi divertite ad imparare argomenti a caso, scegliete una pagina casualmente (guardate il modulo `random`) e leggete una sezione o due. Senza riguardo all’ordine nel quale leggete le sezioni di questo manuale, vi può essere d’aiuto iniziare con il capitolo 2, “Tipi, eccezioni e funzioni built-in”, visto che il resto del manuale richiede familiarità con questo materiale.

Che lo spettacolo abbia inizio!

CAPITOLO

DUE

Oggetti built-in

I nomi per le eccezioni built-in, funzioni ed un numero di costanti, si trovano in una tabella dei simboli separata. Questa tabella viene interrogata da ultima quando l'interprete cerca il significato di un nome, così i nomi locali e globali definiti dall'utente possono sovrascrivere i nomi built-in. I tipi built-in vengono qui descritti insieme per ottenere un facile riferimento.[1]

Le tabelle in questo capitolo documentano la priorità degli operatori elencati in ordine di priorità ascendente (all'interno di una tabella) e riuniscono gli operatori che hanno la stessa priorità nello stesso riquadro. Gli operatori binari dello stesso gruppo con la medesima priorità da sinistra a destra. (Il gruppo degli operatori unari da destra verso sinistra, ma qui non avete una reale scelta.) Vedete il capitolo 5 del *Manuale di Riferimento Python* per una visione completa sulla priorità degli operatori.

2.1 Funzioni built-in

L'interprete Python ha un numero di funzioni built-in che sono sempre disponibili. Vengono qui elencate in ordine alfabetico.

__import__(*name*[, *globals*[, *locals*[, *fromlist*]]])

Questa funzione viene invocata dall'istruzione `import`. Esiste nativamente e potete sostituirla con un'altra funzione che abbia un'interfaccia compatibile, per cambiare la semantica dell'istruzione `import`. Per gli esempi di cosa potreste fare con questa istruzione, vedete la libreria standard alle voci dei moduli `ihooks` e `rexec`. Vedete anche il modulo built-in `imp`, che definisce alcune utili operazioni al di fuori delle quali potrete costruire le vostre funzioni `__import__()` personalizzate.

Per esempio, l'istruzione '`import spam`' risulta nella seguente chiamata: `__import__('spam', globals(),, locals(), [])`; l'istruzione '`from spam.ham import eggs`' risulta in '`__import__('spam.ham', globals(), locals(), ['eggs'])`'. Notate che anche se `locals()` e `['eggs']` vengono passate come argomenti, la funzione `__import__()` non assegna la variabile locale chiamata `['eggs']`; questo viene compiuto dal codice successivo, generato per l'istruzione import. (Difatti, l'implementazione standard non usa sempre gli argomenti *locals*, ed usa i *globals* solo per determinare il contesto relativo al package dell'istruzione `import`).

Quando il nome *name* della variabile è nella forma `package.module`, normalmente, viene restituito il package di alto livello (il nome che precede il primo punto), *non* il modulo chiamato da *name*. Tuttavia, quando viene fornito un argomento *fromlist* non vuoto, viene restituito il modulo chiamato da *name*. Questo è stato fatto per mantenere la compatibilità con il bytecode generato per differenti tipologie di istruzioni import; quando si usa '`import spam.ham.eggs`', il package di alto livello `spam` deve essere importato nello spazio dei nomi, ma quando si usa '`from spam.ham import eggs`', il sotto pacchetto `spam.ham` deve essere usato per cercare la variabile `eggs`. Per aggirare questo comportamento usate `getattr()` per estrarre il componente desiderato. Per esempio potreste definire il seguente assistente:

[1] La maggior parte delle descrizioni difettano nella spiegazione delle eccezioni che possono essere sollevate — questo verrà corretto nelle future versioni di questo manuale.

```
def my_import(name):
    mod = __import__(name)
    components = name.split('.')
    for comp in components[1:]:
        mod = getattr(mod, comp)
    return mod
```

abs(*x*)
Restituisce il valore assoluto di un numero. L'argomento può essere un numero intero semplice o long, o un numero in virgola mobile. Se l'argomento è un numero complesso, viene restituita la sua grandezza reale.

basestring()
Questo tipo astratto è la superclasse per `str` e `unicode`. Non può essere chiamato o istanziato, ma può essere usato per testare se un oggetto è un'istanza di `str` o di `unicode`. `isinstance(obj, basestring)` è equivalente a `isinstance(obj, (str, unicode))`. Nuovo nella versione 2.3.

bool([*x*])
Converte un valore in un booleano, usando la normale procedura di verifica della verità. Se *x* è falso o omesso, questa restituisce `False` (falso); altrimenti restituisce `True` (vero). `bool` è anche una classe, sotto classe di `int`. La classe `bool` non può contenere altre sotto classi. Le sue uniche istanze sono `False` e `True`.

Nuovo nella versione 2.2.1. Modificato nella versione 2.3: Se non viene fornito nessun argomento, questa funzione restituisce `False`.

callable(*object*)
Restituisce vero se l'argomento oggetto *object* sembra essere chiamabile, altrimenti restituisce falso. Anche nel caso in cui venga restituito vero, una chiamata a *object* può comunque non avere successo; se il risultato è falso però, una chiamata a *object* non potrà mai avere successo. Notate che le classi sono chiamabili se hanno un metodo `__call__()`.

chr(*i*)
Restituisce una stringa di un carattere il cui codice ASCII è l'intero *i*. Per esempio, `chr(97)` restituisce la stringa `'a'`. Questo è l'inverso di `ord()`. L'argomento deve essere compreso nell'intervallo [0..255]; Viene sollevata un'eccezione `ValueError` se *i* è fuori da questo intervallo.

classmethod(*function*)
Restituisce un metodo di classe per una funzione *function*.

Un metodo di classe riceve la classe come primo argomento implicito, proprio come un metodo istanziato riceve l'istanza. Per dichiarare un metodo di una classe, usate questa forma:

```
class C:
    def f(cls, arg1, arg2, ...): ...
    f = classmethod(f)
```

Può essere chiamato sia su una classe (come `C.f()`) che su un'istanza (come in `C().f()`). La chiamata viene ignorata eccetto che per quella classe. Se un metodo della classe viene chiamato per una classe derivata, l'oggetto classe derivata viene passato come primo argomento implicito.

I metodi delle classi sono differenti da quelli del C++ o dai metodi statici di Java. Se li volete, vedete `staticmethod()` in questa sezione. Nuovo nella versione 2.2.

cmp(*x*, *y*)
Confronta i due oggetti *x* e *y* e restituisce un intero a seconda del risultato. Il valore d'uscita è negativo se *x* `<` *y*, zero se *x* `==` *y* e quindi positivo se *x* `>` *y*.

compile(*string, filename, kind*[, *flags*[, *dont_inherit*]])
Compila la stringa *string* in un codice oggetto. Il codice oggetto può essere eseguito tramite un'istruzione `exec` o valutato con una chiamata ad `eval()`. L'argomento contenente il nome del file, *filename*, dovrebbe essere il file dal quale viene letto il codice; passate qualche valore riconoscibile se non viene letto da un file (`'<string>'` viene usato spesso). La tipologia *kind*, di argomento specifica quale tipo di codice deve essere compilato; può essere `'exec'` se la stringa è composta da una sequenza di istruzioni; `'eval'` se è composta da una singola espressione, o `'single'` se è composta da una singola istruzione interattiva (in

quest'ultimo caso l'istruzione valuta che l'espressione abbia un qualsiasi valore, altrimenti viene stampato `None`).

Quando compilate istruzioni su più righe, adottate due accorgimenti: la fine della riga deve essere rappresentata da un singolo carattere di fine riga (`'\n'`), e l'input deve essere terminato come minimo da un carattere di fine riga. Se la fine della riga viene rappresentata da `'\r\n'`, usate il metodo delle stringhe `replace()` per cambiarla in `'\n'`.

Gli argomenti facoltativi *flags* e *dont_inherit* (che sono una novità di Python 2.2) controllano quali istruzioni future (vedete la PEP 236) interesseranno la compilazione della stringa. Se nessuno dei due argomenti è presente (o entrambi sono zero) il codice viene compilato con quelle future istruzioni effettivamente presenti nel codice che si sta chiamando a compilare. Se l'argomento *flags* è dato e *dont_inherit* non lo è (o è zero), l'istruzione futura specificata dall'argomento *flags* viene usata in congiunzione a quelle che sarebbero state comunque utilizzate. Se *dont_inherit* è un intero diverso da zero, allora l'argomento *flags* è vero — le istruzioni future di fatto vengono ignorate dalla chiamata per la compilazione.

Le istruzioni future vengono specificate da bit che possono essere bit per bit or-ed e anche per specificare istruzioni multiple. Il bit-fields richiesto per specificare una data caratteristica può essere trovato come l'attributo `compiler_flag` nell'istanza `_Feature` nel modulo `__future__`.

`complex`(`[`*real*`[`, *imag*`]]`)

Crea un numero complesso con il valore reale *real* più l'immaginario *imag*J*, o converte una stringa o numero in un numero complesso. Se il primo parametro è una stringa, viene interpretato come un numero complesso, e la funzione deve essere chiamata senza un secondo parametro. Il secondo parametro non può essere una stringa. Ogni argomento può essere di un qualsiasi tipo numerico (inclusi i complessi). Se *imag* viene omesso, lo si considera zero per definizione, e la funzione serve come una funzione di conversione numerica del genere `int()`, `long()` e `float()`. Se entrambi gli argomenti vengono omessi, restituisce `0j`.

`delattr`(*object*, *name*)

Questa è simile a `setattr()`. Gli argomenti sono un oggetto ed una stringa. La stringa deve essere il nome di uno degli attributi dell'oggetto. La funzione cancella gli attributi nominati e provvede ad assegnare l'oggetto. Per esempio `delattr(`*x*`, '`*foobar*`')` è equivalente a `del` *x*`.`*foobar*.

`dict`(`[`*mapping-or-sequence*`]`)

Restituisce un nuovo dizionario inizializzato da un argomento posizionale facoltativo o da un insieme di argomenti di chiavi. Se non viene dato nessun argomento, restituisce un nuovo dizionario vuoto. Se l'argomento posizionale è un oggetto mappa, restituisce un dizionario che ha le stesse chiavi e gli stessi valori degli oggetti mappa. Altrimenti l'argomento posizionale deve essere una sequenza, un contenitore che supporti l'iterazione, o un oggetto iteratore. Ciascuno degli elementi dell'argomento deve essere inoltre di uno di questi tipi, ed ognuno deve contenere a turno esattamente due oggetti. Il primo viene usato come una chiave in un nuovo dizionario e il secondo come il valore della chiave. Se una chiave data viene vista più di una volta, l'ultimo valore ad essa associato viene conservato nel nuovo dizionario.

Se sono dati gli argomenti chiave, le chiavi stesse con i loro valori associati, vengono aggiunte agli elementi del dizionario. Se una chiave è specificata sia nell'argomento posizionale sia nell'argomento chiave, il valore associato alla chiave viene conservato nel dizionario. Per esempio, tutto questo restituisce un dizionario equivalente a `{uno: 2, due: 3}`:

- `dict({'uno': 2, 'due': 3})`
- `dict({'uno': 2, 'due': 3}.items())`
- `dict({'uno': 2, 'due': 3}.iteritems())`
- `dict(zip(('uno', 'due'), (2, 3)))`
- `dict([['due', 3], ['uno', 2]])`
- `dict(uno=2, due=3)`
- `dict([(['uno', 'due'][i-2], i) for i in (2, 3)])`

Nuovo nella versione 2.2. Modificato nella versione 2.3: Aggiunto il supporto per la costruzione di un dizionario da un argomento chiave.

dir([*object*])

Senza argomenti, restituisce l'elenco dei nomi presenti nella locale tavola dei simboli corrente. Con un argomento, cerca di restituire un elenco di attributi validi per quell'oggetto. Questa informazione viene estratta dall'attributo `__dict__`, se definito, e dalla classe o tipo di oggetto. La lista non è necessariamente completa. Se l'oggetto è un modulo oggetto, la lista contiene i nomi degli attributi dei moduli. Se l'oggetto è un tipo o un oggetto di classe, la lista contiene il nome dei suoi attributi e ricorsivamente degli attributi delle loro basi. Diversamente, la lista contiene il nome degli attributi dell'oggetto, il nome degli attributi della classe e ricorsivamente gli attributi delle classi di base. La lista risultante è ordinata alfabeticamente. Per esempio:

```
>>> import struct
>>> dir()
['__builtins__', '__doc__', '__name__', 'struct']
>>> dir(struct)
['__doc__', '__name__', 'calcsize', 'error', 'pack', 'unpack']
```

Note: Siccome `dir()` viene fornito principalmente per averne il vantaggio nell'uso del prompt interattivo, esso cerca di fornire un utile insieme di nomi, piuttosto che di fornire un voluminoso e rigorosamente definito insieme di nomi, e il suo livello di dettaglio potrà cambiare con successive versioni dell'interprete.

divmod(*a*, *b*)

Prende come argomenti due numeri (non complessi) e restituisce una coppia di numeri consistenti nel loro quoziente e resto, quando si opera una divisione long. Con tipi di operandi misti, si applicano le regole per l'aritmetica binaria. Per interi long o semplici, il risultato è lo stesso di `(`*a*` / `*b*`, `*a*` % `*b*`)`. Per i numeri in virgola mobile il risultato è `(`*q*`, `*a*` % `*b*`)`, dove *q* è solitamente `math.floor(`*a*` / `*b*`)` ma potrebbe essere inferiore di una unità. In ogni caso *q* `*` *b* `+` *a* `%` *b* è molto vicino ad *a*, se *a* `%` *b* non è zero e possiede lo stesso segno di *b*, e `0 <= abs(`*a*` % `*b*`) < abs(`*b*`)`.

Modificato nella versione 2.3: l'uso di `divmod()` con numeri complessi è deprecato.

enumerate(*iterable*)

Restituisce un oggetto enumerate. *iterable* deve essere una sequenza, un iteratore, o qualche altro oggetto che supporti l'iterazione. Il metodo `next()` dell'oggetto restituito da `enumerate()`, restituisce a sua volta una tupla contenente un contatore (da 0) e il corrispondente valore ottenuto iterando sull'*iterable* indicato. `enumerate()` è utile per ottenere una serie indicizzata: `(0, seq[0])`, `(1, seq[1])`, `(2, seq[2])`, ... Nuovo nella versione 2.3.

eval(*expression*[, *globals*[, *locals*]])

Gli argomenti sono una stringa e due dizionari facoltativi. L'espressione *expression* viene analizzata e valutata come un'espressione Python (parlando tecnicamente, una lista di condizioni) usando i dizionari *globals* e *locals* come spazi dei nomi globali e locali. Se il dizionario *globals* è presente e manca '__builtins__', quello corrente viene utilizzato prima che *expression* venga analizzata. Il significato è che normalmente *expression* ha pieno accesso al modulo standard `__builtin__` e l'ambiente soggetto a restrizioni viene propagato. Se il dizionario *locals* viene omesso, diviene predefinito il dizionario *globals*. Se entrambi i dizionari vengono omessi, l'espressione viene eseguita nell'ambiente dove `eval` viene chiamata. Il valore restituito è il risultato dell'espressione valutata. Errori di sintassi vengono riportati come eccezioni. Esempio:

```
>>> x = 1
>>> print eval('x+1')
2
```

Questa funzione può anche impiegarsi per eseguire oggetti di codice arbitrario (come quello creato con `compile()`). In questo caso, viene passato il codice di un oggetto invece di una stringa. Il codice oggetto deve essere stato compilato passando `'eval'` come tipologia (*kind*) di argomento.

Suggerimento: l'esecuzione dinamica di istruzioni viene supportata dall'istruzione `exec`. L'esecuzione di istruzioni da un file viene supportata dalla funzione `execfile()`. Le funzioni `globals()` e `locals()` restituiscono rispettivamente i dizionari globali e locali, che potrebbero essere utili per l'uso di `eval()` o `execfile()`.

execfile(*filename*[, *globals*[, *locals*]])

Questa funzione è simile all'istruzione `exec`, ma analizza un file invece che una stringa. È differente

dall'istruzione `import` in cui non si usa il modulo administration — legge il file incondizionatamente e non crea un nuovo modulo.[2]

Gli argomenti sono il nome di un file e due dizionari facoltativi. Il file viene analizzato e valutato come una sequenza di istruzioni Python (in modo simile ad un modulo), usando i dizionari *globals* e *locals* come spazi dei nomi globali e locali. Se il dizionario *locals* viene omesso, quello predefinito è *globals*. Se entrambi i dizionari vengono omessi, l'espressione viene eseguita nell'ambiente dove viene chiamato `execfile()`. Il valore restituito è `None`.

Avvertenze: Le azioni predefinite di *locals* vengono descritte per le funzioni `locals()` più avanti: non dovrebbero essere tentate modifiche al dizionario *locals* predefinito. Passategli un esplicito dizionario *locals* se avete bisogno di vedere gli effetti del codice sullo spazio dei nomi locale (*locals*) dopo il risultato della funzione `execfile()`. `execfile()` non può essere usato in modo attendibile per modificare le variabili locali di una funzione.

`file`(*filename*[, *mode*[, *bufsize*]])

Restituisce un nuovo file oggetto (descritto precedentemente nella sezione 2.3.9, "File oggetto"). I primi due argomenti sono gli stessi, come per `stdio fopen()`: *filename* è il nome del file che deve essere aperto, *mode* indica il modo e la modalità in cui deve aprirsi il file: `'r'` per la lettura, `'w'` per la scrittura (troncando un file esistente), e `'a'` per aggiungere (in *alcuni* sistemi UNIX significa che il testo viene inserito *tutto* alla fine del file, senza riguardo per la posizione di ricerca).

I modi `'r+'`, `'w+'` ed `'a+'` aprono il file per aggiornarlo (notate che `'w+'` tronca il file). Aggiungete `'b'` al modo per aprire il file in modalità binaria, su sistemi che differenziano tra file binari e file di testo (altrimenti ignorato). Se il file non può essere aperto, viene sollevata un'eccezione `IOError`.

Un'aggiunta al valore predefinito per `fopen()`, riguardo alla modalità di apertura *mode*, potrebbe essere `'U'` o `'rU'`. Se Python è stato compilato con il supporto universale ai fine riga (predefinito) il file verrà aperto come un file di testo, ma le righe termineranno con un `'\n'`, il carattere di fine riga convenzionale di Unix, `'\r'` il carattere Macintosh convenzionale o `'\r\n'`, il carattere Windows convenzionale. Tutte queste rappresentazioni vengono viste come `'\n'` dal programma Python. Se Python è stato compilato senza il supporto universale ai fine riga il *mode* supportato `'U'` è lo stesso del modo testo normale. Notate anche che i file oggetto così aperti hanno anche un attributo chiamato `newlines` che ha valore `None` (se nessun fine riga è stato ancora visto), `'\n'`, `'\r'`, `'\r\n'`, o una tupla contenente tutti i caratteri di fine riga visti.

Se *mode* viene omesso, il valore predefinito è `'r'`. Aprendo un file binario, si dovrebbe collegare `'b'` al valore della *modalità* scelta per una migliore la portabilità. (Questo è utile su sistemi che non trattano i file binari e quelli di testo differentemente, dove serve come documentazione). L'argomento facoltativo *bufsize* specifica la dimensione desiderata del file di buffer: 0 significa non bufferizzato, 1 significa riga bufferizzata, ogni altro valore positivo indica l'uso di un buffer (approssimativamente) di quella misura. Un *bufsize* negativo significa l'uso predefinito di quello di sistema, che è di solito una riga bufferizzata per i dispositivi tty e completa bufferizzazione per gli altri file. Se omesso, viene adottato il sistema predefinito.[3]

Il costruttore `file()` è nuovo in Python 2.2. L'ortografia precedente, `open()`, viene mantenuta per la compatibilità ed è un alias per `file()`.

`filter`(*function*, *list*)

Costruisce una lista dagli elementi di *list* per i quali la funzione *function* restituisce vero. *list* può essere una sequenza, un contenitore che supporta iterazioni, o un iteratore. Se *list* è una stringa o una tupla, il risultato fornisce lo stesso tipo; altrimenti viene restituita sempre una lista. Se *function* ha valore `None`, ne viene assunta l'identità attribuita alla funzione, quindi tutti gli elementi di *list* che risultino falsi (zero o vuoti) vengono rimossi.

Notate che `filter(function,` *list*`)` è equivalente a `[elemento for elemento in` *list* `if function(elemento)]` se la funzione non è `None` e se `[elemento for elemento in` *list* `if elemento]` se la funzione è `None`.

`float`([*x*])

Converte una stringa o un numero in un numero in virgola mobile. Se l'argomento è una stringa, deve contenere un decimale possibilmente indicato o un numero in virgola mobile, possibilmente posizionato

[2] Viene usato piuttosto raramente, da non garantire che faccia parte di una istruzione.

[3] Specificare la dimensione del buffer attualmente non ha effetto sui sistemi che non hanno `setvbuf()`. L'interfaccia per specificare la dimensione del buffer non è stata fatta usando il metodo chiamato `setvbuf()`, perché questo potrebbe causare un dump core dopo che ogni chiamata di I/O è stata effettuata, e non esiste un modo certo per determinare se questo sia il caso.

tra due spazi vuoti. Altrimenti, l'argomento potrà essere un numero semplice o un intero di tipo long o un numero in virgola mobile, e verrà restituito un numero in virgola mobile con lo stesso valore (entro la precisione in virgola mobile di Python). Se nessun argomento viene fornito, restituisce `0.0`.

Note: Quando passati ad una stringa, i valori per NaN ed Infinito possono essere restituiti a seconda della libreria C in uso. Lo specifico insieme di stringhe accettate che causano questi valori dipende interamente dalla libreria C ed è noto che può essere variabile.

`frozenset`([*iterable*])
Restituisce un oggetto frozenset i cui elementi vengono presi da *iterable*. I frozenset sono insiemi che non hanno metodi di aggiornamento, ma possono mescolarsi ed usarsi come membri di altri insiemi o come chiavi di un dizionario. Gli elementi di un frozenset devono essere essi stessi immutabili. Per rappresentare degli insiemi di insiemi, gli insiemi interni dovrebbero anche essere oggetti `frozenset`. Se *iterable* non è specificato, restituisce un nuovo insieme vuoto. `frozenset([])`. Nuovo nella versione 2.4.

`getattr`(*object, name*[, *default*])
Restituisce il valore dell'attributo con nome di un oggetto *object*. *name* deve essere una stringa. Se la stringa è il nome di uno degli attributi dell'oggetto, il risultato è il nome di quell'attributo. Per esempio, `getattr(x, 'foobar')` è equivalente a `x.foobar`. Se l'attributo nominato non esiste, viene restituito il valore predefinito, se fornito, altrimenti viene sollevata un'eccezione `AttributeError`.

`globals`()
Restituisce un dizionario rappresentante la corrente tabella dei simboli globale. Questo è sempre il dizionario del modulo corrente (dentro una funzione o metodo, questo è il modulo dove il dizionario viene definito, non il modulo dal quale è stato chiamato).

`hasattr`(*object, name*)
Gli argomenti sono un oggetto ed una stringa. Il risultato è `True` se la stringa è il nome, *name*, di uno degli attributi dell'oggetto, `False` se non lo è. Questo viene implementato dalla chiamata `getattr(`*object, name*`)` e vedendo se solleva un'eccezione oppure no.

`hash`(*object*)
Restituisce il valore dell'hash dell'oggetto *object* (se ne ha uno). I valori degli hash sono degli interi. Vengono usati per confrontare velocemente le chiavi di un dizionario durante la consultazione. I valori numerici che confrontati risultino uguali hanno lo stesso valore di hash (anche se sono di tipi differenti, come nel caso di 1 e 1.0).

`help`([*object*])
Invoca l'aiuto di sistema built-in. (Questa funzione viene intesa per un uso interattivo). Se non vengono forniti argomenti, l'aiuto interattivo di sistema parte sulla console dell'interprete. Se l'argomento è una stringa, questa viene ricercata come il nome di un modulo, funzione, classe, metodo, parola chiave o indice degli argomenti della documentazione, dopodiché viene stampata una pagina di aiuto sulla console. Se l'argomento è un qualsiasi altro tipo di oggetto, viene generata una pagina di aiuto per l'oggetto. Nuovo nella versione 2.2.

`hex`(*x*)
Converte un numero intero (di qualsiasi dimensione) in una stringa esadecimale. Il risultato è una espressione Python valida. Notate: questo produce sempre una costante senza segno. Per esempio, su una macchina a 32 bit, `hex(-1)` produce `'0xffffffff'`. Quando valutato su una macchina con la stessa dimensione della parola, questa costante viene valutata come -1; ad una differente dimensione della parola aumenta il numero positivo o solleva un'eccezione `OverflowError`.

`id`(*object*)
Restituisce l''identità' di un oggetto. Questo è un intero (o un intero long) che viene garantito per essere unico e costante per questo oggetto durante il suo ciclo di vita. Due oggetti che hanno cicli di vita disgiunti possono avere lo stesso valore di `id()`. (Nota implementativa: questo è l'indirizzo dell'oggetto).

`input`([*prompt*])
Equivalente a `eval(raw_input(`*prompt*`))`. **Avvertenze:** Questa funzione non protegge dagli errori degli utenti! Si aspetta una valida espressione Python come input; se l'input non è sintatticamente valido, verrà sollevata un'eccezione `SyntaxError`. Altre eccezioni possono essere sollevate se si verifica un errore durante la valutazione. (In altre parole, qualche volta è esattamente quello di cui avete bisogno quando scrivete un veloce script per uso esperto).

Se il modulo `readline` è stato caricato, allora `input()` verrà usato per fornire funzionalità per la scrittura su riga di comando, e per la memoria storica dei comandi impartiti.

Considerate l'uso delle funzioni `raw_input()` per il generale input degli utenti.

`int`(*x*[, *radix*])
Converte una stringa o un numero in un intero semplice. Se l'argomento è una stringa, deve contenere un numero decimale rappresentabile come un intero Python, possibilmente posizionato tra due spazi vuoti. Il parametro *radix* fornisce la base per la conversione e può essere un qualsiasi numero intero nell'intervallo [2,36], o zero. Se *radix* è zero, la base adeguata viene calcolata in base al contenuto della stringa; l'interpretazione è la stessa utilizzata per gli interi costanti. Se la base *radix* è specificata e *x* non è una stringa, viene sollevata un'eccezione di tipo `TypeError`. Altrimenti, l'argomento può essere un numero intero semplice, un intero di tipo long o un numero in virgola mobile. La conversione di un numero in virgola mobile in un intero tronca il numero (verso lo zero). Se l'argomento è esterno all'intervallo degli interi, al suo posto viene restituito un intero long. Se non ci sono argomenti dati, viene restituito `0`.

`isinstance`(*object*, *classinfo*)
Restituisce vero se l'argomento *object* è un'istanza dell'argomento *classinfo*, o di una (diretta o indiretta) sotto classe di questo. Restituisce ancora vero se *classinfo* è un tipo di oggetto e *object* è un oggetto di quel tipo. Se *object* non è un'istanza di classe o un oggetto del tipo dato, la funzione restituisce sempre falso. Se *classinfo* non è un oggetto classe o un oggetto tipo, potrebbe essere una tupla della classe o di quel tipo di oggetto, o potrebbe contenere ricorsivamente altre tuple simili (altri tipi di sequenze non vengono accettati). Se *classinfo* non è una classe, un tipo o una tupla di classi, tipi o tuple, viene sollevata un'eccezione `TypeError`. Modificato nella versione 2.2: Aggiunto il supporto per tupla di tipi.

`issubclass`(*class*, *classinfo*)
Restituisce vero se la classe *class* è una sotto classe (diretta o indiretta) di *classinfo*. Una classe viene considerata una sotto classe di sé stessa. *classinfo* può essere una tupla di oggetti della classe, nel quale caso ogni voce in *classinfo* verrà controllata. Negli altri casi, verrà sollevata un'eccezione `TypeError`. Modificato nella versione 2.3: Aggiunto il supporto per una tupla di informazione sui tipi.

`iter`(*o*[, *sentinel*])
Restituisce un oggetto iteratore. Il primo argomento viene interpretato in modo molto differente, a seconda della presenza del secondo argomento. Senza il secondo argomento, *o* deve essere una collezione di oggetti che supporti il protocolllo iterativo (vedete il metodo `__iter__()`), o deve supportare il protocollo sequenza (il metodo `__getitem__()` con numeri interi come argomenti, che iniziano da `0`). Se non supporta nessuno di questi protocolli verrà sollevata un'eccezione `TypeError`. Se il secondo argomento, *sentinel*, viene dato, allora *o* deve essere un oggetto chiamabile. L'iteratore creato in questo caso chiamerà *o* senza argomenti per ogni chiamata al suo metodo `next()`; se il valore restituito è uguale a *sentinel*, verrà sollevata l'eccezione `StopIteration`, altrimenti verrà restituito il valore. Nuovo nella versione 2.2.

`len`(*s*)
Restituisce la lunghezza (il numero di elementi) di un oggetto. L'argomento deve essere una sequenza (stringa, tupla o lista) o un oggetto mappabile (dizionario).

`list`([*sequence*])
Restituisce una lista i cui elementi sono gli stessi e nello stesso ordine degli elementi della sequenza *sequence*. *sequence* può essere una sequenza, un contenitore che supporti le iterazioni, o un oggetto iteratore. Se *sequence* è già una lista, ne viene fatta una copia e restituita, in modo simile a *sequence*`[:]`. Per esempio, `list('abc')` restituisce `['a', 'b', 'c']` e `list( (1, 2, 3) )` restituisce `[1, 2, 3]`. Se non viene dato alcun argomento, restituisce una nuova lista vuota, `[]`.

`locals`()
Aggiorna e restituisce un dizionario rappresentante la locale tabella dei simboli corrente. **Avvertenze:** Il contenuto di questo dizionario non dovrebbe essere modificato; i cambiamenti non possono interessare i valori delle variabili locali usate dall'interprete.

`long`([*x*[, *radix*]])
Converte una stringa o un numero in un intero long. Se l'argomento è una stringa, deve contenere un numero reale di dimensione arbitraria, possibilmente posizionato tra due spazi vuoti. L'argomento *radix* viene interpretato allo stesso modo di `int()` e può essere fornito soltanto quando *x* è una stringa. Altrimenti, l'argomento può essere un numero semplice, un intero long o un numero in virgola mobile, e viene restituito un intero long di uguale valore. La conversione di numeri in virgola mobile in interi tronca (verso zero) il

numero. Se non vengono forniti argomenti restituisce `0L`.

`map`(*function, list, ...*)
Applica la funzione *function* ad ogni elemento della lista *list* e restituisce una lista dei risultati. Se vengono passati ulteriori argomenti di *list*, *function* deve prendere quegli argomenti ed applicarli agli elementi di tutte le liste in parallelo; se una lista è più corta di un'altra, viene estesa con elementi `None`. Se la *function* è `None`, viene assunta la sua funzione identità; se ci sono molteplici liste di argomenti, `map()` restituisce una lista consistente in tuple, contenenti i corrispondenti elementi di tutte le liste (una sorta di operazione di trasposizione). Gli argomenti di *list* possono essere ogni tipo di sequenza; il risultato è sempre una lista.

`max`(*s*[, *args...*])
Con il singolo argomento *s*, restituisce il più grande elemento di una sequenza non vuota (come una stringa, una tupla o una lista). Con più di un argomento, restituisce il più grande degli argomenti.

`min`(*s*[, *args...*])
Con il singolo argomento *s*, restituisce il più piccolo elemento di una sequenza non vuota (come una stringa, una tupla o una lista). Con più di un argomento, restituisce il più piccolo degli argomenti.

`object`()
Restituisce un nuovo oggetto privo di caratteristiche. `object()` è la base per tutti i nuovi stili di classi. Ha i metodi che sono comuni a tutte le istanze di nuovi stili di classi. Nuovo nella versione 2.2.

Modificato nella versione 2.3: Questa funzione non accetta nessun argomento. Precedentemente accettava argomenti ma li ignorava.

`oct`(*x*)
Converte un numero intero (di qualsiasi dimensione) in una stringa ottale. Il risultato è un'espressione Python valida. Notate: questo restituisce sempre una costante senza segno. Per esempio, su una macchina a 32 bit, `oct(-1)` restituisce `'037777777777'`. Quando viene valutata su una macchina con la stessa dimensione della parola, questa costante viene valutata -1; ad una differente dimensione della parola, può diventare un grande numero positivo o sollevare un'eccezione `OverflowError`.

`open`(*filename*[, *mode*[, *bufsize*]])
Un alias per la funzione `file()` descritta precedentemente.

`ord`(*c*)
Restituisce il valore ASCII di una stringa di un carattere, o di un carattere Unicode. E.g., `ord('a')` restituisce l'intero `97`, `ord(u'\u2020')` restituisce `8224`. Questo è il contrario di `chr()` per le stringhe e di `unichr()` per i caratteri Unicode.

`pow`(*x*, *y*[, *z*])
Restituisce *x* elevato alla potenza *y*; se *z* è presente, restituisce *x* elevato a *y*, modulo *z* (una computazione più efficiente di `pow(`*x*`, `*y*`) % `*z*). Gli argomenti devono avere tipi numerici. Con tipi di operandi misti, vengono applicate le regole coercitive per gli operatori binari aritmetici. Per gli operandi int e long int, il risultato ha lo stesso tipo dell'operando (dopo la coercizione) a meno che il secondo argomento risulti negativo; in quel caso, tutti gli argomenti vengono convertiti in float e viene restituito un risultato float. Per esempio, `10**2` restituisce `100`, ma `10**-2` restituisce `0.01`. (Quest'ultima caratteristica è stata aggiunta in Python 2.2. In Python 2.1 e precedenti, se entrambi gli argomenti erano di tipo intero ed il secondo argomento era negativo, veniva sollevata un'eccezione). Se il secondo argomento è negativo, il terzo argomento deve essere omesso. Se *z* è presente, *x* e *y* devono essere di tipo intero, e *y* non deve essere negativo. (Questa restrizione è stata aggiunta in Python 2.2. In Python 2.1 e precedenti, i tre argomenti floating di `pow()` restituivano, dipendentemente dalla piattaforma, un numero in virgola mobile, variabile a seconda dell'arrotondamento in virgola mobile).

`property`([*fget*[, *fset*[, *fdel*[, *doc*]]]])
Restituisce la proprietà dell'attributo per le classi di nuovo stile (classi che derivano da `object`).

fget è una funzione per ottenere il valore di un attributo, allo stesso modo *fset* è una funzione per assegnare un valore e *fdel* è una funzione per cancellare un attributo. Un uso tipico è quello di definite l'attributo gestito x:

```
class C(object):
    def getx(self): return self.__x
    def setx(self, value): self.__x = value
    def delx(self): del self.__x
    x = property(getx, setx, delx, "Io sono la proprietà di 'x'.")
```

Nuovo nella versione 2.2.

range([*start,*] *stop*[, *step*])

Questa è una versatile funzione per creare liste contenenti progressioni aritmetiche. Viene soprattutto utilizzata nei cicli `for`. Gli argomenti devono essere degli interi semplici. Se l'argomento relativo al passo *step* viene omesso, il suo valore predefinito viene assunto uguale a `1`. Se l'argomento *start* viene omesso, il suo valore predefinito viene assunto uguale a `0`. La forma completa restituisce una lista di interi semplici `[`*start*`, `*start* ` + `*step*`, `*start* ` + 2 * `*step*`, ...]`. Se *step* è positivo, l'ultimo elemento è il più grande *start* ` + `*i* ` * `*step* minore di *stop*; se *step* è negativo, l'ultimo elemento è il più grande *start* ` + `*i* ` * `*step* più grande di *stop*. *step* non deve essere zero (altrimenti viene sollevata un'eccezione `ValueError`). Esempio:

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(1, 11)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> range(0, 30, 5)
[0, 5, 10, 15, 20, 25]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(0, -10, -1)
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
>>> range(0)
[]
>>> range(1, 0)
[]
```

raw_input([*prompt*])

Se l'argomento *prompt* è presente, viene scritto sullo standard output, senza il carattere di fine riga. La funzione quindi legge una riga dall'input, la converte in una stringa (togliendo il carattere di nuova riga) e la restituisce. Quando viene letta la EOF, viene sollevata un'eccezione `EOFError`. Esempio:

```
>>> s = raw_input('--> ')
--> Monty Python's Flying Circus
>>> s
"Monty Python's Flying Circus"
```

Se il modulo `readline` viene caricato, `raw_input()` lo userà per elaborare le righe inserite e le funzionalità dello storico dei comandi.

reduce(*function, sequence*[, *initializer*])

Applica la funzione *function*, che richiede contemporaneamente due argomenti, agli elementi della sequenza *sequence*, da sinistra verso destra, così da ridurre la sequenza ad un singolo valore. Per esempio, `reduce(lambda x, y: x+y, [1, 2, 3, 4, 5])` calcola `(((1+2)+3)+4)+5)`. L'argomento di sinistra, *x*, è il valore accumulato e l'argomento di destra, *y*, il valore aggiornato dalla sequenza. Se l'inizializzatore facoltativo *initializer* è presente, viene posizionato prima degli elementi nella sequenza di calcolo, e serve da valore predefinito quando la sequenza è vuota. Se *initializer* non viene dato e *sequence* contiene solo un elemento, viene restituito il primo elemento.

reload(*module*)

Ricarica il modulo *module* precedentemente importato. L'argomento deve essere un modulo oggetto, che deve essere stato importato con successo precedentemente. Questo è utile se avete modificato il file sorgente del modulo usando un editor esterno e volete provare la nuova versione senza lasciare l'interprete Python. Il valore restituito è il modulo oggetto (lo stesso dell'argomento *module*).

Quando `reload(module)` viene eseguito:

- Il codice del modulo Python viene ricompilato e quindi eseguito nuovamente il codice a livello di modulo, definendo un insieme di oggetti che legano i nomi ai dizionari relativi ai moduli. La funzione `init` dei moduli di estensione non viene chiamata in un secondo tempo.
- Come con gli altri oggetti, in Python i vecchi oggetti vengono bonificati dopo che i conteggi dei loro riferimenti raggiungono il valore zero.
- I nomi, nello spazio dei nomi del modulo vengono aggiornati per puntare agli oggetti nuovi o modificati.
- Altri riferimenti ai vecchi oggetti (così come a nomi esterni al modulo) non devono essere ricollegati ai riferimenti dei nuovi oggetti e devono essere aggiornati in ogni spazio dei nomi, dove avvenga e se questo comportamento è desiderato.

Esistono alcuni altri avvertimenti:

Se il modulo è sintatticamente corretto ma la sua inizializzazione fallisce, la prima istruzione `import` che lo riguarda potrebbe non collegare il suo nome localmente, ma immagazzinarlo in un modulo oggetto (parzialmente inizializzato), `sys.modules`. Per ricaricare il modulo dovete eseguire nuovamente un `import` (questo collegherà il nome al modulo oggetto parzialmente inizializzato) prima di poterlo ricaricare con `reload()`).

Quando un modulo viene ricaricato, il suo dizionario (contenente le variabili globali del modulo) viene mantenuto. La ridefinizione dei nomi sovrascriverà le vecchie definizioni, quindi questo non è in generale un problema. Se la nuova versione del modulo non definisce un nome che invece era definito nella vecchia versione, ressta la vecchia definizione. Questa caratteristica può essere usata a vantaggio del modulo se mantiene una tabella globale o una cache degli oggetti — con una istruzione `try` esso può valutare la presenza della tabella, e saltare la sua inizializzazione se lo si desidera.

```
try:
    cache
except NameError:
    cache = {}
```

Non è generalmente molto utile ricaricare il built-in o i moduli dinamicamente, ad eccezione che per `sys`, `__main__` e `__builtin__`. In molti casi, comunque, i moduli di estensione non sono destinati ad essere inizializzati più di una volta, e potrebbero fallire in modo arbitrario quando ricaricati.

Se un modulo importa oggetti da un altro modulo usando `from` ... `import` ..., la chiamata `reload()` per l'altro modulo non ridefinisce gli oggetti importati da esso — un modo per aggirare questo comportamento consiste nel rieseguire l'istruzione `from`, un'altro consiste nell' utilizzare `import` e dei nomi qualificati (*module.name*) al suo posto.

Se il modulo istanzia le istanze di una classe, il ricaricamento del modulo che definisce la classe non influenza il metodo di definizione delle istanze — queste continuano ad usare la vecchia definizione di classe. È vero anche per le classi derivate.

`repr`(*object*)
Restituisce una stringa contenente la rappresentazione stampabile di un oggetto. Questo è lo stesso valore reso dalle conversioni (apici inversi). È utile qualche volta potere accedere a questa operazione come ad una funzione ordinaria. Per molti tipi di dati, questa funzione tenta di restituire una stringa con lo stesso valore di un oggetto, una volta passata ad `eval()`.

`reversed`(*seq*)
Restituisce un iteratore inverso. *seq* deve essere un oggetto che supporti il protocollo sequenza (il metodo `__len__()` e il metodo `__getitem__()` con argomenti interi partendo da `0`). Nuovo nella versione 2.4.

`round`(*x*[, *n*])
Restituisce un valore in virgola mobile *x* arrotondato a *n* cifre dopo il punto decimale. Se *n* è omesso, per definizione viene impostato a zero. Il risultato è un numero in virgola mobile. I valori vengono arrotondati al più vicino multiplo di 10 della potenza *n*, a cui viene attribuito segno negativo: se due multipli sono ugualmente vicini, l'arrotondamento viene fatto partendo da 0 (così, per esempio `round(0.5)` è `1.0` e `round(-0.5)` è `-1.0`).

set([*iterable*])
Restituisce un insieme di elementi presi da *iterable*. Gli elementi devono essere immutabili. Per rappresentare insiemi di insiemi, l'insieme interno dovrebbe essere un insieme di oggetti `frozenset`. Se *iterable* non viene specificato, restituisce un nuovo insieme vuoto, `set([])`. Nuovo nella versione 2.4.

setattr(*object, name, value*)
Questo è la controparte di `getattr()`. Gli argomenti sono un oggetto, una stringa ed un valore arbitrario. La stringa può chiamare un attributo esistente o un nuovo attributo. La funzione assegna il valore all'attributo, se l'oggetto lo permette. Per esempio, `setattr(`*x*`, '`*foobar*`', 123)` è equivalente a *x*`.`*foobar* `= 123`.

slice([*start,*] *stop*[, *step*])
Restituisce un oggetto fetta rappresentante l'insieme degli indici specificati da `range(`*start*`, `*stop*`, `*step*`)`. Gli argomenti *start* e *step* per definizione sono `None`. Gli oggetti fetta hanno attributi in sola lettura `start`, `stop` e `step`, che semplicemente restituiscono i valori degli argomenti (o i loro predefiniti). Non hanno altre funzionalità esplicite; tuttavia vengono usati da Numerical Python ed altre estensioni di terze parti. Gli oggetti fetta vengono generati anche quando viene usata una sintassi indicizzata. Per esempio: '`a[start:stop:step]`' o '`a[start:stop, i]`'.

sorted(*iterable*[, *cmp*[, *key*[, *reverse*]]])
Restituisce una nuova lista ordinata dagli elementi in *iterable*. Gli argomenti facoltativi *cmp*, *key* e *reverse* hanno lo stesso significato di quelli del metodo `list.sort()`. Nuovo nella versione 2.4.

staticmethod(*function*)
Restituisce un metodo statico per la funzione *function*.

Un metodo statico non riceve un primo argomento implicito. Per dichiarare un metodo statico usate la forma:

```
class C:
    def f(arg1, arg2, ...): ...
    f = staticmethod(f)
```

Può essere chiamato sia nella classe (come in `C.f()` che in una istanza (come in `C().f()`). L'istanza viene ignorata eccetto che per la sua classe.

I metodi statici in Python sono simili a quelli trovati in Java o in C++. Per concetti più avanzati, vedete `classmethod()` in questa sezione. Nuovo nella versione 2.2.

str([*object*])
Restituisce una stringa contenente una rappresentazione stampabile di un oggetto. Per le stringhe, questa funzione restituisce la stringa stessa. La differenza con `repr(`*object*`)` è che `str(`*object*`)` non sempre tenta di restituire una stringa che sia accettabile da `eval()`; il suo obiettivo è restituire una stringa stampabile. Se non vengono forniti argomenti, restituisce la stringa vuota, `''`.

sum(*sequence*[, *start*])
Somma gli elementi della sequenza *sequence* cominciando da *start*, da sinistra a destra, e restituisce il totale. Il valore predefinito di *start* è `0`. Gli elementi della sequenza *sequence* sono normalmente numeri e non è consentito che siano stringhe. Il veloce e corretto sistema per concatenare una sequenza di stringhe è chiamare `''.join(`*sequence*`)`. Notate che `sum(range(`*n*`), `*m*`)` è equivalente a `reduce(operator.add, range(`*n*`), `*m*`)`. Nuovo nella versione 2.3.

super(*type*[, *object-or-type*])
Restituisce una superclasse di *type*. Se il secondo argomento viene omesso, il super oggetto restituito è slegato. Se il secondo argomento è un oggetto, `isinstance(`*object*`, `*type*`)` deve essere vero. Se il secondo argomento è un tipo, `issubclass(`*type2*`, `*type*`)` deve essere vero. `super()` lavora solo con le classi di nuovo stile.

Un uso tipico per chiamare un metodo cooperativo di superclasse è:

```
class C(B):
    def meth(self, arg):
        super(C, self).meth(arg)
```

Nuovo nella versione 2.2.

tuple([*sequence*])
Restituisce una tupla i cui elementi sono gli stessi e nello stesso ordine degli elementi della sequenza *sequence*. *sequence* può essere una sequenza, un contenitore che supporta l'iterazione, o un oggetto iteratore. Se *sequence* è già una tupla, viene restituita immutata. Per esempio, `tuple('abc')` restituisce `('a', 'b', 'c')` e `tuple([1, 2, 3])` restituisce `(1, 2, 3)`. Se non vengono forniti argomenti, viene restituita una nuova tupla vuota, `()`.

type(*object*)
Restituisce il tipo dell'oggetto *object*. Il valore restituito è un tipo di oggetto. Il modulo standard `types` definisce nomi per tutti i tipi built-in che non abbiano già dei nomi built-in. Per esempio:

```
>>> import types
>>> x = 'abc'
>>> if type(x) is str: print "Una stringa"
...
Una stringa
>>> def f(): pass
...
>>> if type(f) is types.FunctionType: print "Una funzione"
...
Una funzione
```

La funzione built-in `isinstance()` è raccomandata per testare il tipo di un oggetto.

unichr(*i*)
Restituisce la stringa Unicode di un carattere il cui codice Unicode è l'intero *i*. Per esempio, `unichr(97)` restituisce la stringa `u'a'`. Questo è l'inverso di `ord()` per le stringhe Unicode. L'argomento deve appartenere all'intervallo [0..65535]. Diversamente viene sollevata un'eccezione `ValueError`. Nuovo nella versione 2.0.

unicode([*object*[, *encoding* [, *errors*]]])
Restituisce una stringa in versione Unicode dell'oggetto *object* usando uno dei seguenti modi:

Se sono forniti *encoding* e/o *errors*, `unicode()` decodifica l'oggetto, che può essere sia una stringa a 8 bit che un buffer di caratteri, usando il codec per la codifica *encoding*. Il parametro *encoding* è una stringa che fornisce il nome di una codifica; se la codifica non è conosciuta, viene sollevata un'eccezione `LookupError`. La gestione degli errori viene fatta in accordo con *errors*; questo specifica il trattamento di caratteri che hanno una codifica di input non valida. Se *errors* è `'strict'` (predefinito) viene sollevata un'eccezione `ValueError` sugli errori, mentre un valore `'ignore'` fa in modo che gli errori vengano ignorati silenziosamente, e un valore `'replace'` fa in modo che il carattere di sostituzione ufficiale Unicode, `U+FFFD`, venga usato per sostituire caratteri di input che non possono essere decodificati. Vedete anche il modulo `codecs`.

Se nessun parametro facoltativo viene fornito, `unicode()` imiterà il comportamento di `str()`, salvo restituire stringhe Unicode anziché stringhe a 8-bit. Più precisamente, se l'oggetto *object* è una stringa Unicode o una sua sotto classe, restituirà quella stringa Unicode senza applicare nessuna decodifica.

Per gli oggetti che forniscono un metodo `__unicode__()`, chiamerà questo metodo senza argomenti per creare una stringa Unicode. Per tutti gli altri oggetti, la versione della stringa a 8 bit o una rappresentazione di questa viene richiesta e quindi convertita in una stringa Unicode usando il codec per la codifica predefinita `'strict'`.

Nuovo nella versione 2.0. Modificato nella versione 2.2: Aggiunto il supporto per `__unicode__()`.

vars([*object*])
Senza argomenti, restituisce un dizionario corrispondente alla tavola dei simboli locale. Con un modulo, classe o un oggetto istanza di classe come argomento (o qualsiasi altra cosa che ha come attributo `__dict__`), restituisce un dizionario corrispondente alla tavola dei simboli degli oggetti. Il dizionario restituito non dovrebbe essere modificato: gli effetti sulla corrispondente tavola dei simboli sono indefiniti.[4]

xrange([*start,*] *stop*[, *step*])
Questa funzione è molto simile a `range()`, ma restituisce un "oggetto xrange" invece di una lista. Questo

[4]Nella corrente implementazione, le variabili locali collegate non possono normalmente essere influenzate in questa maniera, ma le variabili possono essere richiamate da altri ambiti (come i moduli). Questo potrebbe cambiare.

è un tipo oscuro di sequenza che produce gli stessi valori della lista corrispondente, senza memorizzarli realmente tutti insieme. Il vantaggio di `xrange()` su `range()` è minimo (poiché `xrange()` deve tuttavia creare dei valori quando vengono richiesti) eccetto quando un intervallo molto ampio di valori viene usato su una macchina con poca memoria o quando tutti gli elementi dell'intervallo non vengono mai usati (come quando un loop è di solito terminato con `break`).

`zip`(*[seq1, ...]*)

Questa funzione restituisce una lista di tuple, dove la tupla *i*-esima contiene l'elemento *i*-esimo da ognuno degli argomenti in sequenza. La lista restituita viene troncata in lunghezza, al valore del più corto argomento della sequenza. Quando esistono sequenze multiple di argomenti che hanno tutte la stessa lunghezza, `zip()` è simile a `map()` con un argomento iniziale `None`. Con una sequenza di un singolo argomento, viene restituita una lista di 1-tupla. Senza argomenti restituisce una lista vuota. Nuovo nella versione 2.0.

Modificato nella versione 2.4: Precedentemente, `zip()` richiedeva almeno un argomento e `zip()` sollevava un'eccezione `TypeError` invece di restituire una lista vuota..

2.2 Funzioni Built-in non essenziali

Esistono varie funzioni built-in che non è essenziale imparare, conoscere o usare, nella moderna programmazione in Python. Sono state mantenute per una compatibilità all'indietro con i programmi scritti con vecchie versioni di Python.

I programmatori Python, gli insegnanti, gli studenti e coloro che scrivono libri sull'argomento dovrebbero sentirsi liberi di escludere queste funzioni senza preoccuparsi di aver tralasciato qualcosa di importante.

`apply`(*function, args*[, *keywords*])

L'argomento della funzione *function* deve essere un oggetto chiamabile (una funzione built-in o un metodo definito dall'utente, o un oggetto classe) e l'argomento *args* deve essere una sequenza. *function* viene chiamata con *args* come lista argomento; il numero degli argomenti è la lunghezza della tupla. Se l'argomento facoltativo *keywords* è presente, deve essere un dizionario le cui chiavi siano stringhe. Specifica gli argomenti *keywords* che devono essere aggiunti alla fine della lista degli argomenti. La chiamata `apply()` è differente dalla chiamata *function*`(`*args*`)`, poiché in quel caso esiste sempre esattamente un argomento. L'uso di `apply()` è equivalente a *function*`(*`*args*`, **`*keywords*`)`. L'uso di `apply()` non è necessario poiché "la sintassi di chiamata estesa", come usata nell'ultimo esempio, è completamente equivalente.

Deprecato dalla versione 2.3. Usare invece la sintassi di chiamata estesa, come descritto oltre.

`buffer`(*object*[, *offset*[, *size*]])

L'oggetto *object* deve essere un oggetto che supporti il buffer di chiamata dell'interfaccia (come stringhe, array e buffer). Un nuovo oggetto buffer deve essere creato con riferimento all'oggetto *object* passato come argomento. L'oggetto buffer deve essere una fetta dall'inizio di *object* (o da uno specificato *offset*). La fetta si estenderà alla fino alla fine di *object* (o avrà la lunghezza fornita dalla dimensione *size* dell'argomento).

`coerce`(*x, y*)

Restituisce una tupla consistente di due argomenti numerici convertiti in un tipo comune, usando le stesse regole utilizzate nelle operazioni aritmetiche.

`intern`(*string*)

Inserisce una stringa *string* nella tavola delle stringhe "interned" e restituisce la stringa inserita – che è *stringa* o una copia. Le stringhe interned sono utili per guadagnare piccoli aumenti di prestazioni in ricerche nei dizionari – se le chiavi nel dizionario sono interned e le chiavi di ricerca sono interned, il confronto delle chiavi (dopo l'hashing), può farsi mediante di puntatori invece che per mezzo di un confronto tra stringhe. Normalmente, i nomi usati nei programmi Python sono automaticamente interned, e i dizionari usati per conservare gli attributi di moduli, classi o istanze hanno chiavi interned. Modificato nella versione 2.3: Le stringhe interned non sono immortali (come in Python 2.2 e versioni precedenti); dovete tenere un riferimento al valore restituito da `intern()` per avere benefici da esso.

2.3 Tipi built-in

Le seguenti sezioni descrivono i tipi standard built-in dell'interprete. Storicamente, i tipi built-in di Python vennero differenziati dai tipi definiti dall'utente, perché non fu possibile usare i tipi built-in come base per l'ereditarietà orientata agli oggetti. Con la versione 2.2 questa situazione ha cominciato a cambiare, anche se il progetto di unificare i tipi definiti dall'utente ed i tipi built-in è ancora lontano dalla completezza.

I tipi principali built-in sono numerici, sequenze, mappature, classi di file, istanze ed eccezioni.

Alcune operazioni vengono supportate da vari tipi di oggetto; in particolare, praticamente tutti gli oggetti possono essere confrontati, testati per il valore di verità e convertiti in una stringa (tramite la notazione `'...'`, la equivalente funzione `repr()`, o tramite la funzione `str()`, leggermente diversa). L'ultima conversione viene usata implicitamente quando un oggetto viene scritto da una istruzione `print`. (Informazioni sull'istruzione istruzione `print` ed altre istruzioni del linguaggio possono essere trovate nel *Manuale di riferimento di Python* e nel *Tutorial Python*.)

2.3.1 Test del valore di verità

Ogni oggetto può essere testato per il valore di verità, per usarlo nelle condizioni `if` o `while`, o come operando delle operazioni booleane qui sotto elencate. I seguenti valori sono considerati falsi :

- `None`
- `False`
- zero per ogni tipo numerico, per esempio `0`, `0L`, `0.0`, `0j`.
- ogni sequenza vuota, per esempio, `''`, `()`, `[]`.
- ogni mappa vuota, per esempio, `{}`.
- istanze di classi definite dall'utente, se la classe definisce un metodo `__nonzero__()` o `__len__()`, quando tale metodo restituisca l'intero zero o il valore `bool False`.[5]

Tutti gli altri valori sono considerati veri — così oggetti di molti tipi sono sempre veri.

Le operazione e le funzioni built-in che forniscono un risultato booleano, restituiscono sempre `0` o `False` per falso e `1` o `True` per vero, salvo indicazioni contrarie. (Eccezione importante: le operazioni booleane '`or`' e '`and`' restituiscono sempre uno dei loro operandi.)

2.3.2 Operazioni booleane

Queste sono le operazioni booleane, ordinate per priorità ascendente:

Operazione	Risultato	Note
x `or` *y*	se *x* è falso, allora *y*, altrimenti *x*	(1)
x `and` *y*	se *x* è falso, allora *x*, altrimenti *y*	(1)
`not` *x*	se *x* è falso, allora `True`, altrimenti `False`	(2)

Note:

(1) Queste valutano soltanto il loro secondo argomento, se richiesto per il loro risultato.

(2) '`not`' ha un livello di priorità inferiore rispetto agli operatori non booleani, così che `not` *a* `==` *b* viene interpretato come `not (`*a* `==` *b*`)` e *a* `==` `not` *b* viene considerato un errore di sintassi.

[5] Ulteriori informazioni su questi metodi speciali possono essere reperite nel *Manuale di riferimento di Python*.

2.3.3 Confronti

Le operazioni di confronto vengono supportate da tutti gli oggetti. Hanno tutte la stessa priorità (che è più alta di quella delle operazioni booleane). Il confronto può essere concatenato arbitrariamente; per esempio, *x* `<` *y* `<=` *z* è equivalente a *x* `<` *y* `e` *y* `<=` *z*, eccetto che *y* viene valutato solo una volta (ma in entrambi i casi *z* non viene valutato affatto quando *x* `<` *y* è falso.

Questa tabella ricapitola le operazioni di confronto:

Operazione	Significato	Note
`<`	minore	
`<=`	minore o uguale	
`>`	maggiore	
`>=`	maggiore o uguale	
`==`	uguale	
`!=`	diverso	(1)
`<>`	diverso	(1)
`is`	identità dell'oggetto	
`is not`	negazione dell'identità dell'oggetto	

Note:

(1) `<>` e `!=` sono ortografie alternative per lo stesso operatore. `!=` è l'ortografia preferita; `<>` è obsoleta.

Oggetti di tipi differenti, con le eccezioni di differenti tipi numerici e differenti tipi stringa, non risultano mai utilizzabili nel confronto di uguaglianza; questi oggetti vengono ordinati coerentemente e non invece in modo arbitrario (in modo tale che l'ordinamento di un array eterogeneo fornisca un risultato coerente). Inoltre, alcuni tipi (per esempio i file oggetto) supportano soltanto una nozione degenere del confronto, ove ognuno dei due oggetti di quel tipo sia diseguale. Tali oggetti vengono ordinati arbitrariamente, e non invece in modo coerente. Gli operatori `<`, `<=`, `>` e `>=`, solleveranno un'eccezione `TypeError` ogni volta che uno degli operandi risulti un numero complesso.

Le istanze di una classe normalmente si confrontano come non uguali, a meno che la classe non definisca il metodo `__cmp__()`. Vedete il *Manuale di riferimento di Python* per informazioni sull'uso di questo metodo ed i suoi effetti sul confronto tra oggetti.

Nota implementativa: Gli oggetti di differenti tipi, con l'eccezione dei numeri, vengono ordinati per il loro nome del tipo; gli oggetti dello stesso tipo che non supportano la proprietà di confronto vengono ordinati per il loro indirizzo.

Altre due operazioni con la stessa priorità sintattica, '`in`' e '`not in`', vengono supportate solo dai tipi sequenza (vedete sotto).

2.3.4 Tipi numerici

Esistono quattro tipi numerici distinti; *interi semplici*, *interi long*, *numeri in virgola mobile* e *numeri complessi*. In aggiunta, i Booleani vengono considerati un sotto tipo di interi semplici. Gli interi semplici (chiamati anche solo interi, *interi*) vengono implementati usando `long` in C, che dà loro come minimo 32 bit di precisione. Gli interi long hanno una precisione illimitata. I numeri in virgola mobile vengono implementati usando `double` in C. Tutti i puntatori alla loro precisione vengono disattivati fino a che non è nota la macchina su cui si sta lavorando.

I numeri complessi hanno una parte reale ed una parte immaginaria, implementate usando `double` in C. Per estrarre queste parti da un numero complesso *z*, si usa *z*`.real` e *z*`.imag`.

I numeri vengono creati da costanti numeriche o come il risultato di funzioni ed operatori built-in. Costanti numeriche intere semplici (inclusi numeri esadecimali e ottali) restituiscono interi semplici fino a che il valore che denotano è troppo grande per essere rappresentato come tale, in quel caso viene restituito un intero long. Le costanti numeriche intere con il suffisso '`L`' o '`l`' restituiscono interi long ('`L`' viene preferito perché '`1l`' sembra troppo simile al numero undici!). Le costanti numeriche contenenti un punto decimale o un segno esponenziale

restituiscono numeri in virgola mobile. L'aggiunta di '`j`' o '`J`' ad una costatnte numerica restituisce un numero complesso con la parte reale uguale a zero. Una costante numerica complessa è la somma di una parte reale ed una parte immaginaria.

Python supporta pienamente gli aritmetici misti: quando un operatore binario aritmetico ha operandi di differente tipo numerico, l'operando con il tipo "narrower" (NdT:stretto) viene allargato a quello dell'altro, dove l'intero semplice è più stretto di quello intero long, a sua volta più stretto di quello in virgola mobile, a sua volta ancora più stretto di quello complesso. Il confronto tra numeri di tipi misti usa le stesse regole. [6] I costruttori `int()`, `long()`, `float()` e `complex()` possono essere usati per produrre numeri di un tipo specifico.

Tutti i tipi numerici (eccetto i complessi) supportano le seguenti operazioni, ricapitolate per priorità ascendente (le operazioni nello stesso box hanno la stessa priorità; tutte le operazioni numeriche hanno una priorità più alta rispetto a quelle di confronto):

Operazione	Risultato
x `+` *y*	somma di *x* e *y*
x `-` *y*	sottrazione di *y* a *x*
x `*` *y*	moltiplicazione di *x* per *y*
x `/` *y*	quoziente di *x* per *y*
x `%` *y*	resto di *x* `/` *y*
`-`*x*	*x* negativizzato
`+`*x*	*x* immutato
`abs(`*x*`)`	valore assoluto della magnitudo di *x*
`int(`*x*`)`	*x* conversione in intero
`long(`*x*`)`	*x* conversione in intero long
`float(`*x*`)`	*x* conversione in virgola mobile
`complex(`*re*`,`*im*`)`	un numero complesso con una parte reale *re* ed una parte immaginaria *im*. Il valore predefinito di *im* è zero.
c`.conjugate()`	coniugazione del numero complesso *c*
`divmod(`*x*`,` *y*`)`	la coppia (*x* `/` *y*, *x* `%` *y*)
`pow(`*x*`,` *y*`)`	*x* elevato alla potenza *y*
x `**` *y*	*x* elevato alla potenza *y*

Note:

(1) Per divisioni di interi (semplici o long) il risultato è un intero. Il risultato viene sempre arrotondato all'intero inferiore: 1/2 è 0, (-1)/2 è -1, 1/(-2) è -1 e (-1)/(-2) è 0. Notate che il risultato è un intero long se ogni operando è un intero long, senza riguardo per il valore numerico.

(2) La conversione da un numero in virgola mobile in un intero (long o semplice) può arrotondare o troncare come in C; vedete le funzioni `floor()` e `ceil()` nel modulo `math` per conversioni meglio definite.

(3) Vedete la sezione 2.1, "Funzioni built-in" per una descrizione completa.

(4) Operatori di base complessi per divisioni, modulo operator e `divmod()`.

Deprecato dalla versione 2.3. Convertite invece in virgola mobile usando `abs()`, se appropriato.

Operazioni bit-string su tipi interi

I tipi interi e long supportano operazioni aggiuntive che hanno significato solo per le bit-string. I numeri negativi vengono trattati come se fossero in complemento a 2 (per gli interi long, questo significa un sufficentemente largo numero di bit che non causino un overflow durante le operazioni).

Le priorità delle operazione binarie bit per bit sono tutte inferiori di quelle delle operazioni numeriche, e superiori a quelle delle comparazioni; le operazioni unarie '`~`' hanno la stessa priorità delle altre operazioni numeriche unarie ('`+`' e '`-`').

Questa tabella illustra le operazioni bit-string ordinate per priorità ascendente (operazioni nello stesso gruppo hanno la stessa priorità):

[6] Di conseguenza, la lista `[1, 2]` viene considerata uguale a `[1.0, 2.0]` e similmente per le tuple.

Operazione	Risultato	Note
x `|` *y*	bit per bit *or* di *x* e *y*	
x `^` *y*	bit per bit *or eclusivo* di *x* e *y*	
x `&` *y*	bit per bit *and* di *x* e *y*	
x `«` *n*	*x* scorrimento a sinistra di *n* bit	(1), (2)
x `»` *n*	*x* scorrimento a destra di *n* bit	(1), (3)
`~`*x*	i bit di *x* invertiti	

Note:

(1) I conteggi negativi dello scorrimento sono illegali e sollevano un'eccezione `ValueError`.

(2) Uno scorrimento di *n* bit a sinistra è equivalente alla moltiplicazione per `pow(2, `*n*`)` senza il controllo sull'overflow.

(3) Uno scorrimento a destra di *n* bit è equivalente alla divisione per `pow(2, `*n*`)` senza il controllo sull'overflow.

2.3.5 Tipi iteratori

Nuovo nella versione 2.2.

Python supporta il concetto di iterazione attraverso i contenitori, implementato usando due distinti metodi; questi vengono usati per consentire alle classi definite dall'utente il supporto all'iterazione. Le sequenze, descritte qui sotto con maggiore dettaglio, supportano sempre i metodi iterativi.

Un metodo ha bisogno di essere definito per contenere oggetti che forniscano il supporto iterativo:

`__iter__()`
Restituisce un oggetto iteratore. L'oggetto viene richiesto per supportare il protocollo iterativo descritto qui sotto. Se un contenitore supporta differenti tipi di iterazione, metodi addizionali possono essere forniti per specifiche richieste di iteratori per quei tipi di iterazione. (Un esempio di un oggetto che supporti forme multiple di iterazione potrebbe essere una struttura ad albero, che supporta entrambi gli attraversamenti in ampiezza e profondità.) Questo metodo corrisponde allo slot `tp_iter` del tipo di struttura per gli oggetti Python nelle API Python/C.

Gli stessi oggetti iteratori vengono richiesti per supportare i seguenti due metodi, che formano insieme il *protocollo iteratore*:

`__iter__()`
Restituisce lo stesso oggetto iteratore. Viene richiesto per permettere sia ai contenitori che agli iteratori di essere usati con le istruzioni `for` e `in`. Questo metodo corrisponde allo slot `tp_iter` per il tipo di struttura per oggetti Python nelle API Python/C.

`next()`
Restituisce l'elemento successivo del contenitore. Se non ci sono ulteriori elementi, viene sollevata un'eccezione di tipo `StopIteration`. Questo metodo corrisponde allo slot `tp_iternext` del tipo di struttura per gli oggetti Python nelle API Python/C.

Python definisce molti oggetti iteratori per supportare l'iterazione su tipi di sequenze generali e specifiche, dizionari, e altre forme più specializzate. I tipi specifici non sono importanti, aldilà della loro implementazione del protocollo iteratore.

L'intenzione del protocollo è quella che quando un metodo iteratore `next()` solleva un'eccezione `StopIteration`, esso continuerà lo stesso in una chiamata successiva. Le esecuzioni che non obbediscono a questa proprietà vengono ritenute interrotte. (Questo vincolo è stato aggiunto in Python 2.3; in Python 2.2, i vari iteratori vengono interrotti secondo questa regola.)

I generatori di Python forniscono un modo conveniente per implementare il protocollo iteratore. Se il metodo dell'oggetto contenitore `__iter__()` viene implementato come un generatore, esso restituirà automaticamente un oggetto iteratore (tecnicamente, un oggetto generatore) sopperendo ai metodi `__iter__()` e `next()`.

2.3.6 Tipi sequenza

Esistono sei tipi sequenza: stringhe, stringhe Unicode, liste, tuple, buffer e oggetti xrange.

Le stringhe costanti vengono scritte tra apici singoli o doppi: `'xyzzy'`, `frobozz`. Vedete il capitolo 2 del *Manuale di riferimento Python* per maggiori informazioni sulle stringhe costanti. Le stringhe Unicode sono molto simili alle stringhe, ma vengono specificate nella sintassi usando un carattere '`u`' che le precede: `u'abc'`, `udef`. Le liste si costruiscono con parentesi quadre, gli elementi vengono separati da virgole: `[a, b, c]`. Le tuple vengono costruite dall'operatore virgola (non all'interno di parentesi quadre) con o senza le parentesi tonde, ma una tupla vuota deve avere le parentesi tonde, come `a, b, c` o `()`. Una tupla costituita da un singolo elemento deve evere virgola in coda, come `(d,)`.

Gli oggetti buffer non vengono direttamente supportati dalla sintassi di Python, ma possono essere creati chiamando la funzione built-in `buffer()`. Non supportano la concatenazione o la ripetizione.

Gli oggetti Xrange sono simili ai buffer in quanto non esiste una specifica sintassi per crearli, ma vengono creati usando la funzione `xrange()`. Non supportano l'affettamento, la concatenazione o la ripetizione, e l'uso di `in`, `not in`, `min()` o `max()` su di essi non ha effetto.

La maggior parte dei tipi sequenza supporta le seguenti operazioni. Le operazioni '`in`' e '`not in`' hanno la stessa priorità delle operazioni di confronto. Le operazioni '`+`' e '`*`' hanno la stessa priorità delle corrispondenti operazioni numeriche.[7]

Questa tabella mostra la sequenza delle operazioni ordinate per priorità ascendente (operazioni nello stesso gruppo hanno la stessa priorità). Nella tabella, *s* e *t* sono sequenze dello stesso tipo; *n*, *i* e *j* sono interi:

Operazione	Risultato	Note
x `in` *s*	`1` se un elemento di *s* è uguale a *x*, altrimenti `0`	(1)
x `not in` *s*	`0` se un elemento di *s* è uguale a *x*, altrimenti `1`	(1)
s `+` *t*	la concatenazione di *s* e *t*	
s `*` *n*, *n* `*` *s*	*n* copie superficiali di *s* concatenate	(2)
s[*i*]	lo i-esimo elemento *i* di *s*, origine 0	(3)
s[*i*:*j*]	fetta di *s* da *i* a *j*	(3), (4)
s[*i*:*j*:*k*]	fetta di *s* da *i* a *j* con passo *k*	(3), (5)
`len(`*s*`)`	lunghezza di *s*	
`min(`*s*`)`	il più piccolo elemento di *s*	
`max(`*s*`)`	il più grande elemento di *s*	

Note:

(1) Quando *s* è una stringa o un oggetto stringa Unicode, le operazioni `in` e `not in` agiscono come una sotto stringa di prova. In versioni di Python precedenti la 2.3, *x* doveva essere una stringa di lunghezza 1. In Python 2.3 e successive, *x* può essere una stringa di qualsiasi lunghezza.

(2) Il valore di *n* meno quello di `0` viene trattato come `0` (che restituisce una sequenza vuota dello stesso tipo di *s*). Notate anche che le copie sono superficiali; le strutture annidate non vengono copiate. Questo spesso confonde i nuovi programmatori Python; considerate:

```
>>> lists = [[]] * 3
>>> lists
[[], [], []]
>>> lists[0].append(3)
>>> lists
[[3], [3], [3]]
```

È accaduto che `lists` è una lista contenente tre copie della lista `[[]]` (una lista di un elemento che contiene una lista vuota), ma la lista contenuta viene condivisa da ogni copia. Potete generare una lista di differenti liste in questo modo:

[7] Devono essere assegnate perché il parser non può sapere il tipo dell'operando.

```
>>> lists = [[] for i in range(3)]
>>> lists[0].append(3)
>>> lists[1].append(5)
>>> lists[2].append(7)
>>> lists
[[3], [5], [7]]
```

(3) Se *i* o *j* è negativo, l'indice è relativo alla fine della stringa: `len(`*s*`) + `*i* o `len(`*s*`) + `*j* vengono sostituiti. Ma notate che `-0` è sempre `0`.

(4) La fetta di *s* da *i* a *j* viene definita come la sequenza di elementi con indice *k* così che *i* `<=` *k* `<` *j*. Se *i* o *j* è più grande di `len(`*s*`)`, usate `len(`*s*`)`. Se *i* viene omesso, usate `0`. Se *j* viene omesso, usate `len(`*s*`)`. Se *i* è più grande o uguale a *j*, la fetta è vuota.

(5) La fetta di *s* da *i* a *j* con passo *k* viene definita come la sequenza di elementi con indice *x* `=` *i* `+` *n*`*`*k* tali che `0 <=` *n* `< abs(i-j)`. Se *i* o *j* è più grande di `len(`*s*`)`, usate `len(`*s*`)`. Se *i* o *j* vengono omessi diventano allora valori "end" (dipendenti dal segno di *k*). Notate, *k* non può essere zero.

Metodi stringa

Questi sono metodi stringa che supportano sia stringhe a 8-bit che oggetti stringa Unicode:

`capitalize``()`
Restituisce una copia della stringa con il solo carattere iniziale maiuscolo.

`center``(`*width*`[`*, fillchar*`])`
Restituisce la centratura di una stringa, in un campo di ampiezza *width*. Il riempimento viene fatto usando il *fillchar* specificato (il predefinito è uno spazio). Modificato nella versione 2.4: Supporto per l'argomento *fillchar*.

`count``(`*sub*`[`*, start*`[`*, end*`]])`
Restituisce il numero di occorrenze della sotto stringa *sub* nella stringa `S[`*start*`:`*end*`]`. Gli argomenti facoltativi *start* e *end* vengono interpretati come nella notazione delle fette.

`decode``([`*encoding*`[`*, errors*`]])`
Decodifica la stringa usando il codec registrato per *encoding* (NdT: codifica). *encoding* è la modalità predefinita per la codifica delle stringhe. *errors* può essere impostato con un differente schema di gestione. Il predefinito è `'strict'`, ad indicare che errori di codifica solleveranno un'eccezione di tipo `ValueError`. Altri possibili valori sono `'ignore'` e `'replace'`. Nuovo nella versione 2.2.

`encode``([`*encoding*`[`*,errors*`]])`
Restituisce una versione codificata della stringa. La codifica predefinita è quella predefinita per la stringa corrente. *errors* può essere impostato con un differente schema di gestione. Il predefinito è `'strict'`, ad indicare che errori di codifica solleveranno un'eccezione di tipo `ValueError`. Altri possibili valori sono `'ignore'` e `'replace'`. Nuovo nella versione 2.0.

`endswith``(`*suffix*`[`*, start*`[`*, end*`]])`
Restituisce `True` se la stringa finisce con lo specificato suffisso *suffix*, altrimenti restituisce `False`. Con il facoltativo *start*, il test inizia da quella posizione. Con il facoltativo *end*, blocca il confronto a quella posizione.

`expandtabs``([`*tabsize*`])`
Restituisce una copia della stringa dove tutti i caratteri tab vengono espansi usando gli spazi. Se *tabsize* non viene fornito, si assume che sia di `8` caratteri.

`find``(`*sub*`[`*, start*`[`*, end*`]])`
Restituisce il più basso indice nella stringa dove viene trovata la sotto stringa *sub*, così che *sub* venga contenuta nell'intervallo [*start*, *end*]. Gli argomenti facoltativi *start* e *end* vengono interpretati come nella notazione relativa alle fette. Restituisce `-1` se *sub* non viene trovata.

`index``(`*sub*`[`*, start*`[`*, end*`]])`
Come `find()`, ma solleva un'eccezione `ValueError` quando la sotto stringa non viene trovata.

isalnum()
Restituisce vero se tutti i caratteri nella stringa sono alfanumerici ed è presente almeno un carattere, falso negli altri casi.

isalpha()
Restituisce vero se tutti i caratteri nella stringa sono alfabetici ed è presente almeno un carattere, falso negli altri casi.

isdigit()
Restituisce vero se tutti i caratteri nella stringa sono cifre ed è presente almeno un carattere, falso negli altri casi.

islower()
Restituisce vero se tutti i caratteri nella stringa sono minuscoli ed è presente almeno un carattere, falso negli altri casi.

isspace()
Restituisce vero se ci sono solo spazi nella stringa ed è presente almeno un carattere, falso negli altri casi.

istitle()
Restituisce vero se la stringa è un "titolo", cioè una stringa con i caratteri iniziali di ogni parola maiuscoli, ed è presente almeno un carattere, per esempio i caratteri maiuscoli possono solo seguire caratteri minuscoli e viceversa. Restituisce falso negli altri casi.

isupper()
Restituisce vero se tutti i caratteri nella stringa sono maiuscoli ed è presente almeno un carattere maiuscolo, falso negli altri casi.

join(*seq*)
Restituisce una stringa che è la concatenazione delle stringhe nella sequenza *seq*. Il separatore tra gli elementi è la stringa che fornisce questo metodo.

ljust(*width*[, *fillchar*])
Restituisce la giustificazione a sinistra di una stringa di larghezza *width*. Il riempimento viene eseguito usando lo specificato *fillchar* (il predefinito è uno spazio). Se *width* è minore di `len(`*s*`)`, viene restituita la stringa originale. Modificato nella versione 2.4: Supporto per l'argomento *fillchar*.

lower()
Restituisce una copia della stringa convertita in minuscolo.

lstrip([*chars*])
Restituisce una copia di una stringa con i caratteri iniziali rimossi. Se *chars* viene omesso o è `None`, i caratteri di spazio vengono rimossi. Se viene fornito e non è `None`, *chars* deve essere una stringa; i caratteri verranno tolti dall'inizio della stringa sulla quale viene chiamato questo metodo. Modificato nella versione 2.2.2: Supporto per l'argomento *chars*.

replace(*old, new*[, *count*])
Restituisce una copia della stringa con tutte le occorrenze della sotto stringa *old* sostituite da *new*. Se l'argomento facoltativo *count* è presente, vengono sostituite solo le prime occorrenze di *count*.

rfind(*sub* [,*start* [,*end*]])
Restituisce il più alto indice nella stringa dove viene rinvenuta la sotto stringa *sub*, così che *sub* venga contenuta all'interno di s[start,end]. Gli argomenti facoltativi *start* e *end* vengono interpretati come nella notazione delle fette. Restituisce `-1` in caso di fallimento.

rindex(*sub*[, *start*[, *end*]])
Come `rfind()` ma solleva un'eccezione `ValueError` quando non viene trovata la sotto stringa *sub*.

rjust(*width*[, *fillchar*])
Restituisce la giustificazione a destra di una stringa di larghezza *width*. Il riempimento viene eseguito usando lo specificato *fillchar* (il predefinito è uno spazio). Se *width* è minore di `len(`*s*`)`, viene restituita la stringa originale. Modificato nella versione 2.4: Supporto per l'argomento *fillchar*.

rsplit([*sep* [,*maxsplit*]])
Restituisce una lista di parole nella stringa, usando *sep* come delimitatore per la stringa. Se *maxsplit* viene fornito, vengono effettuate al più *maxsplit* separazioni, *esattamente* quelle. Se *sep* non viene specificato o è

None, ogni spazio viene considerato un separatore. Nuovo nella versione 2.4.

rstrip([*chars*])
Restituisce una copia della stringa con i caratteri finali rimossi. Se *chars* viene omesso o è None, i caratteri di spazio vengono rimossi. Se viene fornito e non è None, *chars* deve essere una stringa; i caratteri verranno tolti dalla fine della stringa sulla quale viene chiamato questo metodo. Modificato nella versione 2.2.2: Supporto per l'argomento *chars*.

split([*sep* [,*maxsplit*]])
Restituisce una lista delle parole contenute nella stringa, usando *sep* come delimitatore per la stringa. Se viene fornito *maxsplit*, verranno eseguite al più *maxsplit* suddivisioni. Se *sep* non viene specificato o è None, ogni spazio nella stringa viene considerato un separatore.

splitlines([*keepends*])
Restituisce una lista delle righe nella stringa, fermandosi all'ultima riga. Le interruzioni di riga non vengono incluse nella lista risultante a meno che *keepends* venga fornito e sia vero.

startswith(*prefix*[, *start*[, *end*]])
Restituisce True se la stringa inizia con *prefix*, altrimenti restituisce False. Con l'opzione *start*, il test sulla stringa inizia da quella posizione. Con l'opzione *end*, il confronto sulla stringa si ferma a quella posizione.

strip([*chars*])
Restituisce una copia della stringa con i caratteri di inizio e fine riga rimossi. Se *chars* viene omesso o è None, gli spazi vengono rimossi. Se viene fornito e non è None, *chars* deve essere una stringa; i caratteri nella stringa verranno eliminati dall'inizio e dalla fine della stringa sulla quale viene chiamato questo metodo. Modificato nella versione 2.2.2: Supporto per l'argomento *chars*.

swapcase()
Restituisce una copia della stringa con i caratteri maiuscoli convertiti in minuscoli e viceversa.

title()
Restituisce una versione della stringa con il primo carattere delle parole maiuscolo e le altre minuscole.

translate(*table*[, *deletechars*])
Restituisce una copia della stringa dove tutte le occorrenze dei caratteri nell'argomento facoltativo *deletechars* vengono rimossi, ed i restanti caratteri vengono mappati per mezzo della tavola di traduzione fornita, che deve essere una stringa di lunghezza 256.

Per gli oggetti Unicode, il metodo translate() non accetta l'argomento facoltativo *deletechars*. Al suo posto, esso restituisce una copia della stringa *s* dove tutti i caratteri vengono mappati per mezzo della tavola di traduzione fornita, che deve essere una mappa di ordinali Unicode per ordinali Unicode, stringhe Unicode o None. I caratteri non mappati restano inalterati. I caratteri mappati a None vengono cancellati. Notate, un approccio più flessibile consiste nel creare un codificatore con mappatura personalizzata, usando il modulo codecs (vedete encodings.cp1251 per un esempio).

upper()
Restituisce una copia della stringa convertita in caratteri maiuscoli.

zfill(*width*)
Restituisce una stringa numerica di zeri alla sinistra di una stringa di lunghezza *width*. La stringa originale viene restituita inalterata se *width* è minore di len(*s*). Nuovo nella versione 2.2.2.

Operazioni sulla formattazione delle stringhe

Le stringhe e gli oggetti Unicode hanno un'unica operazione built-in; l'operatore % (modulo). Questo è anche conosciuto come *formattatore* di stringhe o operatore di *interpolazione* di stringhe. Fornendo dei valori *formato* %*valori* (dove formato è una stringa o un oggetto Unicode), le specifiche di conversione % in *formato* vengono sostituite con zero o più elementi di *valori*. L'effetto è simile a quello di sprintf() nel linguaggio C. Se *formato* è un oggetto Unicode, o se ognuno degli oggetti è stato convertito usando la conversione %s degli oggetti Unicode, anche il risultato sarà un oggetto Unicode.

Se *formato* richiede un singolo argomento, *valori* potrebbe essere un singolo oggetto non tupla.[8] Altrimenti, *valori*

[8] Per formattare soltanto una tupla dovreste fornire una tupla singleton il cui unico elemento sia la tupla da formattare.

deve essere una tupla con l'esatto numero di elementi specificato dal formato della stringa, o un singolo oggetto mappato (per esempio un dizionario).

Uno specificatore di conversione contiene due o più caratteri ed ha i seguenti componenti, che devono presentarsi in quest'ordine :

1. Il carattere '%', che segna l'inizio dello specificatore.
2. La chiave di mappatura (facoltativa), consistente in una sequenza di caratteri racchiusi tra parentesi tonde (per esempio, `(nomeacaso)`).
3. Opzioni di conversione (facoltativo), che influenzano il risultato di alcuni tipi di conversione.
4. Campo di larghezza minima (facoltativo). Se specificato con un '`*`' (asterisco), la larghezza attuale viene letta dal prossimo elemento della tupla in *valori*, e l'oggetto da covertire viene dopo il campo di larghezza minima, e precisione facoltativa.
5. Precisione (facoltativo), viene fornita come un '`.`' (punto) seguito dalla precisione. Se specificato come '`*`' (un asterisco), la larghezza attuale viene letta dall'elemento successivo della tupla in *valori*, ed il valore da convertire viene dopo la precisione.
6. Modificatore di lunghezza (facoltativo).
7. Tipo conversione.

Quando l'argomento è un dizionario (o un'altro tipo di mappatura), la formattazione nella stringa *deve* includere, tra parentesi, una chiave del dizionario, inserita immediatamente dopo il carattere '%'. La chiave di mappatura seleziona il valore che deve essere formattato dalla mappatura. Per esempio:

```
>>> print '%(language)s has %(#)03d quote types.' % \
          {'language': "Python", "#": 2}
Python has 002 quote types.
```

In questo caso nessuno specificatore `*` può essere trovato nella formattazione (poiché richiede una lista di parametri sequenziali).

Le opzioni di conversione sono:

Opzione	Significato
#	Il valore di conversione userà l'"alternate form" (dove definito piu' avanti).
0	La conversione riempirà di zeri, per valori numerici.
-	Il valore convertito viene sistemato a sinistra (sovrascrivendo la conversione '`0`' se vengono forniti entrambi).
	(uno spazio) Uno spazio bianco dovrebbe essere lasciato prima di un numero positivo (o una stringa vuota) prodotto d
+	Un carattere segno ('`+`' o '`-`') precederà la conversione (sovrascrivendo l'opzione space).

Il modificatore di lunghezza può essere `h`, `l`, e `L` può essere presente, ma viene ignorato come non necessario per Python.

I tipi di conversione sono:

Conversione	Significato	Note
d	Numero intero decimale con segno.	
i	Numero intero decimale con segno.	
o	Ottale senza segno.	(1)
u	Decimale senza segno.	
x	Esadecimale senza segno (minuscolo).	(2)
X	Esadecimale senza segno (maiuscolo).	(2)
e	Numero in virgola mobile, in formato esponenziale (minuscolo).	
E	Numero in virgola mobile, in formato esponenziale (maiuscolo).	
f	Decimale in virgola mobile.	
F	Decimale in virgola mobile.	
g	Lo stesso di 'e' se l'esponente è più grande di -4 o minore della precisione, 'f' altrimenti.	
G	Lo stesso di 'E' se l'esponente è più grande di -4 o minore della precisione, 'F' altrimenti.	
c	Carattere singolo (accetta interi o stringhe di singoli caratteri).	
r	Stringa (converte ogni oggetto Python usando `repr()`).	(3)
s	Stringa (converte ogni oggetto Python usando `str()`).	(4)
%	Nessun argomento viene convertito, riporta un carattere '%' nel risultato.	

Note:

(1) La forma alternativa ottiene il risultato di inserire uno zero iniziale ('0'), fra il riempimento di sinistra e la formattazione del numero, se il primo carattere del risultato non è già uno zero.

(2) La forma alternativa ottiene il risultato di inserire un iniziale `'0x'` o `'0X'` (dipendente da quale tra i formati 'x' o 'X' sia stato usato) tra la parte sinistra e la formattazione del numero, se il carattere iniziale del risultato non è già zero.

(3) La conversione `%r` venne aggiunta in Python 2.0.

(4) Se l'oggetto o la formattazione fornita è una stringa `unicode`, la stringa risultante sarà anch'essa `unicode`.

Poiché le stringhe Python hanno una lunghezza esplicita, le conversioni `%s` non assumono che `'\0'` sia la fine della stringa.

Per ragioni di sicurezza, le precisioni in virgola mobile vengono bloccate a 50; le conversioni `%f` per i numeri di valore assoluto oltre 1e25 vengono sostituite da conversioni `%g`. [9] Tutti gli altri errori sollevano eccezioni.

Operazioni ulteriori sulle stringhe vengono definite nei moduli standard `string` e `re`.

Tipi XRange

Il tipo `xrange` è una sequenza immutabile che viene comunemente usata per i cicli. Il vantaggio del tipo `xrange` è quello di essere un oggetto che impiegherà sempre lo stesso quantitativo di memoria, non importa la misura dell'intervallo che rappresenta. Non ci sono consistenti incrementi di prestazioni.

Gli oggetti XRange hanno veramente poche opzioni: supportano solamente le funzioni di indicizzazione, di iterazione e la `len()`.

Tipi sequenza mutabile

Gli oggetti lista supportano ulteriori operazioni che permettono modifiche locali dell'oggetto. Altri tipi di sequenze mutabili (quando aggiunte al linguaggio) potrebbero anche supportare queste operazioni. Le stringhe e le tuple sono tipi di sequenze immutabili: questi oggetti non possono essere modificati una volta creati. Le seguenti operazioni vengono definite su una sequenza di tipo mutabile (dove *x* è un oggetto arbitrario):

[9] Questi numeri sono ragionevolmente arbitrari. Vengono intesi per evitare di stampare serie infinite di cifre insignificanti senza impedirne l'uso corretto e senza dover conoscere con precisione i valori della virgola mobile su una macchina particolare.

Operazione	Risultato	Note
`s[i] = x`	elemento *i* di *s* viene sostituito da *x*	
`s[i:j] = t`	la fetta di *s* da *i* a *j* viene sostituita da *t*	
`del s[i:j]`	uguale a `s[i:j] = []`	
`s[i:j:k] = t`	gli elementi di `s[i:j:k]` vengono sostituiti da quelli di *t*	(1)
`del s[i:j:k]`	rimuove gli elementi di `s[i:j:k]` dalla lista	
`s.append(x)`	uguale a `s[len(s):len(s)] = [x]`	(2)
`s.extend(x)`	ugualea a `s[len(s):len(s)] = x`	(3)
`s.count(x)`	restituisce il numero di *i* per cui `s[i] == x`	
`s.index(x[, i[, j]])`	restituisce il più piccolo *k*, tale che `s[k] == x` e `i <= k < j`	(4)
`s.insert(i, x)`	uguale a `s[i:i] = [x]`	(5)
`s.pop([i])`	uguale a `x = s[i]; del s[i]; restituisce x`	(6)
`s.remove(x)`	uguale a `del s[s.index(x)]`	(4)
`s.reverse()`	inverte gli elementi di *s* localmente	(7)
`s.sort([cmp[, key[, reverse]]])`	ordina gli elementi di *s* localmente	(7), (8), (9), (10)

Note:

(1) *t* deve avere la stessa lunghezza della fetta che sta rimpiazzando.

(2) L'implementazione in C di Python ha accettato storicamente i parametri multipli ed implicitamente li ha uniti nelle tuple; questo non funziona più in Python 2,0. L'uso di questa specie di funzionalità viene deprecato da Python 1.4.

(3) Solleva un'eccezione quando *x* non è un oggetto lista.

(4) Solleva un'eccezione `ValueError` quando *x* non viene trovato in *s*. Quando un indice negativo viene passato come secondo o terzo parametro al metodo `index()`, vi viene aggiunta la lunghezza della lista, come per gli indici delle fette. Se l'indice è ancora negativo, viene troncato a zero, come per gli indici delle fette. Modificato nella versione 2.3: Precedentemente, `index()` non aveva argomenti per specificare le posizioni di start e stop.

(5) Quando un indice negativo viene passato come primo parametro del metodo `insert()`, vi viene aggiunta la lunghezza della lista, come per gli indici delle fette. Modificato nella versione 2.3: Precedentemente, tutti gli indici negativi venivano troncati a zero.

(6) Il metodo `pop()` viene supportato solamente dai tipi lista ed array. L'argomento facoltativo *i* è per definizione `-1`, così per definizione l'ultimo elemento viene rimosso e rstituito.

(7) I metodi `sort()` e `reverse()` modificano la lista localmente, per economia di spazio, quando ordinano o invertono una grande lista. Per ricordarvi che operano come effetto secondario, non restituiscono liste ordinate o invertite.

(8) Il metodo `sort()` accetta argomenti facoltativi per controllare i confronti.

cmp specifica una funzione di confronto personalizzata di due argomenti (lista elementi) che dovrebbe restituire un negativo, zero o un numero positivo, in funzione del primo argomento considerato più piccolo, uguale, o più grande del secondo argomento: '*cmp*=`lambda x,y: cmp(x.lower(), y.lower())`'

key specifica una funzione di un argomento che viene usato per estrarre una chiave di confronto da ogni elemento della lista: '*cmp*=`str.lower`'

reverse è un valore booleano. Se impostato a `True`, gli elementi della lista vengono elencati come se ogni confronto venisse invertito.

In generale, i processi di conversione *key* e *reverse* sono più veloci di quelli che specificano la funzione equivalente *cmp*. Questo perché *cmp* viene chiamato più volte per ogni elemento della lista, mentre *key* e *reverse* toccano ogni elemento della lista solo una volta.

Modificato nella versione 2.3: Il supporto per `None` come un equivalente per l'omissione, *cmp* è stato aggiunto.

Modificato nella versione 2.4: Il supporto per *key* e *reverse* è stato aggiunto.

(9) A partire da Python 2.3, il metodo `sort()` viene garantito come stabile. `sort()` è stabile se garantisce di non cambiare l'ordine relativo degli elementi uguali che si confrontano — questo aiuta per ordinare in più passaggi (per esempio, ordinare per dipartimento, quindi per grado di stipendio).

(10) Mentre una lista può essere ordinata, l'effetto di tentare di mutarla, o anche esaminarla, non viene definito. L'implementazione in C di Python 2.3 e successivi fa apparire la lista vuota durante l'ordinamento, e solleva un'eccezione di tipo `ValueError` se può rilevare che la lista è stata modificata durante l'operazione.

2.3.7 Tipi set

Un oggetto *set* (NdT: insieme) è una collezione non ordinata di valori immutabili. Gli usi comuni includono l'esame dei membri dell'insieme, la rimozione dei duplicati da una sequenza, e la computazione di operazioni matematiche quali intersezione, unione, differenza, e differenza simmetrica. Nuovo nella versione 2.4.

Come altre collezioni, gli insiemi supportano *x* `in` *set*, `len(`*set*`)` e `for` *x* `in` *set*. Essendo collezioni non ordinate, gli insiemi non registrano la posizione degli elementi o l'ordine di inserzione. Di conseguenza, gli insiemi non supportano l'indicizzazione, l'affettamento o altri comportamenti tipici delle sequenze.

Esistono correntemente due tipi di insiemi built-in, `set` e `frozenset`. Il tipo `set` è mutabile — il contenuto può essere cambiato usando metodi come `add()` e `remove()`. Poiché è mutabile, non ha un valore hash e non può venire usato come una chiave di dizionario o un elemento di un altro insieme. Il tipo `frozenset` è immutabile e supporta l'hash — il suo contenuto non può venire alterato dopo la creazione; tuttavia, può venire usato come una chiave di dizionario o come un elemento di un altro insieme.

Le istanze di `set` e `frozenset` forniscono le seguenti operazioni:

Operazione	Equivalenza	Risultato
`len(`*s*`)`		cardinalità dell'insieme *s*
x `in` *s*		verifica *x* come membro di *s*
x `not in` *s*		verifica che *x* non sia un membro di *s*
s`.issubset(`*t*`)`	*s* `<=` *t*	verifica se ogni elemento in *s* è in *t*
s`.issuperset(`*t*`)`	*s* `>=` *t*	verifica se ogni elemento in *t* è in *s*
s`.union(`*t*`)`	*s* | *t*	nuovo insieme con elementi sia di *s* che di *t*
s`.intersection(`*t*`)`	*s* & *t*	nuovo insieme con elementi comuni tra *s* e *t*
s`.difference(`*t*`)`	*s* - *t*	nuovo insieme con elementi in *s* ma non in *t*
s`.symmetric_difference(`*t*`)`	*s* ^ *t*	nuovo insieme con gli elementi di *s* o *t* ma non entrambi
s`.copy()`		nuovo insieme, una copia superficiale di *s*

Notate, le versioni dei metodi non operatori di `union()`, `intersection()`, `difference()`, `symmetric_difference()`, `issubset()` e `issuperset()` accetteranno ogni iterabile come argomento. Invece i loro operatori base controparte richiedono che i propri argomenti debbano essere insiemi. Questo preclude errori di costruzione come `set('abc') & 'cbs'` in favore di un più leggibile `set('abc').intersection('cbs')`.

`set` e `frozenset` supportano il confronto tra insiemi. Due insiemi sono uguali se e solo se ogni elemento di ogni insieme è contenuto nell'altro (ognuno è un sotto insieme dell'altro). Un insieme è minore dell'altro insieme se e solo se il primo insieme è un insieme di proprietà del secondo insieme (è un sotto insieme ma non è uguale). Un insieme è più grande di un altro insieme se e solo se il primo insieme è un superinsieme proprietario del secondo insieme (è un superinsieme, ma non è uguale).

Il sotto insieme ed il confronto di uguaglianza non generalizzano ad una funzione di ordinamento completa. Per esempio, ogni due insiemi disgiunti sono non uguali e non sono sottoinsiemi uno dell'altro, così *tutte* le seguenti uguaglianze restituiscono `False`: *a*<*b*, *a*==*b* o *a*>*b*. Di conseguenza, gli insiemi non implementano il metodo `__cmp__`.

Poichè gli insiemi definiscono solo parzialmente l'ordinamento (relazioni tra sottoinsiemi), l'output del metodo `list.sort()` risulta indefinito per le liste di insiemi.

Per convenienza nell'implementazione di insiemi di insiemi, i metodi `__contains__()`, `remove()` e `discard()` automaticamente cercano corrispondenze con istanze della classe `set` e le loro controparti

`frozenset`, all'interno di un insieme. Per esempio, `set('abc') in set([frozenset('abc')])` restituisce `True`.

La seguente tabella elenca le operazioni disponibili per classi `set` che non applicano l'istanza immutabile di `frozenset`:

Operazione	Equivalenza	Risultato
s`.update(`*t*`)`	*s* |= *t*	restituisce l'insieme *s* con elementi aggiunti da *t*
s`.intersection_update(`*t*`)`	*s* &= *t*	restituisce l'insieme *s* dei soli valori comuni riscontrati anche in
s`.difference_update(`*t*`)`	*s* -= *t*	restituisce l'insieme *s* dopo la rimozione degli elementi comuni
s`.symmetric_difference_update(`*t*`)`	*s* ^= *t*	restituisce l'insieme *s* con elementi da *s* o *t* ma non da entrambi
s`.add(`*x*`)`		aggiunge l'elemento *x* all'insieme *s*
s`.remove(`*x*`)`		rimuove *x* dall'insieme *s*; solleva l'eccezione `KeyError` se non
s`.discard(`*x*`)`		rimuove *x* dall'insieme *s* se presente
s`.pop()`		rimuove e restituisce un elemento arbitrario da *s*; solleva l'eccez
s`.clear()`		rimuove tutti gli elementi dall'insieme *s*

Notate, le versioni non operatore dei metodi `update()`, `intersection_update()`, `difference_update()` e `symmetric_difference_update()` accetteranno ogni iterabile come un argomento.

2.3.8 Tipi mappa

Un oggetto *mappa* tiene traccia dei valori immutabili per gli oggetti arbitrari. Le mappe sono oggetti mutabili. Correntemente esiste solo un tipo mappa, il dizionario. Le chiavi di un dizionario sono valori quasi arbitrari. Solo valori contenenti liste, dizionari o altri tipi mutabili (che sono confrontati per valore piuttosto che per identità dell'oggetto) non possono venire usati come chiavi. I tipi numerici usati per le chiavi obbediscono alle regole normali per il confronto numerico: se due numeri confrontati sono uguali (come `1` e `1.0`), allora possono essere usati in modo intercambiabile per indicizzare le stesse voci del dizionario.

I dizionari vengono creati mettendo una lista di coppie *chiave*`:` *valore* separati da virgole, tra parentesi graffe, per esempio: `{'jack': 4098, 'sjoerd': 4127}` o `{4098: 'jack', 4127: 'sjoerd'}`.

Le seguenti operazioni vengono definite nelle mappe (dove *a* e *b* sono mappe, *k* è una chiave, e *v* ed *x* sono oggetti arbitrari):

Operazione	Risultato	Note
`len(`*a*`)`	il numero degli elementi in *a*	
a`[`*k*`]`	l'elemento di *a* con chiave *k*	(1)
a`[`*k*`] = `*v*	assegna *a*`[`*k*`]` a *v*	
`del` *a*`[`*k*`]`	rimuove *a*`[`*k*`]` da *a*	(1)
a`.clear()`	rimuove tutti gli elementi da `a`	
a`.copy()`	copia superficiale di `a`	
a`.has_key(`*k*`)`	`True` se *a* ha una chiave *k*, altrimenti `False`	
k `in` *a*	equivalente a *a*.has_key(*k*)	(2)
k `not in` *a*	equivalente a `not` *a*.has_key(*k*)	(2)
a`.items()`	una copia della lista *a* di coppie (*chiave*, *valore*)	(3)
a`.keys()`	una copia della lista *a* di chiavi	(3)
a`.update(`[*b*]`)`	aggiorna (e sovrascrive) le coppie di chiavi/valori da *b*	(9)
a`.fromkeys(`*seq*[`, `*value*]`)`	crea un nuovo dizionario con chiavi da *seq* e valori impostati a *value*	(7)
a`.values()`	una copia dei valori della lista *a*	(3)
a`.get(`*k*[`, `*x*]`)`	*a*`[`*k*`]` se *k* `in` *a*, altrimenti *x*	(4)
a`.setdefault(`*k*[`, `*x*]`)`	*a*`[`*k*`]` se *k* `in` *a*, altrimenti *x* (anch'essa impostata)	(5)
a`.pop(`*k*[`, `*x*]`)`	*a*`[`*k*`]` se *k* `in` *a*, altrimenti *x* (e rimozione di k)	(8)
a`.popitem()`	rimuove e restituisce una coppia di valori arbitrari (*chiave*, *valore*)	(6)
a`.iteritems()`	restituisce un iteratore su coppie (*chiave*, *valore*)	(2), (3)
a`.iterkeys()`	restituisce un iteratore su chiavi di mappe	(2), (3)
a`.itervalues()`	restituisce un iteratore su valori di mappe	(2), (3)

Note:

(1) Solleva un'eccezione `KeyError` se *k* non è compresa nella mappa.

(2) Nuovo nella versione 2.2.

(3) Chiavi e valori vengono elencati in modo casuale. Se `items()`, `keys()`, `values()`, `iteritems()`, `iterkeys()` e `itervalues()` vengono chiamati senza effettuare modifiche al dizionario, le liste corrisponderanno direttamente. Questo permette la creazione di coppie (*valore*, *chiave*) usando `zip()`: '`coppie = zip(`*a*`.values(), `*a*`.keys())`'. Le stesse relazioni valgono per i metodi `iterkeys()` e `itervalues()`: '`coppie = zip(`*a*`.itervalues(), `*a*`.iterkeys())`' forniscono lo stesso valore per `coppie`. Un altro modo per creare la stessa lista è: '`coppie = [(v, k) for (k, v) in `*a*`.iteritems()]`'.

(4) Non solleva mai un'eccezione se *k* non è nella mappa, ma restituisce *x*. *x* è facoltativo; quando *x* non viene fornito e *k* non è nella mappa, viene restituito `None`.

(5) `setdefault()` è simile a `get()`, tranne per il caso in cui *k* non sia presente, *x* viene sia restituito che inserito nel dizionario come il valore di *k*.

(6) `popitem()` è utile ad iterare distruttivamente su un dizionario, viene usato spesso negli algoritmi degli insiemi.

(7) `fromkeys()` è un metodo di classe che restituisce un nuovo dizionario. Il valore *value* predefinito è `None`. Nuovo nella versione 2.3.

(8) `pop()` solleva un'eccezione di tipo `KeyError` quando nessuna chiave predefinita viene fornita e la chiave non è stata trovata. Nuovo nella versione 2.3.

(9) `update()` accetta sia un altro oggetto mappa che un iterabile formato da una coppia chiave/valore (come una tupla o altro iterabile di lunghezza due). Se vengono specificati degli argomenti *keyword* (NdT: parola chiave), la mappa viene aggiornata con quelle coppie chiave/valore: '`d.update(red=1, blue=2)`'. Modificato nella versione 2.4: Permette che l'argomento sia un iterabile formato da una coppia chiave/valore e consente argomenti a parola chiave.

2.3.9 File oggetto

I file oggetto vengono implementati utilizzando il package `stdio` del C, e possono essere creati con il costruttore built-in `file()` descritto nella sezione 2.1, "Funzioni built-in"[10] I file oggetto vengono restituiti anche da degli altri metodi e funzioni built-in, come `os.popen()`, `os.fdopen()` ed il metodo `makefile()` degli oggetti socket.

Quando un'operazione su file fallisce per motivi legati all'I/O, viene sollevata l'eccezione `IOError`. Questo include situazioni dove l'operazione non viene definita per qualche ragione, come un `seek()` su un dispositivo tty o la scrittura di un file aperto in sola lettura.

I file hanno i seguenti metodi:

`close``()`
: Chiude il file. Un file chiuso non può essere più letto o scritto. Ogni operazione che richiede l'apertura di quel file solleverà un'eccezione del tipo `ValueError` dopo che il file sarà stato chiuso. È concesso di chiamare più di una volta il metodo `close()`.

`flush``()`
: Svuota il buffer interno, come `fflush()` di `stdio`. Questo può essere una no-op su qualche oggetto simile a file.

`fileno``()`
: Restituisce l'intero "descrittore di file" che viene usato dall'implementazione sottostante per la richiesta di operazioni di I/O da parte del sistema operativo. Questo può essere utile per altre interfacce di basso livello che usano i descrittori di file, come il modulo `fcntl` o `os.read()` e simili. **Note:** Gli oggetti simili a file che non hanno un reale descrittore di file *non* dovrebbero fornire questo metodo!

`isatty``()`
: Restituisce `True` se il file è connesso ad un dispositivo tty(-simile), altrimenti `False`. **Note:** Se un simile a file non è associato ad un file reale, questo metodo *non* dovrebbe essere implementato!

`next``()`
: Un file oggetto è il suo iteratore personale, per esempio `iter(`*f*`)` restituisce *f* (a meno che *f* non sia chiuso). Quando un file viene usato come iteratore, tipicamente in un ciclo `for` (per esempio, `for line in f: print line`), il metodo `next()` viene chiamato ripetutamente. Questo metodo restituisce la successiva riga di input, o solleva un'eccezione di tipo `StopIteration` quando viene raggiunto l'EOF. Per avere un ciclo `for` più efficiente, che iteri su ogni riga del file (un'operazione molto comune), il metodo `next()` usa un buffer nascosto read-ahead. Come conseguenza dell'uso di un buffer read-ahead, combinando il metodo `next()` con un altro metodo per i file (come `readline()`), l'associazione non funzionerà correttamente. Tuttavia, usando `seek()` per inserire il file in una posizione assoluta si riuscirà a svuotare il buffer read-ahead. Nuovo nella versione 2.3.

`read``([`*size*`])`
: Legge al più una quantità (*size*) di byte da un file (di meno se viene letto l'EOF prima di ottenere la quantità *size* di byte). Se l'argomento *size* è negativo o viene omesso, legge tutti i dati fino a che non viene raggiunto l'EOF. I byte vengono restituiti come un oggetto stringa. Viene restituita una stringa vuota quando l'EOF viene incontrato immediatamente. (Per certi file , come ttys, ha senso continuare la lettura dopo che è stato incontrato l'EOF.) Notate che questo metodo può chiamare la funzione C sottostante `fread()` più di una volta, per acquisire più byte ed arrivare il più vicino possibile alla dimensione (*size*). Notate anche che, quando in modo non bloccante, possono essere restituiti meno dati di quelli richiesti, se non viene passato il parametro *size*.

`readline``([`*size*`])`
: Legge un'intera riga dal file. Un codice di controllo di fine riga viene catturato nella stringa (ma è assente quando il file finisce con una riga incompleta).[11] Se l'argomento *size* è presente e non negativo, rappresenta il conteggio massimo dei byte (inclusi i caratteri di fine riga) e può restituire una riga incompleta. Una stringa vuota viene restituita *solo* quando viene trovato immediatamente un EOF. **Note:** Diversamente dal

[10] file() è nuovo in Python 2.2. Il più vecchio built-in open() è un alias per `file()`.

[11] Il vantaggio di lasciare il carattere di fine riga è che restituisce una stringa vuota, inequivocabile segno della fine del file. È anche utilizzabile (nei casi in cui può interessare, per esempio, se voleste fare una copia esatta del file mentre ne state analizzando le righe) per avvertire se l'ultima riga del file finisce con un carattere di fine riga oppure no (sì, questo succede!).

metodo `fgets()` di `stdio`, la stringa restituita contiene caratteri nulli (`'\0'`) se vengono incontrati nell'input.

`readlines`([*sizehint*])
Legge fino a EOF usando `readline()`, e restituisce una lista contenente le righe lette. Se l'argomento facoltativo *sizehint* è presente, invece di leggere fino a EOF, legge le righe intere il cui valore approssimativo ammonta a *sizehint* byte (possibilmente dopo l'arrotondamento superiore alla misura del buffer interno). Usando l'implementazione di oggetti con interfaccia simile a file, si potrà scegliere di ignorare *sizehint* se non può essere implementato, o se non può esserlo efficientemente.

`xreadlines`()
Questo metodo restituisce le medesime funzionalità di `iter(f)`. Nuovo nella versione 2.1. **Deprecato dalla versione 2.3.** Usate invece '`for` *line* `in` *file*'.

`seek`(*offset*[, *whence*])
Imposta la corrente posizione del file, come `fseek()` di `stdio`. L'argomento *whence* è facoltativo e per definizione viene impostato a `0` (posizionamento assoluto del file); altri valori sono `1` (ricerca relativa alla posizione corrente) e `2` (ricerca relativa alla fine del file). Non ci sono valori restituiti. Notate che se il file viene aperto in modalità appending (NdT: aggiunta) (modo `'a'` o `'a+'`) tutte le operazioni `seek()` non verranno effettuate alla prossima scrittura. Se il file viene aperto solo in scrittura in modalità append (modo `'a'`), questo metodo è essenzialmente un no-op, ma rimane utile per file aperti in modalità append con la lettura abilitata (modo `'a+'`). Se il file viene aperto in modalità testo (mode `'t'`), solo gli offset restituiti da `tell()` sono ammessi. L'uso di altri offset causa comportamenti imprevisti.

Notate che non tutti i file oggetto sono soggetti al metodo `seek()`.

`tell`()
Restituisce la corrente posizione del file, come il metodo `ftell()` di `stdio`.

`truncate`([*size*])
Tronca la dimensione del file. Se l'agomento facoltativo *size* è presente, il file viene troncato a (al massimo) quella misura. La misura viene predefinita alla posizione corrente. La posizione corrente nel file non viene cambiata. Notate che se la misura specificata eccede la misura del file corrente, il risultato dipende dalla piattaforma: risultati possibili includono che il file resti immutato, aumenti fino alla misura specificata come se fosse zero-filled, o incrementi fino alla misura specifica con nuovo contenuto non specificato. Disponibilità: Windows e la maggior parte delle varianti UNIX.

`write`(*str*)
Scrive una stringa nel file. Non ci sono valori restituiti. Fino a che viene bufferizzata, la stringa non può essere mostrata nel file prima che vengano chiamati i metodi `flush()` o `close()`.

`writelines`(*sequence*)
Scrive una sequenza di stringhe nel file. La sequenza può essere ogni oggetto iterabile che produce stringhe. Non ci sono valori restituiti. (Il nome viene inteso per ricercare `readlines()`; `writelines()` non aggiunge separatori di riga.)

File supporta il protocollo iteratore. Ogni iterazione restituisce lo stesso risultato di *file*`.readline()`, e l'iterazione finisce quando il metodo `readline()` restituisce una stringa vuota.

I file oggetto possono offrire un numero di altri interessanti attributi. Questi non sono richiesti per gli oggetti simile a file, ma dovrebbero essere implementati se hanno senso per il particolare oggetto.

`closed`
Un valore booleano indicante il corrente stato del file oggetto. Questo è un attributo in sola lettura; il metodo `close()` cambia il valore. Può non essere disponibile su tutti gli oggetti simile a file.

`encoding`
La codifica usata dal file. Quando delle stringhe Unicode vengono scritte in un file, vengono convertite in stringhe di byte, usando questa codifica. In aggiunta, quando il file è connesso ad un terminale, l'attributo prende la codifica che il terminale usa abitualmente (questa informazione potrebbe essere sbagliata se l'utente ha il terminale configurato male). L'attributo è in sola lettura e non può essere presente su tutti gli oggetti simile a file. Potrebbe essere anche `None`, in quel caso il file userà la codifica predefinita del sistema per convertire le stringhe Unicode.

Nuovo nella versione 2.3.

mode
Il modo I/O per il file. Se il file è stato creato usando la funzione built-in `open()`, questo sarà il valore del parametro *mode*. Questo è un attributo in sola lettura e può non essere presente in tutti gli oggetti simile a file.

name
Se il file oggetto è stato creato usando `open()`, il nome del file. Altrimenti, alcune stringhe che indichino il sorgente del file oggetto, nella forma '`<...>`'. Questo è un attributo in sola lettura e può non essere presente in tutti gli oggetti simile a file.

newlines
Se Python è stato compilato con l'opzione **--with-universal-newlines** al momento del **configure** (predefinita), questo attributo in sola lettura esiste, e per i file aperti in modalità di lettura universal newline (NdT: fine riga), tiene traccia dei tipi di fine riga incontrati durante la lettura del file. I valori che possono essere presi in considerazione sono `'\r'`, `'\n'`, `'\r\n'`, `None` (sconosciuto, non vengono più letti i fine riga) o una tupla contenente tutti i tipi di fine riga visti, per indicare i fine riga multipli che sono stati trovati. Per i file che non vengono letti nel modo universal newline il valore di questi attributi sarà `None`.

softspace
Valore booleano che indica se uno spazio necessita di essere stampato prima di un altro valore quando si usa l'istruzione `print`. Le classi che stanno tentando di simulare un file oggetto dovrebbero avere anche un attributo `softspace` scrivibile, che dovrebbe essere inizializzato a zero. Questo sarà automatico per la maggior parte delle classi implementate in Python (dovrà essere usata una certa cutela per gli oggetti che sovrascrivono gli attributi di accesso); i tipi implementati in C dovranno fornire un attributo `softspace` scrivibile. **Note:** Questo attributo non viene usato per controllare l'istruzione `print`, ma permetterà l'implementazione di `print` per tenere traccia del suo stato interno.

2.3.10 Altri tipi built-in

L'interprete supporta diversi altri tipi di oggetti. La maggior parte di loro supporta solo una o due operazioni.

Moduli

L'unica operazione speciale su un modulo è l'attributo d'accesso: *m*`.`*name*, dove *m* è un modulo e *name* accede ad un nome definito nella tabella dei simboli di *m*. Possono venire assegnati anche attributi ai moduli. (Notate che l'istruzione `import` non è, strettamente parlando, una operazione su un modulo oggetto; `import` *foo* non richiede un modulo chiamato *foo* esistente, piuttosto richiede una *definizione* (esterna) da qualche parte per un modulo chiamato *foo*.)

Un membro speciale di ogni modulo è `__dict__`. Questo è il dizionario contenente la tabella dei simboli dei moduli. Modificando questo dizionario si cambierà l'attuale tabella dei simboli dei moduli, ma non è possibilie assegnare direttamente l'attributo `__dict__` (potete scrivere *m*`.__dict__['a'] = 1`, che definisce *m*`.a` per essere `1`, ma non potete scrivere *m*`.__dict__ = {}`). Modificare direttamente `__dict__` non è raccomandato.

I moduli costruiti nell'interprete vengono scritti come questo: `<module 'sys' (built-in)>`. Se vengono caricati da un file, sono scritti così: `<module 'os' from '/usr/local/lib/python2.3/os.pyc'>`.

Classi ed istanze di classi

Per questi vedete i capitoli 3 e 7 del *Manuale di riferimento di Python*.

Funzioni

Gli oggetti funzione vengono creati dalle definizioni di funzione. L'unica operazione su un oggetto funzione è la chiamata effettuabile su di esso: *func*`(`*lista-argomenti*`)`.

Esistono solo due varietà di oggetti funzione: funzioni built-in e funzioni definite dall'utente. Supportano entrambe le stessa operazione (chiamata della funzione), ma l'implementazione è differente, da qui la differenza dei tipi oggetto.

L'implementazione aggiunge due attributi speciali in sola lettura: *f*`.func_code` è una funzione dell'*oggetto codice* (vedete più avanti) e *f*`.func_globals` è il dizionario usato per lo spazio dei nomi globali della funzione (questo è lo stesso di *m*`.__dict__` dove *m* è il modulo nel quale la funzione *f* era stata definita).

Gli oggetti funzione supportano anche l'acquisizione e l'impostazione arbitraria di attributi, che possono venire usati, per esempio, per attaccare metadata a funzioni. L'attributo regolare notazione-punto viene usato per prendere ed impostare questi attributi. *Notate che la corrente implementazione supporta soltanto gli attributi su funzioni definite dall'utente. Gli attributi delle funzioni sulle funzioni built-in potranno essere supportate in futuro.*

Le funzioni hanno un altro attributo speciale, *f*`.__dict__` (ovvero *f*`.func_dict`), che contiene lo spazio dei nomi usato per supportare gli attributi della funzione. `__dict__` e `func_dict` possono essere accessibili direttamente o impostati da un oggetto dizionario. Un dizionario della funzione non può essere cancellato.

Metodi

I metodi sono funzioni che vengono chiamate usando la notazione attributo. Ne esistono di due tipi: metodi built-in (come `append()` su liste) e metodi di istanze di classe. I metodi bult-in vengono descritti con i tipi che li supportano.

L'implementazione aggiunge due attributi speciali in sola lettura per i metodi delle istanze di classe: *m*`.im_self` è l'oggetto sul quale il metodo opera, e *m*`.im_func` è la funzione che implementa il metodo. La chiamata *m*`(`*arg-1*`, `*arg-2*`, ..., `*arg-n*`)` è completamente equivalente alla chiamata *m*`.im_func(`*m*`.im_self, `*arg-1*`, `*arg-2*`, ..., `*arg-n*`)`.

I metodi delle istanze di classe sono sia *legati* che *slegati*, a seconda che il metodo attraverso con il quale vi si accede sia rispettivamente un'istanza o una classe. Quando un metodo è slegato, il suo attributo `im_self` sarà `None` e se chiamato, un esplicito oggetto `self` dovrà essere passato come primo argomento. In questo caso, `self` deve essere una istanza del metodo di classe slegato (o una sotto classe di quella classe), altrimenti verrà sollevata un'eccezione di tipo `TypeError`.

Come gli oggetti funzione, gli oggetti metodi accettano gli attributi arbitrari che possono venire loro forniti. Tuttavia, poiché i metodi degli attributi vengono attualmente immagazzinati nell'oggetto funzione sottostante (`meth.im_func`), impostare i metodi degli attributi, per metodi sia legati che slegati, non è consentito. Tentare di impostare un metodo per un attributo solleverà un'eccezione `TypeError`. Per impostare i metodi degli attributi, è necessario assegnarli esplicitamente nell'oggetto funzione sottostante:

```
class C:
    def method(self):
        pass

c = C()
c.method.im_func.whoami = 'my name is c'
```

Vedete il *Manuale di riferimento di Python* per ulteriori informazioni.

Oggetti codice

Gli oggetti codice vengono usati per l'implementazione della rappresentazione di codice eseguibile Python "pseudo compilato", come corpo di una funzione. Differiscono dagli oggetti funzione perché non contengono un riferimento al loro ambiente di esecuzione globale. Gli oggetti codice vengono restituiti dalla funzione bult-in `compile()` e possono essere estratti dagli oggetti funzione attraverso il loro attributo `func_code`.

Un oggetto codice può essere eseguito o valutato, passandolo (invece della stringa sorgente) all'istruzione `exec` o alla funzione built-in `eval()`.

Vedete il *Manuale di riferimento di Python* per ulteriori informazioni.

Oggetti tipo

Gli oggetti tipo rappresentano i vari tipi di oggetto. Ad un oggetto tipo si accede attraverso la funzione built-in `type()`. Non ci sono speciali operazioni sui tipi. Il modulo standard `types` definisce i nomi per tutti i tipi standard built-in.

I tipi vengono scritti come questo: `<type 'int'>`.

L'oggetto Null

Questo oggetto viene restituito dalle funzioni che non restituiscono esplicitamente un valore. Non supporta operazioni speciali. Esiste esattamente un oggetto null, chiamato `None` (un nome built-in).

Viene scritto come `None`.

L'oggetto ellittico

Questo oggetto viene usato per estendere la notazione delle fette (vedete il *Manuale di riferimento di Python*). Non supporta operazioni speciali. Esiste esattamente un oggetto ellittico, chiamato `Ellipsis` (un nome built-in).

Viene scritto come `Ellipsis`.

Valori booleani

I valori booleani sono le due costanti oggetto `False` e `True`. Vengono usati per rappresentare i valori di verità (anche se altri valori possono anche essere considerati falsi o veri). Nei contesti numerici (per esempio quando usate come argomento per un operatore aritmetico), possono comportarsi rispettivamente come gli interi 0 e 1. La funzione built-in `bool()` può essere usata per assegnare ogni valore ad un booleano, se il valore può essere interpretato come un valore di verità (vedete la sezione Verifica del valore di verità).

Vengono scritti rispettivamente come `False` e `True`.

Oggetti interni

Vedete il *Manuale di riferimento di Python* per questa informazione. Descrive gli oggetti stack frame, gli oggetti traceback, e gli oggetti fetta.

2.3.11 Attributi speciali

L'implementazione aggiunge pochi attributi speciali in sola lettura a parecchi tipi di oggetto, dove sono rilevanti. Alcuni di questi non vengono riportati dalla funzione built-in `dir()`.

`__dict__`
: Un dizionario o altro oggetto mappa, usato per conservare gli attributi (scrivibili) di un oggetto.

`__methods__`
: **Deprecato dalla versione 2.2.** Usate la funzione built-in `dir()` per visualizzare la lista degli attributi di un oggetto. Questo attributo non sarà più disponibile.

`__members__`
: **Deprecato dalla versione 2.2.** Usate la funzione built-in `dir()` per visualizzare la lista degli attributi di un oggetto. Questo attributo non sarà più disponibile.

`__class__`
: La classe alla quale l'istanza di classe appartiene.

`__bases__`
: La tupla delle classi di base di una classe oggetto. Se non ci sono classi di base, verrà restituita una tupla vuota.

__name__
Il nome della classe o il tipo.

2.4 Eccezioni built-in

Le eccezioni dovrebbero essere classi oggetto. Le eccezioni vengono definite nel modulo `exceptions`. Questo modulo non ha mai bisogno di essere importato esplicitamente: le eccezioni vengono fornite nello spazio dei nomi built-in, alla stessa maniera del modulo `exceptions`.

Note: Nelle versioni passate di Python le eccezioni stringa venivano supportate. In Python 1.5 e nelle versioni più recenti, tutte le eccezioni standard sono state convertite in classi oggetto e gli utenti vengono incoraggiati a fare altrettanto. Le eccezioni stringa solleveranno un'eccezione `PendingDeprecationWarning`. In versioni future, verrà rimosso il supporto per le eccezioni delle stringhe.

Due differenti oggetti stringa con lo stesso valore vengono considerati due eccezioni differenti. Questo è stato fatto per forzare i programmatori ad usare il nome delle eccezioni invece del loro valore stringa, quando specificano i gestori delle eccezioni. Il valore stringa di tutte le eccezioni built-in è il loro nome, ma questo non è un requisito per le eccezioni definite dall'utente o dalle eccezioni definite dai moduli della libreria.

Per le classi delle eccezioni, in una istruzione `try` con una clausola `except` che menziona una classe particolare, quella clausola gestisce anche ogni classe di eccezione derivata da quella classe (ma non la classe eccezione dalla quale è derivata). Due classi di eccezioni non relazionate attraverso una classe derivata non saranno mai equivalenti, anche se hanno lo stesso nome.

Le eccezioni built-in elencate qui sotto possono essere generate dall'interprete o dalle funzioni built-in. Fatta eccezione dove indicato, hanno un "valore associato" indicante la causa dettagliata dell'errore. Questo può essere una stringa o una tupla contenente vari elementi informativi (p.es., un errore nel codice ed una stringa che spiega il codice). Il valore associato è il secondo argomento per l'istruzione `raise`. Per le eccezioni delle stringhe, il valore associato stesso verrà raccolto in una variabile chiamata come il secondo argomento della clausola `except` (se esiste). Per le classi eccezione, quella variabile riceve l'istanza dell'eccezione. Se la classe eccezione deriva dalla classe standard originaria `Exception`, il valore associato è presente come istanza di eccezione dell'attributo `args`, e possibilmente anche su altri attributi.

Il codice dell'utente può sollevare un'eccezione built-in. Questo può venire usato per testare un generatore di eccezioni o per riportare una condizione di errore "appositamente pensata" per la situazione nella quale l'interprete solleva l'eccezione stessa; ma fate attenzione, che non esiste nulla per evitare un errore improprio prodotto nel codice utente.

Le classi delle eccezioni built-in possono essere sotto classi, create per definire nuove eccezioni; i programmatori sono incoraggiati a derivare almeno le nuove eccezioni dalla classe base `Exception`. Ulteriori informazioni sulla definizione delle eccezioni sono disponibili nel *Tutorial Python* sotto la voce "Eccezioni definite dall'utente".

Le seguenti eccezioni vengono usate soltanto come classi di base per altre eccezioni.

exception `Exception`
La classe originaria per le eccezioni. Tutte le eccezioni built-in derivano da questa classe. Anche tutte le eccezioni definite dall'utente dovrebbero derivare da questa classe, ma questo non viene (ancora) imposto. La funzione `str()`, quando applicata ad un'istanza di questa classe (o a più classi derivate), restituisce il valore stringa dell'argomento o degli argomenti, oppure una stringa vuota se nessun argomento è stato passato al costruttore. Quando usata come una sequenza, questa accede agli argomenti dati al costruttore (per retrocompatibilità con il vecchio codice). Gli argomenti sono anche disponibili come una tupla delle istanze dell'attributo `args`.

exception `StandardError`
La classe base per tutte le eccezioni built-in eccetto `StopIteration` e `SystemExit`. `StandardError` stessa è derivata dalla classe originaria `Exception`.

exception `ArithmeticError`
La classe base per quelle eccezioni built-in che vengono sollevate per vari errori aritmetici: `OverflowError`, `ZeroDivisionError`, `FloatingPointError`.

exception `LookupError`

La classe base per le eccezioni che vengono sollevate quando una chiave o un indice usato in una mappa o in una sequenza non è valida: `IndexError`, `KeyError`. Questa può essere sollevata direttamente da `sys.setdefaultencoding()`.

exception `EnvironmentError`
La classe base per le eccezioni che possono verificarsi al di fuori del sistema Python: `IOError`, `OSError`. Quando eccezioni di questo tipo vengono create con una 2-tuple, il primo elemento è disponibile nell'istanza dell'attributo `errno` (si presuppone essere un errore numerico), ed il secondo elemento è disponibile sull'attributo `strerror` (solitamente è il messaggio di errore associato). La tupla stessa è anche disponibile nell'attributo `args`. Nuovo nella versione 1.5.2.

Quando una eccezione di tipo `EnvironmentError` viene istanziata con una 3-tuple, i primi due elementi sono disponibili come sopra, mentre il terzo elemento è disponibile nell'attributo `filename`. Tuttavia, per retrocompatibilità, l'attributo `args` contiene solo 2-tuple dei primi due argomenti del costruttore.

L'attributo `filename` è `None` quando questa eccezione viene creata con un numero di argomenti diverso da 3. Gli attributi `errno` e `strerror` sono anche `None` quando l'istanza viene creata con un numero di argomenti diverso da 2 o 3. In quest'ultimo caso, `args` contiene letteralmente il costruttore di argomenti sotto forma di tupla.

Le seguenti eccezioni sono quelle che vengono attualmente sollevate.

exception `AssertionError`
Viene sollevata quando un'istruzione `assert` fallisce.

exception `AttributeError`
Viene sollevata quando un riferimento ad un attributo o ad un assegnamento fallisce. (Nel caso in cui un oggetto non supporti affatto i riferimenti ad un attributo o gli assegnamenti ad un attributo, viene sollevata un'eccezione di tipo `TypeError`).

exception `EOFError`
Viene sollevata quando una delle funzioni built-in (`input()` o `raw_input()`) incontra un codice di controllo indicante la la fine del file (EOF) senza avere letto alcun dato. (N.B.: I metodi `read()` e `readline()` del file oggetto restituiscono una stringa vuota quando incontrano un EOF.)

exception `FloatingPointError`
Viene sollevata quando un'operazione in virgola mobile fallisce. Questa eccezione è sempre definita, ma può venire sollevata quando Python viene configurato con l'opzione **--with-fpectl**, oppure quando il simbolo `WANT_SIGFPE_HANDLER` viene definito nel file 'pyconfig.h'.

exception `IOError`
Viene sollevata quando una operazione di I/O (come un'istruzione `print`, la funzione built-in `open()` o il metodo di un file oggetto) fallisce per una ragione legata all'I/O, p.es. "file not found" o "disk full".

Questa classe deriva da `EnvironmentError`. Vedete la discussione relativa per maggiori informazioni sugli attributi delle istanze di eccezione.

exception `ImportError`
Viene sollevata quando un'istruzione `import` fallisce nella ricerca della definizione del modulo, o quando l'importazione in un ciclo `from ... import` fallisce la ricerca del nome che deve essere importato.

exception `IndexError`
Viene sollevata quando una sequenza in un sottoscript è fuori dall'intervallo. (Gli indici delle fette vengono troncati silenziosamente per rientrare nell'intervallo consentito; un'eccezione di tipo `TypeError` viene sollevata se un indice non è un intero semplice).

exception `KeyError`
Viene sollevata quando una chiave di mappa (dizionario) non viene trovata nell'insieme delle chiavi esistenti.

exception `KeyboardInterrupt`
Viene sollevata quando l'utente preme la combinazione interrupt (normalmente `Control-C` or `Delete`). Durante l'esecuzione, un controllo per gli interrupt viene fatto regolarmente. Gli interrupt digitati quando una funzione built-in `input()` o `raw_input()` è in attesa di input, sollevano ugualmente questa eccezione.

exception `MemoryError`

Viene sollevata quando una operazione esegue un out of memory ma la situazione può essere ancora recuperata (cancellando alcuni oggetti). Il valore associato è una stringa indicante quale tipo di operazione (interna) è stata eseguita out of memory. Notate che in questa situazione, a causa dell' architettura di gestione della memoria sottostante (la funzione C `malloc()`), l'interprete non sempre riesce ad effettuare un completo recupero; tuttavia solleva un'eccezione, in modo da poter stampare una traccia dello stack, nel caso in cui sia stata provocata da un programma in esecuzione.

exception `NameError`
Viene sollevata quando un nome locale o globale non viene trovato. Questo si applica solo ai nomi non qualificati. Il valore associato è un messaggio di errore che include il nome che non può essere trovato.

exception `NotImplementedError`
Questa eccezione viene derivata da `RuntimeError`. Nelle classi base definite dall'utente, i metodi astratti dovrebbero sollevare questa eccezione quando richiedono delle classi derivate per sovrascrivere il metodo. Nuovo nella versione 1.5.2.

exception `OSError`
Questa classe viene derivata dal `EnvironmentError` e viene usata principalmente come l'eccezione `os.error` del modulo `os`. Vedete `EnvironmentError`, in precedenza, per la descrizione dei possibili valori associati. Nuovo nella versione 1.5.2.

exception `OverflowError`
Viene sollevata quando il risultato di un'operazione aritmetica è troppo grande per essere rappresentato. Questo non può accadere per gli interi long (che dovrebbero semmai sollevare un'eccezione `MemoryError` piuttosto che desistere dall'operazione). A causa della mancanza di uno standard nel trattamento della virgola mobile in C, la maggior parte delle operazioni in virgola mobile non vengono controllate. Per gli interi semplici, tutte le operazioni che possono causare un overflow vengono controllate, tranne quelle di left shift (NdT: operazioni di scorrimento a sinistra), le cui applicazioni tipiche rendono preferibile la perdita di qualche bit piuttosto che sollevare un'eccezione.

exception `ReferenceError`
Questa eccezione viene sollevata quando un riferimento debole ad un proxy, creato dalla funzione `weakref.proxy()`, viene usato per accedere ad un attributo del referente dopo che è stato aggiunto ai dati inutili da elaborare dalla garbage collection. Per maggiori informazioni sui riferimenti deboli, vedete il modulo `weakref`. Nuovo nella versione 2.2: Precedentemente conosciuto come l'eccezione `weakref.ReferenceError` .

exception `RuntimeError`
Viene sollevata quando viene rilevato un errore che non ricade in nessuna delle altre categorie. Il valore associato è una stringa indicante precisamente che cosa è andato storto. (Questa eccezione è sostanzialmente una reliquia da una precedente versione dell'interprete, non viene più molto usata.)

exception `StopIteration`
Viene sollevata dal metodo `next()` di un iteratore, per segnalare che non ci sono ulteriori valori. Deriva da `Exception` piuttosto che da `StandardError`, poiché questo non viene considerato un errore neli suoi impieghi normali. Nuovo nella versione 2.2.

exception `SyntaxError`
Viene sollevata quando il parser incontra un errore di sintassi. Questo può verificarsi nell'istruzione `import`, in un'istruzione `exec`, in una chiamata alle funzioni built-in `eval()` o `input()`, o nella lettura dello script iniziale o dello standard input (anche interattivamente).

Istanze di questa classe hanno attributi `filename`, `lineno`, `offset` e `text` per un più facile accesso ai dettagli. L'`str()` dell'istanza di eccezione restituisce solo il messaggio.

exception `SystemError`
Viene sollevata quando l'interprete trova un errore interno, ma la situazione non sembra così seria da farci abbandonare tutte le speranze. Il valore associato è una stringa indicante che cosa è andato storto (in termini di basso livello).

Dovreste riportare questa stringa all'autore o al manutentore del vostro interprete Python. Siate sicuri di riportare la versione dell'interprete Python (`sys.version`; viene anche stampata all'avvio di una sessione interattiva di Python), l'esatto messaggio di errore (l'eccezione associata al valore) e se possibile il sorgente del programma che ha innescato l'errore.

exception `SystemExit`
Questa eccezione viene sollevata dalla funzione `sys.exit()`. Quando non è stata modificata, l'interprete Python esce; non viene stampata alcuna traccia. Se il valore associato è un semplice intero, specifica lo stato di uscita del sistema (passato come la funzione C `exit()`); se è `None`, lo stato di uscita è zero; se ha un altro tipo (come una stringa), il valore dell'oggetto viene stampato e lo stato di uscita è uno.

Le istanze hanno un attributo `code` che viene impostato per lo stato di uscita proposto o messaggio di errore (predefinito per `None`). Inoltre, questa eccezione deriva direttamente da `Exception` e non da `StandardError`, poiché tecnicamente non è un errore.

Una chiamata a `sys.exit()` viene tradotta in una eccezione in modo tale che i gestori di pulizia (le clausole `finally` delle istruzioni `try`) possano essere eseguiti, e così che il programma per il debug possa eseguire uno script senza correre il rischio di perderne il controllo. La funzione `os._exit()` può venire usata se è assolutamente necessario uscire immediatamente (per esempio, nel processo figlio dopo una chiamata a `fork()`).

exception `TypeError`
Viene sollevata quando un'operazione o una funzione viene applicata ad un oggetto di tipo inappropriato. Il valore associato è una stringa contenente i dettagli sul tipo errato.

exception `UnboundLocalError`
Viene sollevata quando viene creato un riferimento ad una variabile locale in una funzione o metodo, ma nessun valore è stato collegato a quella variabile. È una sotto classe di `NameError`. Nuovo nella versione 2.0.

exception `UnicodeError`
Viene sollevata quando interviene un errore di codifica o decodifica riferita ad Unicode. È una sotto classe di `ValueError`. Nuovo nella versione 2.0.

exception `UnicodeEncodeError`
Viene sollevata quando un errore, con riferimento ad Unicode, interviene durante la codifica. È una sotto classe di `UnicodeError`. Nuovo nella versione 2.3.

exception `UnicodeDecodeError`
Viene sollevata quando un errore, con riferimento ad Unicode, interviene durante la decodifica. È una sotto classe di `UnicodeError`. Nuovo nella versione 2.3.

exception `UnicodeTranslateError`
Viene sollevata quando un errore, con riferimento a Unicode, interviene durante la traduzione. È una sotto classe di `UnicodeError`. Nuovo nella versione 2.3.

exception `ValueError`
Viene sollevata quando un'operazione o funzione built-in ricevono un argomento che ha il tipo giusto ma valore inappropriato, e la situazione non viene descritta da una eccezione più precisa, come `IndexError`.

exception `WindowsError`
Viene sollevata quando interviene un errore specifico di Windows o quando il numero dell'errore non corrisponde ad un valore `errno`. I valori `errno` e `strerror` vengono creati dai valori restituiti dalle funzioni `GetLastError()` e `FormatMessage()` dell'API della piattaforma Windows. È una sotto classe di `OSError`. Nuovo nella versione 2.0.

exception `ZeroDivisionError`
Viene sollevata quando il secondo argomento di una divisione o di una operazione su di un modulo è zero. Il valore associato è una stringa indicante il tipo degli operandi e dell'operazione.

Le seguenti eccezioni vengono usate come categorie di avvertimenti; vedete il modulo `warnings` per maggiori informazioni.

exception `Warning`
Classe base della categoria degli avvertimenti.

exception `UserWarning`
Classe base degli avvertimenti generata dal codice utente.

exception `DeprecationWarning`
Classe base degli avvertimenti relativi a caratteristiche deprecate.

exception `PendingDeprecationWarning`
Classe base degli avvertimenti riguardanti caratteristiche che verranno deprecate in futuro.

exception `SyntaxWarning`
Classe base di avvertimenti riguardanti sintassi dubbie.

exception `RuntimeWarning`
Classe base per avvertimenti riguardanti comportamenti dubbi durante l'esecuzione.

exception `FutureWarning`
Classe base per avvertimenti riguardanti i costrutti che cambieranno semantica in futuro.

La gerarchia delle classi per le eccezioni built-in è:

```
Exception
 +-- SystemExit
 +-- StopIteration
 +-- StandardError
 |    +-- KeyboardInterrupt
 |    +-- ImportError
 |    +-- EnvironmentError
 |    |    +-- IOError
 |    |    +-- OSError
 |    |         +-- WindowsError
 |    +-- EOFError
 |    +-- RuntimeError
 |    |    +-- NotImplementedError
 |    +-- NameError
 |    |    +-- UnboundLocalError
 |    +-- AttributeError
 |    +-- SyntaxError
 |    |    +-- IndentationError
 |    |         +-- TabError
 |    +-- TypeError
 |    +-- AssertionError
 |    +-- LookupError
 |    |    +-- IndexError
 |    |    +-- KeyError
 |    +-- ArithmeticError
 |    |    +-- OverflowError
 |    |    +-- ZeroDivisionError
 |    |    +-- FloatingPointError
 |    +-- ValueError
 |    |    +-- UnicodeError
 |    |        +-- UnicodeEncodeError
 |    |        +-- UnicodeDecodeError
 |    |        +-- UnicodeTranslateError
 |    +-- ReferenceError
 |    +-- SystemError
 |    +-- MemoryError
 +---Warning
      +-- UserWarning
      +-- DeprecationWarning
      +-- PendingDeprecationWarning
      +-- SyntaxWarning
      +-- OverflowWarning
      +-- RuntimeWarning
      +-- FutureWarning
```

2.5 Costanti built-in

Un piccolo numero di costanti vive nello spazio dei nomi built-in. Queste sono:

`False`
: Il valore falso di tipo `bool` (NdT: booleano). Nuovo nella versione 2.3.

`True`
: Il valore vero di tipo `bool`. Nuovo nella versione 2.3.

`None`
: Il solo valore di `types.NoneType`. `None` viene usato frequentemente per rappresentare l'assenza di un valore, come quando argomenti predefiniti non vengono passati ad una funzione.

`NotImplemented`
: Valori speciali che possono essere restituiti da metodi speciali di "confronto ricco" (`__eq__()`, `__lt__()` e simili), per indicare che il confronto non è implementato, con rispetto per l'altro tipo.

`Ellipsis`
: Valore speciale usato in congiunzione con la sintassi estesa delle fette.

CAPITOLO

TRE

Servizi runtime Python

I moduli descritti in questo capitolo forniscono una vasta gamma di servizi relativi all'interprete Python e la sua interazione con il suo ambiente. Di seguito una veduta d'insieme:

sys	Accesso a parametri e funzioni specifiche del sistema in uso.
gc	Interfaccia per il rilevatore ciclico garbage collector.
weakref	Supporto per riferimenti deboli e dizionari deboli.
fpectl	Controllo per la gestione delle eccezioni nei numeri in virgola mobile.
atexit	Registrazione ed esecuzione di funzioni di pulizia.
types	Nomi per i tipi built-in.
UserDict	Wrapper per classi di oggetti dizionario.
UserList	Wrapper per classi di oggetti lista.
UserString	Wrapper per classi di oggetti stringa.
operator	Tutti gli operatori standard come le funzioni built-in.
inspect	Estrarre informazioni e codice sorgente da oggetti in tempo reale.
traceback	Stampa o recupera la traccia dello stack.
linecache	Questo modulo fornisce un accesso casuale a righe individuali contenute in un file di testo.
pickle	Converte oggetti Python in un flusso di dati in byte e li ripristina.
cPickle	Version of `pickle` più veloce, ma non derivabile in sotto classi.
copy_reg	Funzioni di supporto al registro di `pickle`.
shelve	Persistenza degli oggetti Python.
copy	Operazioni di copia superficiale e profonda.
marshal	Converte oggetti Python in un flusso di byte e li ripristina (con differenti vincoli).
warnings	Emette messaggi di avviso, e ne controlla la disposizione.
imp	Accesso all'implementazione dell'istruzione `import`.
pkgutil	Utility per il supporto delle estensioni dei package.
code	Classi di base per interpreti Python interattivi.
codeop	Compilare codice (incompleto, in talune circostanze) Python.
pprint	Data pretty printer.
repr	Implementazione alternativa di `repr()` con limiti sulla dimensione.
new	Interfaccia alla creazione in runtime dell'implementazione di oggetti.
site	Un metodo standard per riferire i moduli alla configurazione specifica della piattaforma.
user	Un metodo standard per il riferimento ai moduli specifici dell'utente.
__builtin__	L'insieme delle funzioni built-in.
__main__	L'ambiente dove vengono eseguiti gli script di alto livello.
__future__	Definizione delle istruzioni future

3.1 `sys` — Parametri e funzioni specifiche per il sistema

Questo modulo fornisce l'accesso ad alcune variabili usate o mantenute dall'interprete, e a funzioni che interagiscono fortemente con l'interprete stesso. È sempre disponibile.

argv
: La lista degli argomenti della riga di comando passati a uno script Python. `argv[0]` è il nome dello script

(dipende dal sistema operativo che questi venga indicato da un percorso completo o meno). Se il comando viene eseguito usando l'opzione **-c** dalla riga di comando dell'interprete, `argv[0]` viene impostata con il contenuto della stringa `'-c'`. Se nessun nome di script viene passato all'interprete Python, `argv` ha lunghezza zero.

`byteorder`
Un indicatore dell'ordine nativo del byte. Questo può avere il valore `'big'` su piattaforme big-endian (il byte più significativo prima) e `'little'` su piattaforme little-endian (byte meno significativo prima). Nuovo nella versione 2.0.

`builtin_module_names`
Una tupla di stringhe che riporta i nomi di tutti i moduli compilati dentro questo interprete Python. (Questa informazione non è disponibile in nessun altro modo – `modules.keys()` riporta solo i moduli importati).

`copyright`
Una stringa contenente il copyright proprio dell'interprete Python.

`dllhandle`
Intero indicante la gestione della DLL di Python. Disponibilità: Windows.

`displayhook`(*valore*)
Se *valore* è diverso da `None`, questa funzione lo stampa su `sys.stdout` e lo salva in `__builtin__._`.

`sys.displayhook` viene chiamata sul risultato della valutazione di un'espressione inserita in una sessione interattiva di Python. La visualizzazione di questi valori può essere personalizzata assegnando a `sys.displayhook` un'altra funzione con un solo argomento.

`excepthook`(*tipo, valore, traceback*)
Questa funzione stampa su `sys.stderr` un dato traceback [NdT:traccia dello stack] e l'eccezione che lo ha creato.

Quando un'eccezione viene sollevata e non raccolta, l'interprete chiama `sys.excepthook` con tre argomenti: la classe dell'eccezione, l'istanza dell'eccezione ed un oggetto traceback. In una sessione interattiva ciò si verifica appena prima che il controllo venga restituito al prompt; in un programma Python, invece, appena prima che il programma stesso termini. La gestione di eccezioni di così alto livello può essere personalizzata assegnando a `sys.excepthook` un'altra funzione con tre argomenti.

`__displayhook__`
`__excepthook__`
Questi oggetti contengono i valori originali di `displayhook` e `excepthook` all'avvio del programma. Essi vengono salvati in modo tale che `displayhook` e `excepthook` possano venire ripristinati nel caso vengano sostituiti con degli oggetti corrotti.

`exc_info`()
Questa funzione restituisce una tupla di tre valori, che forniscono informazioni riguardanti l'eccezione correntemente gestita. L'informazione restituita è specifica sia del thread che del frame dello stack corrente. Se quest'ultimo non sta gestendo un'eccezione, l'informazione viene presa dal frame dello stack chiamante, o dal suo chiamante, e così via fino a quando viene trovato un frame dello stack che sta gestendo un'eccezione. Qui, "che sta gestendo un'eccezione" deve intendersi "che sta eseguendo o che ha eseguito una clausola d'eccezione". Di ogni frame dello stack, è accessibile soltanto l'informazione riguardante l'eccezione gestita più di recente.

Se nessuna eccezione viene momentaneamente gestita da qualche parte sullo stack, viene restituita una tupla contenente tre valori `None`. Altrimenti, i valori restituiti sono `(`*tipo*`,` *valore*`,` *traceback*`)`. Il loro significato è il seguente: *tipo* riceve il tipo dell'eccezione che si sta gestendo (un oggetto classe); *valore* riceve il cosiddetto parametro dell'eccezione (il suo *valore associato* o il secondo argomento da `sollevare`, che è sempre un'istanza di classe se il tipo dell'eccezione è un oggetto classe); *traceback* riceve un oggetto di tipo traceback (cfr. Reference Manual) che comprende lo stack della chiamata a partire dal punto in cui si è verificata originariamente l'eccezione.

Se viene chiamata `exc_clear()`, questa funzione restituisce tre valori `None` fino a quando non viene sollevata un'altra eccezione nel thread corrente oppure lo stack d'esecuzione non ritorna a un frame (d'esecuzione) dove viene gestita un'altra eccezione.

Avvertenze: Assegnare il valore di ritorno di *traceback* ad una variabile locale in una funzione che stia gestendo un'eccezione provocherà un riferimento circolare. Ciò impedirà che qualsiasi cosa, referenziata

da una variabile locale nella stessa funzione oppure da traceback, venga trattata dal garbage-collector. Dal momento che la maggior parte delle funzioni non ha bisogno di accedere al valore di traceback, la soluzione migliore è quella di usare qualcosa del tipo `exctype, value = sys.exc_info()[:2]`, per estrarre soltanto il tipo e il valore dell'eccezione. Se si ha veramente bisogno del traceback, assicuratevi di eliminarlo dopo l'uso (la cosa migliore sarebbe un'istruzione `try` ... `finally`) oppure chiamare `exc_info()` all'interno di una funzione che non gestisca lei stessa un'eccezione. **Note:** A partire dalla versione 2.2 di Python, tali circoli viziosi vengono automaticamente corretti quando la garbage collection viene abilitata e divengono irrangiungibili, ma rimane comunque più efficiente evitarne la creazione.

`exc_clear``()`
Questa funzione cancella tutte le informazioni relative all'eccezione corrente oppure all'ultima verificatasi nel thread corrente. Dopo una chiamata a questa funzione, `exc_info()` restituirà tre valori `None`, fino a quando non viene sollevata un'altra eccezione nel thread corrente oppure lo stack d'esecuzione ritorna ad un frame (d'esecuzione) dove si sta gestendo un'altra eccezione.

Questa funzione è necessaria soltanto in poche ed oscure situazioni. Queste includono sistemi di registrazione e gestione di errori, che restituiscono informazioni sull'ultima eccezione oppure su quella corrente. La funzione può anche venire usata anche per cercare di liberare risorse e dare l'avvio alla terminazione di un oggetto, sebbene non esiste garanzia su quali oggetti verranno liberati, se ve ne sono. Nuovo nella versione 2.3.

`exc_type`
`exc_value`
`exc_traceback`
Deprecato dalla versione 1.5. Usate piuttosto `exc_info()`.

Poichè queste sono variabili globali, non sono specifiche del corrente thread, quindi il loro uso non è sicuro in un programma multi-threaded. Quando nessuna eccezione viene gestita, `exc_type` viene impostato su `None` e gli altri due restano indefiniti.

`exec_prefix`
Una stringa che fornisce il prefisso in cui viene situata la directory specifica dove i file Python, in funzione della piattaforma in uso, vengono installati. In modo predefinito, la stringa è `'/usr/local'`. Può venire impostata al momento della compilazione con l'argomento **--exec-prefix** nello script **configure**. Specificatamente, tutti i file di configurazione (es. il file header 'pyconfig.h') vengono installati nella directory `exec_prefix + '/lib/python`*version*`/config'` e i moduli di libreria condivisi in `exec_prefix + '/lib/python`*version*`/lib-dynload'`, dove *version* è uguale a `version[:3]`.

`executable`
Una stringa che fornisce il nome dell'eseguibile binario per l'interprete Python, nei sistemi dove questo ha senso.

`exit``([`*arg*`])`
Esce da Python. Viene implementata sollevando l'eccezione `SystemExit`, cosicché le azioni di pulizia specificate dalle clausole finally delle istruzioni `try` vengono eseguite, ed è possibile intercettare il tentativo di uscita ad un livello più esterno. L'argomento facoltativo *arg* può essere un intero che fornisce lo status d'uscita (predefinito è zero) o un altro tipo di oggetto. Se è un intero, zero viene considerato "terminato con successo" e qualsiasi valore diverso da zero viene considerato "terminato in modo anomalo" da shell e simili. Molti sistemi richiedono che tale valore sia compreso nell'intervallo 0-127, altrimenti producono risultati indefiniti. Alcuni sistemi hanno una convenzione per assegnare uno specifico significato ai codici di uscita, ma sono generalmente poco sviluppati; i programmi UNIX generalmente usano il 2 per errori di sintassi da riga di comando e 1 per tutti gli altri tipi di errore. Se un altro tipo di oggetto viene passato, `None` è equivalente a passare zero, qualsiasi altro oggetto viene stampato sullo `sys.stderr`, e il risultato nel codice di uscita è 1. In particolare, `sys.exit(messaggio di errore)` è la maniera più veloce di uscire da un programma quando si verifica un errore.

`exitfunc`
Questo valore non viene attualmente definito dal modulo, ma può venire impostato dall'utente (o da un programma) per specificare un'azione di pulizia all"uscita del programma. Quando impostato, dovrebbe essere una funzione senza parametri. Questa funzione verrà chiamata quando l'interprete esce. Solamente una funzione può venire installata in questo modo; per consentire che alla fine del programma vengano chiamate funzioni multiple, usate il modulo `atexit`. **Note:** La funzione exit non viene chiamata quando

il programma è chiuso da un segnale, quando viene intercettato un errore fatale in Python, o quando viene chiamata `os._exit()`.

`getcheckinterval``()`
Restituisce il "check interval" dell'interprete; vedete `setcheckinterval()`. Nuovo nella versione 2.3.

`getdefaultencoding``()`
Restituisce il nome della corrente stringa predefinita per la codifica, usata dall'implementazione Unicode. Nuovo nella versione 2.0.

`getdlopenflags``()`
Restituisce il corrente valore delle opzioni [NdT: flag] che vengono usate per le chiamate `dlopen()`. Le opzioni costanti vengono definite nei moduli `dl` e `DLFCN`. Disponibilità: UNIX. Nuovo nella versione 2.2.

`getfilesystemencoding``()`
Restituisce il nome della codifica usata per convertire i nomi Unicode dei file nei nomi file di sistema, o `None` se viene usata la codifica predefinita di sistema. Il valore del risultato dipende dal sistema operativo:

- Su Windows 9x, la codifica è "mbcs".
- Su Mac OS X, la codifica è "utf-8".
- Su Unix, la codifica dipende dalle preferenze dell'utente, secondo il risultato di nl_langinfo(CODESET), o None se nl_langinfo(CODESET) fallisce.
- Su Windows NT+, i nomi dei file sono nativi Unicode, quindi non viene eseguita alcuna conversione.

Nuovo nella versione 2.3.

`getrefcount``(`*oggetto*`)`
Restituisce il conteggio dei riferimenti dell'*oggetto*. Il conteggio risultante è generalmente più alto di quello che potreste aspettarvi, perché include il riferimento (temporaneo) ad un argomento di `getrefcount()`.

`getrecursionlimit``()`
Restituisce il valore attuale del limite di ricorsione, ovvero la profondità massima dello stack dell'interprete Python. Tale limite previene che una ricorsione infinita provochi uno stack overflow di C, e un crash di Python. Questo parametro può essere impostato con `setrecursionlimit()`.

`_getframe``([`*profondità*`])`
Restituisce un oggetto di tipo frame dallo stack delle chiamate. Se viene specificato il parametro facoltativo intero della variabile *profondità*, restituisce il frame che sta al di sotto della cima dello stack del numero di chiamate indicato. Se il valore è più grande della profondità dell'intero stack, allora viene sollevata l'eccezione `ValueError`. Il valore predefinito per la variabile *profondità* è zero e in questo caso viene restituito il frame in cima allo stack.

Questa funzione dovrebbe essere utilizzata solamente per scopi speciali o interni (all'interprete).

`getwindowsversion``()`
Restituisce una tupla che contiene cinque componenti, che descrivono la versione di Windows attualmente in esecuzione. Gli elementi sono *major*, *minor*, *build*, *platform* e *text*. *text* contiene una stringa mentre tutti gli altri valori sono interi.

platform può assumere uno dei seguenti valori:

0 (`VER_PLATFORM_WIN32s`)Win32s o Windows 3.1.

1 (`VER_PLATFORM_WIN32_WINDOWS`)Windows 95/98/ME

2 (`VER_PLATFORM_WIN32_NT`)Windows NT/2000/XP

3 (`VER_PLATFORM_WIN32_CE`)Windows CE.

Questa funzione incapsula la funzione Win32 `GetVersionEx()`; consultate la documentazione Microsoft per ulteriori informazioni su questi valori.

Disponibilità: Windows. Nuovo nella versione 2.3.

`hexversion`
Il il numero di versione codificato come un singolo intero. Viene garantito che aumenti ad ogni versione, comprendendo un adeguato supporto per versioni che non siano di produzione. Per esempio, per verificare che l'interprete Python sia almeno alla versione 1.5.2 utilizzate:

```
if sys.hexversion >= 0x010502F0:
    # uso di qualche funzionalità avanzata
    ...
else:
    #  usa un'implementazione alternativa o l'invio di un
    #+ avvertimento all'utente
    ...
```

Questa viene chiamata '`hexversion`' (letteralmente: versione esadecimale) poiché assomiglia a qualcosa di comprensibile solo quando viene vista attraverso la funzione built-in `hex()`. Per una codifica più leggibile potete utilizzare il valore di `version_info`. Nuovo nella versione 1.5.2.

`last_type`
`last_value`
`last_traceback`

Queste tre variabili non vengono sempre definite; vengono impostate quando un'eccezione non viene gestita e l'interprete stampa un messaggio d'errore e una traceback dello stack. L'utilizzo per cui sono state introdotte consiste nel permettere l'importazione di un modulo debugger durante l'utilizzo interattivo, ed iniziare un debugging successivo al crash senza dover rieseguire il comando che ha causato l'errore. L'utilizzo tipico è '`import pdb; pdb.pm()`' per far partire il debugger; per ulteriori informazioni vedete il capitolo 9, "Il debugger di Python".

Il significato delle variabili è lo stesso di quello dei valori restituiti da `exc_info()`, vista precedentemente. Poiché esiste un solo thread interattivo, la thread-safety non è un problema per queste variabili, a differenza di quanto accade per exc_type `exc_type` etc..

`maxint`

Il più grande numero intero positivo supportato dal tipo intero standard di Python. Questo valore è almeno `2**31-1`. Il più grande numero intero negativo è `-maxint-1` — l'asimmetria è frutto dell'aritmetica binaria in complemento a 2.

`maxunicode`

Un intero che rappresenta il più grande code point per un carattere Unicode. Questo valore dipende dall'opzione di configurazione che specifica se i caratteri Unicode vengono memorizzati come UCS-2 o UCS-4.

`modules`

Questo è un dizionario che mappa i nomi dei moduli nei moduli che sono già stati caricati. Il dizionario può venire manipolato per forzare il ricaricamento degli stessi moduli o per altri trucchi del genere. Ricordate che eliminare un modulo da questo dizionario *non* è la stessa cosa che invocare il metodo `reload()` sul corrispondente oggetto di tipo modulo.

`path`

Una lista di stringhe che specificano i percorsi di ricerca per trovare i moduli richiesti. Vengono inizializzate dalla variable d'ambiente PYTHONPATH, oltre che predefinite in funzione dell'installazione effettuata.

Inizializzata all'avvio del programma, il primo elemento di questa lista, `path[0]`, è la directory contenente lo script utilizzato per invocare l'interprete Python. Se la directory dello script non è disponibile (p.es. se l'interprete viene invocato interattivamente, o se lo script viene letto direttamente dallo standard input), `path[0]` è una stringa vuota, che indica a Python di cercare i moduli prima nella directory corrente. Attenzione che la directory dello script viene inserita *prima* delle altre voci risultanti in PYTHONPATH.

Un programma è libero di modificare questa lista per i propri scopi.

Modificato nella versione 2.3: le stringhe Unicode non vengono più ignorate.

`platform`

Questa stringa contiene l'identificatore del tipo di piattaforma, p.es. `'sunos5'` o `'linux1'`. Può venire utilizzata per aggiungere componenti specifici del sistema a `path`, per istanza.

`prefix`

Una stringa contenente la directory specifica nel sistema dove vengono installati i file di Python indipendenti dalla piattaforma; normalmente, questa stringa contiene `'/usr/local'`. Può venire impostata in fase di compilazione con l'opzione **--prefix** nello script di **configure**. La principale collezione dei mo-

duli di libreria di Python viene installata nella directory `prefix + '/lib/python`*version*`'`, mentre gli header file indipendenti dal sistema (tutti, eccetto 'pyconfig.h') vengono collocati in `prefix + '/include/python`*version*`'`, dove *version* è uguale a `version[:3]`.

ps1
ps2

Stringhe che specificano il prompt principale e quello secondario dell'interprete. Vengono definite solo se l'interprete è nel modo interattivo. Il loro valore iniziale, in questo caso, è `'>>> '` e `'... '`. Se un oggetto non stringa viene assegnato ad entrambe le variabili, la funzione `str()` viene valutata ogni volta che l'interprete si prepara a leggere un nuovo comando interattivo; queste stringhe possono venire usate per implementare un prompt dinamico.

setcheckinterval(*intervallo*)

Imposta il "check interval" dell'interprete. Questo valore intero determina quanto spesso l'interprete deve controllare delle cose che sono soggette ad un comportamento periodico, come l'attivazione dei thread e la gestione dei segnali. Abitualmente è `100`, il che significa che il controllo viene fatto ogni 100 istruzioni Python virtuali. Impostandolo a un valore maggiore è possibile incrementare le prestazioni per i programmi che fanno uso di thread. Impostandolo a un valore `<=` 0, il controllo viene effettuato ad ogni istruzione virtuale, massimizzando la responsività a scapito di un maggiore carico di lavoro.

setdefaultencoding(*nome*)

Imposta il valore corrente di codifica usata dall'implementazione Unicode. Se *nome* non corrisponde a nessuna tra le codifiche disponibili, viene sollevata l'eccezione `LookupError`. Questa funzione è designata per venire usata soltanto dall'implementazione del modulo `site`, e dove necessario, da `sitecustomize`. Una volta usata dal modulo `site`, viene rimossa dallo spazio dei nomi del modulo `sys`. Nuovo nella versione 2.0.

setdlopenflags(*n*)

Imposta le opzioni usate dall'interprete per le chiamate `dlopen()`, come quando l'interprete carica dei modulo di estensione. Tra i vari compiti, abilita un metodo (lento) di risoluzione dei simboli nel momento in cui un modulo viene importato, se chiamato come `sys.setdlopenflags(0)`. Per condividere i simboli attraverso i moduli di estensione, si deve chiamare `sys.setdlopenflags(dl.RTLD_NOW | dl.RTLD_GLOBAL)`. I nomi simbolici per le opzioni del modulo possono essere trovate sia nel modulo `dl` che nel modulo `DLFCN`. Se `DLFCN` non è disponibile, può essere generato da '/usr/include/dlfcn.h' usando lo script **h2py**. Disponibilità: UNIX. Nuovo nella versione 2.2.

setprofile(*profilefunc*)

Imposta la funzione del profile di sistema , che permette di implementare un profiler per il codice sorgente Python direttamente in Python stesso. Vedete nel capitolo 10 per maggiori informazioni circa il profiler di Python. La funzione di profile di sistema viene invocata allo stesso modo della funzione di sistema trace (verificate `settrace()`), ma non viene chiamata per ogni riga eseguita del codice (soltanto sulle chiamate e sui valori restituiti, ma un evento di restituzione viene riportato solo quando un'eccezione viene precedentemente impostata). La funzione è specifica per i thread, ma non c'è modo per il profiler di conoscere i segnali che contestualmente vengono scambiati tra i thread, e così non ha senso usarla quando sono presenti thread multipli. Inoltre, il risultato del suo valore non viene utilizzato, cosicché può restituire semplicemente `None`.

setrecursionlimit(*limite*)

Imposta la massima profondità dello stack utilizzato dall'interprete Python a *limite*. Questo limite previene ricorsioni infinite, cause di overflow dello stack C e crash di Python.

Il massimo limite possibile dipende dalla piattaforma in uso. Un utente può aver bisogno di impostare un limite più alto quando ha un programma che richieda una ricorsione profonda, e una piattaforma che supporti un limite più alto. Questo dovrebbe essere fatto con cura, perché un limite troppo alto può causare un crash.

settrace(*tracefunc*)

Imposta la funzione trace di sistema, che consente di implementare in Python un debugger di codice sorgente Python. Vedete la sezione 9.2, "Come funziona" nel capitolo sul debugger Python. La funzione è specifica per i thread; per fare in modo che il debbuger supporti il multi-threads, deve essere registrata tramite `settrace()`, per ciascun thread che viene esaminato dal debugger.

stdin

stdout
stderr
File oggetto corrispondenti a standard input, standard output e standard error dell'interprete. `stdin` viene usato per tutti gli input dell'interprete, ad eccezione degli script che non includono le chiamate `input()` e `raw_input()`. `stdout` viene usato per l'output di `print`, per le istruzioni, e per gli inviti di `input()` e `raw_input()`. L'invito dell' interprete ed i suoi messaggi di errore (la maggior parte di essi) vengono inviati allo `stderr`. `stdout` e `stderr` non hanno bisogno di essere dei file oggetto built-in: ogni oggetto è accettabile finché possiede un metodo `write()` che prenda una stringa argomento. (Il cambiamento di questi oggetti non influisce sui flussi I/O standard, per processi eseguiti da `os.popen()`, `os.system()`, o della famiglia di funzioni di `exec*()` nel modulo `os`.)

__stdin__
__stdout__
__stderr__
Questi oggetti contengono i valori originali di `stdin`, `stderr` e `stdout` all'inizio del programma. Vengono usati durante la finalizzazione e potrebbero essere utili per ripristinare gli attuali file che sappiamo essere file oggetto funzionanti, nel caso in cui siano stati sovrascritti con un oggetto corrotto.

tracebacklimit
Quando questa variabile viene impostata ad un valore intero, determina il numero massimo dei livelli della informazione di traceback stampata quando si verifica un'eccezione non gestibile. Il valore predefinito è `1000`. Quando viene impostata a `0` o inferiore, tutta l'informazione della traceback viene soppressa e viene stampata soltanto il tipo di eccezione ed il valore.

version
Una stringa contenente la versione dell'interprete Python, insieme alle informazioni supplementari del numero di compilazione e del compilatore utilizzato. Ha un valore nel formato *'version (#build_number, build_date, build_time) [compiler]'*. I primi tre caratteri vengono usati per identificare la versione nelle directory d'installazione (dove appropriato su ciascuna piattaforma). Un esempio:

```
>>> import sys
>>> sys.version
'1.5.2 (#0 Apr 13 1999, 10:51:12) [MSC 32 bit (Intel)]'
```

api_version
La versione dell'API C per questo interprete. I programmatori possono trovarlia utile quando la versione del debbuging entra in conflitto tra Python ed i moduli di estensione. Nuovo nella versione 2.3.

version_info
Una tupla contenente i cinque componenti del numero della versione: *major*, *minor*, *micro*, *releaselevel* e *serial*. Tutti i valori eccetto *releaselevel* sono interi; il livello della release viene identificato come `'alpha'`, `'beta'`, `'candidate'` o `'final'`. Il `version_info` corrispondente alla versione Python 2.0 è: `(2, 0, 0, 'final', 0)`. Nuovo nella versione 2.0.

warnoptions
Questo è il dettaglio dell'implementazione dell'ambiente degli avvertimenti; non modificate questo valore. Fate riferimento al modulo `warnings` per avere più informazioni sull'ambiente degli avvertimenti.

winver
Il numero di versione usato per formare le chiavi di registro su piattaforme Windows. Viene immagazzinata come stringa di risorsa 1000 nella DLL di Python. Il valore viene normalmente costituito dai primi tre caratteri di `version`. Viene fornito nel modulo `sys` per scopi informativi; la modifica di questo valore non effetti sulle chiavi di registro usate da Python. Disponibilità: Windows.

Vedete anche:

Modulo `site` (sezione 3.28):
Questo descrive come usare i files .pth per estendere `sys.path`.

3.2 gc — L'interfaccia Garbage Collector

Il modulo `gc` è disponibile solo se l'interprete viene compilato con l'opzione del garbage collector (NdT: raccoglitore della garbage) ciclico (abilitata in modo predefinito). Se non viene abilitata, viene sollevata l'eccezione `ImportError` nel tentativo di importare questo modulo.

Il modulo fornisce un' interfaccia al garbage collector facoltativo. Prevede le possibilità di disabilitare il raccoglitore, di regolare la frequenza di raccolta e di impostare le opzioni di debug. Fornisce inoltre accesso ad oggetti irragiungibili che il raccoglitore trova ma non può liberare. Poichè il raccoglitore integra il conteggio dei riferimenti già usati in Pyhton, è possibile disabilitarlo se si è sicuri che il programma non creerà dei riferimenti ciclici. La raccolta automatica può venire disabilitata chiamando `gc.disable()`. Per fare il debug di un programma difettoso chiamate `gc.set_debug(gc.DEBUG_LEAK)`.

Il modulo `gc` fornisce le seguenti funzioni:

`enable``()`
Abilita automaticamente la garbage collection (NdT: raccolta garbage).

`disable``()`
Disabilita automaticamente la garbage collection.

`isenabled``()`
Restituisce vero se la garbage collection automatica viene abilitata.

`collect``()`
Avvia una raccolta completa. Tutte le generazioni vengono esaminate e viene restituito il numero degli oggetti irragiungibili.

`set_debug``(`*flags*`)`
Imposta le opzioni di debugging della garbage collection. Le informazioni di debug verranno scritte su `sys.stderr`. Vedete più avanti per un elenco delle opzioni di debug che possono venire combinate usando le operazioni sui bit per il controllo del debug.

`get_debug``()`
Restituisce le opzioni di debug correntemente impostate.

`get_objects``()`
Restituisce una lista di tutti gli oggetti tracciati dal raccoglitore, ad esclusione della lista restituita. Nuovo nella versione 2.2.

`set_threshold``(`*threshold0*`[`*, threshold1*`[`*, threshold2*`]])`
Imposta la soglia della garbage collection (la frequenza della raccolta). Impostando *threshold0* a zero la raccolta viene disabilitata.

Il GC classifica gli oggetti in tre generazioni, a seconda del numero di raccolte a cui sono sopravvissuti. I nuovi oggetti vengono posti nella generazione più giovane (generazione `0`). Se un oggetto sopravvive alla raccolta, viene spostato nella successiva generazione più vecchia. Poichè la generazione `2` è la più vecchia, gli oggetti in quella generazione rimangono lì dopo una raccolta. Per poter decidere il momento necessario per avviare la raccolta, il raccoglitore tiene traccia del numero degli oggetti allocati e deallocati dall'ultima raccolta. Quando il numero di allocazioni meno il numero delle deallocazioni è superiore a *threshold0*, la raccolta inizia. Inizialmente viene esaminata solo la generazione `0`. Se la generazione `0` viene stata esaminata un numero di volte superiore a *threshold1* da quando è stata esaminata la generazione `1`, allora anche la generazione `1` viene presa in considerazione. In maniera simile *threshold2* controlla il numero di raccolte della generazione `1` prima di raccogliere la generazione `2`.

`get_threshold``()`
Restituisce le soglie correnti di acquisizione come una tupla di valori `(`*threshold0*`,` *threshold1*`,` *threshold2*`)`.

`get_referrers``(`**objs*`)`
Restituisce la lista di oggetti che vengono direttamente riferiti a ciascuno degli objs. Questa funzione individuerà solo quei contenitori che supportano la garbage collection; i tipi di estensione che si possono riferire ad altri oggetti ma che non supportano la garbage collection non verranno individuati.

Notate che gli oggetti che sono già stati dereferenziati, ma che vivono nei cicli e che non sono ancora stati raccolti dal garbage collector possono venire elencati tra i risultati riferibili. Per ottenere solo oggetti correntemente attivi chiamate `collect()` prima di chiamare `get_referrers()`.

Dovete fare attenzione quando usate oggetti restituiti da `get_referrers()` perché alcuni di questi po-

trebbero essere ancora in costruzione e quindi in uno stato temporaneamente non valido. Evitate di usare `get_referrers()` per altri scopi oltre al debugging.

Nuovo nella versione 2.2.

`get_referents`(**objs*)
Restituisce una lista di oggetti direttamente riferiti a ciascuno degli argomenti. I riferimenti restituiti sono quegli oggetti esaminati dagli argomenti dei metodi di livello C `tp_traverse` (se ci sono), e possono non tutti momentaneamente essere oggetti direttamente raggiungibili. I metodi `tp_traverse` vengono supportati solo dagli oggetti che supportano la garbage collection, e vengono richiesti solamente per esaminare oggetti che possono venire impiegati all'interno di un ciclo. Così, per esempio, se un intero è direttamente raggiungibile da un argomento, quell'oggetto intero può o meno apparire nell'elenco dei risultati.

Nuovo nella versione 2.3.

La variabile seguente viene fornita per accessi in sola lettura (potete cambiare il suo valore, ma non dovreste riassegnarla):

`garbage`
Una lista di oggetti che il raccoglitore trova essere irraggiungibili ma che potrebbero non essere liberati (oggetti non raccoglibili). In modo predefinito, la lista contiene solo oggetti con metodo `__del__()`.[1] Oggetti che hanno metodi `__del__()` e che sono parte di un ciclo di riferimento, fanno sì che un intero ciclo di riferimenti non sarà raccoglibile, inclusi gli oggetti non necessariamente presenti nel ciclo ma raggiungibili solo da esso. Python non raccoglie automaticamente questi cicli, perché, normalmente, non è possibile per Python stesso conoscere un ordine sicuro per lanciare il metodo `__del__()`. Se conoscete un sistema sicuro, potete forzare l'operazione esaminando la *garbage* list, ed interrompendo esplicitamente i cicli dovuti agli oggetti presenti nella lista. Notate che questi oggetti vengono mantenuti vivi fintanto che si trovano nella *garbage* list, per cui, per eliminarli definitivamente dovranno essere rimossi dalla *garbage* list stessa. Per esempio, dopo aver interrotto dei cicli, richiamate del `del gc.garbage[:]` per svuotare la lista. È generalmente preferibile evitare il problema, non creando cicli contenenti oggetti con metodi `__del__()`, e *garbage* può venire esaminata per verificare che questi cicli non vengano creati.

Se `DEBUG_SAVEALL` è stata impostata, tutti gli oggetti non raggiungibili verranno aggiunti alla lista invece che liberati.

Le seguenti costanti vengono fornite per venire usate con `set_debug()`:

`DEBUG_STATS`
Stampa le statistiche durante la raccolta. Questa informazione può essere utile mentre si regola la frequenza di raccolta.

`DEBUG_COLLECTABLE`
Stampa informazioni sugli oggetti riciclabili individuati.

`DEBUG_UNCOLLECTABLE`
Stampa informazioni sugli oggetti non riciclabili individuati (oggetti che risultano essere non raggiungibili ma che non possono veniree liberati dal raccoglitore). Questi oggetti verranno aggiunti alla `garbage` list.

`DEBUG_INSTANCES`
Quando `DEBUG_COLLECTABLE` o `DEBUG_UNCOLLECTABLE` viene impostata, stampa informazioni circa le istanze di oggetti trovate.

`DEBUG_OBJECTS`
Quando `DEBUG_COLLECTABLE` o `DEBUG_UNCOLLECTABLE` viene impostata, stampa informazioni circa gli oggetti istanze diversi da quelli rinvenuti.

`DEBUG_SAVEALL`
Quando impostata, tutti gli oggetti non raggiungibili trovati verranno aggiunti a *garbage* piuttosto che venire liberati. Questo comportamento può risultare utile per fare il debug di un programma difettoso.

`DEBUG_LEAK`
Le opzioni per il debugging necessarie al raccoglitore per stampare le informazioni circa un programma difettoso (uguali a `DEBUG_COLLECTABLE | DEBUG_UNCOLLECTABLE | DEBUG_INSTANCES | DEBUG_OBJECTS | DEBUG_SAVEALL`).

[1] Prima di Python 2.2, la lista conteneva tutti gli oggetti istanziati in cicli non raggiungibili, non soltanto quelli con metodi `__del__()`.

3.3 `weakref` — Riferimenti deboli

Nuovo nella versione 2.1.

Il modulo `weakref` permette al programmatore Python di creare *riferimenti deboli* agli oggetti.

Di seguito, il termine *referente* indica l'oggetto a cui ci si riferisce con un riferimento debole.

Un riferimento debole a un oggetto non è sufficiente per tenere un oggetto in vita: quando l'unico riferimento restante a un referente è un riferimento debole, la garbage collection è libera di distruggere il referente e riusare la sua memoria per qualche altra cosa. Il principale uso di un riferimento debole è quello di implementare cache o assegnamenti contenenti grandi oggetti, dove è preferibile che un grande oggetto non debba essere tenuto vivo per il solo fatto che appare in una cache o in un assegnamento. Per esempio, se si ha un gran numero di immagini binarie di oggetti, si potrebbe desiderare di associare un nome a ciascuna di esse. Se si utilizza un dizionario Python per assegnare nomi a immagini, o immagini a nomi, gli oggetti immagine potrebbero restare in vita solo perché appaiono come valori o chiavi all'interno dei dizionari. Le classi `WeakKeyDictionary` e `WeakValueDictionary` fornite attraverso il modulo `weakref` rappresentano un'alternativa, usando riferimenti deboli per costruire assegnamenti che non mantengono vivi gli oggetti solo perché essi appaiono in un assegnamento di oggetti. Se, per esempio, un oggetto immagine è un valore in un `WeakValueDictionary`, quando gli ultimi riferimenti a quell'oggetto immagine sono dei riferimenti deboli mantenuti da assegnamenti deboli, la garbage collection può richiedere l'oggetto, e la sua corrispondente registrazione nella mappa dei riferimenti deboli viene semplicemente cancellata.

`WeakKeyDictionary` e `WeakValueDictionary` usano riferimenti deboli nella loro implementazione, impostando una chiamata di ritorno di funzioni sui riferimenti deboli che notificano ai dizionari deboli quando una chiave o un valore viene richiesto dalla garbage collection. Molti programmatori potrebbero verificare che l'uso di uno di questi tipi di dizionari deboli è tutto ciò di cui necessitano - di regola non è necessario creare i propri riferimenti deboli direttamente. L'implementazione di basso livello usata dal dizionario debole viene mostrata dal modulo `weakref` per beneficiarne negli usi più avanzati.

Non tutti gli oggetti possono venire riferiti debolmente; gli oggetti che godono di questa proprietà sono istanze di classe, funzioni scritte in Python (ma non in C), e metodi (sia diretti che indiretti). Tipi di estensione possono venire facilmente creati per supportare i riferimenti deboli; per ulteriori informazioni vedete la sezione 3.3.3 "Riferimenti deboli nei tipi di estensione".

`ref`(*oggetto*[, *callback*])

Restituisce un riferimento debole ad un *oggetto*. L'oggetto originale può venire recuperato chiamando l'oggetto di riferimento se il referente è ancora vivo; se invece non lo è più, chiamare il riferimento all'oggetto comporterà la restituzione di `None`. Se viene fornita una *callback*, questa verrà usata quando l'oggetto sarà prossimo alla finalizzazione; l'oggetto con riferimenti deboli verrà passato come parametro alla chiamata di ritorno; il referente non sarà più disponibile.

È consentito che riferimenti deboli multipli vengano costruiti intorno allo stesso oggetto. Le chiamate di ritorno registrate per ogni riferimento debole verranno sfruttate in ordine cronologico dalla più recente (cioè da quella registrata per ultima) alla più vecchia (quella registrata per prima).

Le eccezioni sollevate dalla chiamata di ritorno verranno indicate sullo standard error, ma non possono essere propagate; vengono gestite esattamente nella stessa maniera delle eccezioni generate dal metodo `__del__()` di un oggetto.

I riferimenti deboli sono hashabili se l'*oggetto* è hashabile. Manterranno il loro valore di hash anche dopo che l'*oggetto* è stato rimosso. Se la funzione `hash()` viene chiamata per la prima volta solo dopo che l'*oggetto* è stato rimosso, la chiamata solleverà un'eccezione `TypeError`.

I riferimenti deboli supportano i test di uguaglianza, ma non di ordinamento. Se i referenti sono ancora in vita, due riferimenti hanno la stessa relazione di uguaglianza che esiste tra gli oggetti referenti (indipendentemente dalle funzioni di *callback*). Se entrambi entrambi i referenti vengono cancellati, i riferimenti sono uguali solo se gli oggetti referenti sono lo stesso oggetto.

`proxy`(*oggetto*[, *callback*])

Restituisce un proxy ad un *oggetto* che usa un riferimento debole. Questo supporta l'uso del proxy nella maggior parte dei contesti, invece di richiedere una esplicita dereferenziazione usata con oggetti che hanno riferimenti deboli. L'oggetto restituito erediterà il tipo di uno dei due, `ProxyType` oppure `CallableProxyType`, a seconda di quale *oggetto* sia chiamabile. Gli oggetti proxy non sono hashabili,

indipendentemente dal referente; questo evita una serie di problemi legati alla loro natura fondamentalmente mutabile, e previene il loro utilizzo come chiavi dei dizionari. La chiamata di ritorno è la stessa come il parametro con lo stesso nome dato alla funzione `ref()`.

`getweakrefcount`(*oggetto*)
Restituisce il numero di riferimenti deboli e i proxy che si riferiscono all'*oggetto*.

`getweakrefs`(*oggetto*)
Restituisce una lista di tutti i riferimenti deboli e oggetti proxy che si riferiscono all'*oggetto*.

class `WeakKeyDictionary`([*dict*])
È una classe di mappatura che riferisce debolmente le chiavi. Le voci nel dizionario verranno scartate quando non c'è più un riferimento forte alla chiave. Questo classe può venire usata per associare dati aggiuntivi ad un oggetto che appartiene ad altre parti di un'applicazione, senza aggiungere attributi a questi oggetti. Può essere di particolare utilità con oggetti che sovrascrivono gli attributi degli accessi.

Note: Avvertenza: Dal momento che un `WeakKeyDictionary` viene costruito sopra un dizionario Python, non deve cambiare dimensione quando vengono eseguite delle iterazioni su di esso. Questo può essere difficile da garantire per un `WeakKeyDictionary` perché le azioni eseguite dal programma durante l'iterazione possono portare alcuni elementi a svanire "per magia" (come effetto collaterale della garbage collection).

class `WeakValueDictionary`([*dict*])
Classe di mappatura che riferisce valori debolmente. Le voci nel dizionario verranno scartate quando non esiste più un riferimento forte al valore.

Note: Avvertenza: Dal momento che un `WeakValueDictionary` viene costruito sopra un dizionario Python, non deve cambiare dimensione quando vengono eseguite delle iterazioni su di esso. Questo può essere difficile da garantire per un `WeakValueDictionary` perché le azioni eseguite dal programma durante l'iterazione possono portare alcuni elementi a svanire "per magia" (come effetto collaterale della garbage collection).

`ReferenceType`
Il tipo di oggetto per oggetti con riferimenti deboli.

`ProxyType`
Il tipo di oggetto per proxy di oggetti che non sono chiamabili.

`CallableProxyType`
Il tipo di oggetto per proxy di oggetti chiamabili.

`ProxyTypes`
Sequenza contenente tutti i tipi di oggetti per i proxy. Questa può rendere più semplice verificare se un oggetto è un proxy senza dover nominare entrambi i tipi di proxy.

exception `ReferenceError`
Eccezione sollevata quando un oggetto proxy viene utilizzato ma l'oggetto sottostante è stato inserito nella garbage collection. Equivale all'eccezione standard `ReferenceError`.

Vedete anche:

PEP 0205, "*Weak References*"
La proposta ed il fondamento logico per questa caratteristica, comprendente i link a precedenti implementazioni e le informazioni riguardanti funzioni simili in altri linguaggi.

3.3.1 Oggetti con riferimenti deboli

Gli oggetti con riferimenti deboli non hanno attributi o metodi, ma permettono di ottenere il referente, se esiste ancora, chiamandolo:

```
>>> import weakref
>>> class Object:
...     pass
...
>>> o = Object()
>>> r = weakref.ref(o)
>>> o2 = r()
>>> o is o2
True
```

Se il referente non esiste più, una chiamata all'oggetto referente restituirà `None`:

```
>>> del o, o2
>>> print r()
None
```

Per verificare che un oggetto con riferimento debole esista ancora, si dovrà utilizzare l'espressione *ref*`() is not None`. Normalmente, il codice di un'applicazione che necessita di usare un riferimento ad un oggetto dovrebbe seguire questo modello:

```
# r è un oggetto con riferimento debole
o = r()
if o is None:
    # il referente è stato inserito nella garbage collection
    print "Oggetto allocato."
else:
    print "Oggetto ancora vivo!"
    o.do_something_useful()
```

L'utilizzo di un test separato per l'"esistenza in vita" può dare luogo a delle interferenze in applicazioni che sfruttino i thread; un altro thread può provocare l'invalidamento di un riferimento debole prima che questo venga chiamato (dal thread attuale - NDT); l'esempio appena mostrato è sicuro sia in applicazioni multi thread che in applicazioni single thread.

3.3.2 Esempio

Questo semplice esempio mostra come un'applicazione può utilizzare gli ID degli oggetti per recuperare gli oggetti che ha già visto in precedenza. Gli ID degli oggetti possono quindi essere utilizzati in altre strutture di dati senza forzare gli oggetti a rimanere vivi, ma gli oggetti, se vivi, possono ancora essere recuperati attraverso il loro ID.

```
import weakref

_id2obj_dict = weakref.WeakValueDictionary()

def remember(obj):
    oid = id(obj)
    _id2obj_dict[oid] = obj
    return oid

def id2obj(oid):
    return _id2obj_dict[oid]
```

3.3.3 Riferimenti deboli nei tipi

Uno degli obiettivi dell'implementazione è quello di permettere ad ogni tipo di partecipare al meccanismo dei riferimenti deboli senza generare l'overhead su quegli oggetti che non beneficiano di questo meccanismo (come i numeri).

Affinché un oggetto sia riferibile in modo debole, l'estensione deve includere nella struttura dell'istanza un campo `PyObject*` per l'utilizzo del meccanismo dei riferimenti deboli; questo deve venire inizializzato a `NULL` dal costruttore dell'oggetto. Deve anche impostare il campo `tp_weaklistoffset` del corrispondente tipo di oggetto dell'offset del campo. Inoltre, deve aggiungere `Py_TPFLAGS_HAVE_WEAKREFS` allo slot tp_flags. Per esempio, l'istanza tipo viene definita con la seguente struttura:

```
typedef struct {
    PyObject_HEAD
    PyClassObject *in_class;       /* L'oggetto classe */
    PyObject      *in_dict;        /* Un dizionario */
    PyObject      *in_weakreflist; /* Lista di riferimenti deboli */
} PyInstanceObject;
```

L'oggetto tipo dichiarato staticamente per le istanze viene così definito:

```
PyTypeObject PyInstance_Type = {
    PyObject_HEAD_INIT(&PyType_Type)
    0,
    "module.instance",

    /* Molte righe omesse per brevità... */

    Py_TPFLAGS_DEFAULT | Py_TPFLAGS_HAVE_WEAKREFS   /* tp_flags */
    0,                                             /* tp_doc */
    0,                                             /* tp_traverse */
    0,                                             /* tp_clear */
    0,                                             /* tp_richcompare */
    offsetof(PyInstanceObject, in_weakreflist), /* tp_weaklistoffset */
};
```

Il costruttore di tipo è responsabile dell'inizializzazione della lista dei riferimenti deboli come `NULL`:

```
static PyObject *
instance_new() {
    /* Altre righe di inizializzazione omesse per brevità */

    self->in_weakreflist = NULL;

    return (PyObject *) self;
}
```

L'unica aggiunta ulteriore è che il distruttore ha bisogno di chiamare il gestore dei riferimenti deboli per pulire ogni riferimento debole rimasto. Questo deve essere fatto prima di ogni altra cosa, da parte del distruttore, ma vien richiesto solamente se la lista dei riferimenti deboli è non `NULL`:

```
static void
instance_dealloc(PyInstanceObject *inst)
{
    /* Allocate temporaneamente se necessario, ma non
       iniziate ancora la distruzione.
     */

    if (inst->in_weakreflist != NULL)
        PyObject_ClearWeakRefs((PyObject *) inst);

    /* Procedete normalmente con la distruzione dell'oggetto. */
}
```

3.4 `fpectl` — Controllo delle eccezioni nei numeri in virgola mobile

Molti computer elaborano le operazioni con numeri in virgola mobile in conformità con il cosiddetto standard IEEE-754. Su un qualsiasi computer, alcune operazioni in virgola mobile producono risultati che non possono venire espressi con normali valori in virgola mobile. Per esempio, provate:

```
>>> import math
>>> math.exp(1000)
inf
>>> math.exp(1000) / math.exp(1000)
nan
```

(L'esempio precedente funziona su molte piattaforme. La DEC Alpha può essere una eccezione). "Inf" è un valore speciale, non numerico, nello standard IEEE-754 che sta per "infinito" mentre "nan" significa "non un numero" ("not a number"). Notate che, aldilà del risultato non numerico, nulla di speciale accade quando chiedete a Python di procedere con questi calcoli. Questo infatti è il comportamento predefinito nello standard IEEE-754, e se a voi funziona, smettete pure di leggere questo paragrafo.

In alcune circostanze, sarebbe meglio sollevare un'eccezione ed interrompere il processo nel punto in cui si rileva l'operazione errata. Il modulo `fpectl` è da usare in queste situazioni. Fornisce il controllo sui numeri in virgola mobile per l'hardware di molte case produttrici, consentendo all'utente la possibilità di abilitare la generazione di un `SIGFPE` ogni volta che dovrebbe venire sollevata una eccezione IEEE-754 come Division by Zero, Overflow, oppure Invalid Operation. Insieme ad un paio di macro wrapper, inserite nel codice C comprendente il sistema Python in uso, `SIGFPE` viene intercettata e convertita nell'eccezione Python `FloatingPointError`.

Il modulo `fpectl` definisce le seguenti funzioni e può sollevare l'eccezione data:

`turnon_sigfpe``()`
: Abilita la generazione di `SIGFPE`, ed imposta un appropriato gestore di segnale.

`turnoff_sigfpe``()`
: Resetta al valore predefinito la gestione delle eccezioni sui numeri in virgola mobile.

exception `FloatingPointError`
: Dopo che è stata eseguita `turnon_sigfpe()`, un'operazione con numeri in virgola mobile che solleva una delle eccezioni IEEE-754, Division by Zero, Overflow, oppure Invalid Operation, solleverà invece questa eccezione standard di Python.

3.4.1 Esempio

L'esempio seguente mostra come avviare ed esaminare il funzionamento del modulo `fpectl`.

```
>>> import fpectl
>>> import fpetest
>>> fpectl.turnon_sigfpe()
>>> fpetest.test()
overflow        PASS
FloatingPointError: Overflow

div by 0        PASS
FloatingPointError: Division by zero
  [ more output from test elided ]
>>> import math
>>> math.exp(1000)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
FloatingPointError: in math_1
```

3.4.2 Limitazioni ed ulteriori considerazioni

Impostare un certo processore affinché intercetti errori di virgola mobile IEEE-754, richiede attualmente un codice utente su una base per-architecture. Potreste dover modificare `fpectl` per controllare il vostro hardware specifico.

La conversione di un'eccezione IEEE-754 in un'eccezione Python richiede che le macro wrapper `PyFPE_START_PROTECT` e `PyFPE_END_PROTECT` vengano inserite nel vostro codice in modo appropriato. Python stesso è stato modificato per supportare il modulo `fpectl`, mentre altri codici di interesse per gli analisti numerici non sono stati modificati.

Il modulo `fpectl` non è thread-safe.

Vedete anche:

Alcuni file nei sorgenti della distribuzione possono essere interessanti per imparare molto su come lavora questo modulo. Il file include 'Include/pyfpe.h' tratta dell'implementazione di questo modulo esaurientemente. 'Modules/fpetestmodule.c' fornisce molti esempi di utilizzo. Ulteriori esempi si possono trovare in 'Objects/floatobject.c'.

3.5 `atexit` — Gestori d'uscita

Nuovo nella versione 2.0.

Il modulo `atexit` definisce una singola funzione per registrare azioni di pulizia. Le funzioni così registrate vengono automaticamente eseguite sulla normale terminazione dell'interprete.

Nota: le funzioni registrate tramite questo modulo non vengono chiamate quando il programma viene ucciso da un segnale, quando viene rilevato un errore fatale di Python o quando viene chiamato `os._exit()`.

Questa è una interfaccia alternativa per la funzionalità fornita dalla variabile `sys.exitfunc`.

Nota: questo modulo è improbabile che lavori correttamente quando usato con altri codici che impostano `sys.exitfunc`. In particolare altri moduli di Python sono liberi di usare `atexit` senza la conoscenza del programmatore. Gli autori che utilizzano `sys.exitfunc` dovrebbero invece convertire i loro codici per poter utilizzare `atexit`. Il modo più semplice per convertire un codice che imposti `sys.exitfunc` è quello di importare `atexit` e registrare la funzione che veniva limitata da `sys.exitfunc`.

`register`(*func*[, **args*[, ***kargs*]])

Registra *func* come una funzione da eseguire alla terminazione del programma. Ciascun argomento facoltativo che viene passato a *func* deve venire passato come argomento di `register()`.

Quando viene terminato normalmente il programma (per esempio, se viene chiamata `sys.exit()` oppure viene completata l'esecuzione del modulo principale), tutte le funzioni registrate vengono chiamate nell'ordine, ultima entrata, prima ad uscire. La consuetudine è che i moduli di livello più basso

vengano normalmente importati prima di quelli di livello più alto e quindi dovranno essere cancellati successivamente.

Vedete anche:

Modulo `readline` (sezione 7.20):
Esempi utili di `atexit` per leggere e scrivere i file dello storico di `readline`.

3.5.1 `atexit` Esempi

Il seguente esempio dimostra come un modulo possa inizializzare un contatore da un file, quando questo venga importato, e salvare automaticamente il valore aggiornato del contatore, quando il programma termini senza affidarsi ad un'applicazione che faccia un'esplicita chiamata all'interno del modulo durante la chiusura.

```
try:
    _count = int(open("/tmp/counter").read())
except IOError:
    _count = 0

def incrcounter(n):
    global _count
    _count = _count + n

def savecounter():
    open("/tmp/counter", "w").write("%d" % _count)

import atexit
atexit.register(savecounter)
```

Gli argomenti chiave e posizionali possono anche venire passati a `register()`, per venire poi passati attraverso la funzione registrata quando questa viene chiamata:

```
def goodbye(nome, aggettivo):
    print 'Ciao, %s, è stato %s incontrarti.' % (nome, aggettivo)

import atexit
atexit.register(goodbye, 'Donny', 'nice')

# or:
atexit.register(goodbye, aggettivo='nice', nome='Donny')
```

3.6 `types` — Nomi per i tipi built-in

Questo modulo definisce i nomi per alcuni tipi di oggetto che vengono usati dall'interprete standard di Python, ma non per i tipi definiti da diversi moduli di estensione. Inoltre, non include alcuni tipi che si presentano durante l'esecuzione come il tipo `listiterator`. È buona norma usare '`from types import *`' — il modulo non esporta alcun nome oltre a quelli qui elencati. Nuovi nomi esportati da versioni future di questo modulo, verrannoo tutti inseriti in '`Type`'.

Un uso tipico è per funzioni che svolgono differenti compiti in base ai loro tipi di argomento, come la seguente:

```
from types import *
def delete(mialista, elemento):
    if type(elemento) is IntType:
       del mialista[elemento]
    else:
       mialista.remove(elemento)
```

A partire da Python 2.2, funzioni fattoriali built-in come `int()` e `str()` vengono considerate anche nomi per indicare i tipi corrispondenti. Questo è il metodo attualmente preferito per accedere al tipo, piuttosto che usare il modulo `types`. In accordo con queste considerazioni, l'esempio precedente dovrebbe potersi scrivere come segue:

```
def delete(mialista, elemento):
    if isinstance(elemento, int):
       del mialista[elemento]
    else:
       mialista.remove(elemento)
```

Il modulo definisce i seguenti nomi:

`NoneType`
Il tipo `None`.

`TypeType`
Il tipo del tipo di oggetto (come restituito anche da `type()`).

`BooleanType`
Il tipo dei booleani `True` e `False`; questo è un alias della funzione built-in `bool`. Nuovo nella versione 2.3.

`IntType`
Il tipo degli interi (p.es. `1`).

`LongType`
Il tipo degli interi long (p.es. `1L`).

`FloatType`
Il tipo dei numeri in virgola mobile (p.es. `1.0`).

`ComplexType`
Il tipo dei numeri complessi (p.es. `1.0j`). Questo è non definito se Python viene compilato senza il supporto per i numeri complessi.

`StringType`
Il tipo delle le stringhe di caratteri (p.es. `'Spam'`).

`UnicodeType`
Il tipo dei caratteri stringa Unicode (p.es. `u'Spam'`). Questo tipo è non definito se Python viene compilato senza il supporto per l'Unicode.

`TupleType`
Il tipo delle tuple (p.es. `(1, 2, 3, 'Spam')`).

`ListType`
Il tipo delle liste (p.es. `[0, 1, 2, 3]`).

`DictType`
Il tipo dei dizionari (p.es. `{'Bacon': 1, 'Ham': 0}`).

`DictionaryType`
Un nome alternativo per definire `DictType`.

`FunctionType`
Il tipo delle funzioni definite dall'utente e delle lambda.

LambdaType
Un nome alternativo per `FunctionType`.

GeneratorType
Il tipo degli oggetti generator-iter (NdT: generatori di sequenze), prodotto dalla chiamata di una funzione generatrice. Nuovo nella versione 2.2.

CodeType
Il tipo del codice oggetto, come quello restituito da `compile()`.

ClassType
Il tipo delle classi definite dall'utente.

InstanceType
Il tipo delle istanze di classi definite dall'utente.

MethodType
Il tipo dei metodi delle istanze di classi definite dall'utente.

UnboundMethodType
Un nome alternativo per `MethodType`.

BuiltinFunctionType
Il tipo delle funzioni built-in come `len()` o `sys.exit()`.

BuiltinMethodType
Un nome alternativo per le `BuiltinFunction`.

ModuleType
Il tipo dei moduli.

FileType
Il tipo degli oggetti file aperti come `sys.stdout`.

XRangeType
Il tipo di oggetti intervallo restituiti da `xrange()`.

SliceType
Il tipo di oggetti restituiti da `slice()`.

EllipsisType
Il tipo di `Ellipsis`.

TracebackType
Il tipo degli oggetti tracebak come quelli che si possono trovare in `sys.exc_traceback`.

FrameType
Il tipo degli oggetti frame (struttura) come quelli che si trovano in `tb.tb_frame` se `tb` è un oggetto traceback.

BufferType
Il tipo degli oggetti buffer creati dalla funzione `buffer()`.

StringTypes
Una sequenza contenente `StringType` e `UnicodeType`, usata per facilitare il controllo di ogni oggetto stringa. Il suo utilizzo è più portabile rispetto a quello di utilizzare una sequenza di due tipi stringa costruiti altrove, poichè contiene soltanto `UnicodeType`, se viene costruita mediante la versione corrente di Python. Per esempio: `isinstance(s, types.StringTypes)`. Nuovo nella versione 2.2.

3.7 `UserDict` — Wrapper per classi di oggetti dizionario

Note: Questo modulo è disponibile solo per compatibilità all'indietro. Se si sta scrivendo codice che non deve lavorare con versioni di Python precedenti alla 2.2, è consigliabile usare direttamente una sotto classe del tipo built-in `dict`.

Questo modulo definisce una classe che si comporta da wrapper per gli oggetti dizionario. È una classe di base

piuttosto utile per le proprie implementazioni delle classi dizionario, che può ereditarne e sovrascriverne i metodi esistenti o aggiungerne di nuovi. In questo modo possono venire aggiunti nuovi comportamenti ai dizionari.

Il modulo definisce anche un mix definendo tutti i metodi dei dizionari per le classi cha già hanno un minimo di interfacce mappate. Questo semplifica enormemente la scrittura di classi che necessitino essere sostituibili per i dizionari (come il modulo shelve).

Il modulo `UserDict` definisce la classe `UserDict` e `DictMixin`:

class `UserDict`([*initialdata*]**)**
Classe che simula un dizionario. I contenuti dell'istanza vengono memorizzati in un dizionario regolare, accessibile attraverso l'attributo `data` dell'istanza `UserDict`. Se *initialdata* viene fornita, `data` viene inizializzato con il suo contenuto; notate che un riferimento ad *initialdata* non verrà mantenuto, consentendone l'uso per altri scopi.

Oltre a supportare metodi e operazioni di mapping (vedete la sezione 2.3.8), l'istanza di `UserDict` fornisce anche i seguenti attributi:

`data`
Un dizionario reale utilizzato per memorizzare il contenuto della classe `UserDict`.

class `DictMixin`()
Definisce tutti i metodi di dizionari, per classi che già possiedono un minimo di interfaccia ai dizionari, includendo `__getitem__()`, `__setitem__()`, `__delitem__()`, and `keys()`.

Questo mix dovrebbe venire usato come superclasse. Aggiungendo ciascuno dei metodi indicati sopra, integra progressivamente maggiori funzionalità. Per esempio, definendo tutti i metodi tranne `__delitem__` si precluderà soltanto `pop` e `popitem` dall'interfaccia completa.

In aggiunta ai quattro metodi di base, viene progressivamente raggiunta maggiore efficienza definendo `__contains__()`, `__iter__()` e `iteritems()`.

Poiché mixin non ha conoscenza del costruttore della sotto classe, non definisce `__init__()` o `copy()`.

3.8 `UserList` — Classe wrapper per oggetti lista

Note: Questo modulo è disponibile solo per compatibilità all'indietro. Se si sta scrivendo codice che non necessita di lavorare con versioni di Python precedenti alla 2.2, é consigliabile generare sotto classi direttamente dal tipo `list` built-in.

Questo modulo definisce una classe che si comporta come un wrapper degli oggetti lista. Si tratta di una classe di base utile per le proprie classi simili alle liste, che può ereditarne e sovrascriverne i metodi esistenti o aggiungerne di nuovi. In questo modo possono venire aggiunte nuove caratteristiche alle liste.

Il modulo `UserList` definisce la classe `UserList`:

class `UserList`([*lista*]**)**
Classe che simula una lista. I contenuti dell'istanza vengono memorizzati in una lista regolare, accessibile attraverso l'attributo `data` dell'istanza di `UserList`. I contenuti dell'istanza vengono inizialmente impostati come una copia di *lista*, avendo come predefinita la lista vuota `[]`; *lista* può essere sia una lista regolare Python, che una istanza di `UserList` (o una sotto classe).

Oltre a supportare i metodi e le operazioni di sequenze mutabili (vedete la sezione 2.3.6), le istanze `UserList` forniscono i seguenti attributi:

`data`
Un oggetto Python lista reale usato per memorizzare il contenuto della classe `UserList`.

Requisiti per derivare sotto classi: Dalle sotto classi di `UserList` ci si aspetta che offrano un costruttore, che possa venire chiamato o senza argomenti o con un argomento. Le operazioni su liste che ritornano una nuova sequenza cercano di creare un'istanza dell'attuale implementazione della classe. Per far ciò, viene assunto che il costruttore possa venire chiamato con un parametro singolo, che è un oggetto sequenza usato come sorgente di dati.

Se una classe derivata non vuole adeguarsi a questi requisiti, tutti i metodi speciali supportati da questa classe

avranno bisogno di venire sovrascritti; per favore consultate i sorgenti per informazioni circa i metodi che si devono fornire in quel caso.

Modificato nella versione 2.0: Python versione 1.5.2 e 1.6 richiedevano anche che il costruttore fosse chiamabile senza parametri, e offrisse un attributo `data` mutabile. Le versioni Python più vecchie non cercavano di creare istanze di classi derivate.

3.9 `UserString` — Classe wrapper per gli oggetti stringa

Note: Questa classe `UserString` da questo modulo è disponibile soltanto per compatibilità all'indietro. Se si sta scrivendo codice che non ha bisogno di lavorare con versioni di Python precedenti alla 2.2, é consigliabile considerare di derivare le classi direttamente dal tipo built-in `str`, in sostituzione dell'uso di `UserString` (non ci sono equivalenti built-in a `MutableString`).

Questo modulo definisce una classe che si comporta come un wrapper per gli oggetti stringa. È una classe base utile per le proprie classi simili alle stringhe, che può ereditane e sovrascriverne i metodi esistenti o aggiungerne di nuovi. In questo caso possono venire aggiunte nuove caratteristiche alle stringhe.

Notate che queste classi sono molto inefficienti rispetto alle stringhe reali o agli oggetti Unicode; questo è soprattutto vero nel caso di `MutableString`.

Il modulo `UserString` definisce le seguenti classi:

class `UserString`([*sequenza*])
Classe che simula un oggetto stringa o un oggetto stringa Unicode. Il contenuto dell'istanza viene memorizzato in una stringa regolare o in un oggetto stringa Unicode, accessibile attraverso l'attributo `data` dell'istanza `UserString`. I contenuti dell'istanza vengono inizialmente impostati come una copia di *sequenza*. *sequenza* può essere sia una stringa Python regolare che una stringa Unicode, un'istanza di `UserString` (o una sotto classe) o una sequenza arbitraria che può venire convertita in una stringa utilizzando la funzione built-in `str()`.

class `MutableString`([*sequenza*])
Questa classe deriva direttamente dalla classe `UserString` indicata precedentemente, e ridefinisce le stringhe in modo tale che siano *mutabili*. Stringhe mutabili non possono venire usate come chiavi di dizionario, perché i dizionari richiedono come chiavi oggetti *immutabili*. La principale intenzione di questa classe è quella di servire come esempio educativo per l'ereditarietà e la necessità di rimuovere (sovrascrivere) il metodo `__hash__()` per intercettare i tentativi di usare un oggetto mutabile come una chiave di dizionario, che sarebbe altrimenti facilmente soggetto ad errori difficili da individuare.

Oltre a supportare i metodi e le operazioni su stringhe e oggetti Unicode (vedete la sezione 2.3.6 "Metodi Stringa"), l'istanza `UserString` fornisce il seguente attributo:

`data`
Una stringa Python reale o un oggetto Unicode usati per memorizzare il contenuto della classe `UserString`.

3.10 `operator` — Operatori standard come funzioni

Il modulo `operator` esporta un insieme di funzioni implementate in C che corrispondono agli operatori intrinseci di Python. Per esempio, `operator.add(x, y)` è equivalente all'espressione `x+y`. I nomi delle funzioni sono quelli usati per i metodi speciali di classe; varianti senza il segno '__' in testa o in coda vengono inoltre fornite per convenienza.

Le funzioni rientrano in categorie che eseguono confronti tra oggetti, operazioni logiche, operazioni matematiche, operazioni sulle sequenze e verifiche su tipi astratti.

Le funzioni di confronto degli oggetti sono utili per tutti gli oggetti, e prendono il nome dall'operatore di confronto ricco che supportano:

`lt`(*a*, *b*)
`le`(*a*, *b*)

eq(*a*, *b*)
ne(*a*, *b*)
ge(*a*, *b*)
gt(*a*, *b*)
__lt__(*a*, *b*)
__le__(*a*, *b*)
__eq__(*a*, *b*)
__ne__(*a*, *b*)
__ge__(*a*, *b*)
__gt__(*a*, *b*)

Eseguono "confronti ricchi" tra *a* e *b*. Nello specifico, `lt(a, b)` è equivalente a *a* `<` *b*, `le(a, b)` è equivalente a *a* `<=` *b*, `eq(a, b)` è equivalente a *a* `==` *b*, `ne(a, b)` è equivalente a *a* `!=` *b*, `gt(a, b)` è equivalente a *a* `>` *b* e `ge(a, b)` è equivalente a *a* `>=` *b*. Notate che, diversamente dalla built-in `cmp()`, queste funzioni possono restituire qualunque valore, che può venire intrepretato o meno come booleano. Fate riferimento al *Python Reference Manual* per maggiori informazioni riguardo i confronti ricchi. Nuovo nella versione 2.2.

Le operazioni logiche sono in genere applicabili a tutti gli oggetti e supportano test della verità, test sull'identità e operazioni booleane:

not_(*o*)
__not__(*o*)

Restituisce il risultato di `not` *o*. (Notate che non esiste il metodo `__not__()` per oggetti istanze; solo il nucleo dell'interprete definisce questa operazione. Il risultato viene influenzato dai metodi `__nonzero__()` e `__len__()`.)

truth(*o*)

Restituisce `True` se *o* è vero, altrimenti `False`. Questo è equivalente all'uso del costruttore `bool`.

is_(*a*, *b*)

Restituisce *a* `is` *b*. Verifica l'identità di oggetti. Nuovo nella versione 2.3.

is_not(*a*, *b*)

Restituisce *a* `is not` *b*. Verifica l'identità di oggetti.

Nuovo nella versione 2.3.

Le operazioni matematiche e bit per bit sono le più numerose:

abs(*o*)
__abs__(*o*)

Restituisce il valore assoluto di *o*.

add(*a*, *b*)
__add__(*a*, *b*)

Restituisce *a* + *b*, per *a* e *b* numerici.

and_(*a*, *b*)
__and__(*a*, *b*)

Restituisce l'and bit per bit di *a* e *b*.

div(*a*, *b*)
__div__(*a*, *b*)

Restituisce *a* `/` *b* quando `__future__.division` non viene effettuata. Questa è anche conosciuta come divisione "classica".

floordiv(*a*, *b*)
__floordiv__(*a*, *b*)

Restituisce *a* `//` *b*. Nuovo nella versione 2.2.

inv(*o*)
invert(*o*)
__inv__(*o*)
__invert__(*o*)

Restitusce l'inverso bit per bit del numero *o*. Questo è l'equivalente di `~`*o*. I nomi `invert()` e

`__invert__()` sono stati aggiunti in Python 2.0.

lshift(*a*, *b*)
__lshift__(*a*, *b*)
Restituisce lo scorrimento a sinistra di *a* di *b* bit.

mod(*a*, *b*)
__mod__(*a*, *b*)
Restituisce *a* `%` *b*.

mul(*a*, *b*)
__mul__(*a*, *b*)
Restituisce *a* `*` *b*, per i numeri *a* e *b*.

neg(*o*)
__neg__(*o*)
Restituisce la negativizzazione di *o*.

or_(*a*, *b*)
__or__(*a*, *b*)
Restituisce l'or bit per bit di *a* e *b*.

pos(*o*)
__pos__(*o*)
Restituisce il positivo di *o*.

pow(*a*, *b*)
__pow__(*a*, *b*)
Restituisce *a* `**` *b*, per i numeri *a* e *b*. Nuovo nella versione 2.3.

rshift(*a*, *b*)
__rshift__(*a*, *b*)
Restituisce lo scorrimento a destra di *a* di *b* bit.

sub(*a*, *b*)
__sub__(*a*, *b*)
Restituisce *a* `-` *b*.

truediv(*a*, *b*)
__truediv__(*a*, *b*)
Restituisce *a* `/` *b* quando `__future__.division` viene effettuata. Questa è anche conosciuta come divisione. Nuovo nella versione 2.2.

xor(*a*, *b*)
__xor__(*a*, *b*)
Restituisce l'or bit per bit esclusivo di *a* e *b*.

Le operazioni che lavorano con sequenze includono:

concat(*a*, *b*)
__concat__(*a*, *b*)
Restituisce *a* + *b* per le sequenza *a* e *b*.

contains(*a*, *b*)
__contains__(*a*, *b*)
Restituisce il risultato del test *b* `in` *a*. Notate gli operatori inversi. La definizione `__contains__()` è stata aggiunta in Python 2.0.

countOf(*a*, *b*)
Restituisce il numero delle occorrenze di *b* in *a*.

delitem(*a*, *b*)
__delitem__(*a*, *b*)
Rimuove il valore di *a* nell'indice *b*.

delslice(*a*, *b*, *c*)
__delslice__(*a*, *b*, *c*)

Cancella la fetta di *a* dall'indice *b* all'indice *c*`-1`.

getitem(*a*, *b*)
__getitem__(*a*, *b*)
Restituisce il valore di *a* nell'indice *b*.

getslice(*a*, *b*, *c*)
__getslice__(*a*, *b*, *c*)
Restituisce la fetta di *a* dall'indice *b* all'indice *c*`-1`.

indexOf(*a*, *b*)
Restituisce l'indice della prima occorrenza di *b* in *a*.

repeat(*a*, *b*)
__repeat__(*a*, *b*)
Restituisce *a* `*` *b* quando *a* è una sequenza e *b* è un intero.

sequenceIncludes(...)
Deprecato dalla versione 2.0. Al suo posto usate `contains()`.

Alias per `contains()`.

setitem(*a*, *b*, *c*)
__setitem__(*a*, *b*, *c*)
Imposta il valore di *a* dall'indice *b* a *c*.

setslice(*a*, *b*, *c*, *v*)
__setslice__(*a*, *b*, *c*, *v*)
Imposta la fetta di *a* dall'indice *b* all'indice *c*`-1` alla sequenza *v*.

Il modulo `operator` definisce anche alcuni predicati per verificare il tipo degli oggetti. **Note:** Va fatta attenzione a non interpretare erroneamente i risultati di queste funzioni; solo `isCallable()` ha le caratteristiche per essere attendibile con oggetti istanza. Per esempio:

```
>>> class C:
...     pass
...
>>> import operator
>>> o = C()
>>> operator.isMappingType(o)
True
```

isCallable(*o*)
Deprecato dalla versione 2.0. In sostituzione usate la funzione built-in `callable()`.

Restituisce vero se l'oggetto *o* può venire chiamato come una funzione, altrimenti falso. Vero viene restituito per funzioni, metodi bound e unbound, classi di oggetti e oggetti istanze che supportino il metodo `__call__()`

isMappingType(*o*)
Restituisce vero se l'oggetto *o* supporta l'interfaccia di mapping. Questo è vero per i dizionari e per tutti gli oggetti istanza. **Avvertenze:** Non esiste un metodo affidabile per verificare se una istanza supporta l'intero protocollo di mapping fintanto che l'interfaccia stessa non viene definita correttamente. Questo rende il test meno utile di quanto altrimenti potrebbe essere.

isNumberType(*o*)
Restituisce vero se l'oggetto *o* rappresenta un numero. Questo è vero per tutti i tipi numerici implementati in C, e per tutti gli oggetti istanza. **Avvertenze:** Non esiste un metodo affidabile per verificare se una istanza supporta l'interfaccia numerica in modo completo fintanto che la classe stessa non viene definita correttamente. Questo rende il test meno utile di quanto altrimenti potrebbe essere.

isSequenceType(*o*)
Restituisce valore vero se l'oggetto *o* supporta il protocollo della sequenza. Restituisce vero per tutti gli oggetti che definiscono una sequenza con i metodi del linguaggio C e per tutti gli altri oggetti istanza. **Avvertenze:** Non esiste un metodo affidabile per verificare se un'istanza supporta l'interfaccia per la sequenza

in modo completo fintanto che l'interfaccia stessa non viene definita correttamente. Questo rende il test meno utile di quanto altrimenti potrebbe essere.

Esempio : Costruite un dizionario che mappi gli ordinali da `0` a `256` nei loro caratteri equivalenti.

```
>>> import operator
>>> d = {}
>>> keys = range(256)
>>> vals = map(chr, keys)
>>> map(operator.setitem, [d]*len(keys), keys, vals)
```

Il modulo `operator` definisce inoltre gli strumenti per attributi generalizzati e per l'aspetto degli elementi. Questo è molto utile per creare velocemente estrattori di campi come argomenti per `map()`, `sorted()`, `itertools.groupby()`, o per altre funzioni che aspettano una funzione in argomento.

`attrgetter`(*attr*)
Restituisce un oggetto chiamabile che recupera *attr* dal proprio operando. Dopo '`f=attrgetter('name')`', la chiamata '`f(b)`' restituisce '`b.name`'. Nuovo nella versione 2.4.

`itemgetter`(*elemento*)
Restituisce un oggetto chiamabile che recupera *elemento* dal proprio operando. Dopo '`f=itemgetter(2)`' la chiamata '`f(b)`' restituisce '`f(b)`'. Nuovo nella versione 2.4.

Esempi:

```
>>> from operator import *
>>> inventory = [('apple', 3), ('banana', 2), ('pear', 5), ('orange', 1)]
>>> getcount = itemgetter(1)
>>> map(getcount, inventory)
[3, 2, 5, 1]
>>> sorted(inventory, key=getcount)
[('orange', 1), ('banana', 2), ('apple', 3), ('pear', 5)]
```

3.10.1 Mappare gli operatori alle funzioni

La seguente tabella evidenzia come, nella sintassi di Python, operazioni astratte corrispondano ai simboli degli operatori e delle funzioni nel modulo `operator`.

Operazione	Sintassi	Funzione
Addizione	*a* + *b*	add(*a*, *b*)
Affettamento	*seq*[*i*:*j*]	getslice(*seq*, *i*, *j*)
And bit per bit	*a* & *b*	and_(*a*, *b*)
Assegnamento di fetta	*seq*[*i*:*j*] = *values*	setslice(*seq*, *i*, *j*, *values*)
Assegnamento indicizzato	*o*[*k*] = *v*	setitem(*o*, *k*, *v*)
Cancellazione di fetta	del *seq*[*i*:*j*]	delslice(*seq*, *i*, *j*)
Cancellazione indicizzata	del *o*[*k*]	delitem(*o*, *k*)
Concatenazione	*seq1* + *seq2*	concat(*seq1*, *seq2*)
Disuguaglianza	*a* != *b*	ne(*a*, *b*)
Divisione	*a* / *b*	div(*a*, *b*) # senza __future__.division
Divisione	*a* / *b*	truediv(*a*, *b*) # con __future__.division
Divisione	*a* // *b*	floordiv(*a*, *b*)
Elevamento a potenza	*a* ** *b*	pow(*a*, *b*)
Formattazione di stringa	*s* % *o*	mod(*s*, *o*)
Identità	*a* is *b*	is_(*a*, *b*)
Identità	*a* is not *b*	is_not(*a*, *b*)
Indicizzazione	*o*[*k*]	getitem(*o*, *k*)
Inversione bit per bit	~ *a*	invert(*a*)
Modulo	*a* % *b*	mod(*a*, *b*)
Moltiplicazione	*a* * *b*	mul(*a*, *b*)
Negazione (aritmetica)	- *a*	neg(*a*)
Negazione (Logica)	not *a*	not_(*a*)
Or bit per bit	*a* | *b*	or_(*a*, *b*)
Or esclusivo bit per bit	*a* ^ *b*	xor(*a*, *b*)
Ordinamento	*a* < *b*	lt(*a*, *b*)
Ordinamento	*a* <= *b*	le(*a*, *b*)
Ordinamento	*a* >= *b*	ge(*a*, *b*)
Ordinamento	*a* > *b*	gt(*a*, *b*)
Ricerche nei contenuti	*o* in *seq*	contains(*seq*, *o*)
Ripetizione di sequenza	*seq* * *i*	repeat(*seq*, *i*)
Scorrimento a sinistra	*a* << *b*	lshift(*a*, *b*)
Scorrimento a destra	*a* >> *b*	rshift(*a*, *b*)
Sottrazione	*a* - *b*	sub(*a*, *b*)
Test della verità	*o*	truth(*o*)
Uguaglianza	*a* == *b*	eq(*a*, *b*)

3.11 `inspect` — Ispezione degli oggetti in tempo reale

Nuovo nella versione 2.1.

Il modulo `inspect` fornisce diverse funzioni utili per acquisire informazioni riguardanti oggetti in tempo reale come ad esempio moduli, classi, metodi, funzioni, traceback, oggetti frame e codice oggetto. Per esempio, può aiutare ad esaminare il contenuto di una classe, recuperare il codice sorgente di un metodo, estrarre e formattare una lista di argomenti per una funzione, oppure dare tutte le informazioni di cui si ha bisogno per visualizzare una traceback dettagliata.

Esistono quattro tipi principali di servizi offerti da questo modulo: controllo del tipo, recupero del codice sorgente, analisi di classi e funzioni, esame dello stack dell'interprete.

3.11.1 Tipi e membri

La funzione `getmembers()` recupera i membri di un oggetto come fossero appartenenti ad una classe o ad un modulo. Le undici funzioni il cui nome inizia con "is" vengono fornite principalmente come scelta conveniente per il secondo argomento di `getmembers()`. Aiutano anche a determinare quando ci si può aspettare di trovare i seguenti attributi speciali:

Tipo	Attributo	Descrizione	Note
modulo	__doc__	stringa di documentazione	
	__file__	nomefile (mancante per i moduli built-in)	
class	__doc__	stringa di documentazione	
	__module__	nome del modulo in cui questa classe viene definita	
metodo	__doc__	stringa di documentazione	
	__name__	nome con il quale questo metodo è definito	
	im_class	oggetto classe richiesto per questo metodo	(1)
	im_func	oggetto funzione contenente l'implementazione del metodo	
	im_self	istanza a cui questo metodo è legata, o `None`	
funzione	__doc__	stringa di documentazione	
	__name__	nome con il quale questa funzione viene definita	
	func_code	codice oggetto contenente il bytecode delle funzioni compilate	
	func_defaults	tupla di ogni valore predefinito per gli argomenti	
	func_doc	(stessa cosa di __doc__)	
	func_globals	spazio dei nomi globali in cui viene definita questa funzione	
	func_name	(stessa cosa di __name__)	
traceback	tb_frame	oggetto frame a questo livello	
	tb_lasti	indice dell'ultima istruzione tentata, in bytecode	
	tb_lineno	numero di linea corrente nel codice sorgente Python	
	tb_next	prossimo oggetto con traceback più interno (chiamato da questo livello)	
frame	f_back	prossimo oggetto frame più esterno (questo è il frame chiamante)	
	f_builtins	spazio dei nomi built-in visto da questo frame	
	f_code	codice oggetto eseguito in questo frame	
	f_exc_traceback	traceback, se sollevato in questo frame, altrimenti `None`	
	f_exc_type	tipo di eccezione, se sollevata in questo frame, altrimenti `None`	
	f_exc_value	valore dell'eccezione, se sollevata in questo frame, altrimenti `None`	
	f_globals	spazio dei nomi globale visto da questo frame	
	f_lasti	indice dell'ultima istruzione tentata, in bytecode	
	f_lineno	numero della linea corrente nel codice sorgente Python	
	f_locals	spazio dei nomi locali visto da questo frame	
	f_restricted	0 o 1 se il frame è in esecuzione in modalità ristretta	
	f_trace	funzione tracciante per questo frame, altrimenti `None`	
code	co_argcount	numero degli argomenti (non inclusi gli argomenti * o **)	
	co_code	stringa o bytecode compilato grezzo	
	co_consts	tupla delle costanti usate nel bytecode	
	co_filename	nome del file in cui questo codice oggetto è stato creato	
	co_firstlineno	numero della prima riga in codice sorgente Python	
	co_flags	bitmap: 1=ottimizzato | 2=newlocals | 4=*arg | 8=**arg	
	co_lnotab	mapping codificato dei numeri di riga negli indici del bytecode	
	co_name	nome cui cui questo codice oggetto viene definito	
	co_names	tupla dei nomi delle variabili locali	
	co_nlocals	numero delle variabili locali	
	co_stacksize	spazio dello stack richiesto nella virtual machine	
	co_varnames	tupla dei nomi degli argomenti e delle variabili locali	
builtin	__doc__	stringa di documentazione	
	__name__	nome originale di questa funzione o metodo	
	__self__	istanza a cui questo metodo è legata, altrimenti `None`	

Nota:

(1) Modificato nella versione 2.2: `im_class` viene usato per riferirsi alla classe che definisce il metodo..

`getmembers`(*object*[, *predicato*])

Restituisce tutti i membri di un oggetto in una lista di coppie (nome, valore) ordinati per nome. Se l'argomento facoltativo *predicato* viene indicato, soltanto i membri per cui il predicato ritorna un valore vero vengono inclusi.

getmoduleinfo(*percorso*)
Restituisce una tupla di valori che descrivono come Python interpreterà il file indicato dal *percorso* se è un modulo, o `None` se non può venire identificato come modulo. La tupla restituita è (*nome*, *suffisso*, *modo*, *mtype*), dove *nome* è il nome del modulo senza il nome dei package che lo contengono, *suffisso* è la parte finale del nome del file (che potrebbe non essere una estensione con punto), *modo* è la modalità `open()` che potrebbe venire usata (`'r'` o `'rb'`), e *mtype* è un intero che indica il tipo di modulo. *mtype* avrà un valore che può venire confrontato con le costanti definite nel modulo `imp`; verificate la documentazione di quel modulo per avere maggiori informazioni sui tipi del modulo stesso.

getmodulename(*percorso*)
Restituisce il nome del modulo nominato dal file *percorso*, non includendo i nomi dei package che lo contengono. Viene usato lo stesso algoritmo che l'interprete utilizza quando cerca i moduli. Se il nome non viene a corrispondere con le convenzioni dell'interprete, viene restituito `None`.

ismodule(*oggetto*)
Restituisce vero se l'oggetto è un modulo.

isclass(*oggetto*)
Restituisce vero se l'oggetto è una classe.

ismethod(*oggetto*)
Restituisce vero se l'oggetto è un metodo.

isfunction(*oggetto*)
Restituisce vero se l'oggetto è una funzione Python oppure una funzione senza nome (lambda).

istraceback(*oggetto*)
Restituisce vero se l'oggetto è una traceback.

isframe(*oggetto*)
Restituisce vero se l'oggetto è un frame.

iscode(*oggetto*)
Restituisce vero se l'oggetto è un codice.

isbuiltin(*oggetto*)
Restituisce vero se l'oggetto è una funzione built-in.

isroutine(*oggetto*)
Restituisce vero se l'oggetto è una funzione definita dall'utente o built-in oppure un metodo.

ismethoddescriptor(*oggetto*)
Restituisce vero se l'oggetto è un descrittore di metodo, ma non se ismethod() o isclass() oppure isfunction() restituiscono vero.

Questo metodo esiste dalla versione 2.2 di Python e, per esempio, restituisce vero con int.__add__. Un oggetto che supera questo test ha l'attributo __get__ ma non __set__; ma al di là di questo, è l'insieme degli attributi che varia. __name__ è di solito rilevante e __doc__ lo è spesso.

Metodi, implementati mediante descrittori, che superano altresì uno degli altri test, restituiscono falso con il test ismethoddescriptor(), semplicemente perché gli altri test offrono più garanzie – potete, ad esempio, contare sul fatto di avere l'attributo im_func (etc) quando un oggetto supera ismethod().

isdatadescriptor(*oggetto*)
Restituisce vero se l'oggetto è un descrittore di dati.

I descrittori di dati hanno sia un attributo __get__ che un attributo __set__. Esempi di descrittori di dati sono le proprietà (definite in Python), i getset ed i membri (definiti in C). Tipicamente, i descrittori di dati avranno anche gli attributi __name__ e __doc__ (le proprietà, i getset ed i membri hanno entrambi questi attributi), ma questo non viene garantito. Nuovo nella versione 2.3.

3.11.2 Recuperare codice sorgente

getdoc(*oggetto*)
Con questo metodo si ottiene la stringa di documentazione di un oggetto. Tutti i caratteri di tabulazione

vengono convertiti in spazi. Per adattare le stringhe in modo tale da indentarle con i blocchi di codice, ogni spazio vuoto che possa essere rimosso in modo uniforme dalla seconda riga in poi, viene tolto.

getcomments(*oggetto*)
Restituisce in una singola stringa qualsiasi linea di commento immediatamente precedente il codice sorgente di oggetto (nel caso sia una classe, una funzione o un metodo), oppure presente all'inizio del file sorgente Python (se oggetto è un modulo).

getfile(*oggetto*)
Restituisce il nome del file (di testo o binario) nel quale oggetto viene definito. Questo metodo fallirà sollevando un'eccezione `TypeError` se oggetto è un modulo built-in, una classe, oppure una funzione.

getmodule(*oggetto*)
Cerca di stabilire in quale modulo un oggetto è stato definito.

getsourcefile(*oggetto*)
Restituisce il nome del file sorgente Python, nel quale è stato definito un oggetto. Questo metodo fallirà sollevando un'eccezione `TypeError` se l'oggetto è un modulo built-in, una classe oppure una funzione.

getsourcelines(*oggetto*)
Restituisce una lista contenente le righe di codice sorgente ed il numero di riga iniziale di un oggetto. L'argomento può essere un modulo, una classe, un metodo, una funzione, una traceback, un frame, oppure codice oggetto. Il codice sorgente viene restituito come lista contenente le righe corrispondenti all'oggetto, ed il numero della riga che indica dove, nel file sorgente originale, viene individuata la prima riga di codice. Un'eccezione `IOError` viene sollevata se il codice sorgente non può essere recuperato.

getsource(*oggetto*)
Restituisce il testo del codice sorgente dell'oggetto. L'argomento può essere un modulo, un metodo, una funzione, una traceback, un frame o un codice oggetto. Il codice sorgente viene restituito come una singola stringa. Se il codice sorgente non può essere recuperato, viene sollevata un'eccezione `IOError`.

3.11.3 Classi e funzioni

getclasstree(*classes*[, *unique*])
Organizza la lista di classi data in una gerarchia di liste annidate. Dove appare una lista annidata, questa contiene classi derivate dalla classe il cui elemento è quello immediatamente precedente nella lista. Ciascun elemento della lista è una tupla di due elementi contenente una classe e la tupla delle sue classi base. Se l'argomento *unique* ha valore vero, la struttura restituita contiene un unico elemento per ogni classe della lista data. Altrimenti, le classi che fanno uso dell'ereditarietà multipla e le loro discendenti appariranno più di una volta.

getargspec(*func*)
Trova i nomi e i valori predefiniti degli argomenti di una funzione. Viene restituita una tupla di quattro elementi: (*args*, *varargs*, *varkw*, *defaults*). *args* è la lista dei nomi degli argomenti, (può contenere liste annidate). *varargs* e *varkw* sono rispettivamente i nomi degli argomenti di tipo `*` e `**`; se non ve ne sono assumono il valore `None`. *defaults* è una tupla di valori predefiniti dell'argomento; se questa tupla ha *n* elementi, questi corrispondono agli ultimi *n* elementi elencati in *args*.

getargvalues(*frame*)
Trova le informazioni sugli argomenti passati in uno specifico frame. Viene restituita una tupla di quattro elementi: (*args*, *varargs*, *varkw*, *locals*). *args* è una lista dei nomi degli argomenti (può contenere liste annidate). *varargs* e *varkw* sono rispettivamente i nomi degli argomenti di tipo `*` e `**`; se non ve ne sono assumono il valore `None`. *locals* è il dizionario degli elementi locali del frame dato.

formatargspec(*args*[, *varargs, varkw, defaults, argformat, varargsformat, varkwformat, defaultformat*])
Formatta una descrizione adatta alla stampa per i quattro valori restituiti da `getargspec()`. Gli altri quattro argomenti sono le corrispondenti funzioni facoltative di formattazione che, se presenti, vengono chiamate per trasformare nomi e valori in stringhe.

formatargvalues(*args*[, *varargs, varkw, locals, argformat, varargsformat, varkwformat, valueformat*])
Formatta una descrizione adatta alla stampa per i quattro valori restituiti da `getargvalues()`. Gli altri quattro argomenti sono le corrispondenti funzioni facoltative di formattazione che se presenti, vengono chiamate per convertire nomi e valori in stringhe.

getmro(*cls*)
Restituisce una tupla composta dalle classi di base utilizzate per realizzare la classe generica cls, e la classe cls stessa, nell'ordine usato per la risoluzione dei metodi. Ogni classe appare una sola volta nella tupla. Notate che l'ordine di risoluzione dei metodi dipende dal tipo di cls. A meno che non stiate usando un metatype definito dall'utente in modo molto particolare, cls sarà il primo elemento della tupla.

3.11.4 Lo stack dell'interprete

Quando si dice che le funzioni descritte più avanti restituiscono dei "frame record", si intende che ognuno dei record è formato da una tupla di sei elementi: l'oggetto frame, il nome del file, il numero della riga di codice corrente, il nome della funzione, una lista di righe di contesto dal codice sorgente, e l'indice della riga corrente all'interno della lista.

Mantenere dei riferimenti agli oggetti frame, come quelli che vengono trovati come primo elemento dei frame record restituiti dalle funzioni qui descritte, può far sì che il vostro programma crei dei cicli di riferimenti. Una volta che un ciclo di riferimenti viene creato, il tempo di vita di tutti gli oggetti che formano il ciclo può diventare molto più lungo del voluto, anche se si è abilitato in Python il rilevatore di cicli, opzionale. Se tali cicli devono venire creati, è importante assicurare che vengano esplicitamente interrotti una volta esaurito il loro compito, per evitare di ritardare la distruzione degli oggetti a cui fa riferimento il ciclo, ed il conseguente aumento della memoria utilizzata.

Sebbene il rilevatore individui questi cicli, la distruzione dei frame (e delle variabili locali) può venire effettuata deterministicamente rimuovendo il ciclo con una clausola `finally`. Questo è anche importante quando la rilevazione dei cicli viene disabilitata durante la compilazione di Python o usando `gc.disable()`. Per esempio:

```
def handle_stackframe_without_leak():
    frame = inspect.currentframe()
    try:
        # fa qualcosa con il frame
    finally:
        del frame
```

L'argomento facoltativo *context*, supportato da molte di queste funzioni, specifica il numero di righe del contesto da restituire, incentrate sulla riga corrente.

getframeinfo(*frame*[, *context*])
Trova le informazioni su un oggetto frame o un traceback. Viene restituita una tupla di 5 elementi, corrispondenti agli ultimi cinque elementi del frame record.

getouterframes(*frame*[, *context*])
Trova una lista di frame record corrispondenti ai frame esterni rispetto a quello fornito. Questi frames rappresentano le chiamate che guidano la creazione del *frame*. La prima voce nell'elenco restituito rappresenta il *frame*; l'ultima voce rappresenta la chiamata più remota sullo stack del *frame*.

getinnerframes(*traceback*[, *context*])
Trova una lista di frame record per il frame del traceback e tutti i frame interni. Questi frame rappresentano chiamate fatte in conseguenza del *frame*. La prima voce nell'elenco rappresenta il *traceback*; l'ultima voce indica dove viene sollevata l'eccezione.

currentframe()
Restituisce l'oggetto frame per le chiamate dello stack.

stack([*context*])
Restituisce una lista di frame record per le chiamate dello stack. La prima voce dell'elenco restituito rappresenta il chiamante; l'ultima voce indica la chiamata più esterna sullo stack.

trace([*context*])
Restituisce una lista di frame record per lo stack tra il frame corrente e quello in cui viene gestita la sollevazione dell'eccezione corrente. La prima voce dell'elenco restituito rappresenta il chiamante; l'ultima voce indica dove viene sollevata l'eccezione.

3.12 `traceback` — Stampa o recupera la traccia dello stack

Questo modulo fornisce un'interfaccia standard per estrarre, formattare e stampare tracce dello stack di programmi Python. Il modulo ripete esattamente il comportamento dell'interprete di Python quando stampa una traccia dello stack. Questo risulta utile quando si vuole stampare tracce dello stack sotto il controllo del programma, come in un "wrapper" (NdT: involucro) attorno all'interprete.

Il modulo usa oggetti traceback — questo è il tipo di oggetto che viene immagazzinato nelle variabili `sys.exc_traceback` (deprecato) e `sys.last_traceback`, e restituito come terzo elemento da `sys.exc_info()`.

Il modulo definisce le funzioni seguenti:

`print_tb`(*traceback*[, *limit*[, *file*]])
Stampa gli elementi della traccia dello stack fino a *limit*, a partire da *traceback*. Se *limit* viene omesso o è `None`, vengono stampati tutti gli elementi. Se *file* viene omesso o è `None`, l'output viene stampato su `sys.stderr`; altrimenti a ricevere l'output devono essere un file aperto o un oggetto file.

`print_exception`(*type*, *value*, *traceback*[, *limit*[, *file*]])
Stampa le informazioni della traccia dello stack relative all'eccezione, fino a *limit* da *traceback* a *file*. Differisce da `print_tb()` nei modi seguenti: (1) se *traceback* non è `None`, stampa la parte iniziale (header) '`Traceback (most recent call last):`'; (2) stampa il *type* di eccezione ed il *valore* dopo la traccia dello stack; (3) se il *type* è `SyntaxError` ed il *value* ha il formato appropriato, stampa la riga dove si è verificato l'errore di sintassi, con un carattere indicatore '`^`', segnalante la posizione approssimativa dell'errore.

`print_exc`([*limit*[, *file*]])
Questa è una scorciatoia per `print_exception(sys.exc_type, sys.exc_value, sys.exc_traceback,` *limit*`,` *file*`)`. (Infatti, usa `sys.exc_info()` per recuperare le stesse informazioni in modalità threadsafe (NdT: di sicuro utilizzo con i thread) invece di usare le variabili deprecate.)

`format_exc`([*limite*[, *file*]])
Questa funzione è simile a `print_exc(`*limit*`)` ma restituisce una stringa invece che stampare su un file. Nuovo nella versione 2.4.

`print_last`([*limite*[, *file*]])
Questa è una scorciatoia per `print_exception(sys.last_type, sys.last_value, sys.last_traceback,` *limit*`,` *file*`)`.

`print_stack`([*f*[, *limit*[, *file*]]])
Questa funzione stampa una traccia dello stack dal punto in cui viene chiamata. L'argomento facoltativo *f* può venire usato per specificare un frame alternativo dello stack da cui partire. Gli argomenti facoltativi *limit* e *file* hanno lo stesso significato che in `print_exception()`.

`extract_tb`(*traceback*[, *limit*])
Restituisce una lista, fino a *limit*, di elementi "pre elaborati" della traccia dello stack, estratti dall'oggetto *traceback* passato alla funzione. È utile per formattare diversamente le tracce dello stack. Se il *limit* viene omesso o è `None`, tutti gli elementi vengono estratti. Un elemento "pre elaborato" della traccia dello stack è una tupla di 4 elementi (*nomefile*, *numero di riga*, *nome funzione*, *testo*) che rappresenta tutte le informazioni che solitamente vengono stampate nella traccia dello stack. Il testo è una stringa con gli spazi vuoti iniziali e di coda rimossi; se il sorgente non è disponibile, la stringa è `None`.

`extract_stack`([*f*[, *limit*]])
Estrae una traccia raw dal frame dello stack corrente. Il valore restituito ha lo stesso formato di quello per `extract_tb()`. Gli argomenti facoltativi *f* e *limit* hanno lo stesso significato di quelli utilizzati per `print_stack()`.

`format_list`(*list*)
Allo stesso modo di come `extract_tb()` o `extract_stack()` restituiscono una lista di tuple, questo comando restituisce una lista di stringhe pronte per la stampa. Ogni stringa, nella lista risultante, corrisponde all'elemento con lo stesso indice nella lista degli argomenti. Ogni stringa termina con un fine riga; le stringhe possono contenere internamente dei fine riga, per quegli elementi la cui riga di testo sorgente non è uguale

a None.

format_exception_only(*type, value*)
Formatta la parte relativa all'eccezione di una traceback. Gli argomenti sono il tipo di eccezione ed il valore così come viene restituito da sys.last_type e sys.last_value. Il valore restituito è una lista di stringhe, ognuna terminante con un fine riga. Normalmente la lista contiene una singola stringa; comunque, per le eccezioni SyntaxError, il valore restituito contiene più linee, che (quando stampate) visualizzano informazioni dettagliate riguardanti il punto si è verificato l'errore sintattico. Il messaggio indicante il tipo di eccezione è sempre l'ultima stringa della lista.

format_exception(*type, value, tb*[, *limit*])
Formatta la traccia dello stack e l'informazione sull'eccezione. Gli argomenti hanno lo stesso significato di corrispondenti della funzione print_exception(). Il valore restituito è una lista di stringhe ciascuna terminante con un fine riga, alcune contenenti a loro volta dei fine riga interni. Quando queste linee vengono concatenate e stampate, il risultato è lo stesso di quello di print_exception().

format_tb(*tb*[, *limit*])
Una scorciatoia per format_list(extract_tb(*tb*, *limit*)).

format_stack([*f*[, *limit*]])
Una scorciatoia per format_list(extract_stack(*f*, *limit*)).

tb_lineno(*tb*)
Questa funzione restituisce l'insieme dei numeri di riga correnti nell'oggetto traceback. Si è resa necessaria in quanto in versioni precedenti a Python 2.3, quando veniva passato l'argomento **-O** a Python, non veniva aggiornato correttamente *tb*.tb_lineno. Questa funzione non viene utilizzata nelle versioni successive a 2.3.

3.12.1 Esempi sulla traceback

Questo semplice esempio implementa un ciclo base read-eval-print (NdT: leggi-valuta-stampa) simile (ma meno utile) al ciclo dell'interprete interattivo standard di Python. Per una più completa implementazione del ciclo dell'interprete, vedete il modulo code.

```
import sys, traceback

def run_user_code(envdir):
    source = raw_input(">>> ")
    try:
        exec source in envdir
    except:
        print "Eccezione in codice utente:"
        print '-'*60
        traceback.print_exc(file=sys.stdout)
        print '-'*60

envdir = {}
while 1:
    run_user_code(envdir)
```

3.13 linecache — Accesso casuale a righe di testo

Il modulo linecache permette di estrapolare qualunque riga da qualsiasi file mentre si cerca di ottimizzare internamente, utilizzando una cache; il caso comune è quello in cui vengono lette molte righe da un singolo file. Questo modulo viene utilizzato dal modulo traceback per recuperare le righe sorgenti col fine di includerle in una traceback formattata.

Il modulo linecache definisce le seguenti funzioni:

getline(*nomefile, lineno*)
Estrapola la riga *lineno* dal file chiamato *nomefile*. Questa funzione non solleverà mai un'eccezione — restituirà '' sugli errori (il carattere di fine riga verrà incluso in tutte le righe trovate).

Se non viene trovato un file chiamato *nomefile* la funzione lo cercherà nel percorso di ricerca del modulo, `sys.path`.

clearcache()
Ripulisce la cache. Utilizzate questa funzione se non sono più necessarie le righe precedentemente lette utilizzando `getline()`.

checkcache()
Controlla la validità della cache. Utilizzate questa funzione se i file nella cache possono aver subito delle modifiche sul disco, e ne richiedete la versione aggiornata.

Esempio:

```
>>> import linecache
>>> linecache.getline('/etc/passwd', 4)
'sys:x:3:3:sys:/dev:/bin/sh\n'
```

3.14 `pickle` — Serializzazione di oggetti Python

Il modulo `pickle` implementa un basilare ma potente algoritmo, per serializzare e deserializzare una struttura di oggetti Python. "Pickling" (NdT: serializzazione) è il processo mediante il quale una gerarchia di oggetti Python viene convertita in un flusso di byte, e "unpickling" (NdT: deserializzazione) è l'operazione inversa, nella quale un flusso di byte viene riconvertito in una gerarchia di oggetti. Pickling (e unpickilng) sono alternativamente noti come "serialization", "marshalling"[2] o "flattening"; comunque sia, per non fare confusione i termini usati qui sono "serializzare" e "deserializzare".

Questa documentazione descrive sia il modulo `pickle` che il `cPickle`.

3.14.1 Relazioni con altri moduli Python

Il modulo `pickle` ha un cugino ottimizzato chiamato modulo `cPickle`. Come implicitamente dice il suo nome, `cPickle` è scritto in C, così da poter essere sino a 1000 volte più veloce di `pickle`. Comunque sia non è in grado di derivare sotto classi dalle classi `Pickler()` e `Unpickler()`, perché in `cPickle` queste sono funzioni, non classi. La maggior parte delle applicazioni non necessitano di questa funzionalità, e possono beneficiare delle migliori prestazioni di `cPickle`. Oltre a questo, le interfacce dei due moduli sono pressoché identiche; l'interfaccia comune viene descritta in questo manuale e le differenze vengono messe in rilievo dove necessario. Nelle discussioni seguenti, si userà il termine "pickle" per descrivere globalmente sia il modulo `pickle` che `cPickle`.

I flussi di dati prodotti dai due moduli vengono garantiti per essere interscambiabili.

Python ha un modulo di serializzazione più primitivo chiamato `marshal`, ma in generale `pickle` dovrebbe essere sempre il modo preferenziale per serializzare oggetti Python. `marshal` esiste principalmente per supportare i file Python '.pyc'.

Il modulo `pickle` differisce da `marshal` per diverse significative caratteristiche:

- Il modulo `pickle` tiene traccia degli oggetti che ha già serializzato, così che i riferimenti sucessivi allo stesso oggetto non verranno nuovamente serializzati. `marshal` non fa questo.

  Ciò ha implicazioni sia per gli oggetti ricorsivi che per la condivisione degli oggetti. Gli oggetti ricorsivi sono oggetti contenenti riferimenti a sé stessi. Non vengono gestiti da marshal, e infatti, l'utilizzo di marshal su oggetti ricorsivi provocherà il crash dell'interprete Python. La condivisione di oggetti si presenta quando

[2] Non confondetelo con il modulo `marshal`

vi sono molteplici riferimenti allo stesso oggetto, in posti differenti nella gerarchia dell'oggetto che è stato serializzato. `pickle` memorizza questi oggetti una volta sola, e assicura che tutti gli altri riferimenti puntino all'originale. Gli oggetti condivisi rimangono condivisi, caratteristica questa molto importante nel caso di oggetti mutevoli.

- `marshal` non può venire usato per serializzare le classi definite dall'utente e le loro istanze. `pickle` può salvare e ripristinare classi ed istanze in maniera trasparente, considerato comunque che la definizione di classe deve essere importabile e presente nello stesso modulo, alla stessa maniera del momento in cui l'oggetto è stato memorizzato.
- Il formato di serializzazione di `marshal` non viene garantito per essere portabile attraverso le varie versioni di Python. Visto che il suo lavoro principale è quello di supportare i files '.pyc', gli sviluppatori di Python si riservano il diritto di cambiare il formato di serializzazione in maniera non retrocompatibile, se sarà necessario. Il formato di serializzazione di `pickle` è garantito essere retrocompatibile nelle varie versioni di Python.

Il modulo `pickle` non viene considerato sicuro contro dati costruiti erroneamente o in modo malizioso. Evitate di deserializzare dati ricevuti da fonte inattendibile o sconosciuta.

Notate che la serializzazione è un concetto più primitivo di quello della persistenza; sebbene `pickle` scriva e legga file oggetto, non gestisce il problema del nominare oggetti persistenti, così come il caso (ancora più complicato) degli accessi simultanei ad oggetti persistenti. Il modulo `pickle` può trasformare un oggetto complesso in un flusso di byte, e può trasformare il flusso di byte in un oggetto con la stessa struttura interna. Forse la cosa più ovvia da fare con questi flussi di byte è quella di scriverli su di un file, ma è altresì concepibile inviarli attraverso una rete o salvarli in un database. Il modulo `shelve` fornisce una semplice interfaccia ad oggetti serializzati o deserializzati su file database in stile DBM.

3.14.2 Il formato di flusso dei dati

Il formato dei dati usati da `pickle` è specifico di Python. Questo ha il vantaggio che non ci sono restrizioni imposte da standard esterni come per XDRR (che non può rappresentare una condivisione dei puntatori); comunque sia, questo significa che programmi non-Python non hanno la capacità di ricostruire oggetti serializzati con Python.

In modo predefinito, il formato dati di `pickle` usa una rappresentazione ASCII stampabile. Questa è leggermente più voluminosa di una rappresentazione binaria. Il grande vantaggio nell'usare ASCII stampabile (e qualche altra caratteristica nella rappresentazione di `pickle`) è che per debugging o per necessità di ripristino, è possibile comprendere il contenuto di file serializzati semplicemente leggendoli con un editor di testi standard.

Esistono attualmente 3 differenti protocolli che possono venire usati per la serializzazione.

- Protocollo versione 0 è l'originale protocollo ASCII, ed è compatibile all'indietro con le versioni più vecchie di Python.
- Protocollo versione 1 è il vecchio formato binario che è compatibile anche con le versioni più vecchie di Python.
- Protocollo versione 2 è stato introdotto in Python 2.3. Fornisce un meccanismo più efficiente per serializzare classi di nuovo stile.

Fate riferimento alla PEP 307 per maggiori informazioni.

Se un *protocol* non viene specificato, viene utilizzata la versione 0. Se il *protocol* viene specificato indicando un valore negativo o `HIGHEST_PROTOCOL`, viene usato il protocollo disponibile con il valore più alto.

Modificato nella versione 2.3: Il parametro *bin* è deprecato e viene fornito soltanto per compatibilità all'indietro con le versioni precedenti. Al suo posto usate il parametro *protocol*.

Un formato binario, che è nettamente più efficiente, può venire scelto specificando un valore vero per l'argomento *bin* al costruttore `Pickler` o alle funzioni `dump()` e `dumps()`. Un *protocol* versione >=1 implica l'uso di un formato binario.

3.14.3 Uso

Per serializzare una gerarchia di oggetti, per prima cosa si deve creare un oggetto pickler, successivamente se ne richiami il metodo `dump()`. Per deserializzare un flusso di dati, si deve creare prima un oggetto unpickler, e poi se ne richiami il metodo `load()`. Il modulo `pickle` fornisce la seguente costante:

`HIGHEST_PROTOCOL`
Il protocollo con versione più alta disponibile. Questo valore può venire passato come un valore *protocol*. Nuovo nella versione 2.3.

Il modulo `pickle` fornisce le seguenti funzioni per rendere questo processo più funzionale:

`dump`(*object, file*[, *protocol*[, *bin*]])
Scrive una rappresentazione serializzata di *object* nell'oggetto file aperto *file*. Questo è equivalente a `Pickler(`*file*`, `*protocol*`, `*bin*`).dump(`*object*`)`.

Se il parametro *protocol* viene omesso, viene utilizzato il protocollo versione 0. Se *protocol* viene specificato come un numero negativo o `HIGHEST_PROTOCOL`, il protocollo con il valore più alto viene utilizzato.

Modificato nella versione 2.3: È stato aggiunto il parametro *protocol*. Il parametro *bin* è deprecato e viene fornito soltanto per compatibilità all'indietro con le versioni precedenti. Al suo posto usate il parametro *protocol*.

Se l'argomento facoltativo *bin* è vero, viene utilizzato il formato binario di pickle; altrimenti, viene utilizzato il (meno efficiente) formato testo di pickle (per compatibilità all'indietro, questo è predefinito).

Il *file* deve avere un metodo `write()` che accetti un argomento di una singola stringa. Può quindi essere un oggetto file aperto in scrittura, un oggetto `StringIO`, o qualsiasi altro oggetto personalizzato che rispetti questa interfaccia.

`load`(*file*)
Legge una stringa dal file oggetto *file* aperto, e lo interpreta come un flusso di dati serializzato, ricostruendo e restituendo l'originale gerarchia dell'oggetto. Questo è l'equivalente di `Unpickler(`*file*`).load()`.

file deve possedere due metodi, un metodo `read()` che prende un argomento di tipo intero e un metodo `readline()` che non richiede argomenti. Entrambi i metodi devono restituire una stringa. Così *file* può essere un oggetto file aperto in lettura, un oggetto `StringIO`, o qualsiasi altro oggetto personalizzato che rispetti questa interfaccia.

Questa funzione determina automaticamente se il flusso dei dati viene scritto in modo binario o meno.

`dumps`(*object*[, *protocol*[, *bin*]])
Restituisce una rappresentazione serializzata dell'oggetto come stringa, invece che scriverlo in un file.

Se il parametro *protocol* viene omesso, si utilizza il protocollo versione 0. Se *protocol* viene specificato con un valore negativo o con `HIGHEST_PROTOCOL`, verrà utilizzata la versione con il protocollo maggiore.

Modificato nella versione 2.3: Il parametro *protocol* è stato aggiunto. Il parametro *bin* è deprecato e viene fornito solo per compatibilità con le versioni precedenti. Al suo posto, usate il parametro *protocol*.

Se l'argomento facoltativo *bin* è vero, pickle viene usato in formato binario; altrimenti (ma meno efficiente) pickle viene usato in formato testo (predefinito).

`loads`(*string*)
Legge da una stringa, una gerarchia dell'oggetto serializzato. I caratteri nella stringa successivi alla rappresentazione dell'oggetto serializzato vengono ignorati.

Il modulo `pickle` definisce anche tre eccezioni:

exception `PickleError`
Una classe comune per le altre eccezioni definite di seguito. Questa eredita da `Exception`.

exception `PicklingError`
Questa eccezione viene sollevata quando un oggetto non serializzabile viene passato al metodo `dump()`.

exception `UnpicklingError`
Questa eccezione viene sollevata quando si verifica un problema effettuando la deserializzazione di un oggetto. Notate che altre eccezioni possono venire sollevate durante le operazioni di deserializzazione,

incluse (ma non necessariamente limitate ad esse) `AttributeError`, `EOFError`, `ImportError` e `IndexError`.

Il modulo `pickle` esporta anche due oggetti chiamabili [3], `Pickler` e `Unpickler`:

class `Pickler`(*file*[, *protocol*[, *bin*]])
Questa classe prende un oggetto file (o un oggetto con le medesime caratteristiche) sul quale scriverà un flusso di dati serializzati.

Se si omette il parametro *protocol*, viene usato protocol 0. Se si specifica *protocol* come un valore negativo, verrà usato il protocollo con versione più alta.

Modificato nella versione 2.3: Il parametro *bin* è deprecato e viene fornito solo per compatibilità all'indietro con le versioni precedenti. Al suo posto, usate il parametro *protocol*.

Se il parametro facoltativo *bin* è vero, indica a pickler di usare il più efficiente formato di serializzazione binario, altrimenti viene usato il formato ASCII (predefinito).

L'oggetto *file* deve avere il metodo `write()` che accetti una singola stringa di argomenti. Può essere perciò un oggetto file aperto, un oggetto `StringIO`, o ogni altro oggetto personalizzato che rispetti questa interfaccia.

Gli oggetti `Pickler` definiscono uno (o due) metodi pubblici:

`dump`(*object*)
Scrive una rappresentazione serializzata di *object* nell'oggetto file aperto dato al costruttore. Sia il formato binario che il formato ASCII verranno usati, dipendentemente dal valore dell'opzione di *bin* passata al costruttore.

`clear_memo`()
Ripulisce il "memo" del pickler. Il memo è la struttura di dati che ricorda quali oggetti sono già stati visti dal pickler, così che oggetti condivisi o ricorsivi vengano serializzati per riferimento e non per valore. Questo metodo è utile quando i pickler vengono riutilizzati.

Note: Prima di Python 2.3, `clear_memo()` era disponibile solo per i serializzatori creati da `cPickle`. Nel modulo `pickle`, i serializzatori hanno un'istanza variabile chiamata `memo` che è un dizionario Python. Così, per ripulire il memo per un modulo serializzatore `pickle`, si potrebbe fare come di seguito:

```
mypickler.memo.clear()
```

In codice che non necessiti di supportare vecchie versioni di Python, usate semplicemente `clear_memo()`.

È possibile effettuare chiamate multiple al metodo `dump()` per la stessa istanza `Pickler`. Queste devono quindi essere equivalenti allo stesso numero di chiamate del metodo `load()` della corrispondente istanza `Unpickler`. Se lo stesso oggetto viene serializzato da chiamate multiple a `dump()`, il `load()` deve rendere tutti i riferimenti allo stesso oggetto. [4]

Gli oggetti `Unpickler` vengono definiti come:

class `Unpickler`(*file*)
Questo prende un oggetto file (o un oggetto con le medesime caratteristiche) da cui leggerà un flusso di dati serializzati. Questa classe determina automaticamente se il flusso di dati è stato scritto in modo binario o meno, quindi non necessita di un'opzione come nella costruzione di `Pickler`.

file deve avere due metodi, un metodo `read()` che prende come argomento un intero, e un metodo `readline()` che non richiede argomenti. Entrambi i metodi devono restituire una stringa. Così *file* può essere un oggetto file aperto in lettura, un oggetto `StringIO`, o qualsiasi altro oggetto personalizzato che rispetti questa interfaccia.

Gli oggetti `Unpickler` hanno uno (o due) metodi pubblici:

[3] Nel modulo `pickle` questi oggetti sono delle classi, che possono diventare sotto classi personalizzate. Tuttavia, nel modulo `cPickle` questi oggetti chiamabili sono funzioni factory e non è possibile derivarne delle sotto classi. Una comune ragione per derivare sotto classi consiste nel voler controllare quale oggetto può attualmente venire deserializzato. Vedete la sezione 3.14.6 per maggiori dettagli.

[4] *Attenzione*: questo viene inteso per la serializzazione di oggetti multipli senza modificare gli oggetti stessi, o loro parti. Se modificate un oggetto e quindi lo serializzate ancora usando la stessa instanza `Pickler`, l'oggetto non verrà serializzato ancora — un riferimento ad esso viene serializzato e `Unpickler` restituirà il vecchio valore, non quello modificato. Ci sono due problemi qui: (1) individuare i cambiamenti, e (2) combinare un insieme minimo di cambiamenti. La Garbage Collection qui può anche diventare un problema.

load()
: Legge una rappresentazione di un oggetto serializzato da un oggetto file aperto passato al costruttore, e restituisce l'oggetto ricostituito gerarchicamente come specificato al suo interno.

noload()
: Questo è identico a `load()` ad eccezione del fatto che non crea alcun oggetto. Ciò è utile principalmente per trovare quelli che vengono chiamati "id persistenti", referenziati all'interno di un flusso di dati serializzato. Vedete la sottostante sezione 3.14.5 per ulteriori dettagli.

  Note: il metodo `noload()` è l'unico attualmente disponibile per gli oggetti `Unpickler` creati dal modulo `cPickle`. I moduli `pickle` non serializzabili non hanno il metodo `noload()`.

3.14.4 Cosa può essere serializzato e deserializzato?

I seguenti tipi possono essere serializzati:

- `None`, `True` e `False`
- Interi, interi long, numeri in virgola mobile, numeri complessi
- stringhe normali ed Unicode
- tuple, liste, insiemi e dizionari contenenti solamente oggetti serializzabili
- funzioni definite nella parte iniziale di un modulo
- funzioni built-in definite nella parte iniziale di un modulo
- classi definite nella parte iniziale di un modulo
- istanze di classe i cui membri `__dict__` o `__setstate__()` siano serializzabili (vedere la sezione 3.14.5 per i dettagli)

I tentativi di serializzare oggetti non serializzabili, solleveranno l'eccezione `PicklingError`; quando questo si verifica, un numero indefinito di bytes nel file sottostante potrebbe essere già stato scritto.

Notate che le funzioni (sia built-in che definite dall'utente) vengono serializzate mediante un nome di riferimento "pienamente qualificato", non mediante un valore. Ciò significa che viene serializzato solo il nome della funzione, mediante il nome del modulo con cui viene definita la funzione stessa. Non vengono serializzati né il codice della funzione né alcuno dei suoi attributi. Perciò il modulo che definisce la funzione deve essere importabile nell'ambiente non serializzabile, e deve contenere l'oggettto nominato, altrimenti verrà sollevata un'eccezione. [5]

Analogamente le classi vengono serializzate in base al nome di riferimento, cosicché vengono applicate le stesse restrizioni adottate nell'ambiente deserializzato. Notate che non viene serializzato nessun dato o codice delle classi, così nel seguente esempio l'attributo di classe `attr` non viene ristabilito nell'ambiente deserializzato:

```
class Foo:
    attr = 'a class attr'

picklestring = pickle.dumps(Foo)
```

Queste restrizioni sono il motivo per cui le funzioni e le classi serializzabili debbano venire definite nelle righe iniziali dei moduli.

Analogamente, quando vengono serializzate istanze di classi, non vengono serializzati attraverso di esse né il loro codice né i loro dati. Vengono serializzati solo i dati di istanze. Questo è stato fatto di proposito, così che sia possibile correggere degli errori in una classe, o aggiungere metodi alla classe, ed ancora caricare oggetti che sono stati creati con una precedente versione della classe. Se prevedete di utilizzare oggetti longevi che possano sopravvivere a molte versioni di una classe, allora può essere conveniente inserire un numero di versione nell'oggetto, in modo tale che le conversioni desiderate possano venire effettuate dal metodo della classe `__setstate__()`.

[5] L'eccezione sollevata probabilmente potrebbe essere una `ImportError` o una `AttributeError`, ma potrebbe anche essere qualcosa di diverso.

3.14.5 Il protocollo pickle

Questa sezione descrive il "pickling protocol" che definisce l'interfaccia fra il pickler/unpickler e gli oggetti che stanno per essere serializzati. Questo protocollo fornisce uno standard per definire, personalizzare e controllare il modo in cui i vostri oggetti vengono serializzati e deserializzati. La descrizione fatta in questa sezione non tratta personalizzazioni specifiche, che possono venire impiegate per rendere un po' più sicuro l'ambiente di deserializzazione da flussi di dati serializzati non fidati; vedete la sezione 3.14.6 per ulteriori dettagli.

Serializzazione e deserializzazione di normali istanze di classe

Quando una istanza di classe serializzata viene deserializzata, normalmente il suo metodo `__init__()` *non* viene invocato. Se si vuole che il metodo `__init__()` venga richiamato nella deserializzazione, una classe di vecchio stile può definire un metodo `__getinitargs__()`, che dovrebbe restituire una *tupla* contenente gli argomenti da passare al costruttore di classi (per es. `__init__()`). Il metodo `__getinitargs__()` viene richiamato all'atto della serializzazione; la tupla che esso restituisce viene incorporata nel pickle per l'istanza.

I tipi di nuovo stile possono fornire il metodo `__getnewargs__()` che viene usato per il protocollo 2. È necessario implementare tale metodo se il tipo stabilisce alcune invarianti interne quando viene creata l'istanza, o se l'allocazione di memoria è influenzata dal valore passato al metodo `__new__()` per il tipo (nelle stringhe e nelle tuple avviene la stessa cosa). Istanze di nuovo stile di tipo `C` vengono create usando

```
obj = C.__new__(C, *args)
```

dove *args* è il risultato che si ottiene chiamando `__getnewargs__()` sull'oggetto originale; se non esiste `__getnewargs__()`, viene assunta una tupla vuota.

Le classi possono anche influire sul modo in cui le proprie istanze vengono serializzate; se la classe definisce il metodo `__getstate__()`, questo viene invocato ed il risultato viene serializzato come il contenuto dell'istanza invece che come contenuto del dizionario dell'istanza. Se non esiste il metodo `__getstate__()`, viene serializzata l'istanza `__dict__`.

Riguardo alla deserializzazione, se la classe definisce anche il metodo `__setstate__()`, viene invocata con lo stato deserializzato.[6] Se non esiste un metodo `__setstate__()`, lo stato serializzato deve essere un dizionario ed i suoi componenti vengono assegnati al nuovo dizionario di istanze. Se una classe definisce sia `__getstate__()` che `__setstate__()`, lo stato dell'oggetto non necessariamente deve essere un dizionario, e questi metodi possono fare ciò che vogliono.[7]

Per classi di nuovo stile, se `__getstate__()` restituisce un valore falso, non verrà invocato il metodo `__setstate__()`.

Serializzare e deserializzare tipi di estensione

Quando il `Pickler` incontra un oggetto di un tipo di cui non sa nulla — come un tipo di estensione — cerca istruzioni sul modo di serializzarlo in due luoghi. Un'alternativa è che l'oggetto implementi un metodo `__reduce__()`. Se fornito, durante la serializzazione `__reduce__()` verrà chiamato senza argomenti, e restituirà o una stringa o una tupla.

Se viene restituita una stringa, nomina una variabile globale il cui contenuto viene serializzato come al solito. Quando viene restituita una tupla, deve essere di lunghezza due o tre, con la seguente semantica:

- Un oggetto chiamabile, che nell'ambiente di deserializzazione deve essere o una classe, un oggetto chiamabile registrato come un "safe constructor" (vedi sotto), o avere un attributo `__safe_for_unpickling__` con valore vero. Altrimenti, verrà sollevata un'eccezione `UnpicklingError` nell'ambiente di deserialiizzazione. Notate che di solito, questo oggetto chiamabile viene serializzato dal nome.

[6] Questi metodi possono venire usati anche per implementare la copia delle istanze della classe.
[7] Questo protocollo viene anche usato per le operazioni di copia sia superficiale che profonda, come definite nel modulo `copy`.

- Una tupla di argomenti per l'oggetto chiamabile, o `None`. **Deprecato dalla versione 2.3.** Usate, invece, una tupla di argomenti
- Facoltativamente, lo stato dell'oggetto, che verrà passato al metodo `__setstate__()` dell'oggetto, come descritto nella sezione 3.14.5. Se l'oggetto non ha il metodo `__setstate__()`, allora, come sopra, il valore deve essere un dizionario che verrà aggiunto all'oggetto `__dict__`.

Sulla deserializzazione, il chiamabile verrà chiamato (a patto che fornisca i criteri di cui sopra), passando nella tupla degli argomenti; dovrebbe restituire l'oggetto deserializzato.

Se il secondo elemento è `None`, allora invece di chiamare l'oggetto chiamabile direttamente, il suo metodo `__basicnew__()` viene chiamato senza argomenti. Dovrebbe anche restituire l'oggetto deserializzato.

Deprecato dalla versione 2.3. Usate, invece, un tupla di argomenti

Un'alternativa per implementare il metodo `__reduce__()` sull'oggetto da serializzare, consiste nel registrare l'oggetto chiamabile con il modulo `copy_reg`. Questo modulo fornisce per i programmi un modo di registrare le "reduction functions" e i costruttori per i tipi definiti dall'utente. Le funzioni reduction hanno le stesse semantica ed interfaccia del metodo `__reduce__()` descritto sopra, eccetto per il fatto che vengono chiamate con un singolo argomento, ossia l'oggetto che deve essere serializzato.

Il costruttore registrato viene richiesto come "safe constructor" ai fini della deserializzazione descritti precedentemente.

Serializzare e deserializzare oggetti esterni

Per ottenere vantaggi dalla persistenza degli oggetti, il modulo `pickle` supporta la nozione relativa di un riferimento ad un oggetto esterno al flusso dei dati serializzati. Questi oggetti vengono riferiti tramite un "id persistente", che consiste in una stringa arbitraria di caratteri ASCII stampabili. La risoluzione di questi nomi non viene definita dal modulo `pickle`, ma viene delegata alle funzioni definite dall'utente su pickler ed unpickler.[8]

Per definire esternamente la risoluzione di un id persistente, si ha bisogno di impostare l'attributo `persistent_id` nell'oggetto serializzatore, e l'attributo `persistent_load` dell'oggetto deserializzatore.

Per serializzare oggetti che hanno un id persistente esterno, il pickler deve avere un metodo personalizzato `persistent_id()` che prenda un oggetto come un argomento e restituisca o `None` oppure l'id persistente per quell'oggetto. Quando viene restituito `None`, il pickler semplicemente serializza l'oggetto come normale. Quando viene restituita una stringa di id persistente, il pickler serializzerà quella stringa attraverso un marcatore, così che il deserializzatore riconosca la stringa come id persistente.

Per deserializzare oggetti esterni, il deserializzatore deve avere una funzione personalizzata `persistent_load()`, che prenda una stringa id persistente e restituisca l'oggetto referenziato.

Ecco uno sciocco esempio che *potrebbe* fare un po' di luce:

[8] Il meccanismo corrente per associare queste funzioni definite dall'utente è leggermente differente per `pickle` e `cPickle`. La descrizione qui fornita funziona allo stesso modo per entrambe le implementazioni. Gli utenti del modulo `pickle` potrebbero usare anche una sotto classe per ottenere gli stessi risultati, sovrascrivendo i metodi `persistent_id()` e `persistent_load()` nelle classi derivate.

```
import pickle
from cStringIO import StringIO

src = StringIO()
p = pickle.Pickler(src)

def persistent_id(obj):
    if hasattr(obj, 'x'):
        return 'il valore %d' % obj.x
    else:
        return None

p.persistent_id = persistent_id

class Integer:
    def __init__(self, x):
        self.x = x
    def __str__(self):
        return 'Il mio nome e' intero %d' % self.x

i = Integer(7)
print i
p.dump(i)

datastream = src.getvalue()
print repr(datastream)
dst = StringIO(datastream)

up = pickle.Unpickler(dst)

class FancyInteger(Integer):
    def __str__(self):
        return 'Io sono il numero %d' % self.x

def persistent_load(persid):
    if persid.startswith('the value '):
        value = int(persid.split()[2])
        return FancyInteger(value)
    else:
        raise pickle.UnpicklingError, 'ID persistente non valido'

up.persistent_load = persistent_load

j = up.load()
print j
```

Nel modulo `cPickle`, l'attributo `persistent_load` dell'unpickler può anche venire impostato su una lista Python, nel qual caso, quando l'unpickler incontra un id persistente, la sua stringa id viene semplicemente accodata a tale lista. Questa funzionalità esiste per far sì che un flusso di dati serializzati possa venire "sniffato" alla ricerca di riferimenti ad oggetti senza istanziare effettivamente tutti gli oggetti serializzati.[9] L'impostazione di `persistent_load` su una lista viene di solito usata congiuntamente a un metodo `noload()` nell'Unpickler.

3.14.6 Classi derivate da unpickler

In modo predefinito, la deserializzazione importerà qualsiasi classe trovata nei dati serializzati. Potete controllare esattamente ciò che viene deserializzato e ciò che viene invocato, personalizzando un proprio unpickler. Sfortunatamente, questa operazione risulterà diversa in funzione del fatto che si utilizzi il modulo `pickle` oppure

[9] Lascio a voi immaginare Guido e Jim seduti insieme a sniffare flussi serializzati nel loro soggiorno.

`cPickle`.[10] Nel modulo `pickle`, si deve derivare una sotto classe da `Unpickler`, ridefinendo il metodo `load_global()`. Tale metodo dovrebbe leggere due linee dal flusso di dati serializzati, in cui la prima riga sarà il nome del modulo contenente la classe, e la seconda riga sarà il nome della classe dell'istanza. Quindi cerca nella classe, possibilmente importando il modulo a cui appartiene, scovando l'attributo, e poi aggiungendo ciò che trova nello stack dell'unpickler. Successivamente, questa classe verrà assegnata all'attributo `__class__` di una classe vuota, come espediente per creare magicamente un'istanza senza chiamare il metodo `__init__()` proprio della classe. Il vostro lavoro (sta a voi decidere di accettarlo o meno), sarebbe quello di avere un metodo `load_global()` che inserisca nello stack dell'unpickler una versione conosciuta come sicura di qualsiasi classe si giudichi sicura da deserializzare. Spetta a voi scrivere una tale classe. Oppure potreste sollevare un errore nel caso desideraste disabilitare del tutto l'unpickling di istanze. Se questo suona come un escamotage, avete ragione. Per questo scopo fate riferimento al codice sorgente.

Le cose sono un po' più pulite con `cPickle`, ma non di molto. Per controllare ciò che viene deserializzato, potete impostare l'attributo `find_global` ad una funzione oppure a `None`. Se il valore è `None` allora qualsiasi tentativo di deserializzare delle istanze solleverà un'eccezione `UnpicklingError`. Se invece è una funzione, allora essa dovrebbe accettare il nome di un modulo e quello di una classe e restituire il corrispondente oggetto di classe. La funzione è responsabile di ricercare la classe e di eseguire qualsiasi import necessario, e può sollevare un errore per prevenire che istanze della classe vengano deserializzate.

La morale della favola è che si dovrebbe prestare molta attenzione all'origine delle stringhe che la vostra applicazione deserializza.

3.14.7 Esempio

Ecco un semplice esempio di come modificare il comportamento di serializzazione (pickling) per una classe. La classe `TextReader` apre un file di testo e restituisce il numero di riga ed il suo contenuto, ogni volta che viene invocato il suo metodo `readline()`. Se viene serializzata un'istanza di `TextReader`, tutti gli attributi *eccetto* l'oggetto membro di tipo file vengono salvati. Quando l'istanza viene deserializzata, il file viene riaperto e l'operazione di lettura riprende dall'ultima posizione. I metodi `__setstate__()` e `__getstate__()` vengono utilizzati per implementare questo comportamento.

[10] Una parola d'avvertimento: i meccanismi qui descritti usano attributi interni e metodi che saranno soggetti a cambiamenti in future versioni di Python. Abbiamo intenzione di fornire, un giorno, un'interfaccia comune per controllare questi comportamenti, che lavorerà sia con `pickle` che con `cPickle`.

```
class TextReader:
    """Stampa numero e contenuto di ogni riga in un file."""
    def __init__(self, file):
        self.file = file
        self.fh = open(file)
        self.lineno = 0

    def readline(self):
        self.lineno = self.lineno + 1
        line = self.fh.readline()
        if not line:
            return None
        if line.endswith("\n"):
            line = line[:-1]
        return "%d: %s" % (self.lineno, line)

    def __getstate__(self):
        odict = self.__dict__.copy() # copia il dizionario perchè stiamo per modificarlo
        del odict['fh']              # rimuove la voce del filehandle
        return odict

    def __setstate__(self,dict):
        fh = open(dict['file'])      # apre nuovamente il file
        count = dict['lineno']       # legge dal file...
        while count:                 # finché il contatore viene ripristinato
            fh.readline()
            count = count - 1
        self.__dict__.update(dict)   # aggiorna gli attributi
        self.fh = fh                 # salva il file oggetto
```

Un semplice uso potrebbe essere il seguente:

```
>>> import TextReader
>>> obj = TextReader.TextReader("TextReader.py")
>>> obj.readline()
'1: #!/usr/local/bin/python'
>>> # (qui altre invocazioni di obj.readline())
... obj.readline()
'7: class TextReader:'
>>> import pickle
>>> pickle.dump(obj,open('save.p','w'))
```

Se desiderate vedere come `pickle` lavora tramite i processi Python, avviate un'altra sessione di Python, prima di continuare. Quello che segue può avvenire sia nello stesso processo che in uno nuovo.

```
>>> import pickle
>>> reader = pickle.load(open('save.p'))
>>> reader.readline()
'8:     "Stampa e numera righe in un file di testo."'
```

Vedete anche:

Modulo `copy_reg` (sezione 3.16):
: Interfaccia Pickle per la registrazione dei costruttori per i tipi di estensione.

Modulo `shelve` (sezione 3.17):
: Database indicizzato di oggetti; usa `pickle`.

Modulo `copy` (sezione 3.18):

Copia di oggetti superficiale e profonda.

Modulo `marshal` (sezione 3.19):
Serializzazione ad alte prestazione dei tipi built-in.

3.15 `cPickle` — Un modulo `pickle` più veloce

Il modulo `cPickle` supporta la serializzazione e la deserializzazione degli oggetti Python, fornendo un'interfaccia e delle funzionalità molto simili a quelle del modulo `pickle`. Ci sono comunque alcune differenze, le più importanti sono le prestazioni e la capacità di derivarne sotto classi.

Primo, `cPickle` può essere fino a 1000 volte più veloce di `pickle` perché il suo codice è implementato in C. Secondo, nel modulo `cPickle` gli oggetti chiamabili `Pickler()` e `Unpickler()` sono funzioni e non classi. Questo significa che non si possono derivare sotto classi personalizzate per serializzare e deserializzare. Molte applicazioni non hanno bisogno di queste funzionalità e trarranno beneficio unicamente dalle eccezionali performance del modulo `cPickle`.

I flussi di dati della serializzazione prodotti da `pickle` e `cPickle` sono identici ed è possibile quindi usare `pickle` e `cPickle` alternativamente con i dati serializzati esistenti. [11]

Esistono altre piccole differenze, nelle API, tra `cPickle` e `pickle`, comunque per la maggior parte delle applicazioni, essi sono intercambiabili. Maggiori informazioni vengono fornite direttamente dalla documentazione nel modulo `pickle`, che include una lista delle differenze documentate.

3.16 `copy_reg` — Funzioni di supporto al registro di `pickle`

Il modulo `copy_reg` fornisce il supporto per i moduli `pickle` e `cPickle`. Il modulo `copy` probabilmente utilizzerà questo modulo di supporto in futuro. Fornisce informazioni sulla configurazione circa i costruttori di oggetti che non sono classi. Alcuni costruttori possono essere funzioni factory o istanze di classe.

`constructor`(*object*)
Dichiara che *object* sia un costruttore valido. Se *object* non è chiamabile (e quindi non valido come costruttore) solleva un'eccezione `TypeError`.

`pickle`(*type*, *fuction*[, *constructor*])
Dichiara che *function* deve essere usata con la funzione "reduction" per oggetti di tipo *type*; *type* non deve essere un oggetto classe "classic". (Le classi classiche vengono gestite diversamente: vedete la documentazione del modulo `pickle` per i dettagli.) *function* dovrebbe restituire una stringa oppure una tupla contenente due o tre elementi.

Il parametro opzionale *constructor*, se indicato, è un oggetto chiamabile che può venire usato per ricostruire l'oggetto quando chiamato con la tupla di argomenti restituita da *function* durante la serializzazione. Se *object* è una classe, oppure *constructor* è un oggetto non chiamabile, viene sollevata l'eccezione `TypeError`.

Vedete il modulo `pickle` per maggiori dettagli sull'interfaccia prevista di *function* e *constructor*.

3.17 `shelve` — Persistenza degli oggetti Python

"shelf" è un oggetto persistente simile al dizionario. La differenza con i database "dbm" è che i valori (non le chiavi!) contenuti in "shelf" possono essere essenzialmente oggetti arbitrari Python – quasiasi cosa che il modulo `pickle` possa gestire. Questo include la maggior parte delle istanze di classe, tipi di dati ricorsivi ed oggetti contenenti molti condivisi oggetti derivati. Le chiavi sono stringhe ordinarie.

[11] Visto che il formato dei dati serializzati è attualmente un piccolo linguaggio di programmazione stack-oriented, e visto che viene presa qualche libertà nel codificare certi oggetti, è possibile che i due moduli producano due differenti flussi di dati dallo stesso oggetto passato in input. Comunque sia, viene garantito che saranno sempre capaci di leggere l'uno i flussi di dati dell'altro.

open(*filename*[,*flag='c'*[,*protocol=None*[,*writeback=False*[,*binary=None*]]]])
Apre un dizionario persistente. Il nome del file specificato è il nome del file per il database sottostante. Come effetto collaterale, un'estensione può venire aggiunta al nome del file e più di un file può essere creato. In modo predefinito, il file del database sottostante viene aperto in lettura e scrittura. Il parametro facoltativo *flag* ha lo stesso significato del parametro *flag* di `anydbm.open`.

Predefinitamente, serializzatori di versione 0 vengono utilizzati per serializzare i valori. La versione del protocollo del serializzatore può venire specificata con il parametero *protocol*. Modificato nella versione 2.3: Il parametro *protocol* è stato aggiunto. Il parametro *binary* è deprecato, e viene fornito solo per compatibilità all'indietro con le versioni precedenti.

In modo predefinito, le mutazioni di elementi modificabili in dizionari persistenti non vengono riscritte in automatico. Se il parametro facoltativo *writeback* viene impostato a *True* (NdT: *Vero*), tutti gli elementi a cui si è avuto accesso vengono inseriti nella cache in memoria, e riscritti al momento della chiusura; questo può rendere più maneggevole il cambiamento dei valori modificabili nel dizionario persistente, ma se molti elementi vengono richiamati, può consumare una gran quantità di memoria per la cache, e può rendere l'operazione di chiusura molto lenta, finché tutti gli elementi richiamati non vengono riscritti (non esiste un modo per determinare quali elementi sono modificabili, e nemmeno per quelli effettivamente modificati).

Gli oggetti shelve supportano tutti i metodi supportati dai dizionari. Questo semplifica la transizione di dizionari basati sugli script in dizionari che richiedono la memorizzazione persistente.

3.17.1 Restrizioni

- La scelta di quale configurazione di database verrà usata (come `dbm`, `gdbm` o `bsddb`) dipende dal tipo di interfaccia disponibile. Per tale motivo non è sicuro aprire direttamente un database usando `dbm`. Il database è anche (sfortunatamente) soggetto alle limitazioni di `dbm`, se usato — questo significa che (la loro rappresentazione serializzata) gli oggetti memorizzati nel database dovrebbero essere ragionevolmente piccoli, e in rari casi la collisione di chiavi può causare il rifiuto del database a consentire ulteriori aggiornamenti.

- A seconda dell'implementazione, chiudere un dizionario persistente può richiedere o meno lo scaricamento dei cambiamenti sul disco. Il metodo `__del__` della classe `Shelf` chiama il metodo `close`, così il programmatore non ha bisogno di effettuare questa operazione esplicitamente.

- Il modulo `shelve` non supporta accessi *concorrenti* in scrittura/lettura agli oggetti shelved. (Accessi simultanei multipli in lettura sono sicuri). Quando un programma ha uno shelf aperto in scrittura, nessun altro programma dovrebbe tenere lo shelf aperto, in lettura o scrittura. Il meccanismo di locking (NdT: chiusure) dei file nei sistemi UNIX può venire usato per risolvere il problema, ma questa metodologia differisce tra le varie versioni UNIX e richiede le conoscenze sulla sua implementazione nel database.

class Shelf(*dict*[, *protocol=None*[, *writeback=False*[, *binary=None*]]])
Una sotto classe di `UserDict.DictMixin` che memorizza valori serializzati in un oggetto *dict*.

In modo predefinitoe, serializzatori di versione 0 vengono utilizzati per serializzare i valori. La versione di protocollo del serializzatore può venire specificata con il parametro *protocol*. Vedete la documentazione di `pickle` per una discussione dei protocolli di serializzazione utilizzati. Modificato nella versione 2.3: Il parametro *protocol* è stato aggiunto. Il parametro *binary* è deprecato, e viene fornito solo per compatibilità all'indietro con le versioni precedenti.

Se il parametro *writeback* è `True`, l'oggetto manterrà una cache di tutti gli elementi modificati, e li riscriverà in *dict* al momento in cui avverranno operazioni di sync o close. Questo consente operazioni naturali su elementi modificabili, ma può consumare molta più memoria e rendere molto lunghi i tempi di sync e close.

class BsdDbShelf(*dict*[, *protocol=None*[, *writeback=False*[, *binary=None*]]])
Una classe derivata di `Shelf` che mette in rilievo `first`, `next`, `previous`, `last` e `set_location`, disponibili nel modulo `bsddb` ma non negli altri moduli di database. L'oggetto *dict* passato al costruttore deve supportare quei metodi. Questo viene generalemente effettuato chiamando uno dei seguenti `bsddb.hashopen`, `bsddb.btopen` oppure `bsddb.rnopen`. I parametri facoltativi – *protocol*, *writeback* e *binary* hanno lo stesso significato che nella classe `Shelf`.

class DbfilenameShelf(*filename*[, *flag='c'*[, *protocol=None*[, *writeback=False*[, *binary=None*]]]])
Una sotto classe di `Shelf` che accetta un nome di file invece che un oggetto come dict. Il file sottostante verrà aperto usando la funzione `anydbm.open`. In modo predefinito, il file verrà creato e aperto sia in

lettura che in scrittura. Il parametro facoltativo *flag* ha qui lo stesso significato che ha nella funzione `open`. I parametri facoltativi *protocol*, *writeback* e *binary* hanno qui lo stesso significato che hanno nella classe `Shelf`.

3.17.2 Esempio

Per ricapitolare l'interfaccia (`key` è una stringa, `data` è un oggetto arbitrario):

```
import shelve

d = shelve.open(filename) # open -- file che può prendere un suffisso
                          # aggiunto da una libreria di basso-livello

d[key] = data   # immagazzina i dati della chiave (sovrascrive i
                # vecchi dati se sta usando una chiave esistente)
data = d[key]   # recupera una COPIA dei dati dalla chiave
                # (solleva KeyError se la chiave non corrisponde)
del d[key]      # cancella i dati immagazzinati nella chiave
                # (solleva KeyError se la chiave non corrisponde)
flag = d.has_key(key)   # vero se la chiave esiste
list = d.keys() # una lista di tutte le chiavi esistenti (lento!)

# visto che d è stata aperta SENZA writeback=True, si faccia attenzione che:
d['xx'] = range(4)  # questa lavora come sperato, ma ...
d['xx'].append(5)   # *questa no!* -- d['xx'] è ANCORA range(4)!!!
# avendo aperto d senza writeback=True, avete bisogno di scrivere accuratamente:
temp = d['xx']      # estrae la copia
temp.append(5)      # modifica la copia
d['xx'] = temp      # memorizza la copia, per renderla persistente
# oppure, d=shelve.open(nomefile,writeback=True) potrebbe lasciarvi
# scrivere d['xx'].append(5) e farla funzionare come deisderato, MA
# potrebbe anche consumare una maggiore quantità di memoria, e rendere
# l'operazione d.close() più lenta.


d.close()       # chiude d
```

Vedete anche:

Modulo `anydbm` (sezione 7.10):
Interfaccia generica allo stile dei database `dbm`.

Modulo `bsddb` (sezione 7.13):
Interfaccia database BSD `db`.

Modulo `dbhash` (sezione 7.11):
Layer sottile attorno al `bsddb` che fornisce una funzione `open` come i moduli degli altri database.

Modulo `dbm` (sezione 8.6):
Interfaccia a database standard UNIX.

Modulo `dumbdbm` (sezione 7.14):
Implementazione portabile dell'interfaccia `dbm`.

Modulo `gdbm` (sezione 8.7):
Interfaccia database GNU, basata sull'interfaccia `dbm`.

Modulo `pickle` (sezione 3.14):
Serializzazione di oggetti usata da `shelve`.

Modulo `cPickle` (sezione 3.15):
Versione ad alte prestazioni di `pickle`.

3.18 copy — Operazioni di copia superficiale e profonda

Questo modulo fornisce operazioni di copia generiche (superficiale e profonda).

Sommario interfaccia:

```
import copy

x = copy.copy(y)        # crea una copia superficiale di y
x = copy.deepcopy(y)    # crea una copia profonda di y
```

Per errori specifici del modulo, viene sollevata l'eccezione `copy.error`.

La differenza tra copia superficiale e profonda è rilevante solo per oggetti composti (oggetti che contengono altri oggetti, come liste o istanze di classe):

- Una *copia superficiale* costruisce un nuovo oggetto complesso e (nei limite del possibile) inserisce in esso i *riferimenti* agli oggetti trovati nell'originale.
- Una *copia profonda* costruisce un nuovo oggetto complesso e, in maniera ricorsiva, inserisce in esso le *copie* degli oggetti trovati nell'originale.

Con la copia profonda, possono spesso presentarsi due problemi che non si verificano con le operazioni di copia superficiale:

- Oggetti ricorsivi (oggetti complessi che, direttamente o indirettamente, contengono un riferimento a sé stessi) possono causare un ciclo ricorsivo.
- Poichè la copia profonda copia *tutto*, potrebbe copiare troppi oggetti, come ad esempio le strutture dati amministrativi, che potrebbero venire condivise tra le copie.

La funzione `deepcopy()` evita questi problemi attraverso le seguenti modalità:

- tenere un dizionario "memo" degli oggetti già copiati durante il corrente passaggio di copia ; e
- lasciare che le classi definite dall'utente sovrascrivano l'operazione di copia o l'insieme dei componenti copiati.

Questa versione non copia tipi come moduli, classi, funzioni, metodi, tracce dello stack, frame dello stack, file, socket, finestre, array o altri tipi simili.

Le classi possono usare la stessa interfaccia per controllare la copia che viene usata per il controllo di serializzazione. Vedete la descrizione del modulo `pickle` per informazioni su questi metodi. Il modulo `copy` non usa il modulo di registrazione `copy_reg`.

Nel caso di una classe che definisca la propria implementazione di copia, si possono utilizzare i metodi speciali `__copy__()` e `__deepcopy__()`. Il primo viene chiamato per implementare l'operazione di copia superficiale. Il secondo viene chiamato per implementare l'operazione di copia profonda; viene passato un argomento, il dizionario memo. Se l'implementazione di `__deepcopy__()` ha necessità di fare una copia profonda di un componente, dovrebbe chiamare la funzione `deepcopy()` con il componente come primo argomento e il dizionario memo come secondo argomento.

Vedete anche:

Modulo `pickle` (sezione 3.14):
Discussione dei metodi speciali usati per supportare il recupero e ripristino dello stato degli oggetti.

3.19 marshal — Serializzazione di oggetti interna a Python

Questo modulo contiene funzioni che possono leggere e scrivere valori Python in formato binario. Il formato è specifico di Python ma indipendente dall'architettura della macchina (p.es., potreste scrivere un valore Python in un file su di un pc, trasportare il file su una Sun e leggerlo lì sopra). I dettagli sul formato non vengono documentati di proposito; potrebbe cambiare attraverso le varie versioni di Python (anche se avviene raramente).[12]

Questo non è un modulo generale di "persistenza". Per la persistenza generale ed il trasferimento di oggetti Python attraverso chiamate RPC, vedete i moduli `pickle` e `shelve`. Il modulo `marshal` esiste principalmente per leggere e scrivere codice "pseudo compilato" per moduli Python in file '.pyc'. Quindi, i manutentori di Python si riservano il diritto di modificare il formato marshal per renderlo incompatibile con le versioni precedenti, se dovesse presentarsene la necessità. Se si stanno serializzando e deserializzando oggetti Python, è consigliato invece utilizzare il modulo `pickle`.

il modulo `marshal` non viene considerato sicuro contro dati errati o costruiti maliziosamente. Non effettuate mai deserializzazioni di tipo unmarshal di dati ricevuti da una fonte non autenticata o non fidata.

Non tutti i tipi di oggetti Python vengono supportati; in generale, solo gli oggetti il cui valore è indipendente da una particolare invocazione di Python possono venire scritti e letti da questo modulo. I seguenti tipi sono supportati: `None`, interi, interi long, numeri in virgola mobile, stringhe, oggetti Unicode, tuple, liste, dizionari e codice oggetto, dove dovrebbe essere sottinteso che tuple, liste o dizionari, vengono supportati se i valori in essi contenuti sono a loro volta supportati; liste e dizionari ricorsivi non dovrebbero venire scritti (causeranno cicli infiniti).

Avvertimento: Su macchine dove il tipo degli `interi long` in C ha più di 32 bit (tipo i DEC alpha), è possibile creare interi Python ordinari più lunghi di 32 bit. Si può allora verificare che se qualche intero viene serializzato con marshal e reinterpretato su di una macchina dove gli `interi long` in C hanno solo 32 bit, venga resituito un intero long Python. Nonostante la differenza di tipo, il valore numerico è lo stesso. (Questo comportamento è nuovo in Python 2.2. Nelle precedenti versioni, tutti tranne i meno significativi dei 32 bit del valore venivano persi, e veniva stampato un messaggio di avviso).

Esistono funzioni di lettura/scrittura sui file simili alle funzioni che operano sulle stringhe.

Il modulo definisce queste funzioni:

`dump`(*value, file*)
: Scrive il valore nel file aperto. Il valore deve essere un tipo supportato. Il file deve essere un file oggetto di tipo aperto come `sys.stdout`, come valore restituito da `open()` o `posix.popen()`. Deve venire aperto in modo binario (`'wb'` o `'w+b'`).

  Se il valore ha (o contiene un oggetto che ha) un tipo non supportato, viene sollevata un'eccezione `ValueError` — ma valori inutili verranno comunque scritti nel file. L'oggetto non verrà corettamente reinterpretato da `load()`.

`load`(*file*)
: Legge e restituisce un valore dal file aperto. Se non viene letto un valore valido, viene sollevata un'eccezione `EOFError`, `ValueError` o `TypeError`. Il file deve essere un file oggetto di tipo aperto in modo binario (`'rb'` o `'r+b'`).

  Avvertenze: Se un oggetto contenente un tipo non supportato è stato serializzato (da marshal) con `dump()`, `load()`, sostituirà `None` ai tipi che non possono venire deserializzati.

`dumps`(*valore*)
: Restituisce la stringa che verrebbe scritta in un file da `dump(`*valore*`, `*file*`)`. Il valore deve essere un tipo supportato. Solleva un'eccezione `ValueError` se il valore ha (o contiene un oggetto che ha) un tipo non supportato.

`loads`(*string*)
: Converte la stringa in un valore. Se non viene trovato un valore valido, viene sollevata un'eccezione tra `EOFError`, `ValueError` e `TypeError`. I caratteri superflui nella stringa vengono ignorati.

[12] Il nome di questo modulo deriva in parte dalla terminologia usata dai progettisti di Modula-3 (tra gli altri), che usano il termine "marshalling" per la trasmissinione di dati in un formato indipendente. Brevemente, "to marshal" significa convertire dei dati da un formato interno ad uno esterno (in un buffer RPC per esempio) e "unmarshalling" indica il procedimento inverso.

3.20 warnings — Controllo dei messaggi di avvertimento

Nuovo nella versione 2.1.

I messaggi di warning (NdT:avvertimento) vengono emessi nelle situazioni in cui è utile avvertire l'utente del verificarsi di qualche condizione in un programma, una condizione però che (di solito) non è così critica da implicare il sollevamento di un'eccezione e la terminazione del programma stesso. Per esempio, potreste voler emettere un messaggio di avvertimento quando un programma sta utilizzando un modulo obsoleto.

I programmatori Python usano gli avvertimenti chiamando la funzione `warn()` definita in questo modulo. (I programmatori C usano `PyErr_Warn()`; vedete il *Python/C API Reference Manual* per ulteriori dettagli).

I messaggi di avvertimento vengono scritti normalmente su `sys.stderr`, ma la loro disposizione può venire modificata in modo flessibile, dall'ignoramento totale fino al loro inserimento all'interno delle eccezioni. La tipologia degli avvertimenti può variare in base alla loro categoria (vedete più sotto), al testo nel messaggio di avvertimento, e al punto da cui vengono emessi. Le ripetizioni di un particolare messaggio di avvertimento provenienti da uno stesso luogo di origine vengono di solito soppresse.

Esistono due livelli nel controllo degli avvertimenti: primo, ogni volta che viene emesso un avvertimento, viene effettuata una scelta se un messaggio debba venire emesso oppure no; successivamente e in caso di esito positivo, il messaggio viene formattato e stampato usando un qualche strumento impostabile dall'utente.

Il compito di capire se è necessario emettere un messaggio di avvertimento o meno, viene svolto da un filtro, che consiste in una sequenza di regole e azioni a cui corrispondere. Delle regole possono venire aggiunte al filtro attraverso la funzione `filterwarnings()` e reimpostate allo stato predefinito mediante la funzione `resetwarnings()`.

La stampa dei messaggi di avvertimento viene eseguita chiamando la funzione `showwarning()`, che può essere ridefinita; l'implementazione predefinita di questa funzione formatta il messaggio attraverso la chiamata a `formatwarning()`, funzione che può a sua volta essere ridefinita.

3.20.1 Categorie degli avvertimenti

Esistono delle eccezioni built-in che rappresentano categorie di avvertimento. La suddivisione è utile per poter filtrare gruppi di avvertimenti. Vengono definite le seguenti classi di categorie warning:

Classe	Descrizione
`Warning`	Questa è la classe base di tutte le classi delle categorie di avvertimento. È una classe derivata di `Exc`
`UserWarning`	La categoria predefinita per `warn()`.
`DeprecationWarning`	Categoria di base per gli avvertimenti relativi alle caratteristiche deprecate.
`SyntaxWarning`	Categoria base per gli avvertimenti relativi a caratteristiche sintattiche dubbie.
`RuntimeWarning`	Categoria base per gli avvertimenti relativi a caratteristiche di runtime dubbie.
`FutureWarning`	Categoria base per gli avvertimento relativi a costrutti che cambieranno la propria semantica in futu

Sebbene queste siano tecnicamente delle eccezioni built-in, vengono documentate qui visto che concettualmente rientrano nel meccanismo degli avvertimenti.

Il codice utente può definire ulteriori categorie di avvertimento creando delle classi derivate da una delle categorie standard. Una categoria di avvertimento deve sempre ereditare la classe `Warning`.

3.20.2 Filtro degli avvertimenti

Il filtro controlla se gli avvertimenti devono essere ignorati, stampati, o considerati errori (sollevando un'eccezione).

Concettualmente, il filtro mantiene una lista ordinata di specificazioni; ciascun avvertimento viene confrontato con ogni filtro della lista delle specificazioni fino a che non viene trovata una corrispondenza; la corrispondenza ne determina la tipologia. Ogni elemento della lista è una tupla della forma (*action*, *message*, *category*, *module*, *lineno*), dove:

- *action* è una delle seguenti stringhe:

Value	Disposition
`error`	converte gli avvertimenti corrispondenti in eccezioni
`ignore`	non stampa mai gli avvertimenti corrispondenti
`always`	stampa sempre gli avvertimenti corrispondenti
`default`	stampa la prima occorrenza degli avvertimenti corrispondenti per ogni punto da cui l'avvertimento viene emes
`module`	stampa la prima occorrenza dell'avvertimento corrispondente, per ogni modulo da cui l'avvertimento viene em
`once`	stampa solo la prima occorrenza dell'avvertimento corrispondente, indipendentemente dal punto da cui provie

- *message* è una stringa contenente un'espressione regolare a cui il messaggio di avvertimento deve corrispondere (la corrispondenza viene compilata per essere sempre insensibile a maiuscole e minuscole)
- *category* è una classe (una classe derivata di `Warning`) della qaule la categoria di avvertimenti deve essere una sotto classe per poter soddisfare le corrispondenze
- *module* è una stringa contenente un'espressione regolare a cui il nome del modulo deve corrispondere (la corrispondenza viene compilata per essere insensibile a maiuscole o minuscole)
- *lineno* è un intero a cui il numero di linea dove l'avvertimento viene emesso deve corrispondere, oppure deve contenere `0` per corrispondere a tutti i numeri di linea

Poiché la classe `Warning` deriva dalla classe built-in `Exception`, per modificare un avvertimento in un errore sollevate semplicemente la categoria `category(message)`.

Il filtro degli avvertimenti viene inizializzato dall'opzione **-W** passata alla riga di comando dell'interprete Python. L'interprete salva gli argomenti per tutte le opzioni **-W** senza interpretarle in `sys.warnoptions`; il modulo `warnings` li interpreta alla loro prima importazione (le opzioni non valide vengono ignorate, dopo la stampa di un messaggio su `sys.stderr`).

3.20.3 Funzioni disponibili

`warn`(*message*[, *category*[, *stacklevel*]])

Emette un avvertimento, o può ignorarlo, oppure sollevare un'eccezione. L'argomento *category*, se indicato, deve essere una classe di una categoria di avvertimenti (vedete sopra); il suo comportamento predefinito consiste nel sollevare un'eccezione `UserWarning`. Alternativamente, *message* può essere un'istanza di `Warning`, nel cui caso *category* verrà ignorata, e verrà usato `message.__class__`. In questo caso il messaggio di testo sarà `str(message)`. Questa funzione solleva un'eccezione se il particolare avvertimento emesso viene modificato in un errore dal filtro degli avvertimenti visto sopra. L'argomento *stacklevel* può venire usato per interfacciare funzioni scritte in Python come la seguente:

```
def deprecation(message):
    warnings.warn(message, DeprecationWarning, stacklevel=2)
```

Questo fa sì che l'avvertimento si riferisca al chiamante di `deprecation()`, piuttosto che all'originale `deprecation()` stesso (poiché il successivo potrebbe fallire lo scopo del messaggio di avvertimento).

`warn_explicit`(*messagge, category, filename, lineno*[, *module*[, *registry*]])

Questa è un'interfaccia di basso livello per la funzionalità di `warn()`, a cui vengono passati esplicitamente il messaggio, la categoria, il filename, il numero di riga ed opzionalmente il nome del modulo e il registry (che dovrebbe essere un dizionario `__warningregistry__` del modulo). Il nome predefinito del modulo corrisponde a quello del filename, a cui viene rimossa l'estensione `.py`; se non viene passato nessun registry, l'avvertimento non viene mai soppresso. *message* deve essere una stringa e *category* una classe derivata di `Warning`, oppure *message* può essere un'istanza `Warning`, nel qual caso *category* verrà ignorata.

`showwarning`(*message, category, filename, lineno*[, *file*])

Scrive un avvertimento in un file. L'implementazione predefinita chiama `formatwarning(`*message*`,` *category*`,` *filename*`,` *lineno*`)` e scrive la stringa risultante in *file*, il cui predefinito è `sys.stderr`.

Potete rimpiazzare questa funzione con una implementazione alternativa assegnandola a `warnings.showwarning`.

`formatwarning`(*message, category, filename, lineno*)
Compone un avvertimento nel modo standard. Restituisce una stringa che può contenere al suo interno i caratteri di fine riga, e terminare con un carattere di fine riga.

`filterwarnings`(*action*[, *message*[, *category*[, *module*[, *lineno*[, *append*]]]]])
Inserisce un elemento nella lista dei filtri degli avvertimenti. L'elemento viene inserito all'inizio della lista, in modo predefinito; se *append* è vero, viene inserito alla fine. Questa funzione verifica i tipi degli argomenti, compila il messaggio e le espressioni regolari del modulo, e li inserisce in una tupla all'inizio del filtro degli avvertimenti. Elementi inseriti successivamente sovrascrivono gli elementi inseriti per primi, se entrambi corrispondono ad un particolare avvertimento. Gli argomenti omessi vengono impostati ad un valore predefinito che soddisfi ogni corrispondenza.

`resetwarnings`()
Azzerano il filtro degli avvertimenti. Questo annulla gli effetti di tutte le precedenti chiamate a `filterwarnings()`, includendo anche quelli delle opzioni sulla riga di comando indicati con **-W**.

3.21 `imp` — Accesso alle caratteristiche interne dell'istruzione `import`

Questo modulo fornisce un'interfaccia ai meccanismi usati per implementare l'istruzione `import`. Definisce le seguenti costanti e funzioni:

`get_magic`()
Restituisce il valore della magic string usata per riconoscere i file con codice bytecode (file ('.pyc'). (Questo valore può differire per ogni versione di Python.)

`get_suffixes`()
Restituisce una lista di triple, ciascuna delle quali descrive un particolare tipo di modulo. Ogni tripla ha la struttura (*suffix*, *mode*, *type*), dove *suffix* è una stringa da aggiungere al nome del modulo per creare il nome del file da cercare, *mode* è la stringa della modalità da passare alla funzione built-in `open()` per aprire file (questa può essere `'r'` per file di testo o `'rb'` per file binari), e *type* è il tipo di file, che può assumere uno dei valori `PY_SOURCE`, `PY_COMPILED`, o `C_EXTENSION`, descritti più avanti.

`find_module`(*name*[, *path*])
Cerca di trovare il modulo *name* nel percorso di ricerca *path*. Se *path* è una lista di nomi di directory, all'interno di ognuna di esse viene eseguita la ricerca in base alle estensioni restituite dalla funzione `get_suffixes()`. Nomi non validi vengono tacitamente ignorati (ma tutti gli elementi della lista devono essere stringhe). Se *path* viene omesso o è uguale a `None`, la ricerca avviene nei percorsi contenuti in `sys.path`, ma dopo aver cercato in alcuni luoghi speciali: innanzitutto prova a cercare un modulo built-in con il nome dato (`C_BUILTIN`), poi tra i moduli frozen (`PY_FROZEN`), e così via in altri luoghi dipendenti dal sistema (su Mac, si cerca tra le risorse (`PY_RESOURCE`); su Windows, nel registro di configurazione, che potrebbe puntare ad uno specifico file).

Se la ricerca ha successo, il valore restituito è una tripla (*file*, *pathname*, *description*) dove *file* è un oggetto file aperto posto all'inizio della tripla, *pathname* è il percorso del file trovato e *description* è una tripla identica a quella restituita dalla funzione `get_suffixes()`, che descrive il tipo di modulo trovato. Se il modulo non è in un file, il *file* restituito è `None`, *filename* è una stringa vuota, e *description* contiene stringhe vuote per i suffissi e modi del modulo; il tipo di modulo viene indicato tra parentesi, come sopra. Se la ricerca non ha successo, viene sollevata l'eccezione `ImportError`. ALtre eccezioni indicano problemi relativi agli argomenti o all'ambiente.

Questa funzione non gestisce i nomi di modulo gerarchici (nomi contenenti punti). Per poter trovare *P.M*, cioè il modulo derivato *M* del package *P*, usate `find_module()` e `load_module()` per trovare e caricare *P*, e quindi `find_module()` con l'argomento *path* impostato a *P*`.__path__`. Nel caso in cui *P* stesso abbia un nome con caratteri punto, applicate il procedimento ricorsivamente.

`load_module`(*name, file, filename, description*)
Carica un modulo precedentemente trovato da `find_module()` (o da qualsiasi altra ricerca che restituisca

un risultato compatibile). Questa funzione fa più che importare il modulo: se il modulo era già stato importato, è equivalente ad un `reload()`! L'argomento *name* indica il nome completo del modulo (compreso il nome del package, se si tratta di un suo modulo derivato). L'argomento *file* è un file aperto e *filename* è il corrispondente nome del file; questi possono essere `None` e `''`, rispettivamente, quando il modulo non viene caricato da un file. L'argomento *description* è una tupla, identica a quella restituita da `get_suffixes()`, che descrive il tipo di modulo da caricare.

Se il caricamento avviene con successo, il valore restituito è l'oggetto modulo; altrimenti, viene sollevata un'eccezione (di solito `ImportError`).

Importante: il chiamante è anche responsabile di chiudere l'argomento *file*, se questo non era `None`, anche quando viene sollevata un'eccezione. Questo scopo viene raggiunto in modo ottimale usando l'istruzione `try` ... `finally`.

`new_module`(*name*)
Restituisce un nuovo oggetto modulo vuoto chiamato *name*. Questo oggetto *non* viene inserito in `sys.modules`.

`lock_held`()
Restituisce `True` se il lock al modulo viene correntemente mantenuto, altrimenti `False`. Su piattaforme che non supportano thread, restituisce sempre `False`.

Su piattaforme che supportano i thread, un thread che esegue un import mantiene un lock interno finchè l'import non viene completato. Questo lock blocca altri thread che pure tentino di eseguire un import fino a quando il primo import non è terminato, prevenendo così la possibilità che gli altri thread vedano oggetti modulo ancora incompleti prima che l'import sia stato completato (anche altri eventuali import presenti vengono bloccati).

`acquire_lock`()
Acquisisce il lock dell'import dell'interprete nel thread corrente. Questo lock dovrebbe venire usato come estensioni al meccanismo di import per garantire che i thread vengano gestiti in maniera sicura quando si importano dei moduli. Su piattaforme che non supportano i thread, questa funzione non fa niente. Nuovo nella versione 2.3.

`release_lock`()
Rilascia il lock dell'interprete per l'import. Su piattaforme che non supportano i thread questa funzione non fa niente. Nuovo nella versione 2.3.

Le seguenti costanti a valori interi, definite in questo modulo, vengono usate per indicare il risultato della ricerca di `find_module()`.

`PY_SOURCE`
Il modulo trovato è un file sorgente.

`PY_COMPILED`
Il modulo trovato è un file compilato in codice oggetto.

`C_EXTENSION`
Il modulo trovato è una libreria condivisa caricabile dinamicamente.

`PY_RESOURCE`
Il modulo trovato è una risorsa Macintosh. Questo valore può essere restituito solo su un Macintosh.

`PKG_DIRECTORY`
Il modulo trovato è una directory package.

`C_BUILTIN`
Il modulo trovato è un modulo built-in.

`PY_FROZEN`
Il modulo trovato è un modulo frozen (vedete `init_frozen()`).

Le seguenti costanti e funzioni sono obsolete; le loro funzionalità vengono rese disponibili tramite `find_module()` o `load_module()`. Vengono mantenute per compatibilità all'indietro.

`SEARCH_ERROR`
Non usata.

init_builtin(*name*)
Inizializza il modulo built-in chiamato *name* e restituisce il suo modulo oggetto. Se il modulo è già stato inizializzato, verrà inizializzato *nuovamente*. Alcuni moduli non possono venire inizializzati due volte — tentando di inizializzarli nuovamente, solleveranno un'eccezione `ImportError`. Se non ci sono moduli built-in chiamati *name*, viene restituito `None`.

init_frozen(*name*)
Inizializza il modulo frozen chiamato *name* e restituisce il suo modulo oggetto. Se il modulo è già stato inizializzato, verrà inizializzato *nuovamente*. Se non esistono moduli frozen chiamati *name*, viene restituito `None`. (I moduli frozen sono moduli scritti in Python il cui oggetto compilato in bytecode viene incorporato in un interprete Python personalizzato, attraverso l'utility Python **freeze**. Vedete, per adesso, 'Tools/freeze/'.)

is_builtin(*name*)
Restituisce `1` se esiste un modulo built-in chiamato *name* che può essere inizializzato nuovamente. Restituisce `-1` se esiste un modulo built-in chiamato *name* che non può essere inizializzato nuovamente (vedete `init_builtin()`). Restituisce `0` se non esistono moduli built-in chiamati *name*.

is_frozen(*name*)
Restituisce `True` se esiste un modulo frozen (vedete `init_frozen()`) chiamato *name*, o `False` se non esiste il modulo.

load_compiled(*name, pathname, file*)
Carica ed inizializza un modulo implementato come codice compilato in bytecode e restituisce il suo modulo oggetto. Se il modulo è già stato inizializzato, verrà inizializzato *nuovamente*. L'argomento *name* viene usato per creare o accedere al modulo oggetto. L'argomento *pathname* punta al file del bytecode compilato. L'argomento *file* è il file del codice bytecode compilato, aperto in lettura in modo binario, all'inizio. Deve essere un file oggetto reale, non una classe definita dall'utente che emula un file.

load_dynamic(*name, pathname*[, *file*])
Carica e inizializza un modulo implementato come libreria condivisa dinamicamente caricabile e restituisce il suo modulo oggetto. Se il modulo è già inizializzato, verrà inizializzato *nuovamente*. Ad alcuni moduli non piace la reinizializzazione e potrebbero sollevare un'eccezione. L'argomento *pathname* deve puntare alla libreria condivisa. L'argomento *name* viene usato per costruire il nome della funzione di inizializzazione: viene chiamata una funzione C esterna chiamata '`init`*name*`()`' nella libreria condivisa. L'argomento facoltativo *file* viene ignorato. (Nota: l'uso delle librerie dinamiche è altamente legato al sistema in uso, e non tutti i sistemi lo supportano).

load_source(*name, pathname, file*)
Carica e inizializza un modulo implementato come file di codice sorgente Python e restituisce il suo modulo oggetto. Se il modulo è già inizializzato, verrà inizializzato *nuovamente*. L'argomento *name* viene usato per creare un modulo oggetto, o accedervi. L'argomento *pathname* punta al file sorgente. L'argomento *file* è il file sorgente, aperto in lettura come testo, dall'inizio. Deve essere un file oggetto reale, non una classe definita dall'utente che emula un file. Notate che se un file compilato in bytecode risulta propriamente corrispondente (con il suffisso '.pyc' o '.pyo') esiste, verrà usato questo piuttosto che far analizzare al parser il file sorgente indicato.

3.21.1 Esempi

Le seguenti funzioni emulano quello che era la definizione standard di importazione fino a Python 1.4 (senza nomi di moduli gerarchici). (Questa *implementazione* non potrà lavorare in quella versione, in quanto `find_module()` è stato esteso e `load_module()` è stato aggiunto nella 1.4.)

```
import imp
import sys

def __import__(name, globals=None, locals=None, fromlist=None):
    # Percorso veloce: controlla se il modulo è già stato importato.
    try:
        return sys.modules[name]
    except KeyError:
        pass

    # se una delle seguenti chiamate solleva un'eccezione,
    # c'è un problema che non possiamo gestire -- lo gestirà il chiamante.

    fp, pathname, description = imp.find_module(name)

    try:
        return imp.load_module(name, fp, pathname, description)
    finally:
        #  Siccome possiamo uscire tramite
        #+ un'eccezione, chiudete fp esplicitamente.
        if fp:
            fp.close()
```

Un esempio più completo che implementa gerarchicamente i nomi del modulo e include una funzione `reload()` può venire trovato nel modulo `knee`. Il modulo `knee` può venire trovato in 'Demo/imputil/' nei sorgenti della distribuzione Python.

3.22 `pkgutil` — Utility per le estensioni dei package

Nuovo nella versione 2.3.

Questo modulo fornisce una singola funzione:

`extend_path`(*path, name*)

Estende il path (NdT: percorso) di ricerca per i moduli che contengono un package. Il corretto utilizzo consiste nel posizionare il seguente codice in un file '__init__.py' del package:

```
from pkgutil import extend_path
__path__ = extend_path(__path__, __name__)
```

Questo aggiungerà al `__path__` del package tutte le sotto directory delle directory su `sys.path`, nominato dopo il package. Questo è molto utile se si vogliono distribuire differenti parti di un singolo package logico come directory multiple.

Inoltre cerca nelle righe iniziali dei file '*.pkg' le corrispondenze di `*` al *name* dell'argomento. Questa funzionalità è simile a quella relativa ai file '*.pth' (vedete il modulo `site` per ulteriori informazioni), tranne per il fatto che non vengono assegnati caratteri speciali alle righe che iniziano con `import`. Un file '*.pkg' viene assegnato come valore di riferimento: ad eccezione dei duplicati che vegono individuati, tutti i valori trovati in un file '*.pkg' vengono aggiunti al path, indipendentemente dal fatto che esistano nel filsystem. (Questa è una caratteristica.)

Se il percorso di input non è una lista (come nel caso dei package frozen) viene restituito invariato. Il percorso di input non viene modificato; viene restituita una copia estesa. Gli argomenti vengono soltanto aggiunti nella coda della copia.

Viene assunto che `sys.path` sia una sequenza. Gli elementi di `sys.path` che non sono stringhe (Unicode o 8-bit) riferite a directory esistenti, vengono ignorati. Gli elementi Unicode all'interno di `sys.path` che causano errori quando utilizzati come nomi dei file, potrebbero causare il sollevamento di un'eccezione da parte della funzione (in linea con il comportamento di `os.path.isdir()`).

3.23 code — Classi di base dell'interprete

Il modulo code fornisce degli strumenti per l'implementazione dei cicli leggi-valuta-stampa in Python. Vengono incluse due classi e funzioni di utilità, che possono venire utilizzate per costruire applicazioni che forniscano un prompt interattivo dell'interprete.

class **InteractiveInterpreter**([*locals*])
Questa classe si occupa della fase di analisi e di interpretazione (lo spazio dei nomi dell'utente); non si occupa invece della bufferizzazione dell'input o della visualizzazione del prompt, oppure di nominare il file di input (il nome del file viene passato sempre in modo esplicito). L'argomento facoltativo *locals* specifica il dizionario nel quale verrà eseguito il codice; viene impostato in modo predefinito su di un dizionario appena creato, con la chiave '__name__' impostata su '__console__' e la chiave '__doc__' impostata su None.

class **InteractiveConsole**([*locals*[, *filename*]])
Emula il comportamento dell'interprete interattivo di Python. Questa classe si fonda su InteractiveInterpreter ed aggiunge il prompt utilizzando le familiari sys.ps1, sys.ps2 e l'input buffering.

interact([*banner*[, *readfunc*[, *local*]]])
Una funzione di utilità per eseguire un ciclo leggi-valuta-stampa. Crea una nuova istanza di InteractiveConsole ed imposta *readfunc* per venire usata come metodo raw_input(), se fornita. Se *local* viene fornita, viene passata al costruttore InteractiveConsole per poterla utilizzare come spazio dei nomi predefinito per l'interprete del ciclo. Il metodo interact() dell'istanza viene quindi eseguito con *banner* passato come il messaggio da visualizzare, se fornito. L' oggetto console viene scartato dopo l'utilizzo.

compile_command(*source*[, *filename*[, *symbol*]])
Questa funzione è utile per i programmi che vogliono emulare il ciclo principale dell'interprete Python (conosciuto come ciclo leggi-valuta-stampa). Il trucco consiste nel determinare quando l'utente ha digitato un comando incompleto che può essere completato digitando del testo ulteriore (in contrapposizione ad un comando completo o ad un errore di sintassi). Questa funzione *quasi sempre* intraprende la stessa decisione di quella del ciclo principale dell'interprete.

source rappresenta la stringa sorgente; *filename* rappresenta il nome del file facoltativo dal quale viene letto il sorgente, preimpostato a '<input>'; e *symbol* rappresenta il simbolo grammaticale di partenza, facoltativo, che dovrebbe essere 'single' (predefinito) oppure 'eval'.

Restituisce un codice oggetto (lo stesso di compile(*source*, *filename*, *symbol*)) se il comando è completo e valido; restituisce None se il comando è incompleto; solleva l'eccezione SyntaxError se il comando è completo e contiene un errore di sintassi, solleva l'eccezione OverflowError oppure ValueError se il comando contiene un carattere non valido.

3.23.1 Oggetti dell'interprete interattivo

runsource(*source*[, *filename*[, *symbol*]])
Compila ed esegue alcuni sorgenti nell'interprete. Gli argomenti sono gli stessi di quelli di compile_command(); il valore predefinito per *filename* è '<input>' e per *symbol* è 'single'. Può verificarsi uno dei seguenti eventi:

- L'input non è corretto; compile_command() solleva un'eccezione (SyntaxError oppure OverflowError). Verrà stampata una traceback della sintassi, invocando il metodo showsyntaxerror(). runsource() restituisce False.
- L'input è incompleto, e vengono richieste ulteriori immissioni; compile_command() restituisce None. runsource() restituisce True.
- L'input è completo; compile_command() restituisce un codice oggetto. Il codice viene eseguito chiamando runcode() (anche questo gestisce eccezioni di run-time, tranne che per SystemExit). runsource() restituisce False.

Il valore restituito può venire utilizzato per decidere se utilizzare `sys.ps1` oppure `sys.ps2` per il prompt della riga successiva.

`runcode`(*code*)
Esegue un codice oggetto. Quando si verifica un'eccezione, viene invocato `showtraceback()` per visualizzare una traceback. Tutte le eccezioni vengono catturate tranne `SystemExit`, alla quale è permesso di propagarsi.

Un'avvertenza a riguardo di `KeyboardInterrupt`: questa eccezione può verificarsi ovunque in questo codice e potrebbe non essere sempre catturata. Il chiamante dovrebbe essere preparato a comportarsi di conseguenza.

`showsyntaxerror`([*filename*])
Visualizza l'errore di sintassi appena verificatosi. Esso non visualizza una traccia dello stack perchè non ne esiste una specifica per gli errori di sintassi. Se *filename* viene fornito, questo viene immesso nell'eccezione al posto del nome del file predefinito, fornito dal parser di Python, in quanto utilizza sempre `'<string>'` quando legge da una stringa. L'output viene scritto dal metodo `write()`.

`showtraceback`()
Visualizza l'eccezione che si è verificata. Togliamo il primo elemento dello stack della traceback, in quanto contenuto all'interno dell'implementazione dell'oggetto interprete. L'output viene scritto dal metodo `write()`.

`write`(*data*)
Scrive una stringa sul flusso dei dati di standard error (`sys.stderr`). Le classi derivate dovrebbero ridefinire il metodo, affinché possano fornire l'output appropriato da utilizzare a seconda delle necessità.

3.23.2 Oggetti della console interattiva

La classe `InteractiveConsole` è una classe derivata di `InteractiveInterpreter`, e quindi offre tutti i metodi degli oggetti interprete, così come le seguenti caratteristiche aggiuntive.

`interact`([*banner*])
Emula la console interattiva di Python. L'argomento facoltativo *banner* specifica il messaggio da stampare prima della interazione iniziale; in modo predefinito, viene stampato un messaggio simile a quello stampato dall'interprete standard di Python, seguito dal nome della classe dell'oggetto console posto fra parentesi (per non confondersi con l'interprete reale – in quanto molto simile!).

`push`(*line*)
Inoltra una riga del testo sorgente verso l'interprete. La riga non dovrebbe avere un carattere finale di fine riga. Può invece avere caratteri di fine riga al suo interno. La riga viene aggiunta ad un buffer, ed il metodo `runsource()` dell'interprete viene chiamato con il contenuto del buffer come sorgente. Se questo indica che il comando è stato eseguito o non è valido, il buffer viene resettato; altrimenti il comando risulta incompleto e il buffer viene lasciato nello stato in cui si trovava prima che la riga venisse aggiunta. Il valore di ritorno è `True` se viene richiesto ulteriore input, `False` se la linea è stata trattata in qualche modo (è la stessa cosa che si verifica con `runsource()`).

`resetbuffer`()
Rimuove dal buffer di input ogni testo sorgente non elaborato.

`raw_input`([*prompt*])
Scrive un prompt e legge una riga. La riga restituita non include il carattere di fine riga. Quando l'utente inserisce la sequenza EOF, viene sollevata l'eccezione `EOFError`. L'implementazione di base usa la funzione built-in `raw_input()`; una classe derivata la può rimpiazzare con una differente implementazione.

3.24 codeop — Compilare codice Python

Il modulo `codeop` fornisce le utility attraverso le quali il ciclo Python leggi-valuta-stampa può venire emulato, così come viene fatto nel modulo `code`. Di conseguenza, probabilmente non vorrete utilizzare il modulo direttamente; se vorrete infatti includere un ciclo simile nel vostro programma, probabilmente vorrete usare il modulo `code`.

Due parti suddividono questo lavoro:

1. Essere capace di predire se una riga di input completa un costrutto Python: in breve, stabilire se stampare successivamente '`>>> `' o '`... `'.
2. Ricordare quali costrutti successivi sono stati inseriti dall'utente, in modo tale che il susseguente input possa venire compilato effettivamente con essi.

Il modulo `codeop` fornisce un metodo per per compiere ciascuna di queste azioni, ed un metodo per compierle entrambe, insieme.

Nel primo caso:

`compile_command`(*source*[, *filename*[, *symbol*]])

Cerca di compilare *source*, che dovrebbe essere una stringa di codice Python, e restitusce un codice oggetto se *source* è codice Python valido. In questo caso, l'attributo filename del codice oggetto sarà *filename*, che in modo predefinito è `'<input>'`. Restituisce `None` se *source non* è codice Python valido, ma è un prefisso di codice Python valido.

Se si verifica un problema con *source*, verrà sollevata un'eccezione. Viene sollevata l'eccezione `SyntaxError` se si presenta della sintassi Python non valida, e `OverflowError` oppure `ValueError` se è presente una costante non valida.

L'argomento *symbol* determina quando *source* viene compilata come una dichiarazione (il valore predefinito è `'single'`) o come espressione (`'eval'`). Qualsiasi altro valore provocherà il sollevamento dell'eccezione `ValueError`.

Avvertenza: È possibile (ma non piacevole) che il parser blocchi l'analisi con un risultato di successo prima che sia stata raggiunta la fine del sorgente; in questo caso, i simboli finali possono venire ignorati piuttosto che causare errori. Per esempio, un backslash seguito da due caratteri di fine riga può essere seguito da garbage arbitraria. Questo comportamento verrà corretto quando le API per il parser verranno migliorate.

class `Compile`()

Le istanze di questa classe hanno a disposizione metodi `__call__()` identici negli effetti alla funzione built-in `compile()`, ma con la differenza che se l'istanza compila una porzione di programma contenente una dichiarazione `__future__`, l'istanza stessa 'ricorda' e compila tutte le successive porzioni del programma con la dichiarazione in vigore.

class `CommandCompiler`()

Le istanze di questa classe hanno a disposizione metodi `__call__()` identici negli effetti alla funzione `compile_command()`, ma con la differenza che se l'istanza compila una porzione di programma contenente una dichiarazione `__future__`, l'istanza stessa 'ricorda' e compila tutte le successive porzioni del programma con la dichiarazione in vigore.

Una nota sulla compatibilità tra le versioni: `Compile` e `CommandCompiler` sono nuove in Python 2.2. Se volete abilitare la caratteristica future-tracking del 2.2 ma mantenere contemporaneamente la compatibilità con la 2.1 e le versioni precedenti, potete scrivere sia:

```
try:
    from codeop import CommandCompiler
    compile_command = CommandCompiler()
    del CommandCompiler
except ImportError:
    from codeop import compile_command
```

che è un cambiamento di impatto ridotto ma introduce possibili stati non voluti nel vostro programma, oppure:

```
try:
    from codeop import CommandCompiler
except ImportError:
    def CommandCompiler():
        from codeop import compile_command
        return compile_command
```

e quindi chiamare `CommandCompiler` ogni volta che avrete la necessità di un nuovo oggetto compilato.

3.25 `pprint` — Modulo per la stampa dei dati in forma elegante

Il modulo `pprint` fornisce la possibilità di stampare "elegantemente" strutture arbitrarie di dati Python in una forma che può venire usata come input per l'interprete. Se le strutture formattate contengono oggetti che non sono dei tipi Python fondamentali, la rappresentazione potrebbe non essere utilizzabile. Questo potrebbe verificarsi nel caso in cui vengano inclusi oggetti come file, socket, classi o istanze, cosi come molti altri oggetti built-in non rappresentabili come costanti Python.

La rappresentazione formattata se può tiene gli oggetti su una singola riga, altrimenti se questa non rientra nella larghezza consentita, la spezza in righe multiple. Costruite esplicitamente oggetti `PrettyPrinter` se avete bisogno di aggiustare il limite della larghezza.

Il modulo `pprint` definisce una classe:

class `PrettyPrinter`(...)

Costruisce un'istanza `PrettyPrinter`. Questo costruttore accetta diversi parametri chiave. Può definirsi un flusso di output utilizzando la parola chiave *stream*; il solo metodo usato sull'oggetto stream è il protocollo dei file, il metodo `write()`. Se non specificato, `PrettyPrinter` utilizza `sys.stdout`. Per controllare la rappresentazione formattata, possono venire usati tre parametri aggiuntivi. Le parole chiave sono *indent*, *depth* e *width*. L'incremento di indentazione che viene aggiunto per ogni livello ricorsivo viene specificato da *indent*; il suo valore predefinito è uno. Altri valori possono rendere l'aspetto dell'output un po' irregolare, ma ciò può rendere più facilmente riconoscibili gli annidamenti. Il numero dei livelli che possono venire stampati è controllato da *depth*; se la struttura di dati da stampare è troppo profonda, il successivo livello contenuto viene sostituito da '`...`'. Non esiste un limite predefinito sulla profondità degli oggetti che devono essere formattati. La larghezza desiderata dell'output viene definita usando il parametro *width*; il suo valore predefinito è otto caratteri. Se non si riesce a formattare una struttura nell'intervallo di lunghezza voluto, si dovrà fare uno sforzo maggiore.

```
>>> import pprint, sys
>>> stuff = sys.path[:]
>>> stuff.insert(0, stuff[:])
>>> pp = pprint.PrettyPrinter(indent=4)
>>> pp.pprint(stuff)
[   [   '',
        '/usr/local/lib/python1.5',
        '/usr/local/lib/python1.5/test',
        '/usr/local/lib/python1.5/sunos5',
        '/usr/local/lib/python1.5/sharedmodules',
        '/usr/local/lib/python1.5/tkinter'],
    '',
    '/usr/local/lib/python1.5',
    '/usr/local/lib/python1.5/test',
    '/usr/local/lib/python1.5/sunos5',
    '/usr/local/lib/python1.5/sharedmodules',
    '/usr/local/lib/python1.5/tkinter']
>>>
>>> import parser
>>> tup = parser.ast2tuple(
...     parser.suite(open('pprint.py').read()))[1][1][1]
>>> pp = pprint.PrettyPrinter(depth=6)
>>> pp.pprint(tup)
(266, (267, (307, (287, (288, (...))))))
```

La classe `PrettyPrinter` supporta diverse funzioni derivate:

`pformat`(*object*[, *indent*[, *width*[, *depth*]]])
Restituisce la rappresentazione formattata di *object* come una stringa. *indent*, *width* e *depth* verranno passati al costruttore `PrettyPrinter` come parametri di formattazione. Modificato nella versione 2.4: Vengono aggiunti i parametri *indent*, *width* e *depth*.

`pprint`(*object*[, *stream*[, *indent*[, *width*[, *depth*]]]])
Stampa la rappresentazione formattata di *object* su *stream*, seguito da un carattere di fine riga. Se *stream* viene omesso, verrà usato `sys.stdout`. Questo può venire usato nell'interprete interattivo al posto di un'istruzione `print` per esaminare i valori. *indent*, *width* e *depth* verranno passati come parametri di formattazione, al costruttore `PrettyPrinter`.

```
>>> stuff = sys.path[:]
>>> stuff.insert(0, stuff)
>>> pprint.pprint(stuff)
[<Recursion on list with id=869440>,
 '',
 '/usr/local/lib/python1.5',
 '/usr/local/lib/python1.5/test',
 '/usr/local/lib/python1.5/sunos5',
 '/usr/local/lib/python1.5/sharedmodules',
 '/usr/local/lib/python1.5/tkinter']
```

Modificato nella versione 2.4: Vengono aggiunti i parametri *indent*, *width* e *depth*.

`isreadable`(*object*)
Determina se la rappresentazione formattata di *object* è "readable", (NdT: "leggibile") oppure può venire utilizzato per ricostruire il valore, usando `eval()`. Questa restituisce sempre falso per gli oggetti ricorsivi.

```
>>> pprint.isreadable(stuff)
False
```

`isrecursive`(*object*)
Determina se *object* richiede una rappresentazione ricorsiva.

Viene inoltre definita un'ulteriore funzione di supporto:

saferepr(*object*)
Restituisce una rappresentazione in forma di stringa di *object*, protetta da strutture dati ricorsive. Se la rappresentazione di *object* presenta un inserimento ricorsivo, allora il riferimento ricorsivo verrà rappresentato come '`<Recursion on` *typename* `with id=`*number*`>`'. La rappresentazione non viene altrimenti formattata.

```
>>> pprint.saferepr(stuff)
"[<Recursion on list with id=682968>, '', '/usr/local/lib/python1.5', '/usr/loca
l/lib/python1.5/test', '/usr/local/lib/python1.5/sunos5', '/usr/local/lib/python
1.5/sharedmodules', '/usr/local/lib/python1.5/tkinter']"
```

3.25.1 Oggetti PrettyPrinter

Le istanze di `PrettyPrinter` hanno i seguenti metodi:

pformat(*object*)
Restituisce la rappresentazione formattata di *object*. Questo inserisce nell'Account le opzioni passate al costruttore `PrettyPrinter`.

pprint(*object*)
Stampa la rappresentazione formattata di *object* sul flusso configurato, seguito da un carattere di fine riga.

I seguenti metodi forniscono le implementazioni per le funzioni corrispondenti degli stessi nomi. Utilizzare questi metodi su un'istanza è evidentemente più efficace, poichè non si ha necessità di creare nuovi oggetti `PrettyPrinter`.

isreadable(*object*)
Determina se la rappresentazione formattata dell'oggetto è "readable", oppure può venire utilizzato per ricostruirne il valore, utilizzando `eval()`. Osservate come questa restituisce sempre falso per gli oggetti ricorsivi. Se viene impostato il parametro *depth* di `PrettyPrinter`, e l'oggetto è più profondo di quanto consentito, restituisce falso.

isrecursive(*object*)
Determina se l'oggetto richiede una rappresentazione ricorsiva.

Questo metodo viene fornito come estensione per consentire alle classi derivate di modificare il modo in cui gli oggetti vengono convertiti in stringhe. L'implementazione predefinita utilizza le caratteristiche proprie della implementazione di `saferepr()`.

format(*object, context, maxlevels, level*)
Restituisce tre valori: la versione formattata di *object* come stringa, un'opzione indicante se il risultato è leggibile, e un'opzione indicante se è stata rilevata una ricorsione. Il primo argomento è l'oggetto che deve essere rappresentato. Il secondo è un dizionario contenente l'`id()` degli oggetti che sono parte del contesto della corrente rappresentazione (contenitori di *object* diretti e indiretti che influenzano la presentazione), così come le chiavi; se risulta necessario dover rappresentare un oggetto a sua volta già rappresentato in *context*, il terzo valore restituito dovrebbe essere vero. Chiamate ricorsive al metodo `format()` dovrebbero aggiungere ulteriori voci per i contenitori, a questa directory. Il terzo argomento, *maxlevels*, stabilisce il limite richiesto per la ricorsione; questo valore sarà `0` se non viene richiesto alcun limite. Questo argomento dovrebbe essere passato alle chiamate ricorsive senza aver subito modifiche. Il quarto argomento, *level*, stabilisce il livello corrente; per le chiamate ricorsive dovrebbe venire passato un valore inferiore di quello della chiamata corrente. Nuovo nella versione 2.3.

3.26 `repr` — Implementazione alternativa di `repr()`

Il modulo `repr` fornisce un mezzo per produrre rappresentazioni di oggetti con limiti imposti sulla dimensione delle stringhe risultanti. Viene utilizzato dal debugger di Python e può venire utilizzato altrettanto bene anche in altri contesti.

Questo modulo fornisce una classe, un'istanza ed una funzione:

class Repr()
Una classe che fornisce servizi di formattazione utili nella implementazione di funzioni simili alla built-in `repr()`; i limiti sulla dimensione per differenti tipi di oggetto vengono aggiunti per evitare la generazione di rappresentazioni troppo lunghe.

aRepr
Un'istanza di `Repr` che viene utilizzata per fornire la funzione `repr()` descritta sotto. Cambiando gli attributi di questo oggetto si interverrà sul limite di dimensione usato da `repr()` e dal debugger Python.

repr(*obj*)
Il metodo `repr()` di `aRepr`. Restituisce una stringa simile a quella restituita dalla funzione built-in avente lo stesso nome, ma con limiti imposti su più dimensioni.

3.26.1 Oggetti Repr

Le istanze `Repr` forniscono diversi membri che possono venire usati per fornire i limiti sulla dimensione per la rappresentazione di differenti tipi di oggetti, e metodi il cui formato specifica i tipi di oggetto.

maxlevel
Profondità limite nella creazione di rappresentazioni ricorsive. Il valore predefinito è `6`.

maxdict
maxlist
maxtuple
Limiti sul numero degli elementi rappresentati per il tipo di oggetto indicato. Il valore predefinito per `maxdict` è `4`, per gli altri, `6`.

maxlong
Massimo numero di caratteri nella rappresentazione di un intero long. Le cifre vengono rimosse a partire dal centro della stringa. Il valore predefinito è `40`.

maxstring
Limite sul numero di caratteri nella rappresentazione della stringa. Notate che come fonte da cui estrarre i caratteri, viene utilizzata la "normale" rappresentazione della stringa: se nella rappresentazione sono necessarie sequenze di escape, queste possono venire tagliate quando la rappresentazione viene abbreviata. Il valore predefinito è `30`.

maxother
Questo limite viene utilizzato per controllare la dimensione dei tipi oggetto per i quali nessuno specifico metodo di formattazione sia disponibile nell'oggetto `Repr`. Viene applicato in una modalità simile a `maxstring`. Il valore predefinito è `20`.

repr(*obj*)
L'equivalente della funzione built-in `repr()` che utilizza la formattazione imposta dall'istanza.

repr1(*obj*, *level*)
Implementazione ricorsiva usata da `repr()`. Utilizza il tipo di *obj* per determinare quale metodo di formattazione debba venire chiamato, passandogli *obj* e *level*. I metodi specifici del tipo devono chiamare `repr1()` per effettuare la formattazione ricorsiva, con *level* `- 1` per il valore di *level* nella chiamata ricorsiva.

repr_*type*(*obj*, *level*)
Metodi di formattazione per tipi specifici vengono implementati come metodi con un nome basato sul nome del tipo. Nel nome del metodo, *type* viene rimpiazzato con `string.join(string.split(type(`*obj*`).__name__, '_'))`. I comandi a questi metodi vengono gestiti da `repr1()`. Metodi per tipi specifici che necessitano di formattare ricorsivamente un valore, dovrebbero effettuare la chiamata a '`self.repr1(`*subobj*`, `*level*` - 1)`'.

3.26.2 Derivare classi da oggetti Repr

L'uso di comandi dinamici di `Repr.repr1()` consente di derivare delle classi da `Repr`, allo scopo di aggiungere il supporto a ulteriori tipi di oggetti built-in, o per modificare la gestione di tipi già supportati. Questo esempio mostra come può venire aggiunto il supporto speciale per i file oggetto.

```
import repr
import sys

class MyRepr(repr.Repr):
    def repr_file(self, obj, level):
        if obj.name in ['<stdin>', '<stdout>', '<stderr>']:
            return obj.name
        else:
            return 'obj'

aRepr = MyRepr()
print aRepr.repr(sys.stdin)          # prints '<stdin>'
```

3.27 new — Creazione di oggetti interni in runtime

Il modulo `new` fornisce un'interfaccia per le funzioni di creazione degli oggetti dell'interprete. Il modulo viene principalmente impiegato nelle funzioni di tipo marshal, quando si deve creare "magicamente" un nuovo oggetto non utilizzando le normali funzioni di creazione. Il modulo fornisce un'interfaccia di basso livello all'interprete, quindi deve venire impiegato la necessaria cautela.

Il modulo `new` definisce le seguenti funzioni:

`instance`(*class*[, *dict*])
Questa funzione crea un'istanza di *class* con il dizionario *dict*, senza chiamare il costruttore `__init__()`. Se *dict* viene omesso o è *None*, un nuovo dizionario vuoto viene creato per la nuova istanza. Notate che non ci sono garanzie che l'oggetto sarà in uno stato consistente.

`instancemethod`(*function, instance, class*)
Questa funzione restituisce un oggetto metodo, limitato da *instance*, o senza sorta di limiti se *instance* è `None`. La funzione *function* deve essere chiamabile.

`function`(*code, globals*[, *name*[, *argdefs*]])
Restituisce una funzione (di natura Python) con i *code* e *globals* forniti. Se *name* viene indicato, deve essere una stringa o `None`. Se è una stringa, la funzione assumerà il *name* indicato, altrimenti il nome della funzione restituita verrà prelevato da *code*`.co_name`. Se *argdefs* viene fornito, deve essere una tupla, e verrà utilizzato per determinare il valore predefinito dei parametri.

`code`(*argcount, nlocals, stacksize, flags, codestring, constants, names, varnames, filename, name, firstlineno, lnotab*)
Questa funzione è un'interfaccia alla funzione C `PyCode_New()`.

`module`(*name*)
Questa funzione restituisce un nuovo modulo oggetto avente come nome il valore di *name*. *name* deve essere una stringa.

`classobj`(*name, baseclasses, dict*)
Questa funzione restituisce un nuovo oggetto classe, con nome uguale a *name*, derivato da *baseclasses* (che dovrebbe essere una tupla di classi) e con spazio dei nomi uguale a *dict*.

3.28 site — Estensione alla configurazione specifica della piattaforma

Questo modulo viene automaticamente importato durante l'inizializzazione. L'importazione automatica può venire soppressa usando l'opzione **-S** dell'interprete.

Importando questo modulo verranno aggiunti, al modulo di ricerca del percorso, i percorsi specifici dell'ambiente.

Inizia il suo lavoro costruendo fino ad un massimo di quattro directory, da quella di testa a quella finale. Per la parte iniziale, utilizza `sys.prefix` e `sys.exec_prefix`; le intestazioni vuote vengono saltate. Per la parte finale, utilizza una stringa vuota (su Macintosh o Windows) o utilizza per primo 'lib/python2.3/site-packages' e quindi 'lib/site-python' (su UNIX). Per ogni diversa combinazione testa-coda, verifica se questa si riferisce ad una directory esistente, e se è così, la aggiunge a `sys.path`, verificando inoltre il nuovo percorso dei file di configurazione.

Un file di configurazione del percorso è un file il cui nome ha la forma '*package*.pth', ed esiste in una delle quattro directory menzionate precedentemente; i suoi contenuti sono degli elementi addizionali (uno per riga) che devono venire aggiunti a sys.path. Elementi non esistenti non vengono mai aggiunti a `sys.path`, ma non viene fatto alcun controllo per verificare se gli elementi si riferiscono ad una directory (piuttosto che ad un file). Nessun elemento viene aggiunto a `sys.path` più di una volta. Le righe vuote e quelle che iniziano con il carattere # vengono saltate. Le righe che iniziano con `import` vengono eseguite.

Per esempio, supponete che `sys.prefix` e `sys.exec_prefix` vengano impostate a '/usr/local'. La libreria Python 2.3.4 viene quindi installata in '/usr/local/lib/python2.3' (dove solo i primi 3 caratteri di `sys.version` vengono utilizzati per costruire il nome del percorso di installazione). Supponete che questa abbia una sotto directory '/usr/local/lib/python2.3/site-packages' con tre sotto directory, 'foo', 'bar' e 'spam', e due percorsi dei file di configurazione, 'foo.pth' e 'bar.pth'. Assumete anche che 'foo.pth' contenga i seguenti:

```
# configurazione del package foo

foo
bar
bletch
```

e che 'bar.pth' contenga:

```
#  configurazione del package bar

bar
```

Quindi le seguenti directory vengono aggiunte a `sys.path`, in questo ordine:

```
/usr/local/lib/python2.3/site-packages/bar
/usr/local/lib/python2.3/site-packages/foo
```

Notate che 'bletch' viene omesso perché non esiste; la directory 'bar' precede la directory 'foo' perché 'bar.pth' viene alfabeticamente prima di 'foo.pth'; e 'spam' viene omesso perché non viene menzionato in nessun file di configurazione dei percorsi.

Dopo questa manipolazione di percorsi, viene effettuato un tentativo per importare un modulo chiamato `sitecustomize` che può effettuare personalizzazioni arbitrarie specifiche per l'ambiente di sistema in oggetto. Se questa importazione fallisce con un'eccezione `ImportError`, il tentativo di importazione viene silenziosamente ignorato.

Notate che per alcuni sistemi non UNIX, `sys.prefix` e `sys.exec_prefix` sono vuoti, e la manipolazioni dei percorsi viene saltata; comunque l'importazione di `sitecustomize` viene ancora tentata.

3.29 `user` — Strumenti aggiuntivi per la configurazione specifica dell'utente

Come policy predefinita (NdT: politica di condotta), l'interprete Python non esegue del codice proprio dell'utente all'avvio di programmi Python. (Solo le sessioni interattive eseguono lo script specificato dalla variabile d'ambiente PYTHONSTARTUP, se esiste).

Comunque, su alcuni programmi o implementazioni specifiche della piattaforma può essere utile consentire agli utenti di avere un file standard di personalizzazione, che si possa avviare qualora un programma lo richieda. Questo modulo implementa tale meccanismo. Un programma che si desidera usi il meccanismo, deve eseguire l'istruzione:

```
import user
```

Il modulo `user` cerca un file '.pythonrc.py' nella directory personale dell'utente e se questo può venire aperto, lo esegue (usando `execfile()`) nel suo (il modulo `user`) spazio dei nomi globale. Errori durante questa fase non vengono gestiti; sta al programma che importa il modulo `user` farlo, se lo si desidera. Si assume che la directory personale venga indicata dalla variabile d'ambiente HOME; se questa non è stata impostata, viene utilizzata la directory corrente.

Il file utente '.pythonrc.py' può concretamente testare `sys.version` se si desidera fare diverse cose in funzione della versione di Python in uso.

Un avviso agli utenti: siate molto conservativi sul luogo in cui metterete il vostro file '.pythonrc.py'. Poiché non si conoscoscono a priori i programmi che lo utilizzerano, non è generalmente una buona idea cambiare il comportamento dei moduli standard e delle funzioni.

Un suggerimento per i programmatori che desiderano utilizzare questo meccanismo: un metodo semplice per lasciare agli utenti la possibilità di specificare opzioni per il vostro package è quello di definire delle variabili nel loro '.pythonrc.py', testate nel vostro modulo. Per esempio, un modulo `spam`, che possiede un alto livello di dettaglio, può cercare una variabile `user.spam_verbose`, come segue:

```
import user

verbose = bool(getattr(user, ''spam_verbose'', 0))
```

La forma in tre argomenti della funzione `getattr()` viene usata nel caso in cui l'utente non abbia definito `spam_verbose` nel proprio file '.pythonrc.py'.

Per programmi con personalizzazioni pesanti, è meglio implementare la lettura di un file di personalizzazione specifico.

Programmi utilizzati in ambiti di sicurezza o di privacy *non* dovrebbero importare questo modulo; un utente può facilmente introdursi nel programma piazzando del codice arbitrario nel file '.pythonrc.py'.

Moduli per uso generale *non* dovrebbero importare questo modulo; potrebbe interferire con l'operazione di import del programma.

Vedete anche:

Modulo `site` (sezione 3.28):
Meccanismo di personalizzazione della piattaforma.

3.30 `__builtin__` — Funzioni built-in

Questo modulo fornisce accesso diretto a tutti gli identificatori 'built-in' di Python; per esempio, `__builtin__.open` è il nome completo della funzione built-in `open()`. Vedete la sezione 2.1, "Funzioni built-in".

3.31 __main__ — Ambiente per gli script di alto livello

Questo modulo rappresenta il campo d'azione (altrimenti anonimo) in cui il programma principale dell'interprete esegue comandi letti sia dallo standard input, che da un file di script, o da un prompt interattivo. È questo l'ambiente in cui l'espressione idiomatica "script condizionale" permette l'esecuzione di uno script:

```
if __name__ == "__main__":
    main()
```

3.32 __future__ — Definizione delle istruzioni future

__future__ è un modulo reale, e si propone tre obiettivi:

- Evitare confusione tra gli strumenti esistenti, che analizzano le istruzioni import e si aspettano di trovare i moduli che stanno importando.
- Assicurarsi che se l'istruzione future viene eseguita in versioni di Python precedenti al 2.1, al massimo causi il sollevamento di eccezioni di runtime (l'import di __future__ fallirà, perché non esistevano moduli con questo nome prima del 2.1).
- Documentare l'introduzione di cambiamenti incompatibili, e quando verranno resi — o se lo sono già — obbligatori. Questo è un metodo di documentazione dinamico, e può essere esaminato pragmaticamente importando il modulo __future__ ed esaminandone il contenuto.

Ogni dichiarazione in '__future__.py' è nella forma:

```
FeatureName = "_Feature(" OptionalRelease "," MandatoryRelease ","
                          CompilerFlag ")"
```

dove, normalmente, *OptionalRelease* è minore di *MandatoryRelease*, ed entrambe sono composte da tuple a cinque elementi, nella stessa forma di sys.version_info:

```
(PY_MAJOR_VERSION, # il numero 2 in 2.1.0a3; un intero
 PY_MINOR_VERSION, # il numero 1; un intero
 PY_MICRO_VERSION, # il numero 0; un intero
 PY_RELEASE_LEVEL, # "alpha", "beta", "candidate" o "final"; stringa
 PY_RELEASE_SERIAL # il numero 3; un intero
)
```

OptionalRelease registra la prima versione in cui la caratteristica introdotta è stata accettata.

Nel caso di una *MandatoryRelease* che non sia stata ancora rilasciata, *MandatoryRelease* predice la versione in cui la nuova caratteristica diverrà parte del linguaggio.

Altrimenti *MandatoryRelease* registra il momento in cui la nuova caratteristica è stata introdotta come facente parte del linguaggio; nelle versioni in questione o in quelle successive, i moduli non necessitano più di un'istruzione future per utilizzare la nuova caratteristica, ma possono continuare ad usarla nella sua importazione.

MandatoryRelease può anche essere None, significa che la novità pianificata verrà eliminata.

Le istanze di classe _Feature possiedono due metodi corrispondenti, getOptionalRelease() e getMandatoryRelease().

CompilerFlag è l'opzione (campo di bit) che dovrebbe venire passato nel quarto argomento della funzione built-in compile() per abilitare la nuova caratteristica nel codice compilato dinamicamente. Questa opzione viene memorizzata nell'attributo compiler_flag sulle istanze _Future.

Nessuna descrizione di funzionalità verrà cancellata da __future__.

CAPITOLO

QUATTRO

Servizi per le stringhe

Il modulo descritto in questo capitolo fornisce un ampio spettro di operazioni per la manipolazione delle stringhe. Ecco una descrizione:

`string`	Operazioni comuni sulle stringhe.
`re`	Operazioni di ricerca e corrispondenza delle espressioni regolari con le regole sintattiche in stile Perl.
`struct`	Interpreta le stringhe come dati binari impacchettati.
`difflib`	Helpers for computing differences between objects.
`fpformat`	Funzioni generali per la formattazione dei numeri in virgola mobile.
`StringIO`	Legge e scrive stringhe come fossero file.
`cStringIO`	Versione più veloce di `StringIO`, ma da cui non si può derivare una classe.
`textwrap`	Text wrapping and filling
`encodings.idna`	Implementazione dell'Internazionalizzazione dei Nomi di Dominio
`unicodedata`	Accesso al Database Unicodee.
`stringprep`	Preparazione delle stringhe, secondo la RFC 3454

Informazioni sui metodi degli oggetti stringa possono venire trovati nella sezione 2.3.6, "Metodi delle stringhe."

4.1 `string` — Operazioni comuni sulle stringhe

Questo modulo definisce alcune costanti utili per controllare le classi di carattere ed alcune utili funzioni sulle stringhe. Per le funzioni sulle stringhe basate sulle espressioni regolari, vedete il modulo `re`.

Le costanti definite in questo modulo sono:

`ascii_letters`
: La concatenazione delle costanti `ascii_lowercase` e `ascii_uppercase` descritte di seguito. Questo valore non dipende dalla localizzazione.

`ascii_lowercase`
: Le lettere minuscole `'abcdefghijklmnopqrstuvwxyz'`. Questo valore non dipende dalla localizzazione e non cambierà.

`ascii_uppercase`
: Le lettere maiuscole `'ABCDEFGHIJKLMNOPQRSTUVWXYZ'`. Questo valore non dipende dalla localizzazione e non cambierà.

`digits`
: La stringa `'0123456789'`.

`hexdigits`
: La stringa `'0123456789abcdefABCDEF'`.

`letters`
: La concatenazione delle stringhe `lowercase` e `uppercase` descritte di seguito. Il valore specifico dipende dalla localizzazione e viene aggiornato quando viene chiamata la funzione `locale.setlocale()`.

lowercase
Una stringa contenente tutti i caratteri che vengono considerati lettere minuscole. Su molti sistemi questa corrisponde alla stringa `'abcdefghijklmnopqrstuvwxyz'`. Non cambiate la sua definizione: l'effetto sulle funzioni `upper()` e `swapcase()` è indefinito. Il valore specifico dipende dalla localizzazione e viene aggiornato quando viene chiamata la funzione `locale.setlocale()`.

octdigits
La stringa `'01234567'`.

punctuation
La stringa di caratteri ASCII che vengono considerati caratteri di punteggiatura nella localizzazione '`C`'.

printable
La stringa di caratteri che vengono considerati stampabili. Questa è una combinazione di `digits`, `letters`, `punctuation` e `whitespace`.

uppercase
Una stringa contenente tutti i caratteri che vengono considerati lettere maiuscole. Su molti sistemi questa corrisponde alla stringa `'ABCDEFGHIJKLMNOPQRSTUVWXYZ'`. Non cambiate la sua definizione: l'effetto sulle funzioni `lower()` e `swapcase()` è indefinito. Il valore specifico dipende dalla localizzazione e viene aggiornato quando viene chiamata la funzione `locale.setlocale()`.

whitespace
Una stringa contenente tutti i caratteri che vengono considerati di spaziatura. Su molti sistemi questa include i caratteri spazio, tabulazione, fine riga, return, formfeed e tabulazioni verticali. Non cambiate la sua definizione: l'effetto sulle funzioni `strip()` e `split()` è indefinito.

Molte delle funzioni fornite da questo modulo vengono anche definite come metodi delle stringhe e degli oggetti Unicode; per maggiori informazioni vedete "Metodi delle stringhe" (sezione 2.3.6). Le funzioni definite in questo modulo sono:

atof(*s*)
Deprecato dalla versione 2.0. Utilizzate la funzione built-in `float()`.

Converte una stringa in un numero in virgola mobile. La stringa deve avere la sintassi standard per un numero in virgola mobile di Python, facoltativamente preceduto dal segno ('`+`' o '`-`'). Notate che si comporta in modo identico alla funzione built-in `float()` quando ad essa viene passata una stringa.

Note: Quando passati in una stringa, i valori NaN e Infinito possono venire restituiti, in funzione della sottostante libreria C. L'insieme specifico delle stringhe accettate che causano questi valori dipende interamente dalla libreria C, che è variabile.

atoi(*s*[, *base*])
Deprecato dalla versione 2.0. Utilizzate la funzione built-in `int()`.

Converte una stringa *s* in un intero in una data *base*. La stringa deve consistere di una o più cifre, facoltativamente precedute dal segno ('`+`' o '`-`'). La *base* predefinita è 10. Se è 0, una base viene scelta in funzione dei caratteri iniziali della stringa (dopo che il segno è stato eliminato): '`0x`' oppure '`0X`' significa 16, '`0`' significa 8, tutto il resto significa 10. Se la *base* è 16, viene accettato il prefisso '`0x`' or '`0X`', sebbene non richiesto. Si comporta in modo identico alla funzione built-in `int()` quando viene ad essa passata una stringa. (Altre note: per un'interpretazione più flessibile delle costanti numeriche, utilizzate la funzione built-in `eval()`.)

atol(*s*[, *base*])
Deprecato dalla versione 2.0. Utilizzate la funzione built-in `long()`.

Converte la stringa *s* in un intero long nella *base* data. La stringa deve consistere di una o più cifre, facoltativamente precedute dal segno ('`+`' or '`-`'). L'argomento *base* ha lo stesso significato della funzione `atoi()`. Un valore finale '`l`' o '`L`' non viene ammesso, a meno che la base non sia 0. Notate che quando invocata senza specificare la *base* o con la *base* impostata a 10, si comporta in modo identico alla funzione built-in `long()` quando viene ad essa passata una stringa.

capitalize(*parola*)
Restituisce una copia della *parola* con soltanto il suo primo carattere in maiuscolo.

capwords(*s*)
Divide l'argomento in parole, usando la funzione `split()`, converte in maiuscolo il primo carattere di

ogni parola utilizzando la funzione `capitalize()`, e riunisce le parole usando la funzione `join()`. Notate che sostituisce i caratteri di spaziatura tra le parole con un singolo spazio ed elimina tutti i caratteri di spaziatura ad inizio e fine stringa.

`expandtabs`(*s*[, *tabsize*])
Espande le tabulazioni in una stringa, per esempio li sostituisce con uno o più spazi, in funzione della colonna corrente e del numero di caratteri corrispondenti al *tabsize* dato. Il numero di colonna viene reimpostato a zero dopo ogni fine riga presente nella stringa. Non interpreta invece gli altri caratteri non stampabili o le sequenze di escape. La dimensione di una tabulazione è predefinita ad 8.

`find`(*s, sub*[, *start*[,*end*]])
Restituisce l'indice minore in *s* in cui viene trovata la sotto stringa *sub* tale che sia interamente contenuta in `s[`*start*`:`*end*`]`. Restituisce `-1` in caso di fallimento. I valori predefiniti per *start* ed *end*, e l'interpretazione dei valori negativi, sono gli stessi di quelli considerati per le fette.

`rfind`(*s, sub*[, *start*[, *end*]])
Come `find()`, ma trova l'indice maggiore.

`index`(*s, sub*[, *start*[, *end*]])
Come `find()`, ma solleva l'eccezione `ValueError` quando la sotto stringa non viene trovata.

`rindex`(*s, sub*[, *start*[, *end*]])
Come `rfind()`, ma solleva l'eccezione `ValueError` quando la sotto stringa non viene trovata.

`count`(*s, sub*[, *start*[, *end*]])
Restituisce il numero di occorrenze (senza sovrapposizioni) della sotto stringa *sub* presenti nella stringa `s[`*start*`:`*end*`]`. I valori predefiniti per *start* ed *end*, e l'interpretazione dei valori negativi, sono gli stessi di quelli considerati per le fette.

`lower`(*s*)
Restituisce una copia di *s*, ma con le lettere maiuscole convertite in minuscole.

`maketrans`(*from, to*)
Restituisce una tabella di traduzione adatta per venire utilizzata con la funzioni `translate()` o `regex.compile()`, che mapperà ogni carattere in *from* nel carattere nella stessa posizione in *to*; *from* e *to* devono avere la stessa lunghezza.

Avvertenze: Non utilizzate stringhe derivate da `lowercase` e `uppercase` come argomenti; in alcune localizzazioni, queste non hanno la stessa lunghezza. Per conversioni da maiuscolo e minuscolo, utilizzate sempre le funzioni `lower()` ed `upper()`.

`split`(*s*[, *sep*[, *maxsplit*]])
Restuisce una lista delle parole della stringa *s*. Se il secondo argomento facoltativo *sep* non è presente o è `None`, le parole vengono separate da stringhe arbitrarie di caratteri di spaziatura (spazi, tabulazioni, fine riga, return, formfeed). Se il secondo argomento *sep* è presente e non è `None`, specifica una stringa da utilizzare come separatore di parole. La lista restituita avrà un elemento in più, aggiunto, rispetto al numero delle occorrenze senza sovrapposizioni del separatore nella stringa. Il valore predefinito del terzo argomento facoltativo *maxsplit* è 0. Se è un valore diverso da zero, vengono effettuate al massimo *maxsplit* divisioni, e la parte rimanente della stringa viene restituita come ultimo argomento della lista (così la lista avrà al più *maxsplit*`+1` elementi).

Il comportamento di split su di una stringa vuota dipende dal valore di *sep*. Se *sep* non viene specificato, o specificato come `None`, il risultato sarà una lista vuota. Se *sep* viene specificato come una qualsiasi stringa, il risultato sarà una lista contenente un elemento che è una stringa vuota.

`rsplit`(*s*[, *sep*[, *maxsplit*]])
Restituisce una lista delle parole della stringa *s*, esaminando *s* dalla fine. A tutti gli effetti, la lista risultante delle parole è la stessa restituita dalla funzione `split()`, ad eccezione di quando il terzo argomento facoltativo viene esplicitamente specificato e diverso da zero. Quando *maxsplit* è un valore diverso da zero, vengono effettuate al più *maxsplit* divisioni della stringa – quelle *più a destra* – e la parte rimanente della stringa viene restituita come primo elemento della lista (così, la lista avrà al più *maxsplit*`+1` elementi). Nuovo nella versione 2.4.

`splitfields`(*s*[, *sep*[, *maxsplit*]])
Questa funzione si comporta in modo identico a `split()`. (In passato, `split()` veniva utilizzata con un

solo argomento, mentre `splitfields()` veniva utilizzata con due soli argomenti.)

`join`(*words*[, *sep*])
Concatena una lista o una tupla di parole, utilizzando le occorrenze di *sep*. Il valore predefinito di *sep* è un singolo spazio. L'espressione '`string.join(string.split(`*s*`, `*sep*`), `*sep*`)`' è sempre equivalente a *s*.

`joinfields`(*words*[, *sep*])
Questa funzione si comporta in modo identico a `join()`. (In passato, `join()` veniva utilizzata con un solo argomento, mentre `joinfields()` veniva utilizzata con due soli argomenti.) Notate che non esiste nessun metodo `joinfields()` per oggetti di tipo stringa; utilizzate piuttosto la funzione `join()`.

`lstrip`(*s*[, *chars*])
Restituisce una copia della stringa con i caratteri iniziali rimossi. Se *chars* viene omesso o è `None`, i caratteri di spaziatura vengono rimossi. Se viene impostato e non è `None`, *chars* deve essere una stringa; i caratteri nella stringa verranno rimossi dall'inizio della stringa su cui viene chiamato questo metodo. Modificato nella versione 2.2.3: Il parametro *chars* non può essere passato in versioni precedenti alla 2.2.

`rstrip`(*s*[, *chars*])
Restituisce una copia della stringa con i caratteri finali rimossi. Se *chars* viene omesso o è `None`, i caratteri di spaziatura vengono rimossi. Se viene impostato e non è `None`, *chars* deve essere una stringa; i caratteri nella stringa verranno rimossi dalla fine della stringa su cui viene chiamato questo metodo. Modificato nella versione 2.2.3: Il parametro *chars* non può essere passato in versioni precedenti alla 2.2.

`strip`(*s*[, *chars*])
Restituisce una copia della stringa con i caratteri iniziali e finali rimossi. Se *chars* viene omesso o è `None`, i caratteri di spaziatura vengono rimossi. Se viene impostato e non è `None`, *chars* deve essere una stringa; i caratteri nella stringa verranno rimossi da entrambe le estremità della stringa su cui il metodo viene chiamato. Modificato nella versione 2.2.3: Il parametro *chars* viene aggiunto. Il parametro *chars* non può essere passato in versioni precedenti alla 2.2.

`swapcase`(*s*)
Restituisce una copia di *s*, ma con le lettere minuscole convertite in maiuscole e viceversa.

`translate`(*s, table*[, *deletechars*])
Cancella da *s* tutti i caratteri che si trovano in *deletechars* (se presente), e traduce i caratteri rimasti utilizzando *table*. *table* deve essere una stringa di 256 caratteri, dove ogni elemento corrisponde alla traduzione del carattere avente il valore corrispodente alla posizione all'interno della stringa di traduzione.

`upper`(*s*)
Restituisce una copia di *s*, ma con le lettere minuscole convertite in maiuscole.

`ljust`(*s, width*)
`rjust`(*s, width*)
`center`(*s, width*)
Queste funzioni rispettivamente giustificano a sinistra, a destra e centralmente una stringa, in un campo di dimensione data. Restituiscono una stringa di larghezza pari ad almeno *width* caratteri, creata riempiendo la stringa *s* con degli spazi fino al raggiungimendo della dimensione data, a destra, a sinistra o ad entrambi i lati. La stringa non viene mai troncata.

`zfill`(*s, width*)
Riempie una stringa numerica a sinistra, con degli 0 fino a che non viene raggiunta la larghezza data. Le stringhe che iniziano con un segno vengono gestite correttamente.

`replace`(*str, old, new*[, *maxreplace*])
Restituisce una copia della stringa *str* con tutte le occorrenze della sotto stringa *old* sostituite dalla stringa *new*. Se l'argomento facoltativo *maxreplace* viene fornito, le prime *maxreplace* occorrenze vengono sostituite.

4.2 `re` — Operazioni con le espressioni regolari

Questo modulo fornisce operazioni per la corrispondenza delle espressioni regolari simile a quelle utilizzate in Perl. Le stringhe utilizzate come modello per le espressioni regolari non possono contenere byte nulli, ma possono specificarli usando la notazione `\number`. Sia i modelli che le stringhe da ricercare possono essere sia stringhe Unicode che stringhe a 8-bit. Il modulo `re` è sempre disponibile.

Le espressioni regolari usano il carattere backslash ('`\`') per indicare le forme speciali o per permettere che i caratteri speciali vengano utilizzati senza invocare il loro significato speciale. Questo coincide con l'utilizzo di Python degli stessi caratteri per gli stessi scopi nelle stringhe costanti; per esempio, per far corrispondere una costante backslash, bisogna scrivere `'\\\\'` come modello, perché l'espressione regolare deve essere '`\\`' e ogni backslash deve essere rappresentato come '`\\`' in una stringa costante in Python.

La soluzione è quella di utilizzare la notazione Python per le stringhe raw nei modelli delle espressioni regolari; i backslash non vengono gestiti in nessun modo speciale in una stringa costante avente il prefisso '`r`'. Perciò `r\n` è una stringa di due caratteri che contiene '`\`' e '`n`', mentre `\n` è una stringa di un carattere che contiene un carattere di fine riga. Normalmente i modelli vengono espressi in codice Python usando questa notazione per le stringhe raw.

Vedete anche:

Mastering Regular Expressions
: Un libro di Jeffrey Friedl sulle espressioni regolari, pubblicato da O'Reilly. La seconda edizione di questo libro non copre più Python, ma la prima edizione trattava in dettaglio la scrittura di buoni modelli di espressioni regolari.

4.2.1 Sintassi delle espressioni regolari

Un'espressione regolare (comunemente abbreviata in RE, dall'Inglese Regular Expression) specifica un insieme di stringhe che vi corrispondono; le funzioni in questo modulo permettono di controllare se una particolare stringa corrisponde ad una data espressione regolare (o se una data espressione regolare corrisponde ad una data stringa, che poi è la stessa cosa).

Le espressioni regolari possono venire concatenate per formare una nuova espressione regolare; se *A* e *B* sono entrambe espressioni regolari, allora anche *AB* è un'espressione regolare. In generale, se una stringa *p* corrisponde ad *A* e un'altra stringa *q* corrisponde a *B*, la stringa *pq* corrisponderà ad *AB*. Questo è vero meno che *A* o *B* contengano operazioni a bassa precedenza, condizioni al contorno tra *A* e *B*, o abbiamo riferimenti di gruppo numerati. Così, espressioni complessse possono facilmente venire costruite partendo da espressioni primitive più semplici, come quelle descritte qui. Per i dettagli sulla teoria e l'implementazione delle espressioni regolari, consultate il libro di Friedl (vedete il riferimento sopra) o un qualsiasi libro sulla creazione di compilatori.

Qui di seguito si troverà una breve spiegazione sul formato delle espressioni regolari. Per maggiori informazioni e una spiegazione più particolareggiata, consultate l'HOWTO sulle Espressioni Regolari, accessibile all'URL http://www.python.org/doc/howto/.

Le espressioni regolari possono contenere sia caratteri speciali che ordinari. Caratteri più ordinari singoli, come '`A`', '`a`', o '`0`', sono le espressioni regolari più semplici; semplicemente corrispondono a se stessi. Potete concatenare i caratteri ordinari, così `last` corrisponde alla stringa `'last'`. (Nel resto di questa sezione, scriveremo le espressioni regolari in `questo modo particolare`, generalmente senza quotatura, e le stringhe da corrispondere tra `'virgolette'`.)

Alcuni caratteri, come '`|`' o '`(`', sono speciali. I caratteri speciali possono indicare sia classi di carattere ordinarie, sia modificatori dell'interpretazione delle espressioni regolari indicate da questi.

I caratteri speciali sono:

'`.`' (Punto, in inglese dot). Nella modalità predefinita, questo corrisponde a qualunque carattere, ad eccezione del fine riga. Se l'opzione `DOTALL` viene specificata, il punto corrisponde a qualunque carattere, fine riga compreso.

'`^`' (Cappelletto, in inglese caret). Corrisponde all'inizio di una stringa, e nella modalità `MULTILINE` corrisponde anche all'inizio di ogni nuova riga, ovvero dopo ogni carattere di fine riga.

'`$`' Corrisponde alla fine della stringa o prima del carattere di fine riga alla fine della stringa; nella modalità `MULTILINE` corrisponde prima di ogni fine riga. `foo` corrisponde sia a 'foo' che

a 'foobar', mentre l'espressione regolare `foo$` corrisponde solo a 'foo'. L'espressione regolare `foo.$` in 'foo1\nfoo2\n' restituisce la corrispondenza 'foo2' normalmente, ma 'foo1' nella modalità `MULTILINE`.

'*' Fa sì che l'espressione regolare risultante corrisponda a 0 o più ripetizioni della precedente espressione regolare, un numero qualunque di ripetizioni è possibile. `ab*` corrisponde ad 'a', 'ab' o 'a' seguita da qualunque numero di 'b'.

'+' Fa sì che l'espressione regolare risultante corrisponda a 1 o più ripetizioni della precedente espressionre regolare. `ab+` corrisponderà ad 'a' seguita da un qualunque numero, diverso da zero, di 'b'; non corrisponde alla stringa 'a'.

'?' Fa sì che l'espressione regolare risultante corrisponda a 0 o 1 ripetizioni della precedente espressione regolare. `ab?` corrisponderà sia ad 'a' che ad 'ab'.

?, +?, ?? I qualificatori '', '+' e '?' sono *ingordi*; corrispondono a tutto il testo possibile. A volte questo comportamento non è quello desiderato; se l'espressione regolare `<.*>` viene fatta corrispondere alla stringa `'<H1>title</H1>'`, corrisponderà all'intera stringa e non solo a `'<H1>'`. Aggiungere il carattere '?' dopo il qualificatore rende la ricerca delle corrispondenze non *non-ingorda* o *minimale*; verrà fatto corrispondere il *minor* numero possibile di caratteri. Usando `.*?` nella precedente espressione vi sarà corrispondenza solamente con `'<H1>'`.

{*m*} Specifica che devono corrispondere esattamente *m* copie della precedente espressione regolare; un numero minore di corrispondenze farà sì che l'intera espressione regolare non corrisponda. Per esempio, `a{6}` corrisponderà esattamente con sei caratteri 'a', ma non con cinque.

{*m*,*n*} Fa sì che la risultante espressione regolare corrisponda da *m* a *n* ripetizioni dell'espressione regolare precedente, cercando di far corrispondere più ripetizioni possibili. Per esempio, `a{3,5}` corrisponderà da tre a cinque caratteri 'a'. Omettendo *m* si specifica un limite inferiore di zero e omettendo *n* si specifica un limite superiore infinito. Come esempio `a{4,}b` corrisponderà con `aaaab` o con un migliaio di caratteri 'a' seguiti da una `b`, ma non ad `aaab`. La virgola non dovrebbe venire omessa, o il modificatore potrebbe venire confuso con la forma descritta in precedenza.

{*m*,*n*}? Fa sì che l'espressione regolare risultante corrisponda da *m* a *n* ripetizioni della precedente espressione regolare, cercando di far corrispondere il *minor numero* di ripetizioni possibili. Questa è la versione non "ingorda" del precedente qualificatore. Per esempio, nella stringa di 6 caratteri `'aaaaaa'`, `a{3,5}` farà corrispondere cinque caratteri 'a', mentre `a{3,5}?` farà corrispondere solo tre caratteri.

'\' Utilizzato sia come sequenza di escape per i caratteri speciali (consentendo la corrispondenza di caratteri come '*', '?' e così via), sia per segnalare una sequenza speciale; le sequenze speciali vengono discusse di seguito.

Se non state utilizzando una stringa raw per rappresentare il modello, ricordatevi che Python usa anche i backslash come una sequenza di escape nelle costanti letterali; se la sequenza di escape non viene riconosiuta dal parser di Python, il backslash ed i caratteri seguenti vengono inclusi nella stringa risultante. Tuttavia, nel caso in cui se Python riconoscesse la sequenza risultante, il backslash dovrebbe venire ripetuto due volte. Questo è complicato e difficile da comprendere, perciò è altamente raccomandato che utilizziate sempre le stringhe raw, tranne che per le espressioni più semplici.

[] Vengono utilizzate per indicare un insieme di caratteri. I caratteri possono venire elencati individualmente, o un intervallo di caratteri può venire indicato fornendo due caratteri separati da un trattino ('-'). I caratteri speciali non sono attivi all'interno degli insiemi. Per esempio, `[akm$]` corrisponderà ognuno dei caratteri 'a', 'k', 'm' o '$'; `[a-z]` corrisponderà a tutte le lettere minuscole, e `[a-zA-Z0-9]` corrisponderà ad ogni lettera o cifra. Le classi di caratteri come `\w` o `\S` (definite di seguito) sono anche acettabili all'interno di un intervallo. Se voleste includere un carattere ']' o '-' in un insieme, fatelo precedere da un backslash o indicatelo come primo carattere dell'intervallo. Il modello `[]]` corrisponderà alla stringa `']'`, per esempio.

Potete far corrispondere i caratteri non compresi in un determinato intervallo. Questo viene indicato includendo il carattere '^' come primo carattere dell'insieme; '^' altrove corrisponderà semplicemente con il carattere '^'. Per esempio, `[^5]` corrisponderà con ogni carattere escluso '5', e `[^^]` corrisponderà con ogni carattere eccetto '^'.

'|' `A|B`, dove A e B possono essere espressioni regolari arbitrarie, crea un'espressione regolare che corrisponderà ad A o a B. Un numero arbitrario di espressioni regolari possono venire separate dal carattere '`|`' in questo modo. Può venire utilizzato anche all'interno dei gruppi (vedete di seguito). Come la stringa bersaglio viene analizzata, le espressioni regolari vengono provate da sinistra a destra. Quando un modello corrisponde completamente, quel ramo viene accettato. Questo significa che quando `A` corrisponde, `B` non verrà testata, anche se dovesse avere una corrispondenza maggiore. In altre parole, l'operatore '`|`' non è mai "ingordo". Per far corrispondere una costante '`|`', utilizzate ⌈`\|`⌋, oppure inseritela in una classe di caratteri, come in ⌈`[|]`⌋.

`(...)` Fa corrispondere qualunque espressione regolare nelle parentesi, e indica l'inizio e la fine di un gruppo; i contenuti di un gruppo possono venire recuperati dopo che una corrispondenza viene verificata, e possono essere nuovamente fatte corrispondere nella stringa con la sequenza speciale ⌈`\`*numero*⌋, descritta di seguito. Per far corrispondere le costanti '`(`' o '`)`', utilizzate ⌈`\(`⌋ o ⌈`\)`⌋ oppure inseritele in una classe di caratteri: ⌈`[(] [)]`⌋.

`(?...)` Questa è una notazione di estensione (un '`?`' che segue un '`(`' non ha significato, altrimenti). Il primo carattere dopo '`?`' determina il significato e l'ulteriore sintassi del costrutto. Le estensioni normalmente non creano un nuovo gruppo; ⌈`(?P<`*nome*`>...)`⌋ è la sola eccezione a questa regola. Di seguito vengono indicate le estenzioni attualmente supportate.

`(?iLmsux)` (Una o più lettere dall'insieme '`i`', '`L`', '`m`', '`s`', '`u`', '`x`'.) Il gruppo corrisponde la stringa vuota; l'insieme delle lettere impostano le opzioni corrispondenti (`re.I`, `re.L`, `re.M`, `re.S`, `re.U`, `re.X`) per l'intera espressione regolare. Questo è utile se volete includere le opzioni come parte dell'espressione regolare, invece che passarle come argomento di *opzione* alla funzione `compile()`.

Notate che il flag ⌈`(?x)`⌋ cambia il modo di analizzare l'espressione. Dovrebbe venire all'inizio della stringa modello, o dopo uno o più caratteri di spaziatura. Se ci sono dei caratteri non di spaziatura prima dell'opzione, i risultati sono indefiniti.

`(?:...)` Una versione delle parentesi che non indica un gruppo. Fa corrispondere qualsiasi espressione regolare contenuta tra parentesi, ma la sotto stringa corrisposta dal gruppo *non può* venire recuperata dopo aver soddisfatto una corrispondenza, e *non può* successivamente venire referenziata nel modello.

`(?P<`*name*`>...)` Simile alle parentesi regolari, ma la sotto stringa corrispondente al gruppo è accessibile attraverso il gruppo simbolico *name*. I nomi dei gruppi devono essere identificatori validi in Python, ed ogni nome di gruppo deve venire definito solo una volta in un'espressione regolare. Un gruppo simbolico è anche un gruppo numerato, come se il gruppo non fosse nominato. Perciò il gruppo chiamato 'id' nell'esempio sottostante può anche venire riferito come il gruppo numerato 1.

Per esempio, se il modello è ⌈`(?P<id>[a-zA-Z_]\w*)`⌋, il gruppo può venire riferito dal suo nome, negli argomenti ai metodi degli oggetti corrispondenti, come `m.group('id')` o `m.end('id')`, e anche attraverso il nome nel testo del modello (per esempio, ⌈`(?P=id)`⌋ ed il testo di sostituzione (come `\g<id>`).

`(?P=`*name*`)` Fa corrispondere qualunque testo precedentemente corrisposto, ad un gruppo chiamato *name*.

`(?#...)` Un commento; il contenuto delle parentesi viene semplicemente ignorato.

`(?=...)` Corrisponde se ⌈`...`⌋ corrisponde successivamente, ma non consuma nessun carattere della stringa. Viene chiamata asserzione di lookahead (NdT: letteralmente guarda avanti). Per esempio, ⌈`Isaac (?=Asimov)`⌋ corrisponderà con `'Isaac '` solo se seguito da `'Asimov'`.

`(?!...)` Corrisponde se ⌈`...`⌋ non corrisponde successivamente. Questa è un'asserzione di lookahead negativa. Per esempio, ⌈`Isaac (?!Asimov)`⌋ corrisponderà ad `'Isaac '` solo se *non* seguito da `'Asimov'`.

`(?<=...)` Corrisponde se la posizione corrente nella stringa viene preceduta da una corrispondenza per ⌈`...`⌋ che termina esattamente alla posizione corrente. Questa viene chiamata un'*asserzione di lookbehind positiva*. ⌈`(?<=abc)def`⌋ troverà un corrispondenza in '`abcdef`', perché il lookbehind controllerà i 3 caratteri predecenti e verificherà se esiste corrispondenza con il modello contenuto. Il modello contenuto deve solo corrispondere stringhe di una data lunghezza fissa, questo significa che ⌈`abc`⌋ o ⌈`a|b`⌋ vengono ammesse, ma ⌈`a*`⌋ e ⌈`a{3,4}`⌋ no. Notate che i modelli che iniziano con un'asserzione

di lookbehind positiva non corrisponderanno mai all'inizio della stringa cercata; desidererete molto probabilmente usare la funzione `search()` piuttosto che la funzione `match()`:

```
>>> import re
>>> m = re.search('(?<=abc)def', 'abcdef')
>>> m.group(0)
'def'
```

Questo esempio cerca una parola preceduta da un trattino:

```
>>> m = re.search('(?<=-)\w+', 'spam-egg')
>>> m.group(0)
'egg'
```

`(?<!...)` Corrisponde se la posizione corrente nella stringa non viene preceduta da una corrispondenza per ⌜`...`⌟. Questa viene chiamata *asserzione di lookbehind negativa*. Similmente all'asserzione di lookbehind positiva, il modello contenuto può solo essere una stringa a lunghezza fissa. I modelli che iniziano con un'asserzione di lookbehind negativa possono corrispondere all'inizio della stringa ricercata.

`(?(`*id/name*`)yes-pattern|no-pattern)` Proverà a corrispondere con ⌜`yes-pattern`⌟ se il gruppo con il dato *id* o il *name* indicato esiste, e con ⌜`no-pattern`⌟ se non esiste. ⌜`|no-pattern`⌟ è opzionale e può venire omesso. Per esempio, ⌜`(<)?(\w+@\w+(?:\.\w+)+)(?(1)>)`⌟ è un semplice sistema di corrispondenza per gli indirizzi e-mail, che corrisponderà con `'<user@host.com>'` e `'user@host.com'`, ma non con `'<user@host.com'`. Nuovo nella versione 2.4.

Le sequenze speciali consistono di '`\`' e di un carattere dalla lista seguente. Se il carattere ordinario non è nella lista, l'espressione regolare risultante corrisponderà al secondo carattere. Per esempio, ⌜`\$`⌟ corrisponderà al carattere '`$`'.

`\`*number* Fa sì che corrisponda il contenuto del gruppo con lo stesso numero. I gruppi vengono numerati a partire da 1. Per esempio, ⌜`(.+) \1`⌟ corrisponde a `'the the'` o a `'55 55'`, ma non a `'the end'`. (notate lo spazio dopo il primo gruppo). Questa sequenza speciale può venire utilizzata soltanto per far corrispondere uno dei primi 99 gruppi. Se la prima cifra di *number* è 0, o se il *number* è di 3 cifre ottali, non verrà interpretato come una corrispondenza su un gruppo, ma come il carattere corrispondente al numero espresso in valore ottale. Tra i caratteri '`[`' e '`]`' di una classe di caratteri, tutti gli escape numerici vengono trattati come caratteri.

`\A` Corrisponde solo all'inizio della stringa.

`\b` Fa sì che corrisponda la stringa vuota, ma solo all'inizio o alla fine di una parola. Una parola viene definita come una sequenza di caratteri alfanumerici e underscore, perciò la fine di una parola viene indicata da un carattere di spaziatura, o un carattere che non sia nè alfanumerico nè un underscore. Notate che `\b` viene definito come il contorno tra `\w` e `\W`, così il preciso insieme di caratteri alfanumerici dipende dai valori delle opzioni `UNICODE` e `LOCALE`. All'interno di un intervallo di caratteri, ⌜`\b`⌟ rappresenta il carattere di backspace, per compatibilità con le stringhe costanti di Python.

`\B` Corrisponde alla stringa vuota, ma solo quando *non* si trova all'inizio o alla fine di una parola. Questo è l'opposto di `\b`, quindi è anche esseo soggetto alle impostazioni di `LOCALE` e `UNICODE`.

`\d` Corrisponde ogni cifra decimale; è equivalente all'insieme ⌜`[0-9]`⌟.

`\D` Corrisponde qualsiasi carattere che non sia una cifra; è equivalente all'insieme ⌜`[^0-9]`⌟.

`\s` Fa sì che corrisponda ogni carattere di spaziatura; è equivalente all'insieme ⌜`[ \t\n\r\f\v]`⌟.

`\S` Fa sì che corrisponda ogni carattere che non sia di spaziatura; è equivalente all'insieme ⌜`[^ \t\n\r\f\v]`⌟.

`\w` Quando le opzioni `LOCALE` e `UNICODE` non vengono specificate, corrisponde a qualsiasi carattere alfanumerico e di trattino basso; questo è equivalente all'insieme `[a-zA-Z0-9_]`. Con `LOCALE`, farà corrispondere l'insieme `[0-9_]` più qualsiasi carattere definito come alfanumerico nella localizzazione corrente. Se `UNICODE` è impostato, questo farà corrispondere i caratteri `[0-9_]` più qualunque carattere classificato come alfanumerico nel database delle proprietà dei caratteri Unicode.

`\W` Quando le opzioni `LOCALE` e `UNICODE` non vengono specificate, fa corrispondere qualsiasi carattere non alfanumerico; questo è equivalente all'insieme `[^a-zA-Z0-9_]`. Con `LOCALE`, farà corrispondere ogni carattere non presente nell'insieme `[0-9_]` e non definito come alfanumerico, nella localizzazione corrente. Se `UNICODE` è impostato, questo farà corrispondere qualsiasi cosa, tranne `[0-9_]` e i caratteri segnati come alfanumerici nel database delle proprietà dei caratteri Unicode.

`\Z` Corrisponde solo alla fine della stringa.

La maggior parte degli escape standard supportati dalle stringhe costanti Python vengono anche accettati dall'analizzatore delle espressioni regolari:

```
\a      \b      \f      \n
\r      \t      \v      \x
\\
```

Gli escape ottali vengono inclusi in una forma limitata: se la prima cifra è uno 0, o ci sono tre cifre ottali, viene considerato un escape ottale. Altrimenti è il riferimento ad un gruppo.

4.2.2 Corrispondenza contro ricerca

Python fornisce due differenti operazioni primitive basate sulle espressioni regolari: match (NdT: corrispondenza) e search (NdT: ricerca). Se siete abituati alla semantica del Perl, l'operazione search è ciò che stavate cercando. Vedete la funzione `search()` ed il corrispondente metodo della compilazione di oggetti espressione regolare.

Notate che match può differire da search usando un'espressione regolare che inizi con '`^`': '`^`' corrisponde solo all'inizio della stringa, o nella modalità `MULTILINE` anche immediatamente dopo un fine riga. L'operazione "match" ha successo solo se il modello corrisponde all'inizio della stringa, senza considerare la modalità, o alla posizione iniziale fornita dal parametro opzionale *pos*, senza considerare se un fine riga lo precede.

```
re.compile("a").match("ba", 1)           # successo
re.compile("^a").search("ba", 1)         # fallimento; 'a' non è all'inizio
re.compile("^a").search("\na", 1)        # fallimento: 'a' non è all'inizio
re.compile("^a", re.M).search("\na", 1)  # successo
re.compile("^a", re.M).search("ba", 1)   # fallimento; non è preceduto da \n
```

4.2.3 Contenuti del modulo

Il modulo definisce le seguenti funzioni e costanti, e un'eccezione:

`compile`(*pattern*[, *flags*])

Compila un modello di espressione regolare in un oggetto espressione regolare, che può venire usato per la corrispondenza, utilizzando i suoi metodi `match()` e `search()`, descritti sotto.

Il comportamento dell'espressione può venire modificato specificando un valore per il campo *flags*. I valori possono essere uno delle seguenti variabili, combinate usando l'OR bit per bit (l'operatore `|`).

La sequenza:

```
prog = re.compile(pat)
result = prog.match(str)
```

è equivalente a

```
result = re.match(pat, str)
```

ma la versione che utilizza `compile()` è più efficiente quando l'espressione verrà usata più di una volta in un singolo programma.

I
IGNORECASE
Effettua una corrispondenza case-insensitive; espressioni come `[A-Z]` corrisponderanno anche alle lettere minuscole. Non viene influenzato dalla localizzazione corrente.

L
LOCALE
Fa in modo che `\w`, `\W`, `\b` e `\B` dipendano dalla localizzazione corrente.

M
MULTILINE
Quando specificato, il carattere del modello '`^`' corrisponde all'inizio della stringa e all'inizio di ogni riga (subito dopo un carattere di fine riga); il carattere del modello '`$`' corrisponde alla fine della stringa e alla fine di ogni riga (immediatamente prima del carattere di fine riga). In modo predefinito, '`^`' corrisponde solo all'inizio della stringa e '`$`' solo alla fine della stringa e subito prima di un fine riga (se presente) alla fine della stringa.

S
DOTALL
Fa in modo che il carattere speciale '`.`' corrisponda ad ogni carattere, incluso il fine riga; senza questa opzione, '`.`' corrisponderà a qualsiasi cosa, *eccetto* il fine riga.

U
UNICODE
Fa in modo che `\w`, `\W`, `\b` e `\B` dipendano dal database delle proprietà dei caratteri Unicode. Nuovo nella versione 2.0.

X
VERBOSE
Questa opzione permette di scrivere espressioni regolari che risultino di aspetto più piacevole. I caratteri di spaziatura all'interno del modello vengono ignorati, a meno che non siano in una classe di caratteri, o preceduti da un backslash che non faccia parte di un escape (NdT: quindi NON '`\\`'), e quando una linea contenga un carattere '`#`' sia in una classe di caratteri sia preceduto da un backslash che non faccia parte di un escape, tutti i caratteri a partire dal carattere '`#`' più a sinistra fino alla fine della stringa vengono ignorati.

`search`(*pattern, string*[, *flags*])
Analizza la *string* cercando una posizione dove il *pattern* delll'espressione regolare genera una corrispondenza, e restituisce una corrispondente istanza di `MatchObject`. Restituisce `None` se nessuna posizione della stringa corrisponde al modello; notate che questo è diverso dal trovare una corrispondenza di lunghezza zero in qualche punto della stringa.

`match`(*pattern, string*[, *flags*])
Se zero o più caratteri all'inizio della *string* corrispondono al *pattern* dell'espressione regolare, restituisce una corrispondenze istanza di `MatchObject`. Restituisce `None` se la stringa non corrisponde al modello; notate che questo è differente rispetto ad una corrispondenza di lunghezza zero.

Note: Se volete localizzare una corrispondenza ovunque nella *string*, utilizzate `search()`.

`split`(*pattern, string*[, *maxsplit* `= 0`])
Divide la *string* rispetto alle occorrenze del *pattern*. Se nel *pattern* sono presenti delle parentesi che indicano dei gruppi, anche il testo di tutti i gruppi nel modello viene restituito come parte della lista risultante. Se *maxsplit* è un valore diverso da zero, vengono effettuate al più *maxsplit* divisioni, e la rimanenza della stringa viene restituita come elemento finale della lista. (Note di incompatibilità: nella versione originale di Python 1.5, *maxsplit* veniva ignorato. Questo comportamento è stato sistemato nelle versioni successive.)

```
>>> re.split('\W+', 'Parole, parole, parole.')
['Parole', 'parole', 'parole', '']
>>> re.split('(\W+)', 'Parole, parole, parole.')
['Parole', ', ', 'parole', ', ', 'parole', '.', '']
>>> re.split('\W+', 'Parole, parole, parole.', 1)
['Parole', 'parole, parole.']
```

Questa funzione combina ed estende le funzionalità delle vecchie `regsub.split()` e `regsub.splitx()`.

`findall`(*pattern, string*)
Restituisce una lista di tutte le corrispondenze (senza sovrapposizioni) del *pattern* in *string*. Se uno o più gruppi sono presenti nel modello, restituisce una lista di gruppi; questa sarà una lista di tuple se il modello ha più di un gruppo. Le corrispondenze vuote vengono incluse nel risultato a meno che non tocchino l'inizio di un'altra corrispondenza. Nuovo nella versione 1.5.2.

`finditer`(*pattern, string*)
Restituisce un iteratore sulle corrispondenze (senza sovrapposizioni) del *pattern* modello dell'espressione regolare sulla *stringa*. Per ogni corrispondenza, l'iteratore restituisce un oggetto corrispondenza. Le corrispondenze vuote vengono incluse nel risultato, a meno che non tocchino l'inizio di un'altra corrispondenza. Nuovo nella versione 2.2.

`sub`(*pattern, repl, string*[, *count*])
Restituisce la stringa ottenuta sostituento le occorrenze di *pattern* (senza sovrapposizioni) nella parte più a sinistra della *string*, con le sostituzioni *repl*. Se il *pattern* non viene trovato, *string* viene restituita invariata. *repl* può essere una stringa o una funzione; se è una stringa, tutti i backslash di escape vengono elaborati. Ad esempio, '`\n`' viene convertito in un singolo carattere di fine riga, '`\r`' viene convertito in un linefeed, e così via. Gli escape sconosciuti, come '`\j`', vengono lasciati soli. I backreference, come '`\6`', vengono sostituiti con la sotto stringa corrispondente al gruppo 6 nel modello. Per esempio:

```
>>> re.sub(r'def\s+([a-zA-Z_][a-zA-Z_0-9]*)\s*\(\s*\):',
...        r'static PyObject*\npy_\1(void)\n{',
...        'def myfunc():')
'static PyObject*\npy_myfunc(void)\n{'
```

Se *repl* è una funzione, viene chiamata per ogni occorrenza (senza sovrapposizioni) del *pattern*. La funzione prende un singolo oggetto corrispondente come argomento, e restituisce la stringa sostitutiva. Per esempio:

```
>>> def dashrepl(matchobj):
....    if matchobj.group(0) == '-': return ' '
....    else: return '-'
>>> re.sub('-{1,2}', dashrepl, 'pro----gram-files')
'pro--gram files'
```

Il modello può essere una stringa o un oggetto RE; se necessitate di specificare opzioni delle espressioni regolari, dovete usare un oggetto RE o utilizzare i modificatori integrati nel modello; per esempio '`sub((?i)b+, x, bbbb BBBB)`' returns `'x x'`.

L'argomento facoltativo *count* indica il massimo numero di occorrenze del modello che devono venire sostituite; *count* deve essere un intero non negativo. Se omesso o zero, tutte le occorrenze verranno sostituite. Le corrispondenze vuote per il modello vengono sostituite solo quando non adiacenti ad una corrispondenza precedente, così '`sub('x*', '-', 'abc')`' restituisce `'-a-b-c-'`.

In aggiunta ai caratteri di escape e ai backreference come descritti sopra, '`\g<name>`' utilizzerà la sotto stringa che corrisposta dal gruppo chiamato '`name`', come definito dalla sintassi `(?P<name>...)`. '`\g<number>`' utilizza il corrispondente numero del gruppo; '`\g<2>`' è quindi equivalente a '`\2`', ma non è ambiguo in una sostituzione come '`\g<2>0`'. '`\20`' verrebbe interpretato come un riferimento al gruppo 20, non come un riferimento al gruppo 2 seguito dalla costante carattere '`0`'. Il backreference '`\g<0>`' sostituisce l'intera sotto stringa che corrisponde all'espressione regolare.

`subn`(*pattern, repl, string*[, *count*])
Effettua la stessa operazione di `sub()`, ma restituisce una tupla (*new_string*, *number_of_subst_made*).

escape(*string*)
Restituisce la *string* con tutti i caratteri non alfanumerici preceduti da un backslash (NdT: '`.`' diventa '`\\.`'); questo è utile se volete far corrispondere una stringa costante arbitraria che può avere metacaratteri delle espressioni regolari al suo interno.

exception error
Eccezione sollevata quando una stringa passata ad una delle funzioni descritte, non è un'espressione regolare valida (per esempio, potrebbe contenere parentesi non bilanciate), o quando si verifica qualche errore durante la compilazione o la corrispondenza. Non è mai un errore se una stringa non contiene corrispondenze per il modello.

4.2.4 Oggetti Espressioni Regolari

Gli oggetti compilati come espressioni regolari supportano i seguenti metodi e attributi:

match(*string*[, *pos*[, *endpos*]])
Se zero o più caratteri all'inizio di *string* corrispondono al modello dell'espressione regolare, restituisce una corrispondente istanza `MatchObject`. Restituisce `None` se la stringa non corrisponde al modello; notate che questo è differente rispetto ad una corrispondenza a lunghezza zero.

Note: Se volete localizzare una corrispondenza ovunque nella *string*, utilizzate `search()`

Il secondo argomento facoltativo *pos* fornisce un indice nella stringa per indicare dove deve iniziare la ricerca; il valore predefinito è `0`. Questo non è completamente equivalente ad affettare la stringa (NdT: to slicing nel testo originale); il carattere `'^'` nel modello corrisponde al reale inizio della stringa e alla posizione immediatamente successiva ad un fine riga, ma non necessariamente all'indice dove la ricerca inizia.

Il parametro facoltativo *endpos* limita la distanza massima in cui verrà effettuata la corrispondenza sulla stringa; si comporta come se la stringa avesse lunghezza pari a *endpos*, così solo i caratteri da *pos* ad *endpos* `- 1` verranno ricercati per una corrispondenza. Se *endpos* è minore di *pos*, nessuna corrispondenza verrà effettuata, altrimenti, se *rx* è compilata in un oggetto corrispondente ad un'espressione regolare, *rx*`.match(`*string*`, 0, 50)` è equivalente a *rx*`.match(`*string*`[:50], 0)`.

search(*string*[, *pos*[, *endpos*]])
Analizza la *string* cercando una posizione in cui il modello dell'espressione regolare generi una corrispondenza, e restituisce una corrispondente istanza `MatchObject`. Restituisce `None` se nessuna posizione della stringa corrisponde al modello; notate che questo è diverso dal trovare una corrispondenza di lunghezza zero in qualche punto della stringa.

I parametri facoltativi *pos* ed *endpos* hanno lo stesso significato indicato per il metodo `match()`.

split(*string*[, *maxsplit* `= 0`])
Identica alla funzione `split()`, utilizzando il modello compilato.

findall(*string*)
Identica alla funzione `findall()`, utilizzando il modello compilato.

finditer(*string*)
Identica alla funzione `finditer()`, utilizzando il modello compilato.

sub(*repl*, *string*[, *count* `= 0`])
Identica alla funzione `sub()`, utilizzando il modello compilato.

subn(*repl*, *string*[, *count* `= 0`])
Identica alla funzione `subn()`, utilizzando il modello compilato.

flags
L'argomento flags (NdT: opzioni) utilizzato quando l'oggetto corrispondente all'espressione è stato compilato, o `0` se nessuna opzione è stata impostata.

groupindex
Un dizionario che associa tutti i nomi dei gruppi simbolici definiti da `(?P<`*id*`>)` ai numeri identificativi dei gruppi. Il dizionario è vuoto se nel modello non vengono utilizzati gruppi simbolici.

pattern
La stringa corrispondente al modello da cui l'oggetto espressione regolare è stato compilato.

4.2.5 Gli oggetti di tipo Match (NdT: corrispondenza)

Le istanze `MatchObject` supportano i seguenti metodi e attributi:

expand(*template*)
Restituisce la stringa ottenuta effettuando le sostituzioni di backslash sulla stringa di maschera *template*, come fatto dal metodo `sub()`. Gli escape come '`\n`' vengono convertiti nei caratteri appropriati, e i backreference numerici ('`\1`', '`\2`') e nominali ('`\g<1>`', '`\g<nome>`') vengono sostituiti dal contenuto dei rispettivi gruppi.

group([*group1, ...*])
Restituisce uno o più sotto gruppi della corrispondenza. Se viene passato un solo argomento, il risultato è una stringa singola; se gli argomenti sono molteplici, il risultato è una tupla con un termine per argomento. Senza argomenti, il valore predefinito di *group1* è zero (l'intera corrispondenza viene restituita). Se un argomento *groupN* è zero, il valore corrispondente restituito è l'intera stringa corrispondente; se è compreso nell'intervallo [1..99], è la stringa corrispondente al gruppo indicato tra parentesi. Se il numero di un gruppo è negativo o maggiore del numero dei gruppi definito nel modello, viene sollevata un'eccezione `IndexError`. Se un gruppo viene contenuto in una parte del modello che non corrisponde, il risultato è `None`. Se un gruppo viene contenuto in una parte del modello che corrisponde molteplici volte, viene restituita l'ultima corrispondenza.

Se l'espressione regolare usa la sintassi `(?P<name>...)`, l'argomento *groupN* può anche essere una stringa che identifica il gruppo in base al suo nome. Se un argomento di tipo stringa non viene usato come nome di gruppo nel modello, viene sollevata un'eccezione di tipo `IndexError`.

Un esempio moderatamente complicato:

```
m = re.match(r"(?P<int>\d+)\.(\d*)", '3.14')
```

dopo la corrispondenza, `m.group(1)` è `'3'`, come lo è `m.group('int')`, e `m.group(2)` è `'14'`.

groups([*default*])
Restituisce una tupla contenente tutti i sotto gruppi della corrispondenza, da 1 fino all'ultimo gruppo presente nel modello, indipendentemente dal loro numero. Il parametro *default* viene usato per i gruppi che non partecipano alla corrispondenza; il valore predefinito è `None`. (Note di imcompatibilità: nella versione originale di Python 1.5, se la tupla fosse stata costituita da un solo elemento, sarebbe stata restituita una stringa. In questi casi, dalle versioni successive (dalla 1.5.1 in su), viene restituita una tupla di un solo elemento.)

groupdict([*default*])
Restituisce un dizionario contenente tutti i sotto gruppi *nominati* della corrispondenza, indicizzati dal nome del sotto gruppo. L'argomento *default* viene utilizzato per i gruppi che non partecipano alla corrispondenza; il valore predefinito è `None`.

start([*group*])
end([*group*])
Restituisce l'indice dell'inizio e della fine della sotto stringa corrispondente al *group*; il valore predefinito di *group* è zero (ad indicare l'intera sotto stringa). Restituisce `-1` se il *group* esiste, ma non contribuisce alla corrispondenza. Per un oggetto di corrispondenza *m* ed un gruppo *g* che ha contribuito alla corrispondenza, la sotto stringa ha corrisposto tramite il gruppo *g* (equivalente a *m*`.group(`*g*`)`) è:

```
m.string[m.start(g):m.end(g)]
```

Notate che `m.start(`*group*`)` uguaglierà `m.end(`*group*`)` se il gruppo *group* corrisponde ad una stringa epsilon (la stringa vuota). Per esempio, dopo *m* `= re.search('b(c?)', 'cba')`, *m*`.start(0)` è 1, *m*`.end(0)` è 2, *m*`.start(1)` ed *m*`.end(1)` sono entrambi 2, e *m*`.start(2)` solleva un'eccezione di tipo `IndexError`.

span([*group*])
Per un `MatchObject` *m*, restituisce la tupla di 2 elementi (*m*`.start(`*group*`)`, *m*`.end(`*group*`)`).

Notate che se il gruppo non contribuisce alla corrispondenza, questa è `(-1, -1)`. Ancora, il valore predefinito di *group* è zero.

`pos`
Il valore di *pos* passato a uno dei metodi `search()` o `match()` del `RegexObject`. Questo è l'indice nella stringa in cui l'automa dell'espressione regolare inizia a cercare.

`endpos`
Il valore di *endpos* passato ai metodi `search()` o `match()` del `RegexObject`. Questi è l'indice nella stringa oltre il quale l'automa dell'espressione regolare non andrà a cercare una corrispondenza.

`lastindex`
L'indice intero dell'ultimo gruppo che ha effettuato una corrisponcenza, o `None` se nessun gruppo ha trovato corrispondenza. Per esempio, le espressioni ⌜`(a)b`⌟, ⌜`((a)(b))`⌟ e ⌜`((ab))`⌟ hanno `lastindex == 1` se applicati alla stringa `'ab'`, mentre l'espressione ⌜`(a)(b)`⌟ avrà `lastindex == 2`, se applicata alla stessa stringa.

`lastgroup`
Il nome dell'ultimo gruppo che ha effettuato una corrispondenza, o `None` se il gruppo non ha un nome, o se nessun gruppo ha trovato corrispondenza.

`re`
L'oggetto corrispondente all'espressione regolare che ha prodotto, attraverso i metodi `match()` o `search()`, questa istanza di `MatchObject`.

`string`
La stringa passata a `match()` o `search()`.

4.2.6 Esempi

Simulare `scanf()`

Python attualmente non ha un equivalente di `scanf()`. . Le espressioni regolari sono generalmente più potenti, sebbene più prolisse, rispetto alle stringhe di formattazione della `scanf()`. La tabella seguente offre alcune tracce più o meno equivalenti tra i token della stringa di formattazione della `scanf()` e le espressioni regolari.

`scanf()` Token	Espressioni regolari
%c	⌜`.`⌟
%5c	⌜`.{5}`⌟
%d	⌜`[-+]?\d+`⌟
%e, %E, %f, %g	⌜`[-+]?(\d+(\.\d*)?|\d*\.\d+)([eE][-+]?\d+)?`⌟
%i	⌜`[-+]?(0[xX][\dA-Fa-f]+|0[0-7]*|\d+)`⌟
%o	⌜`0[0-7]*`⌟
%s	⌜`\S+`⌟
%u	⌜`\d+`⌟
%x, %X	⌜`0[xX][\dA-Fa-f]+`⌟

Per estrarre il nomefile e i numeri da una stringa come questa

```
/usr/sbin/sendmail - 0 errors, 4 warnings
```

potreste usare una stringa di formattazione per `scanf()` come

```
%s - %d errors, %d warnings
```

L'equivalente espressione regolare potrebbe essere

```
(\S+) - (\d+) errors, (\d+) warnings
```

Evitate la ricorsione

Se create espressioni regolari che richiedono all'automa di effettuare molte ricorsioni, potreste incappare in un'eccezione di tipo RuntimeError con un messaggio (`maximum recursion limit`) che vi avvisa del superamento del limite massimo per la ricorsione. Per esempio,

```
>>> import re
>>> s = 'Begin ' + 1000*'a very long string ' + 'end'
>>> re.match('Begin (\w| )*? end', s).end()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "/usr/local/lib/python2.3/sre.py", line 132, in match
    return _compile(pattern, flags).match(string)
RuntimeError: maximum recursion limit exceeded
```

Potete spesso ristrutturare la vostra espressione regolare per evitare la ricorsione.

A partire da Python 2.3, il semplice uso del modello `*?` è un caso speciale per evitare la ricorsione. Così, l'espressione regolare di cui sopra può evitare la ricorsione venendo rimaneggiata come `Begin [a-zA-Z0-9_ ]*?end`. Come ulteriore beneficio, questa espressione regolare verrà eseguita più velocemente rispetto al suo equivalente ricorsivo.

4.3 `struct` — Interpreta le stringhe come dati binari impacchettati

Questo modulo effettua conversioni tra i valori Python e le strutture C rappresentate come stringhe Python. Utilizza delle *stringhe di formattazione* (spiegate di seguito) come descrizioni compatte del layout delle strutture C, e della conversione intesa da e verso i valori Python. Può venire impiegato per gestire dati binari memoprizzati su file o provienienti da una connessione di rete, come esempio di sorgenti di dati, fra i tanti possibili.

Il modulo definisce le sequenti eccezioni e funzioni:

exception `error`
Eccezione sollevata in varie occasioni; l'argomento è una stringa che descrive cosa non ha funzionato.

`pack`(*fmt, v1, v2, ...*)
Restituisce una stringa contenente i valori *v1, v2, ...* impacchettati in accordo con la formattazione data. Gli argomenti devono corrispondere esattamente ai valori richiesti dalla formattazione.

`unpack`(*fmt, string*)
Spacchetta la stringa (presumibilmente impacchettata da `pack(`*fmt*`, ...)`) in accordo col formato indicato. Il risultato è una tupla, anche se contiene esattamente un elemento. La stringa deve contenere esattamente la quantità di dati richiesta dalla stringa di formattazione. (`len(`*string*`)` deve essere uguale a `calcsize(`*fmt*`)`).

`calcsize`(*fmt*)
Restituisce la dimensione della struttura (e quindi della stringa) corrispondente al formato indicato.

I formatti dei caratteri hanno il seguente significato; la conversione tra C e i valori Python deve essere ovvia dati i loro tipi:

Formato	Tipo C	Python	Note
'x'	pad byte	nessun valore	
'c'	char	stringa di lunghezza 1	
'b'	signed char	intero	
'B'	unsigned char	intero	
'h'	short	intero	
'H'	unsigned short	intero	
'i'	int	intero	
'I'	unsigned int	long	
'l'	long	intero	
'L'	unsigned long	long	
'q'	long long	long	(1)
'Q'	unsigned long long	long	(1)
'f'	float	float	
'd'	double	float	
's'	char[]	stringa	
'p'	char[]	stringa	
'P'	void *	intero	

Note:

(1) I codici di conversione 'q' e 'Q' sono disponibili in modo nativo solo se la piattaforma del compilatore C supporta il tipo C `long long`, o, su Windows, `__int64`. Sono sempre disponibili nei modi standard. Nuovo nella versione 2.2.

Un carattere di formattazione può essere preceduto da un intero indicante il conteggio delle ripetizioni. Per esempio, la stringa di formattazione `'4h'` è equivalente a `'hhhh'`.

I caratteri di spaziatura tra i formati vengono ignorati; un contatore e il suo formato non devono venire separati da caratteri di spaziatura.

Per il carattere di formattazione 's', il contatore viene interpretato come la dimensione della stringa, non come un contatore delle ripetizioni come negli altri caratteri di formattazione; per esempio, `'10s'` indica una singola stringa di 10 byte, mentre `'10c'` indica 10 caratteri. Per l'impacchettamento, la stringa viene troncata o riempita con byte nulli, in numero tale da poterla adattare. Per lo spacchettamento, la stringa risultante ha sempre esattamente lo specificato numero di byte. Come caso speciale, `'0s'` indica una singola stringa vuota (mentre `'0c'` indica 0 caratteri).

Il carattere di formattazione 'p' codifica una stringa Pascal, cioè una breve stringa a lunghezza variabile memorizzata in un numero fisso di byte. Il contatore indica il numero totale di byte immagazzinati. Il primo byte immagazzinato indica la lunghezza della stringa, o 255, in caso sia inferiore. I byte della stringa vengono a seguire. Se la stringa passata a `pack()` è troppo lunga (maggiore del contatore meno 1), solo i count-1 byte iniziali della stringa vengono immagazzinati. Se la stringa è più corta di count-1, viene riempita di byte nulli così che vengano utilizzati esattamente count byte. Notate che per `unpack()`, il carattere di formattazione 'p' consuma esattamente count byte, ma la stringa restituita non può contenere mai più di 255 caratteri.

Per i caratteri di formattazione 'I', 'L', 'q' e 'Q' il valore di ritorno è un intero long di Python.

Per il carattere di formattazione 'P', il valore di ritorno è un intero semplice o long di Python, in funzione della dimensione necessaria per contenere un puntatore, una volta che sia stato convertito ad un valore intero. Un puntatore `NULL` verrà sempre restituito come il valore intero Python `0`. Quando si impacchettano valori della dimensione di un puntatore, possono venire utilizzati interi semplici o long di Python. Per esempio, i processori Alpha e Merced usano puntatori a 64 bit, ovvero un intero long Python viene utilizzato per poter contenere il puntatore; le altre piattaforme utilizzano puntatori a 32 bit, e utilizzeranno un intero semplice Python.

In modo predefinito, i numeri C vengono rappresentati usando il formato e il byte order nativi, e correttamente allineati nell' utilizzo dei byte di riempimento se necessario (in accordo con le regole utilizzate dal compilatore C).

Alternativamente, il primo carattere della stringa di formattazione può venire utilizzato per indicare byte order, dimensione e allineamento dei dati impacchettati, in accordo con la tabella seguente:

Carattere	Byte order	Dimensione e allineamento
‘@’	nativo	nativo
‘=’	nativo	standard
‘<’	little-endian	standard
‘>’	big-endian	standard
‘!’	rete (= big-endian)	standard

Se il primo carattere non è uno di questi, viene assunto ‘@’.

Il byte order nativo può essere big-endian o little-endian, a seconda del sistema ospite. Per esempio, i processori Motorola e Sun sono big-endian; i processori Intel e DEC sono little-endian.

La dimensione e l’allineamento nativi vengono determinati utilizzando l’espressione `sizeof` del compilatore C. Questo viene sempre combinato con il byte order nativo.

La dimensione e l’allineamento standard sono come riportato di seguito: nessun allineamento viene richiesto per nessun tipo (perciò dovete utilizzare i byte di riempimento); `short` è di 2 byte; `int` e `long` sono di 4 byte; `long long` (`__int64` su Windows) è di 8 byte; `float` e `double` sono numeri in virgola mobile in formato IEEE rispettivamente di 32 e 64 bit.

Notate la differenza tra ‘@’ e ‘=’: entrambi utilizzano il byte order nativo, ma la dimensione e l’allineamento dell’ultimo sono standarizzati.

La forma `‘!’` è disponibile per quelle povere anime che si vantano di non ricordare se il byte order di rete è big-endian o little-endian.

Non c’è modo di indicare un byte order non nativo (forzare lo scambio dei byte); utilizzate la scelta appropriata di ‘<’ o ‘>’.

Il carattere di formattazione ‘P’ è disponibile solo nel formato byte order nativo (selezionato come predefinito o con il carattere di byte order ‘@’). Il carattere di byte order ‘=’ sceglie se usare l’ordinamento little-endian o big-endian in base al sistema ospite. Il modulo struct non interpreta questo come ordinamento nativo, perciò il formato ‘P’ non è disponibile.

Esempi (usando sempre byte order, dimensione e allineamento nativi, su una macchina big-endian):

```
>>> from struct import *
>>> pack('hhl', 1, 2, 3)
'\x00\x01\x00\x02\x00\x00\x00\x03'
>>> unpack('hhl', '\x00\x01\x00\x02\x00\x00\x00\x03')
(1, 2, 3)
>>> calcsize('hhl')
8
```

Suggerimento: per allineare la fine di una struttura ad un richiesto allineamento di tipo particolare, terminate il formato con il codice per quel tipo, con un contatore di ripetizioni di zero. Per esempio, il formato `'llh0l'` specifica due byte di riempimento alla fine, assumendo che i long vengano allineati a 4 byte. Questo funziona solo quando si utilizzano dimensione ed allineamento nativi; dimensione e allineamento standard non fanno rispettare nessun allineamento.

Vedete anche:

Modulo `array` (sezione 5.13):
Immagazzinamento di dati binari omogenei.

Modulo `xdrlib` (sezione 12.17):
Impacchettamento e spacchettamento di dati XDR.

4.4 `difflib` — Aiuti per calcolare le differenze

Nuovo nella versione 2.1.

class `SequenceMatcher`

Questa è una classe flessibile, per la comparazione delle coppie di sequenze di qualsiasi tipo, a condizione che gli elementi di sequenza siano chiavi valide per una tabella di hash (NdT: hashables). L'algoritmo fondamentale è più vecchio, e un poco più sofisticato, di quello pubblicato alla fine degli anni '80 da Ratcliff e Obershelp col nome iperbolico di "gestalt pattern matching". L'idea è quella di trovare la più lunga sotto sequenza corrispondente, che non contenga elementi "spazzatura" (l'algoritmo di Ratcliff e Obershelp non si occupa della spazzatura). La stessa idea viene poi applicata ricorsivamente alle parti della sequenza che si trovano a sinistra e a destra della sotto sequenza in corrispondenza. Questo non produce la minima alterazione sulle sequenze, ma tende a produrre corrispondenze che "appaiono giuste" agli osservatori umani.

Timing (NdT: Complessità temporale): L'algoritmo base di Ratcliff-Obershelp ha complessità temporale cubica nel caso peggiore, ma normalmente quadratica. `SequenceMatcher` è quadratica nel peggiore dei casi, ed ha una prevedibilità di comportamento che dipende in modo piuttosto complicato sul numero degli elementi che si trovano in comune tra le due sequenza; il caso migliore è lineare.

class `Differ`

Questa è una classe per confrontare sequenze di linee di testo, e generare delle differenze o variazioni leggibili da un umano. Differ usa `SequenceMatcher` sia per confrontare le sequenze di linee, sia per confrontare le sequenze di caratteri all'interno di linee simili (near-matching).

Ogni riga di variazioni di `Differ` inizia con un codice di due lettere:

Codice	Significato
`'- '`	linea unica nella sequenza 1
`'+ '`	linea unica nella sequenza 2
`'  '`	linea comune ad entrambe le sequenze
`'?  '`	linea non presente in nessuna delle sequenze in input

Le linee che iniziano con '`? `' tentano di guidare l'occhio alle differenze tra le linee e, dove non presenti, in entrambe le sequenze di input. Queste linee possono confondere se la sequenza contiene dei caratteri di tabulazione.

`context_diff`(*a, b*[, *fromfile*[, *tofile* [, *fromfiledate*[, *tofiledate*[, *n* [, *lineterm*]]]]]])

Confronta *a* e *b* (liste di stringhe); restituisce la variazione (un generatore che genera le righe variate) nel formato contestuale di diff.

Il formato contestuale di diff è un modo compatto di mostrare solo le linee che sono state modificate, più poche linee di contesto. I cambiamenti vengono mostrati in uno stile prima/dopo. Il numero di linee di contesto viene indicato da *n*, il cui valore predefinito è tre.

In modo predefinito, le linee di controllo di diff (quelle con `***` o `---`) vengono create con un carattere di fine riga nella loro parte finale. Questo è utile in modo tale che gli input creati da `file.readlines()` risultino in diff utilizzabili in `file.writelines()`, dato che entrambi gli input e gli output terminano con dei caratteri di fine riga.

Per gli input che non terminano con un carattere di fine riga, impostate l'argomento *lineterm* al valore "", così che l'output sia uniformemente privo del fine riga.

Il formato contestuale di diff normalmente ha un header (NdT: intestazione) che inizia con i nomi dei file e la data di modifica. Ognuno di questi può venire specificato utilizzando delle stringhe per *fromfile*, *tofile*, *fromfiledate* e *tofiledate*. Le date di modifica vengono normalmente espresse nel formato restituito da `time.ctime()`. Se non specificato, il valore predefinito delle stringhe corrisponde a spazi bianchi.

'Tools/scripts/diff.py' è un front-end da linea di comando a questa funzione.

Nuovo nella versione 2.3.

`get_close_matches`(*word, possibilities*[, *n*[, *cutoff*]])

Restituisce una lista delle migliori corrispondenze "abbastanza buone". *word* è una sequenza per cui vengono richieste corrispondenze ravvicinate (NdT: close matches), e *possibilities* è una lista di sequenze su cui vengono ricercate le corrispondenze *word* (tipicamente una lista di stringhe).

L'argomento facoltativo *n* (valore predefinito `3`) è il massimo numeri di corrispondenze ravvicinate da restituire; *n* dev'essere maggiore di `0`.

L'argomento facoltativo *cutoff* (valore predefinito `0.6`) è un numero in virgola mobile nell'intervallo [0,1]. Le possibilità che non raggiungono un punteggio di somiglianza con *word* almeno equivalente a questo valore vengono ignorate.

Le migliori (non più di *n*) corrispondenze tra le possibilità vengono restituite in una lista, ordinate in base al punteggio di similitudine, le più simili per prime.

```
>>> get_close_matches('appel', ['ape', 'apple', 'peach', 'puppy'])
['apple', 'ape']
>>> import keyword
>>> get_close_matches('wheel', keyword.kwlist)
['while']
>>> get_close_matches('apple', keyword.kwlist)
[]
>>> get_close_matches('accept', keyword.kwlist)
['except']
```

ndiff(*a*, *b*[, *linejunk*[, *charjunk*]])

Confronta *a* e *b* (liste di stringhe); restituisce la variazione nello stile `Differ` (un generatore che restituisce le righe variate).

Le chiavi facoltative *linejunk* e *charjunk* sono per le funzioni filtro (oppure `None`):

linejunk: Una funzione che accetta una singola stringa argomento, e che restituisce vero se la stringa è spazzatura, falso se non lo è. Il valore predefinito è `None`, a partire da Python 2.3. Prima di allora, il valore predefinito era la funzione a livello di modulo `IS_LINE_JUNK()`, che filtra le linee senza caratteri visibili, ad eccezione di un singolo carattere cancelletto ('#'). A partire da Python 2.3, la sottostante classe `SequenceMatcher` effettua un'analisi dinamica delle linee che sono così frequenti da costituire rumore, e questo generalmente lavora meglio del sistema adottato precedentemente alla versione 2.3.

charjunk: Una funzione che accetta un carattere (una stringa di lunghezza uno), e restituisce vero se il carattere è spazzatura falso altrimenti. Il valore predefinito è la funzione a livello di modulo `IS_CHARACTER_JUNK()`, che filtra tutti i caratteri di spaziatura (uno spazio o un tab; nota: è una cattiva idea includere i fine riga tra questi!)

'Tools/scripts/ndiff.py' è un front-end da linea di comando a questa funzione.

```
>>> diff = ndiff('one\ntwo\nthree\n'.splitlines(1),
...              'ore\ntree\nemu\n'.splitlines(1))
>>> print ''.join(diff),
- one
?  ^
+ ore
?  ^
- two
- three
?  -
+ tree
+ emu
```

restore(*sequence*, *which*)

Restituisce una delle due sequenze che hanno generato una variazione.

Fornendo una *sequence* prodotta da `Differ.compare()` o `ndiff()`, estrae le linee originate da file 1 o 2 (parametro *which*), rimuovendo i prefissi delle linee.

Esempio:

```
>>> diff = ndiff('one\ntwo\nthree\n'.splitlines(1),
...              'ore\ntree\nemu\n'.splitlines(1))
>>> diff = list(diff) # materializza in una lista la variazione generata
>>> print ''.join(restore(diff, 1)),
one
two
three
>>> print ''.join(restore(diff, 2)),
ore
tree
emu
```

unified_diff(*a*, *b*[, *fromfile*[, *tofile* [, *fromfiledate*[, *tofiledate*[, *n* [, *lineterm*]]]]]])
Confronta *a* e *b* (liste di stringhe); restituisce una variazione (un generatore che genera le righe variate) nel formato unificato di diff.

I diff unificati sono un modo compatto per mostrare solo le linee che sono cambiate, più alcune righe di contesto. I cambiamenti vengono mostrati in uno stile inline (invece che in blocchi separati prima/dopo). Il numero di righe di contesto viene indicato da *n*, il cui valore predefinito è tre.

In modo predefinito, le righe di controllo di diff (quelle con `---`, `+++` o `@@`) vengono create con un fine riga nella loro parte finale. Questo è utile perché gli input creati da `file.readlines()` risultino in diff utilizzabili per `file.writelines()`, dato che sia l'input che l'output terminano con un fine riga.

Per input che non sono terminati da una newline, impostate l'argomento *lineterm* a `''` così che l'output sia uniformemente privo del fine riga.

Il formato diff contestuale normalmente ha una intestazione indicante i nomi dei file e la data di modifica. Questi possono essere specificati utilizzando stringhe per *fromfile*, *tofile*, *fromfiledate* e *tofiledate*. La data di modifica viene normalmente espressa nel formato restituito da `time.ctime()`. Se non specificato, il valore delle stringhe corrisponde a spazi bianchi.

'Tools/scripts/diff.py' è un front-end da riga di comando per questa funzione.

Nuovo nella versione 2.3.

IS_LINE_JUNK(*line*)
Restituisce vero per le righe che possono venire ignorate. La riga *line* può venire ignorata se è vuota o contiene un singolo '#', altrimenti non può venire ignorata. Utilizzata come valore predefinito per il parametro *linejunk* in `ndiff()` prima di Python 2.3.

IS_CHARACTER_JUNK(*ch*)
Restituisce vero per i caratteri che possono venire ignorati. Il carattere *ch* può venire ignorato se *ch* è uno spazio o un tab, altrimenti non è ignorabile. Utilizzato come valore predefinito per il parametro *charjunk* in `ndiff()`.

Vedete anche:

Pattern Matching: The Gestalt Approach
(http://www.ddj.com/documents/s=1103/ddj8807c/)
Discussione di un algoritmo di similitudine, di John W. Ratcliff e D. E. Metzener. Pubblicato su *Dr. Dobb's Journal* nel luglio del 1988.

4.4.1 Oggetti SequenceMatcher

La classe `SequenceMatcher` ha questo costruttore:

class SequenceMatcher([*isjunk*[, *a*[, *b*]]])
L'argomento facoltativo *isjunk* deve essere `None` (il valore predefinito) o una funzione a singolo argomento, che accetti per argomento un elemento della sequenza, e restituisca vero se e solo se l'elemento è "spazzatura" e dovrebbe essere ignorato. Passare `None` per *b* è equivalente a passare `lambda x: 0`; vale a dire, nessun elemento viene ignorato. Per esempio, passate:

```
lambda x: x in " \t"
```

se state confrontando delle righe come sequenze di caratteri, e non volete che si sincronizzino sugli spazi bianchi o sui caratteri di tabulazione.

Gli argomenti facoltativi *a* e *b* sono sequenze da confrontare; il valore predefinito di entrambe è la stringa vuota. Gli elementi di entrambe le sequenze devono essere chiavi valide di una tabella di hash.

Gli oggetti `SequenceMatcher` hanno i sequenti metodi:

set_seqs(*a*, *b*)
Imposta le due sequenze che dovranno essere confrontate.

`SequenceMatcher` computa e memorizza informazioni dettagliate sulla seconda sequenza, così se volete confrontare una sequenza rispetto a molte sequenze, utilizzate `set_seq2()` per impostare la sequenza usata più di frequente una volta sola e chiamate `set_seq1()` ripetutamente, una per ognuna delle altre sequenze.

set_seq1(*a*)
Imposta la prima sequenze che deve essere confrontata. La seconda sequenza che dovrà essere confrontata non viene cambiata.

set_seq2(*b*)
Imposta la seconda sequenza che deve essere confrontata. La prima sequenza che dovrà essere confrontata non viene cambiata.

find_longest_match(*alo, ahi, blo, bhi*)
Trova il più lungo blocco corrispondente in *a*[*alo*:*ahi*] e *b*[*blo*:*bhi*].

Se *isjunk* viene omesso o è None, get_longest_match() restituisce (*i*, *j*, *k*) così che *a*[*i*:*i*+*k*] è uguale a *b*[*j*:*j*+*k*], dove *alo* <= *i* <= *i*+*k* <= *ahi* e *blo* <= *j* <= *j*+*k* <= *bhi*. Per tutti i (*i'*, *j'*, *k'*) che rispettano queste condizioni, anche le condizioni aggiuntive *k* >= *k'*, *i* <= *i'*, e se *i* == *i'*, *j* <= *j'* vengono soddisfatte. In altre parole, di tutti i blocchi massimali di corrispondenza, restituisce quello che inizia per primo in *a*, e di tutti i blocchi massimali di corrispondenza che iniziano per primi in *a*, restituisce quello che inizia per primo in *b*.

```
>>> s = SequenceMatcher(None, " abcd", "abcd abcd")
>>> s.find_longest_match(0, 5, 0, 9)
(0, 4, 5)
```

Se *isjunk* viene fornito, prima viene determinato il più lungo blocco di corrispondenza come indicato sopra, ma con con la restrizione aggiuntiva che nessun elemento spazzatura appaia nel blocco. Quindi il blocco viene esteso il più possibile facendo corrispondere (solo) elementi spazzatura da entrambi i lati. Perciò il blocco risultante non fa mai corrispondere sulla spazzatura ad eccezione di quando spazzatura identica si trova adiacente ad una corrispondenza interessante.

Ecco lo stesso esempio di prima, ma considerando gli spazi bianchi come spazzatura. Questo previene che ' abcd' corrisponda direttamente a ' abcd' alla fine della seconda sequenza. Invece solo 'abcd' può corrispondere e corrisponde l''abcd' più a sinistra nella seconda sequenza:

```
>>> s = SequenceMatcher(lambda x: x==" ", " abcd", "abcd abcd")
>>> s.find_longest_match(0, 5, 0, 9)
(1, 0, 4)
```

Se nessun blocco corrisponde, questo restituisce (*alo*, *blo*, 0).

get_matching_blocks()
Restituisce una lista di triple che descrive la sequenza corrispondente. Ogni tripla è nella forma (*i*, *j*, *n*) e significa che *a*[*i*:*i*+*n*] == *b*[*j*:*j*+*n*]. Le triple sono funzioni crescenti monotone in *i* and *j*.

L'ultima tripla è fittizia e ha il valore (len(*a*), len(*b*), 0). È l'unica tripla con *n* == 0.

```
>>> s = SequenceMatcher(None, "abxcd", "abcd")
>>> s.get_matching_blocks()
[(0, 0, 2), (3, 2, 2), (5, 4, 0)]
```

get_opcodes()
Restituisce una lista di tuple da 5 elementi che descrivono come trasformare *a* in *b*. Ogni tupla è nella forma (*tag*, *i1*, *i2*, *j1*, *j2*). La prima tupla ha *i1* == *j1* == 0 e le rimanenti tuple hanno *i1* uguale a *i2* della tupla precedente e, inoltre, *i1* uguale al precedente *i2*.

I valori *tag* sono stringhe, con i sequenti significati:

Valore	Significato
'replace'	*a*[*i1*:*i2*] dovrebbe venire sostituito da *b*[*j1*:*j2*].
'delete'	*a*[*i1*:*i2*] dovrebbe venire cancellato. Notate che *j1* == *j2* in questo caso.
'insert'	*b*[*j1*:*j2*] dovrebbe venire inserito in *a*[*i1*:*i1*]. Notate che *i1* == *i2* in questo caso.
'equal'	*a*[*i1*:*i2*] == *b*[*j1*:*j2*] (le sotto sequenze sono uguali).

Per esempio:

```
>>> a = "qabxcd"
>>> b = "abycdf"
>>> s = SequenceMatcher(None, a, b)
>>> for tag, i1, i2, j1, j2 in s.get_opcodes():
...    print ("%7s a[%d:%d] (%s) b[%d:%d] (%s)" %
...           (tag, i1, i2, a[i1:i2], j1, j2, b[j1:j2]))
 delete a[0:1] (q) b[0:0] ()
  equal a[1:3] (ab) b[0:2] (ab)
replace a[3:4] (x) b[2:3] (y)
  equal a[4:6] (cd) b[3:5] (cd)
 insert a[6:6] () b[5:6] (f)
```

`get_grouped_opcodes`([*n*])
: Restituisce un generatore di gruppi con fino a *n* righe di contesto.

  Partendo con il gruppo restituito da `get_opcodes()`, questo metodo lo divide in più piccoli gruppi di cambiamenti ed elimina gli intervalli coinvolti, che non hanno cambiamenti.

  I gruppi vengono restituiti nello stesso formato di `get_opcodes()`. Nuovo nella versione 2.3.

`ratio`()
: Restituisce una misura della similarità delle sequenze, come un numero in virgola mobile nell'intervallo [0,1].

  Dove T è il numero totale di elementi in entrambe le sequenze e M il numero delle corrispondenze, questo è 2.0*M / T. Notate che questo è 1.0 se le sequenze sono identiche, e `0.0` se non hanno niente in comune.

  Questo è costoso in termini computazionali se `get_matching_blocks()` o `get_opcodes()` non sono ancora stati chiamati, nel quale caso potreste voler provare prima `quick_ratio()` o `real_quick_ratio()`, per definire un limite superiore.

`quick_ratio`()
: Restituisce un limite superiore di `ratio()` in modo relativamente rapido.

  Questo non viene definito in altro modo che come un limite superiore su `ratio()`, ma è più rapido da calcolare.

`real_quick_ratio`()
: Restituisce un limite superiore di `ratio()` molto velocemente.

  Questo non viene definito in altro modo che come un limite superiore su `ratio()`, ma è più rapido da calcolare sia di `ratio()` che di `quick_ratio()`.

I tre metodi che restituiscono il rapporto di corrispondenza rispetto al totale dei caratteri, possono dare risultati differenti dovuti ai diversi livelli di approssimazione, anche se `quick_ratio()` e `real_quick_ratio()` sono sempre almeno grandi quanto `ratio()`.

```
>>> s = SequenceMatcher(None, "abcd", "bcde")
>>> s.ratio()
0.75
>>> s.quick_ratio()
0.75
>>> s.real_quick_ratio()
1.0
```

4.4.2 Esempi di SequenceMatcher

Questo esempio confronta due stringhe, considerando gli spazi bianchi come "spazzatura":

```
>>> s = SequenceMatcher(lambda x: x == " ",
...                     "private Thread currentThread;",
...                     "private volatile Thread currentThread;")
```

`ratio()` restituisce un numero in virgola mobile in [0,1], che misura la similarità delle sequenze. A lume di naso, un valore di `ratio()` superiore a 0.6 indica che le sequenze hanno corrispondenze di vicinanza:

```
>>> print round(s.ratio(), 3)
0.866
```

Se siete solo interessati nel sapere dove le sequenze corrispondono, è pratico `get_matching_blocks()`:

```
>>> for block in s.get_matching_blocks():
...     print "a[%d] and b[%d] match for %d elements" % block
a[0] and b[0] match for 8 elements
a[8] and b[17] match for 6 elements
a[14] and b[23] match for 15 elements
a[29] and b[38] match for 0 elements
```

Notate che l'ultima tupla restituita da `get_matching_blocks()` è sempre fittizia, `(len(`*a*`), len(`*b*`), 0)` e questo è il solo caso in cui l'ultimo elemento della tupla (numero di elementi che corrispondono) è `0`.

Se volete sapere come cambiare la prima sequenza nella seconda, utilizzate `get_opcodes()`:

```
>>> for opcode in s.get_opcodes():
...     print "%6s a[%d:%d] b[%d:%d]" % opcode
 equal a[0:8] b[0:8]
insert a[8:8] b[8:17]
 equal a[8:14] b[17:23]
 equal a[14:29] b[23:38]
```

Vedete anche la funzione `get_close_matches()` in questo modulo, che mostra come del semplice codice costruito su `SequenceMatcher` possa essere utilizzato per fare del lavoro molto utile.

4.4.3 Oggetti Differ

Notate che le variazioni generate da `Differ` non fanno caso alle differenze **minime**. Al contrario, le differenze minime sono spesso contro intuitive, poiché si sincronizzano ovunque possibile, a volte corrispondono accidentalmente a 100 pagine di distanza. Restringere i punti di sincronizzazione alle corrispondenze contigue permette di mantenere un minimo senso dell'orientamento, al costo occasionale di produrre un diff più lungo.

La classe `Differ` ha questo costruttore:

class `Differ`(\[*linejunk*\[, *charjunk*\]\])

I parametri facoltativi di tipo keyword *linejunk* e *charjunk* servono per le funzioni filtro (o sono `None`):

linejunk: Una funzione che accetta una singola stringa come argomento e restituisce vero se la stringa è spazzatura. Il valore predefinito è `None`, ad indicare che nessuna riga viene considerata spazzatura.

charjunk: Una funzione che accetta un singolo carattere come argomento (una stringa di lunghezza 1) e restituisce vero se il carattere è spazzatura. Il valore predefinito è `None`, ad indicare che nessun carattere viene considerato spazzatura.

Gli oggetti di tipo `Differ` vengono utilizzati (generati da variazioni) tramite un singolo metodo:

`compare`(*a*, *b*)

Confronta due sequenze di righe e genera la variazione (una sequenza di righe).

Ogni sequenza deve contenere una stringa individuale di una sola riga, terminata da un fine riga. Queste sequenze possono essere ottenute dal metodo `readlines()` da oggetti di tipo file. La variazione generata inoltre consiste di stringhe terminate da dei caratteri di fine riga, pronte per venire stampate così come sono dal metodo `writelines()` di un oggetto di tipo file.

4.4.4 Esempio di Differ

Questo esempio confronta due testi. Prima impostiamo i testi, sequenze di stringhe individuali composte da una singola riga, e terminanti con un fine riga (queste sequenze possono anche venire ottenute dal metodo `readlines()` degli oggetti di tipo file):

```
>>> text1 = '''  1. Beautiful is better than ugly.
...   2. Explicit is better than implicit.
...   3. Simple is better than complex.
...   4. Complex is better than complicated.
... '''.splitlines(1)
>>> len(text1)
4
>>> text1[0][-1]
'\n'
>>> text2 = '''  1. Beautiful is better than ugly.
...   3.   Simple is better than complex.
...   4. Complicated is better than complex.
...   5. Flat is better than nested.
... '''.splitlines(1)
```

Successivamente istanziamo un oggetto di tipo Differ:

```
>>> d = Differ()
```

Notate che quando istanziamo un oggetto di tipo `Differ`, possiamo passare delle funzioni per filtrare righe e caratteri "spazzatura". Vedete il costruttore `Differ()` per i dettagli.

Infine, confrontiamo le due stringhe:

```
>>> result = list(d.compare(text1, text2))
```

`result` è una lista di stringhe, perciò stampiamole graziosamente:

```
>>> from pprint import pprint
>>> pprint(result)
['    1. Beautiful is better than ugly.\n',
 '-   2. Explicit is better than implicit.\n',
 '-   3. Simple is better than complex.\n',
 '+   3.   Simple is better than complex.\n',
 '?     ++                                \n',
 '-   4. Complex is better than complicated.\n',
 '?            ^                     ---- ^  \n',
 '+   4. Complicated is better than complex.\n',
 '?           ++++ ^                      ^  \n',
 '+   5. Flat is better than nested.\n']
```

Come una singola stringa di più righe, appare così:

```
>>> import sys
>>> sys.stdout.writelines(result)
    1. Beautiful is better than ugly.
-   2. Explicit is better than implicit.
-   3. Simple is better than complex.
+   3.   Simple is better than complex.
?     ++
-   4. Complex is better than complicated.
?           ^                     ---- ^
+   4. Complicated is better than complex.
?          ++++ ^                      ^
+   5. Flat is better than nested.
```

4.5 `fpformat` — Conversioni dei numeri in virgola mobile

Il modulo `fpformat` definisce le funzioni per trattare la rappresentazione dei numeri in virgola mobile in stile Python, puro al 100%. **Note:** Questo modulo non è necessario: tutto quello che viene qui discusso può essere fatto attraverso l'operatore di interpolazione di stringhe %.

Il modulo `fpformat` definisce le seguenti funzioni e un'eccezione:

`fix`(*x, digs*)
Formatta *x* come `[-]ddd.ddd` con *digs* cifre dopo il punto e almeno una cifra prima. Se *digs* `<= 0`, il punto decimale viene soppresso.

x può essere sia un numero che una stringa che rappresenti un numero. *digs* è un intero.

Il valore restituito è una stringa.

`sci`(*x, digs*)
Formatta *x* come `[-]d.dddE[+-]ddd` con *digs* cifre dopo il punto ed esattamente una cifra prima. Se *digs* `<= 0`, viene mantenuta una cifra, e il punto viene soppresso.

x può essere sia un numero reale sia una stringa che rappresenti un numero reale. *digs* è un intero.

Il valore restituito è una stringa.

exception `NotANumber`
Eccezione sollevata quando una stringa passata a `fix()` o `sci()` come parametro *x*fix() non rappresenta un numero. Questa è una classe derivata di `ValueError` quando le eccezioni standard sono stringhe. Il valore dell'eccezione è la stringa formattata impropriamente che ha provocato il sollevamento dell'eccezione.

Esempio:

```
>>> import fpformat
>>> fpformat.fix(1.23, 1)
'1.2'
```

4.6 `StringIO` — Legge e scrive stringhe come file

Questo modulo implementa una classe simile a file, `StringIO`, che legge e scrive un buffer stringa (anche conosciuto come *file di memoria*). Vedete la descrizione degli oggetti file per le operazioni (sezione 2.3.9).

class `StringIO`([*buffer*])
Quando un oggetto `StringIO` viene creato, può venire inizializzato ad una stringa esistente, passando la stringa al costruttore. Se nessuna stringa viene fornita, `StringIO` inizierà vuoto.

L'oggetto `StringIO` accetta sia stringhe Unicode che stringhe a 8 bit, ma mischiare i due tipi può richiedere qualche accorgimento. Se entrambi i tipi di stringa vengono utilizzati, le stringhe a 8 bit che non possono venire interpretate come caratteri ASCII a 7 bit (che usano l'ottavo bit) causeranno il sollevamento dell'eccezione `UnicodeError` nel momento in cui verrà chiamato `getvalue()`.

I sequenti metodi degli oggetti `StringIO` richiedono una menzione speciale:

`getvalue``()`
Recupera l'intero contenuto del "file" in qualunque momento prima che il metodo `close()` degli oggetti `StringIO` venga chiamato. Vedete le note sopra per informazioni sul fatto di poter mescolare stringhe a 8 bit e Unicode; questa mescolanza può provocare il sollevamento di `UnicodeError`.

`close``()`
Libera il buffer di memoria.

4.7 `cStringIO` — Versione più veloce di `StringIO`

Il modulo `cStringIO` fornisce un'interfaccia simile a quella del modulo `StringIO`. Un uso intensivo di oggetti di tipo `StringIO.StringIO` può essere reso più efficiente utilizzando al suo posto la funzione `StringIO()` di questo modulo.

Poiché questo modulo fornisce una funzione factory che restituisce oggetti di tipo built-in, non avete nessun modo per costruirne una propria versione utilizzando le classi derivati. Utilizzate il modulo originale `StringIO`, in quel caso.

Diversamente dai file di memoria implementati dal modulo `StringIO`, quelli forniti da questo modulo non sono in grado di accettare stringhe Unicode che non possono venire codificate come stringhe ASCII semplici.

Un'altra differenza dal modulo `StringIO` è che chiamando `StringIO()` con un parametro di tipo stringa, crea un oggetto di sola lettura. Al contrario di un oggetto creato senza un parametro di tipo stringa, non ha un medoto write. Questi oggetti non sono generalmente visibili. Appaiono nei traceback come `StringI` e `StringO`.

Vengono forniti i seguenti oggetti dato:

`InputType`
L'oggetto tipo degli oggetti creati chiamando `StringIO` con un parametro di tipo stringa.

`OutputType`
L'oggetto tipo degli oggetti restituiti chiamando `StringIO` senza parametri.

Esiste anche un'API C a questo modulo; fate riferimento ai sorgenti del modulo per ulteriori informazioni.

4.8 `textwrap` — Involucro e riempimento del testo

Nuovo nella versione 2.3.

Il modulo `textwrap` fornisce due utili funzioni, `wrap()` e `fill()`, di `TextWrapper`, la classe che effettua tutto il lavoro, e una funzione di utilità `dedent()`. Se dovete solo creare un involucro (NdT: wrap) o riempire (NdT: fill) una o due stringhe, le prime due funzioni dovrebbero essere sufficienti; altrimenti, dovreste usare un'istanza di `TextWrapper`.

`wrap`(*text*[, *width*[, ...]])
Crea un involucro per il singolo paragrafo in *text* (una stringa) così che ogni riga sia lunga al massimo *width* caratteri. Restituisce una lista di righe di output, senza i caratteri di fine riga.

Gli argomenti opzionali di tipo keyword corrispondono agli attributi dell'istanza di `TextWrapper`, documentati più avanti. Il valore predefinito per *width* è `70`.

`fill`(*text*[, *width*[, ...]])
Crea un involucro per il singolo paragrafo in *text*, e restituisce una singola stringa contenente il paragrafo nel nuovo formato. `fill()` è una scorciatoia per

```
"\n".join(wrap(text, ...))
```

In particolare, `fill()` accetta esattamente gli stessi argomenti di tipo keyword di `wrap()`.

Sia `wrap()` che `fill()` lavorano creando un'istanza di `TextWrapper`, e chiamando un singolo metodo su di essa. Quell'istanza non viene riutilizzata, così per applicazioni che creano involucri o riempiono molte stringhe di testo, potrebbe essere più efficace creare il proprio oggetto `TextWrapper`.

Una funzione ulteriore di utilità, `dedent()`, viene fornita per rimuovere l'indentazione dalle stringhe che hanno degli spazi bianchi non desiderati alla sinistra del testo.

`dedent`(*text*)

Rimuove tutti gli spazi bianchi che possono venire uniformemente rimossi alla sinistra di ogni riga di *text*.

Questa viene tipicamente usata per per rendere le stringhe con il sistema della quotatura tripla, allineate con il margine sinistro dello schermo (o altro dispositivo di output), mantenendo comunque la forma indentata nel codice sorgente.

Per esempio:

```
def test():
    # Termina la prima riga con \ per evitare la riga vuota!
    s = '''\
    hello
      world
    '''
    print repr(s)          # stampa '    hello\n      world\n    '
    print repr(dedent(s))  # stampa 'hello\n  world\n'
```

class `TextWrapper`(...)

Il costruttore `TextWrapper` accetta un numero di argomenti opzionali di tipo keyword. Ogni argomento corrisponde ad un attributo dell'istanza, cosi' per esempio

```
wrapper = TextWrapper(initial_indent="* ")
```

è equivalente a

```
wrapper = TextWrapper()
wrapper.initial_indent = "* "
```

Potete riutilizzare lo stesso oggetto `TextWrapper` più volte, e potete cambiare qualsiasi argomento facoltativo attraverso l'assegnazione diretta degli attributi dell'istanza tra i vari utilizzi.

Gli attributi dell'istanza di `TextWrapper` (e gli argomenti di tipo keyword del costruttore) sono i seguenti:

`width`

(valore predefinito: `70`) La massima lunghezza delle righe determinate dall'involucro creato. A meno che nel testo di input non vi siano parole individuali più lunghe di `width`, `TextWrapper` garantisce che non vengano mandate in output delle righe più lunghe di `width` caratteri.

`expand_tabs`

(valore predefinito: `True`) Se vero, tutti i caratteri di tabulazione nel testo verranno espansi in spazi, utilizzando il metodo `expand_tabs()` di *text*.

`replace_whitespace`

(valore predefinito: `True`) Se vero, ogni carattere di spaziatura (come definito da `string.whitespace`) rimanente dopo l'espansione dei caratteri di tabulazione, verrà sostituito da un singolo spazio. **Note:** Se `expand_tabs` è falso e `replace_whitespace` è vero, ogni carattere di tabulazione verrà sostituito da un singolo spazio, che *non* è la stessa cosa dell'espansione delle tabulazioni.

`initial_indent`

(valore predefinito: `''`) La stringa che viene preposta alla prima riga dell'output realizzato. Viene inclusa nel conteggio della lunghezza della prima riga.

subsequent_indent
(valore predefinito: ″) La stringa che viene preposta a tutte le righe dell'output realizzato, ad eccezione della prima. Viene inclusa nel conteggio della lunghezza di tutte le righe, ad eccezione della prima.

fix_sentence_endings
(valore predefinito: `False`) Se vero, `TextWrapper` tenta di riconoscere la fine delle frasi, e si assicura che le frasi vengano sempre separate da esattamente due spazi. Questo è in genere quanto si desidera per del testo in un font monospaziato. Comunque, l'algoritmo di riconoscimento è imperfetto: assume che la fine di ogni frase consista di una lettera minuscola seguita da un carattere tra '`.`', '`!`' o '`?`', eventualmente seguito da '' o ''', seguito da uno spazio. Un problema tipico di questo algoritmo, è che non è in grado di riconoscere la differenza tra "Dr." in

```
[...] Dr. Frankenstein's monster [...]
```

e "Spot." in

```
[...] See Spot. See Spot run [...]
```

Il valore predefinito di `fix_sentence_endings` è falso.

Poiché l'algoritmo di riconoscimento della frasi si affida a `string.lowercase` per la definizione di "lettere minuscole" e utilizza la convenzione di utilizzare due spazi dopo un punto per separare le frasi nella stessa riga, è specifico per i testi in lingua inglese.

break_long_words
(valore predefinito: `True`) Se vero, le parole più lunghe di `width` verranno spezzate per assicurare che nessuna riga sia più lunga di `width`. Se falso, le parole lunghe non verranno spezzate, e alcune righe potranno risultare più lunghe di `width`. (Le parole lunghe verranno poste in una riga a sè stante, per minimizzare la quantità eccedente rispetto a `width`.)

`TextWrapper` fornisce anche due metodi pubblici, analoghi alle funzioni di convenienza a livello di modulo:

wrap(*text*)
Crea un involucro per il singolo paragrafo in *text* (una stringa) così che ogni riga sia lunga al massimo `width` caratteri. Tutte le opzioni dell'involucro vengono prese dagli attributi dell'istanza di `TextWrapper`. Restituisce una lista di righe in output, senza i caratteri di fine riga.

fill(*text*)
Crea un involucro per il singolo paragrafo in *text* e restituisce una singola stringa contenente il paragrafo realizzato.

4.9 codecs — Registro dei codec e classi base

Questo modulo definisce le classi di base per i codec standard Python (encoder e decoder) e fornisce l'accesso al registro delle codifiche interno a Python, che gestisce le codifiche stesse e i processi di controllo della gestione degli errori.

Definisce le seguenti funzioni:

register(*search_function*)
Registra una funzione di ricerca codec. Le funzioni di ricerca si aspettano di ricevere un argomento, il nome da decodificare in lettere tutte minuscole, e restituisce una tupla di funzioni (*encoder*, *decoder*, *stream_reader*, *stream_writer*) prendendo i seguenti argomenti:

encoder e *decoder*: questi devono essere funzioni o metodi che possiedano la stessa interfaccia dei metodi `encode()`/`decode()` delle istanze di Codec (vedete Codec Interface). Le funzioni e i metodi si aspettano di lavorare in modalità stateless (NdT: senza stato).

stream_reader e *stream_writer*: questi hanno le funzioni factory che forniscono le seguenti interfacce:

`factory(`*stream*`, `*errors*`='strict')`

Le funzioni factory devono restituire oggetti che forniscano le interfacce definite rispettivamente dalle classi base `StreamWriter` e `StreamReader`. I codec dello stream (NdT: flusso) possono mantenere lo stato.

Valori possibili per gli errori sono `'strict'` (solleva un'eccezione in caso di errore di codifica), `'replace'` (sostituisce dati malformati con un accettabile marcatore di sostituzione, come un '?'), `'ignore'` (ignora dati malformati e continua senza ulteriori notifiche), `'xmlcharrefreplace'` (sostituisce con un appropriato riferimento a carattere XML (solo per la codifica)) e `'backslashreplace'` (sostituisce con sequenze di escape indicate dal backslash (solo per la codifica)) qualsiasi altro errore di gestione, definito attraverso `register_error()`.

Nel caso in cui una funzione di ricerca non trovi una data codifica, dovrebbe restituire `None`.

`lookup`(*encoding*)
Cerca una tupla di codec nel registro delle codifiche di Python, e restituisce la tupla della funzione, come indicato precedentemente.

Le decodifiche vengono prima cercate nella cache del registro. Se non trovate, viene controllata la lista delle funzioni di ricerca registrate. Se nessuna tupla di codec viene trovata, allora viene sollevata un'eccezione `LookupError`. Altrimenti, la tupla del codec viene inserita nella cache e restituita al chiamante.

Per semplificare l'accesso alle diverse codifiche, il modulo fornisce queste ulteriori funzioni che utilizzano `lookup()` per la loro ricerca:

`getencoder`(*encoding*)
Cerca un codec per il dato *encoding*, e restituisce la sua funzione di codifica.

Solleva un'eccezione `LookupError` nel caso in cui non possa trovare la codifica.

`getdecoder`(*encoding*)
Cerca un codec per il dato *encoding*, e restituisce la sua funzione di decodifica.

Solleva un'eccezione `LookupError` nel caso in cui non possa trovare la codifica.

`getreader`(*encoding*)
Cerca un codec per il dato *encoding*, e restituisce la sua classe StreamReader o la funzione factory.

Solleva un'eccezione `LookupError` nel caso in cui non possa trovare la codifica.

`getwriter`(*encoding*)
Cerca un codec per il dato *encoding*, e restituisce la sua classe StreamWriter o la funzione factory.

Solleva un'eccezione `LookupError` nel caso in cui non possa trovare la codifica.

`register_error`(*name, error_handler*)
Registra la funzione di gestione degli errori *error_handler* sotto il nome *name*. *error_handler* verrà chiamato durante la codifica e decodifica nel caso di un errore, quando *name* viene specificato come parametro d'errore.

Per la codifica, *error_handler* verrà chiamato con un'istanza `UnicodeEncodeError`, che contiene informazioni circa la posizione dell'errore. Il gestore dell'errore deve sollevare questa o una differente eccezione, o restituire una tupla con una sostituzione per la parte non codificabile dell'input, ed una posizione da cui la codifica possa continuare. Il codificatore codificherà la sostituzione, e continuerà la codifica dell'input originale alla posizione specificata. Valori di posizione negativi verranno considerati come relativi alla fine della stringa di input. Se la posizione risultante è fuori dai limiti, verrà sollevata un'eccezione `IndexError`.

Decodifica e traduzione lavorano in modo simile, eccetto per il fatto che `UnicodeDecodeError` o `UnicodeTranslateError` verranno passati al gestore, e che la sostituzione fornita dal gestore degli errori verrà inserita direttamente nell'output.

`lookup_error`(*name*)
Restituisce il gestore dell'errore precedentemente registrato sotto il nome *name*.

Solleva un'eccezione `LookupError` nel caso in cui non possa trovare il gestore.

`strict_errors`(*exception*)
Implementa una gestione dell'errore `strict` (NdT: rigorosa).

`replace_errors`(*exception*)
Implementa la gestione dell'errore `replace`.

`ignore_errors`(*exception*)
Implementa la gestione dell'errore `ignore`.

xmlcharrefreplace_errors_errors(*exception*)
Implementa la gestione dell'errore `xmlcharrefreplace`.

backslashreplace_errors_errors(*exception*)
Implementa la gestione dell'errore `backslashreplace`.

Per semplificare il lavoro con i file o i flussi codificati, il modulo definisce anche queste funzioni di utilità:

open(*filename, mode*[, *encoding*[, *errors*[, *buffering*]]])
Apre un file codificato usando il modo *mode* indicato, e ne restituisce un involucro per la versione in grado di fornire codifica e decodifica trasparenti.

Note: La versione realizzata con l'involucro accetterà solo il formato oggetto definito dai codec, per esempio, oggetti Unicode per la maggior parte dei codec built-in. Anche il risultato è dipendente dal codec e normalmente sarà Unicode.

encoding specifica la codifica che verrà usata per i file.

errors può venire indicata per definire la gestione degli errori. Predefinitamente viene considerata `'strict'`, che solleverà un'eccezione `ValueError` nel caso in cui si verifichino errori di codifica.

buffering ha lo stesso significato della funzione built-in `open()`. La sua impostazione predefinita è una riga bufferizzata.

EncodedFile(*file, input*[, *output*[, *errors*]])
Restituisce una versione realizzata mediante un involucro del file, che fornisce in modo trasparente la traduzione di codifica.

Stringhe scritte nel file realizzato mediante involucro vengono interpretate in accordo all'*input* di codifica indicato, e quindi scritte nel file originale come stringhe utilizzando la codifica *output*. La codifica intermedia sarà tipicamente Unicode, ma questo dipende dai codec specificati.

Se *output* non viene indicato, il suo valore predefinito sarà *input*.

errors può venire indicato per definire la gestione degli errori. Il suo valore predefinito è `'strict'`, che solleva un'eccezione `ValueError` nel caso in cui venga rilevato un errore di codifica.

Il modulo fornisce anche le seguenti costanti, utili per leggere e scrivere su file dipendenti dalla piattaforma:

BOM
BOM_BE
BOM_LE
BOM_UTF8
BOM_UTF16
BOM_UTF16_BE
BOM_UTF16_LE
BOM_UTF32
BOM_UTF32_BE
BOM_UTF32_LE
Queste costanti definiscono diverse codifiche del byte order mark (BOM) Unicode, usato nei flussi di dati in UTF-16 e UTF-32 per indicare il byte order utilizzato nello stream o nel file, ed in UTF-8 come firma Unicode. `BOM_UTF16` è sia `BOM_UTF16_BE` che `BOM_UTF16_LE`, a seconda del byte order nativo della piattaforma, `BOM` è un alias per `BOM_UTF16`, `BOM_LE` per `BOM_UTF16_LE` e `BOM_BE` per `BOM_UTF16_BE`. Gli altri rappresentano il BOM nelle codifiche UTF-8 e UTF-32.

4.9.1 Classi base di Codec

Il modulo `codecs` definisce un insieme di classi base che definiscono l'interfaccia, e che possono anche venire usate per scrivere rapidamente i propri codecs da utilizzare con Python.

Ogni codec deve definire quattro interfacce perché sia usabile come codec in Python: stateless encoder (NdT: codificatore senza stato), stateless decoder (NdT: decodificatore senza stato), stream reader (NdT: lettore di flusso) e stream writer (NdT: scrittore di flusso). Gli stream reader e writer di solito riutilizzano gli stateless encoder/decoder per implementare i protocolli file.

La classe `Codec` definisce l'interfaccia per gli encoders/decoders stateless.

Per semplificare e standardizzare la gestione degli errori, i metodi `encode()` e `decode()` possono implementare differenti schemi di gestione dell'errore, fornendo l'argomento stringa *errors*. I seguenti valori stringa vengono definiti ed implementati da tutti i codec standard di Python:

Valore	Significato
`'strict'`	Solleva un'eccezione `UnicodeError` (o una classe derivata); questo è il comportamento predefi
`'ignore'`	Ignora il carattere e continua con il successivo.
`'replace'`	Sostituisce con un opportuno carattere di rimpiazzo. Python userà il CARATTERE SOSTITUTIV
`'xmlcharrefreplace'`	Sostituisce con l'appropriato riferimento a carattere XML (solo per la codifica).
`'backslashreplace'`	Sostituisce con una sequenza di escape protetto (solo per la codifica).

L'insieme dei valori ammessi può venire esteso tramite `register_error`.

Oggetti Codec

La classe `Codec` definisce questi metodi, che a loro volta definiscono le interfacce di funzione negli encoder/decoder stateless:

`encode`(*input*[, *errors*])

Codifica l'oggetto *input* e restituisce una tupla (oggetto output, lunghezza consumata). Poichè i codec non vengono limitati ad essere usati con Unicode, la codifica converte un oggetto Unicode in una stringa semplice, usando un particolare insieme di caratteri di codifica (per esempio: `cp1252` o `iso-8859-1`).

errors definisce il metodo di gestione degli errori da utilizzare. Predefinita è una gestione `'strict'`.

Il metodo può non memorizzare lo stato nell'istanza `Codec`. Usate `StreamCodec` per i codec che devono mantenere lo stato per eseguire una codifica/decodifica efficiente.

Il codificatore deve essere in grado di gestire un input di lunghezza zero, e restituire in questa situazione, un oggetto vuoto come oggetto di uscita.

`decode`(*input*[, *errors*])

Decodifica l'oggetto *input* e restituisce una tupla (oggetto output, lunghezza consumata). In un contesto Unicode, la decodifica converte una stringa semplice, codificata con un particolare insieme di caratteri, codificandola in un oggetto Unicode.

input deve essere un oggetto che fornisca il segmento buffer `bf_getreadbuf`. Le stringhe Python, gli oggetti buffer ed i file mappati in memoria sono esempi di oggetti che forniscono questo segmento.

errors definisce il metodo di gestione degli errori da utilizzare. Predefinita è una gestione `'strict'`.

Il metodo può non memorizzare lo stato nell'istanza `Codec`. Utilizzate `StreamCodec` per le codifiche che debbano mantenere lo stato per poter ottenere una codifica/decodifica efficiente.

Il decodificatore deve essere in grado di gestire input di lunghezza zero, e restituire in questa situazione, un oggetto vuoto come tipo di oggetto risultante.

Le classi `StreamWriter` e `StreamReader` forniscono un'interfaccia generica di lavoro che può venire utilizzata per implementare, molto facilmente, dei nuovi moduli derivati di codifica. Vedete `encodings.utf_8` per un esempio.

Oggetti StreamWriter

La classe `StreamWriter` è una classe derivata di `Codec` e definisce i seguenti metodi che ogni stream writer deve definire per essere compatibile con il registro dei codec Python.

class `StreamWriter`(*stream*[, *errors*])

Costruttore per un'istanza `StreamWriter`.

Tutti i writers di flusso devono fornire questa interfaccia del costruttore. Sono liberi di aggiungere degli argomenti chiave addizionali, ma solo quelli qui definiti vengono usati dal registro Python dei codec.

stream deve essere un oggetto simile a file aperto in modalità di scrittura dati (modo binario).

La classe `StreamWriter` può implementare differenti schemi di gestione dell'errore, fornendo l'argomento chiave *errors*. Questi sono i parametri definiti:

- `'strict'` Solleva un'eccezione `ValueError` (o una classe derivata); questo è il predefinito.
- `'ignore'` Ignora il carattere e continua con il successivo.
- `'replace'` Sostituisce con un opportuno carattere di rimpiazzo.
- `'xmlcharrefreplace'` Sostituisce con un riferimento di carattere XML appropriato
- `'backslashreplace'` Sostituisce con una sequenza di escape protetta.

L'argomento *errors* verrà assegnato ad un attributo con lo stesso nome. Assegnamenti a questo attributo rendono possibile il cambio tra differenti strategie di gestione dell'errore durante la vita dell'oggetto `StreamWriter`.

L'insieme dei valori consentiti per l'argomento *errors* può venire esteso con `register_error()`.

`write`(*object*)
Scrive il contenuto codificato dell'oggetto nello stream.

`writelines`(*list*)
Scrive la lista concatenata di stringhe nello stream (possibilmente riutilizzando il metodo `write()`).

`reset`()
Svuota e azzera i buffer del codec usati per il mantenimento dello stato.

La chiamata a questo metodo dovrebbe assicurare che i dati in uscita vengano messi in uno stato chiaro, che consenta l'aggiunta di nuovi dati senza dover ricontrollare l'intero stream per recuperare lo stato.

In aggiunta ai metodi già indicati, `StreamWriter` deve anche ereditare tutti gli altri metodi e attributi dello stream sottostante.

Oggetti StreamReader

La classe `StreamReader` è una classe derivata di `Codec` e definisce i seguenti metodi che ogni lettore di flusso deve definire affinché sia compatibile con il registro dei codec di Python.

class `StreamReader`(*stream*[, *errors*])
Costruttore per l'istanza `StreamReader`.

Tutti i lettori di flusso devono fornire questa interfaccia per il costruttore. Sono liberi di aggiungere ulteriori argomenti chiave, ma solo quelli qui definiti vengono usati dal registro Python dei codec.

stream deve essere un oggetto simile a file, aperto in modalità lettura dati (modo binario).

La classe `StreamWriter` può implementare differenti schemi di gestione degli errori, fornendo l'argomento chiave *errors*. Questi sono i parametri definiti:

- `'strict'` Solleva un'eccezione `ValueError` (o una classe derivata); questo è il predefinito.
- `'ignore'` Ignora il carattere e continua con il successivo.
- `'replace'` Sostituisce con un opportuno carattere di rimpiazzo.

L'argomento *errors* verrà assegnato ad un attributo con lo stesso nome. Assegnazioni a questo attributo rendono possibile il cambio con differenti strategie di gestione dell'errore durante la vita dell'oggetto `StreamWriter`.

L'insieme dei valori consentiti per l'argomento *errors* può venire esteso con `register_error()`.

`read`([*size*])
Decodifica i dati dallo flusso, e restituisce l'oggetto risultante.

size indica approssimativamente il numero massimo di bytes da leggere dal flusso per la decodifica. Il decodificatore può modificare questa impostazione in base alle necessità. Il valore predefinito -1 indica di leggere e decodificare il più possibile. *size* viene utilizzato per prevenire la decodifica di file troppo grandi in un solo passaggio.

Il metodo dovrebbe usare un'attenta strategia di lettura, cioè dovrebbe la massima quantità possibile di dati, consentita dala definizione della codifica e dalla misura indicata, per esempio se sono disponibili sul flusso delle estensioni facoltative di codifica o dei marcatori di stato, anche questi devono venir letti.

`readline`(*[size]*)
Legge una riga dallo stream di input e restituire i dati decodificati.

Diversamente dal metodo `readlines()`, questo metodo eredita la conoscenza dell'interruzione di riga dal metodo `readlines()` dello stream sottostante – attualmente non esiste supporto per l'interruzione di riga usando il codec decoder, a causa di mancanze di righe nel buffer. Le classi derivate dovrebbero comunque, se possibile, tentare di implementare questo metodo, usando la loro conoscenza dell'interruzione di riga.

size, se indicato, viene passato come argomento dimensione nel metodo `readline()` dello stream.

`readlines`(*[sizehint]*)
Legge tutte le righe disponibili nel flusso di input e le restituisce come una lista di righe.

Le interruzioni di riga vengono implementate usando il metodo decoder di codifica, e vengono incluse nell'elenco dei valori.

sizehint, se indicato, viene passato come argomento *size*, nel metodo `read()` dello stream.

`reset`()
Azzera i buffer dei codec utilizzati per il mantenimento dello stato.

Notate che nessun riposizionamento nello stream dovrebbe venire messo in atto. Questo metodo viene utilizzato principalmente come sistema di recupero da errori di decodifica.

In aggiunta ai metodi sopra indicati, `StreamReader` deve anche ereditare tutti gli altri metodi e attributi dallo stream sottostante.

Le prossime due classi base vengono incluse per comodità. Non sono necessarie al registro dei codec, ma il loro utilizzo può tornare utile.

Oggetti StreamReaderWriter

La classe `StreamReaderWriter` permette il rivestimento dei flussi che lavorano sia in modalità lettura che di scrittura.

La fase di progettazione consiste allora nel fatto che possiate usare le funzioni factory restituite da funzione `lookup()`, per costruire l'istanza.

class `StreamReaderWriter`(*stream, Reader, Writer, errors*)
Crea una istanza `StreamReaderWriter`. *stream* deve essere un oggetto simile a file. *Reader* e *Writer* devono essere funzioni factory o classi in grado di fornire le interfacce `StreamReader` e `StreamWriter`. La gestione degli errori viene fatta nello stesso modo di quello definito per i flussi di lettura e scrittura.

Le istanze `StreamReaderWriter` definiscono le interfacce combinate delle classi `StreamReader` e `StreamWriter`. Esse ereditano tutti gli altri metodi e attributi dal flusso sottostante.

Oggetti StreamRecoder

La classe `StreamRecoder` fornisce una visione frontend - backend dei dati in codifica, qualche volta utile quando ci si trova a gestire differenti ambienti di codifica.

La sua struttura è tale che possiate utilizzare le funzioni factory restituite dalla funzione `lookup()` per costruire l'istanza.

class `StreamRecoder`(*stream, encode, decode, Reader, Writer, errors*)
Crea una instanza `StreamRecoder` che implementa una conversione a due vie: *encode* e *decode* lavorano sul frontend (l'input a `read()` e l'output a `write()`) mentre *Reader* e *Writer* lavorano sul backend (lettura e scrittura sul flusso).

Possono venire usati questi oggetti per eseguire una trasparente, diretta, ricodifica da, per esempio, Latin-1 a UTF-8 e viceversa.

stream deve essere un oggetto simile a file.

encode e *decode* devono aderire all'interfaccia `Codec`, *Reader* e *Writer* devono essere funzioni di base o classi in grado di fornire oggetti rispettivamente delle interfacce `StreamReader` e `StreamWriter`.

encode e *decode* sono necessarie per la traduzione del frontend, *Reader* e *Writer* per la traduzione del backend. Il formato intermedio usato viene determinato dai due insiemi di codec, per esempio il codec Unicode userà Unicode come codifica intermedia.

La gestione degli errori viene fatta nello stesso modo di quello definito per i flussi di lettura e scrittura.

Le istanze `StreamRecoder` definiscono le interfacce combinate delle classi `StreamReader` e `StreamWriter`. Ereditano tutti gli altri metodi ed attributi del flusso sottostante.

4.9.2 Codifica standard

Python viene distribuito con un certo numero di codifiche built-in, tutte implementate come funzioni C, o con dizionari come tabelle mappate. La tabella seguente elenca le codifiche per nome, insieme ad alcuni alias comuni, ed al linguaggio per cui viene utilizzata la codifica indicata. Né la lista degli alias né la lista dei linguaggi vogliono essere complete. Notate che le variazioni nella sintassi, che differiscono anche solo nel maiuscolo/minuscolo o utilizzano un trattino invece di un trattino basso, vengono comunque considerati sinonimi validi.

Molti insiemi di caratteri supportano le stesse lingue. Queste variano nei singoli caratteri (per esempio il simbolo dell'EURO può essere supportato o meno), e nell'assegnazione dei caratteri alle posizioni del codice. Per le lingue europee in particolare, esistono tipicamente le seguenti varianti:

- una codifica ISO 8859
- una pagina di codici Microsoft Windows che tipicamente deriva da un insieme di codici 8859, ma con i caratteri di controllo sostituiti da caratteri grafici
- una pagina di codici IBM EBCDIC
- una pagina di codici IBM PC, ASCII compatibile

Codec	**Alias**	**Linguaggi**
ascii	646, us-ascii	Inglese
big5	big5_tw, csbig5	Cinese Tradi
cp037	IBM037, IBM039	Inglese
cp424	EBCDIC-CP-HE, IBM424	Ebraico
cp437	437, IBM437	Inglese
cp500	EBCDIC-CP-BE, EBCDIC-CP-CH, IBM500	Europa dell'O
cp737		Greco
cp775	IBM775	Lingue baltic
cp850	850, IBM850	Europa dell'O
cp852	852, IBM852	Europa centr
cp855	855, IBM855	Bulgaro, Bie
cp856		Ebraico
cp857	857, IBM857	Turco
cp860	860, IBM860	Portoghese
cp861	861, CP-IS, IBM861	Islandese
cp862	862, IBM862	Ebraico
cp863	863, IBM863	Canadese
cp864	IBM864	Arabo
cp865	865, IBM865	Danese, Nor
cp869	869, CP-GR, IBM869	Greco
cp874		Thailandese
cp875		Greco
cp932	932, ms932, mskanji, ms_kanji	Giapponese
cp949	949, ms949, uhc	Coreano
cp950	950, ms950	Cinese tradiz

Codec	Alias	Linguaggi
cp1006		Urdu
cp1026	ibm1026	Turco
cp1140	ibm1140	Europa dell’
cp1250	windows-1250	Europa centr
cp1251	windows-1251	Bulgaro, Bie
cp1252	windows-1252	Europa dell’
cp1253	windows-1253	Greco
cp1254	windows-1254	Turco
cp1255	windows-1255	Ebraico
cp1256	windows1256	Arabo
cp1257	windows-1257	Lingue baltic
cp1258	windows-1258	Vietnamita
euc_jp	eucjp, ujis, u_jis	Giapponese
euc_jisx0213	jisx0213, eucjisx0213	Giapponese
euc_kr	euckr, korean, ksc5601, ks_c_5601, ks_c_5601_1987, ksx1001, ks_x_1001	Coreano
gb2312	chinese, csiso58gb231280, euc_cn, euccn, eucgb2312_cn, gb2312_1980, gb2312_80, iso_ir_58	Cinese Semp
gbk	936, cp936, ms936	Cinese Unifi
gb18030	gb18030_2000	Cinese Unifi
hz	hzgb, hz_gb, hz_gb_2312	Cinese Semp
iso2022_jp	csiso2022jp, iso2022jp, iso_2022_jp	Giapponese
iso2022_jp_1	iso2022jp_1, iso_2022_jp_1	Giapponese
iso2022_jp_2	iso2022jp_2, iso_2022_jp_2	Giapponese,
iso2022_jp_3	iso2022jp_3, iso_2022_jp_3	Giapponese
iso2022_jp_ext	iso2022jp_ext, iso_2022_jp_ext	Giapponese
iso2022_kr	csiso2022kr, iso2022kr, iso_2022_kr	Coreano
latin_1	iso-8859-1, iso8859-1, 8859, cp819, latin, latin1, L1	Europa dell’
iso8859_2	iso-8859-2, latin2, L2	Europa centr
iso8859_3	iso-8859-3, latin3, L3	Esperanto, M
iso8859_4	iso-8859-4, latin4, L4	Lingue baltic
iso8859_5	iso-8859-5, cyrillic	Bulgaro, Bie
iso8859_6	iso-8859-6, arabic	Arabo
iso8859_7	iso-8859-7, greek, greek8	Greco
iso8859_8	iso-8859-8, hebrew	Ebraico
iso8859_9	iso-8859-9, latin5, L5	Turco
iso8859_10	iso-8859-10, latin6, L6	Lingue nordi
iso8859_13	iso-8859-13	Lingue baltic
iso8859_14	iso-8859-14, latin8, L8	Lingue Celte
iso8859_15	iso-8859-15	Europa dell’
johab	cp1361, ms1361	Coreano
koi8_r		Russo
koi8_u		Ucraino
mac_cyrillic	maccyrillic	Bulgaro, Bie
mac_greek	macgreek	Greco
mac_iceland	maciceland	Islandese
mac_latin2	maclatin2, maccentraleurope	Europa centr
mac_roman	macroman	Europa dell’
mac_turkish	macturkish	Turco
ptcp154	csptcp154, pt154, cp154, cyrillic-asian	Kazaco
shift_jis	csshiftjis, shiftjis, sjis, s_jis	Giapponese
shift_jisx0213	shiftjisx0213, sjisx0213, s_jisx0213	Giapponese
utf_16	U16, utf16	Tutte le lingu
utf_16_be	UTF-16BE	Tutte le lingu
utf_16_le	UTF-16LE	Tutte le lingu
utf_7	U7	Tutte le lingu
utf_8	U8, UTF, utf8	Tutte le lingu

Un certo numero di codec è specifico di Python, cosicché il loro nome non ha significato al di fuori di Python stesso. Alcuni di questi non convertono da stringhe Unicode a stringhe di byte, ma invece usano la proprietà dell'architettura del codice macchina di Python in modo tale che ogni funzione biettiva con un argomento, possa venire considerata come una codifica.

Per i codec listati sotto, il risultato nella direzione di "codifica" è sempre una stringa di byte. Il risultato nella direzione di "decodifica" viene elencato nella tabella come un tipo operando.

Codec	Alias	Tipo operando	Scopo
base64_codec	base64, base-64	byte string	Converte l'operando in MIME base64
bz2_codec	bz2	byte string	Comprime l'operando usando bz2
hex_codec	hex	byte string	Converte l'operando in una rappresentazion
idna		Unicode string	Implementa RFC 3490. Nuovo nella versi
mbcs	dbcs	Unicode string	Solo per Windows: codifica l'operando in a
palmos		Unicode string	Codifica di PalmOS 3.5
punycode		Unicode string	Implementa RFC 3492. Nuovo nella versi
quopri_codec	quopri, quoted-printable, quotedprintable	byte string	Converte l'operando in MIME quotato stan
raw_unicode_escape		Unicode string	Produce una stringa assimilabile come cost
rot_13	rot13	byte string	Restituisce la cifratura dell'operando in alg
string_escape		byte string	Produce una stringa assimilabile a una cost
undefined		any	Solleva un'eccezione per tutte le conversion
unicode_escape		Unicode string	Produce una stringa assimilabile a una cost
unicode_internal		Unicode string	Restituisce la rappresentazione interna dell
uu_codec	uu	byte string	Converte l'operando usando uuencode
zlib_codec	zip, zlib	byte string	Comprime l'operando usando gzip

4.9.3 `encodings.idna` — Nomi di Dominio Internazionalizzati in Applicazioni (IDNA)

Nuovo nella versione 2.3.

Questo modulo implementa la RFC 3490 (Nomi di Dominio Internazionalizzati nelle Applicazioni) e la RFC 3492 (Nameprep: un Profilo Stringprep per Nomi di Dominio Internazionalizzati (IDN). Costruisce sopra la codifica `punycode` e su `stringprep`.

Queste RFC, insieme definiscono un protocollo per il supporto dei caratteri non ASCII nei nomi di dominio. Un nome di dominio contenente caratteri non ASCII (come "www.Alliancefrançaise.nu") viene convertito in una codifica compatibile con ASCII(ACE, come "www.xn–alliancefranaise-npb.nu"). La forma ACE del nome di dominio viene poi usata in tutti i luoghi in cui i caratteri arbitrari non sono consentiti dal protocollo, come richieste DNS, campi Host: HTTP, e così via. La conversione viene effettuata dall'applicazione, se possibile in modo non visibile all'utente: l'applicazione dovrebbe convertire in modo trasparente le etichette di dominio Unicode a IDNA al volo, e riconvertire le etichette ACE a Unicode prima di visualizzarle all'utente.

Python supporta questa conversione in diversi modi: il codec `idna` permette la conversione tra Unicode e ACE. Inoltre, il modulo `socket` converte in modo trasparente nomi di host Unicode a ACE, cosicché le applicazioni non hanno bisogno di venire informate circa la conversione dei nomi di host, quando questi vengono passati al modulo socket. Sopra tutto questo, i moduli che possiedono nomi di host come parametri di funzioni, come `httplib` e `ftplib`, accettano nomi di host Unicode (`httplib` invia inontre in modo trasparente un nome di host IDNA nel campo Host:, se invia quel campo a tutti).

Quando si ricevono nomi di host dall'esterno (come in una ricerca di risoluzione inversa) non viene effettuata nessuna conversione automatica verso Unicode: le Applicazioni che desiderano presentare all'utente qualche nome di host, dovrebbero decodificarle dall'Unicode.

Il modulo `encodings.idna` implementa anche la procedura nameprep, che esegue alcune normalizzazioni sui nomi degli host, per poter presentare i dati come non sensibili alle differenze tra maiuscolo e minuscolo dei Nomi di Dominio Internazionali, e per unificare i caratteri simili. La funzione nameprep può venire utilizzata direttamente, se lo si desidera.

`nameprep`(*label*)

Restituisce la versione nameprep di *label*. L'implementazione gestisce attualmente stringhe di richiesta, così `AllowUnassigned` risulta vero.

`ToASCII`(*label*)
Converte un'etichetta in ASCII, come specificato nella RFC 3490. Si assume che `UseSTD3ASCIIRules` sia falso.

`ToUnicode`(*label*)
Converte un'etichetta in Unicode, come specificato nella RFC 3490.

4.10 `unicodedata` — Database Unicode

Questo modulo fornisce l'accesso al database dei caratteri Unicode che definisce le proprietà di ciascun carattere Unicode. I dati di questo database si basano sul file 'UnicodeData.txt' nella versione 3.2.0, disponibile presso l'ftp://ftp.unicode.org/.

Il modulo usa gli stessi nomi e simboli definit dal file UnicodeData File Format 3.2.0 (vedete http://www.unicode.org/Public/UNIDATA/UnicodeData.html). Definisce le seguenti funzioni:

`lookup`(*name*)
Cerca il carattere a partire da *name*. Se il carattere con il dato nome *name* viene trovato, restituisce il corrispondente carattere Unicode. Se non trovato, viene sollevata l'eccezione `KeyError`.

`name`(*unichr*[, *default*])
Restituisce il nome assegnato dal carattere Unicode *unichr* come stringa. Se nessun nome *name* è definito, viene restituito il predefinito, *default*, altrimenti, se non indicato, viene sollevata l'eccezione `ValueError`.

`decimal`(*unichr*[, *default*])
Restituisce il valore decimale assegnato al carattere Unicode *unichr* come intero. Se nessun valore è definito, viene restituito il predefinito, *default*, altrimenti, se non indicato, viene sollevata l'eccezione `ValueError`.

`digit`(*unichr*[, *default*])
Restituisce il valore della cifra assegnata al carattere Unicode *unichr* come intero. Se nessun valore è definito, viene restituito il valore predefinito, *default*, altrimenti, se non indicato, viene sollevata l'eccezione `ValueError`.

`numeric`(*unichr*[, *default*])
Restituisce il valore numerico assegnato al carattere Unicode *unichr* come numero in virgola mobile. Se nessun valore è definito, viene restituito il valore predefinito, *default*, altrimenti, se non indicato, viene sollevata l'eccezione `ValueError`.

`category`(*unichr*)
Restituisce la categoria generale assegnata al carattere Unicode *unichr* come stringa.

`bidirectional`(*unichr*)
Restituisce la categoria bidirezionale assegnata al carattere Unicode *unichr* come stringa. Se nessun valore è definito, viene restituita una stringa vuota.

`combining`(*unichr*)
Restituisce la classe canonica combinata, assegnata al carattere Unicode *unichr* come intero. Restituisce `0` se nessuna classe combinata viene definita.

`mirrored`(*unichr*)
Restituisce la proprietà speculare assegnata al carattere Unicode *unichr* come intero. Restituisce `1` se il carattere viene identificato come "speculare" in un testo bidirezionale, altrimenti `0`.

`decomposition`(*unichr*)
Restituisce il carattere estratto dalla mappa assegnato al carattere Unicode *unichr* come stringa. Viene restituita una stringa vuota nel caso in cui nessuna mappa sia definita.

`normalize`(*form*, *unistr*)
Restituisce la normale forma *form* per la stringa Unicode *unistr*. Valori validi per *form* sono 'NFC', 'NFKC', 'NFD' e 'NFKD'.

Lo standard Unicode definisce varie forme di normalizzazione di una stringa Unicode, basate sulla defi-

nizione di equivalenze canoniche ed equivalenze compatibili. In Unicode, diversi caratteri possono essere espressi in diversi modi. Per esempio, il carattere U+00C7 (LATIN MAIUSCOLO LETTERA C CON ACCENTO CEDILLA) può anche venire espresso come sequenza U+0043 (LATIN MAIUSCOLO LETTERA C) U+0327 (COMBINATO CEDILLA).

Per ogni carattere, esistono due forme normali: forma normale C e forma normale D. La forma normale D (NFD) è conosciuta anche come decomposizione canonica, e traduce ciascun carattere nella sua forma decomposta. La forma normale C (NFC) prima applica una decomposizione canonica, quindi nuovamente compone i caratteri ricombinandoli.

Oltre queste due forme, esistono due forme aggiuntive basate sull'equivalenza di compatibilità. In Unicode, alcuni caratteri vengono supportati così come lo sarebbero se venissero unificati con altri caratteri. Per esempio, U+2160 (ROMAN NUMERALE UNO) è esattamente la stessa cosa di U+0049 (LATIN MAIUSCOLO LETTER I). Comunque, viene supportato in Unicode per mantenere la compatibilità con gli insiemi di caratteri esistenti (per esempio gb2312).

La forma normale KD (NFKD) applicherà la decomposizione compatibile, ad esempio sostituire tutti caratteri di compatibilità con i loro equivalenti. La forma normale KC (NFKC) prima applica la decomposizione compatibile, seguita dalla composizione canonica.

Nuovo nella versione 2.3.

In aggiunta, il modulo rende disponibile la seguente costante:

`unidata_version`
La versione del database Unicode usato in questo modulo.

Nuovo nella versione 2.3.

4.11 `stringprep` — Preparazione delle stringhe per Internet

Quando si devono identificare delle cose (come ad esempio i nomi degli host) in Internet, è spesso necessario confrontare queste identificazioni attraverso un meccanismo di "uguaglianza". Il modo con cui questo confronto viene effettuato può dipendere dal dominio dell'applicazione, per esempio se deve essere sensibile a maiuscole e minuscole o meno. Può anche risultare necessario restringere il campo delle possibili identificazioni, per consentire le identificazoni composte di soli caratteri "stampabili".

La RFC 3454 definisce una procedura per "preparare" le stringhe Unicode nei protocolli internet. Prima di inviare le stringhe al cavo, queste vengono elaborate dalla procedura di preparazione, dopo la quale si trovano in una certa forma normalizzata. La RFC definisce un insieme di tabelle, che possono essere combinate in profili. Ogni profilo deve definire quale tabella utilizzare, e quali altre parti facoltative della procedura `stringprep` devono entrare a far parte del profilo. Un esempio di un profilo di `stringprep` è `nameprep`, che viene utilizzata per i nomi di dominio internazionalizzati.

Il modulo `stringprep` espone solo le tabelle dell'RFC 3454. Poiché queste tabelle sarebbero molto grandi da rappresentare come dizionari o liste, il modulo usa internamente il database dei caratteri Unicode. Lo stesso codice sorgente del modulo viene generato utilizzando l'utility `mkstringprep.py`.

Come risultato, queste tabelle vengono esposte come funzioni, non come strutture dati. Ci sono due tipi di tabelle nella RFC: insiemi e mappe. Per un insieme, `stringprep` fornisce la "funzione caratteristica", per esempio una funzione che restituisce vero se il parametro è parte dell'insieme. Per le mappe, fornisce la funzione di mappatura: data la chiave, restituisce il valore associato. Di seguito una lista di tutte le funzioni disponibili nel modulo.

`in_table_a1`(*code*)
Determina se *code* è presente nella tabella tableA.1 (Codice non assegnato in Unicode 3.2).

`in_table_b1`(*code*)
Determina se *code* è presente nella tabella tableB.1 (Codice normalmente mappato a nulla).

`map_table_b2`(*code*)
Restituisce il valore mappato per *code* in accordo con la tabella tableB.2 (Mappa per case-folding usata con NFKC).

`map_table_b3`(*code*)

Restituisce il valore mappato per *code* in accordo con la tabella tableB.3 (Mappa per case-folding usata senza normalizzazione).

in_table_c11(*code*)
Determina se *code* è presente nella tabella tableC.1.1 (Caratteri di spaziatura ASCII).

in_table_c12(*code*)
Determina se *code* è presente nella tabella tableC.1.2 (Caratteri di spaziatura non ASCII).

in_table_c11_c12(*code*)
Determina se *code* è presente nella tabella tableC.1 (Caratteri di spaziatura, unione di C.1.1 e C.1.2).

in_table_c21(*code*)
Determina se *code* è presente nella tabella tableC.2.1 (Caratteri di controllo ASCII).

in_table_c22(*code*)
Determina se *code* è presente nella tabella tableC.2.2 (Caratteri di controllo non ASCII).

in_table_c21_c22(*code*)
Determina se *code* è presente nella tabella tableC.2 (Caratteri di controllo, unione di C.2.1 e C.2.2).

in_table_c3(*code*)
Determina se *code* è presente nella tabella tableC.3 (Utilizzo privato).

in_table_c4(*code*)
Determina se *code* è presente nella tabella tableC.4 (Non-caratteri nel codice punti).

in_table_c5(*code*)
Determina se *code* è presente nella tabella tableC.5 (Codici surrogati).

in_table_c6(*code*)
Determina se *code* è presente nella tabella tableC.6 (Non appropriati per testo semplice).

in_table_c7(*code*)
Determina se *code* è presente nella tabella tableC.7 (Non appropriati per rappresentazioni canoniche).

in_table_c8(*code*)
Determina se *code* è presente nella tabella tableC.8 (Cambiano la proprietà di visualizzazione o vengono deprecati).

in_table_c9(*code*)
Determina se *code* è presente nella tabella tableC.9 (Caratteri con tag).

in_table_d1(*code*)
Determina se *code* è presente nella tabella tableD.1 (Caratteri con proprietà bidirezionali “R” o “AL”).

in_table_d2(*code*)
Determina se *code* è presente nella tabella tableD.2 (Caratteri con proprietà bidirezionale “L”).

CAPITOLO

CINQUE

Servizi vari

I moduli descritti in questo capitolo forniscono vari servizi che sono disponibili in tutte le versioni di Python. Qui ora ne verrà data una visione d'insieme:

pydoc	Generatore di documentazione e sistema di aiuto in linea.
doctest	Un framework per verificare gli esempi presenti nelle docstring.
unittest	Ambiente per i test delle unità di codice (unittest).
test	Il package dei test di regressione contiene le suite di test per python.
test.test_support	Supporto per Python ai test di regressione.
math	Funzioni matematiche (`sin()` etc.).
cmath	Funzioni matematiche per i numeri complessi.
random	Genera numeri pseudo casuali usando varie comuni distribuzioni.
whrandom	Generatore di numeri pseudo casuali in virgola mobile.
bisect	Algoritmo di bisezione di array per la ricerca binaria.
collections	Tipi di dato contenitore ad alte prestazioni
heapq	Algoritmo heap queue (a.k.a. algoritmo per la coda con priorità che utillizza uno heap).
array	Array efficienti di valori numerici tipizzati uniformemente.
sets	Implementazione di insiemi composti da elementi distinti.
itertools	Funzioni che creano iteratori per cicli efficienti.
ConfigParser	Analizzatore dei file di configurazione.
fileinput	Iterazione simile al Perl su righe provenienti da molteplici flussi di input, con capacità di "salvare sul
xreadlines	Iterazione efficiente sulle righe di un file.
calendar	Funzioni per lavorare con i calendari, incluse alcune emulazioni del programma UNIX **cal**.
cmd	Creazione di interpreti a riga di comando.
shlex	Semplice analizzatore lessicale per linguaggi simili alla shell di UNIX.

5.1 pydoc — Generatore di documentazione e sistema di aiuto in linea

Nuovo nella versione 2.1.

Il modulo `pydoc` genera automaticamente la documentazione dai moduli Python. La documentazione può presentarsi come testo su una console, può essere servita ad un browser Web o salvata in un file HTML.

La funzione built-in `help()` invoca il sistema di aiuto in linea nell'interprete, che usa `pydoc` per generare la sua documentazione come testo sulla console. La stessa documentazione può anche essere vista fuori dall'interprete di Python eseguendo **pydoc** come un script al prompt dei comandi del sistema operativo. Per esempio, eseguendo

```
pydoc sys
```

sulla shell si vedrà la documentazione riguardante il modulo `sys` in uno stile simile a quello delle pagine dei manuali visibili su UNIX con il comando **man**. L'argomento di **pydoc** può essere il nome di una funzione, modulo o pacchetto, od un riferimento ad una classe, metodo, o una funzione entro un modulo, o un modulo in un

package. Se l'argomento passato a **pydoc** somiglia ad un percorso (ovverosia contiene un separatore di percorso per il vostro sistema operativo come ad esempio lo slash in UNIX), e si riferisce ad un file esistente in codice Python, la documentazione viene prodotta per quel file.

Specificando l'opzione **-w** prima dell'argomento si otterrà una documentazione HTML che verra scritta al di fuori dal file, nella directory corrente, invece che mostrare il testo sulla console.

Specificando **-k** prima dell'argomento la ricerca avverrà solo nelle righe introduttive di tutti i moduli disponibili che contengono le chiavi date, in modo simile a quanto presente nel comando **man** di UNIX. La riga introduttiva di un modulo è la prima riga della sua stringa di documentazione.

Si può anche usare **pydoc** per far partire un server HTTP su un macchina locale che renderà disponibile documentazione ai browser che visiteranno le pagine web. **pydoc -p 1234** farà partire un server HTTP sulla porta 1234 permettendo di visitare la documentazione all'indirizzo `http://localhost:1234/`, inserendo la stringa nel vostro browser Web preferito. **pydoc -g** inizializzerà il server ed in aggiunta aprirà una minima interfaccia grafica basata su `Tkinter` per aiutarvi a cercare le pagina nella documentazione.

Quando **pydoc** genera documentazione utilizza l'ambiente ed il percorso correnti per individuare i moduli. In questo modo, invocando **pydoc spam** verrà documentata esattamente la stessa versione del modulo che avreste ottenuto lanciando l'interprete Python e digitando '`import spam`'.

Per quanto riguarda la documentazione dei moduli, si assume che risiedano in http://www.python.org/doc/current/lib/. Per questo può essere di primaria importanza impostare le variabili d'ambiente PYTHONDOCS ad un URL o ad una directory locale differenti che contengono le pagine del Library Reference Manual.

5.2 `doctest` — Verifica che le docstring rappresentino la realtà.

Il modulo `doctest` effettua una ricerca nelle docstring di un modulo cercando del testo che somigli ad una sessione interattiva del Python, quindi esegue tutte le sessioni per verificare che funzionino come mostrato. Ecco un esempio piccolo ma completo:

```
"""
Questo è un modulo d'esempio.

L'esempio rappresenta una funzione, factorial.  Per esempio,

>>> factorial(5)
120
"""

def factorial(n):
    """Restituisce il fattoriale di n , un intero esatto >= 0.

    Se il risultato è abbastanza piccolo per adattarsi ad un int,
    restituisce un int. Altrimenti un long.

    >>> [factorial(n) for n in range(6)]
    [1, 1, 2, 6, 24, 120]
    >>> [factorial(long(n)) for n in range(6)]
    [1, 1, 2, 6, 24, 120]
    >>> factorial(30)
    265252859812191058636308480000000L
    >>> factorial(30L)
    265252859812191058636308480000000L
    >>> factorial(-1)
    Traceback (most recent call last):
        ...
    ValueError: n must be >= 0

    Fattoriali di un numero in virgola mobile sono OK, ma la frazione
    deve essere un intero esattamente :
    >>> factorial(30.1)
    Traceback (most recent call last):
        ...
    ValueError: n must be exact integer
    >>> factorial(30.0)
    265252859812191058636308480000000L

    Deve anche non essere ridicolmente grande:
    >>> factorial(1e100)
    Traceback (most recent call last):
        ...
    OverflowError: n too large
    """
```

```
    import math
    if not n >= 0:
        raise ValueError("n must be >= 0")
    if math.floor(n) != n:
        raise ValueError("n must be exact integer")
    if n+1 == n:  # catch a value like 1e300
        raise OverflowError("n too large")
    result = 1
    factor = 2
    while factor <= n:
        try:
            result *= factor
        except OverflowError:
            result *= long(factor)
        factor += 1
    return result

def _test():
    import doctest, example
    return doctest.testmod(example)

if __name__ == "__main__":
    _test()
```

Se lanciate il programma ‘example.py’ direttamente da riga di comando, `doctest` mostra la sua magia:

```
$ python example.py
$
```

Non c’è nessun messaggio! È normale e significa che tutti gli esempi funzionano. Aggiungendo **-v** allo script, `doctest` stampa un resoconto dettagliato di ciò che sta testando ed un riassunto alla fine:

```
$ python example.py -v
Running example.__doc__
Trying: factorial(5)
Expecting: 120
ok
0 of 1 examples failed in example.__doc__
Running example.factorial.__doc__
Trying: [factorial(n) for n in range(6)]
Expecting: [1, 1, 2, 6, 24, 120]
ok
Trying: [factorial(long(n)) for n in range(6)]
Expecting: [1, 1, 2, 6, 24, 120]
ok
Trying: factorial(30)
Expecting: 265252859812191058636308480000000L
ok
```

E così di seguito, eventualmente finendo con:

```
Trying: factorial(1e100)
Expecting:
Traceback (most recent call last):
    ...
OverflowError: n too large
ok
0 of 8 examples failed in example.factorial.__doc__
2 items passed all tests:
   1 tests in example
   8 tests in example.factorial
9 tests in 2 items.
9 passed and 0 failed.
Test passed.
$
```

É tutto quello di cui avete bisogno per per cominciare ad essere produttivi con `doctest`! Le stringhe in 'doctest.py' contengono informazioni dettagliate riguardo tutti gli aspetti di `doctest`. In questa sede abbiamo toccato solo gli argomenti principali.

5.2.1 Normale utilizzo

Nell'uso corrente, si sostituisce ogni modulo `M` con il modulo da testare:

```
def _test():
    import doctest, M           # sostituite M con il nome del vostro modulo
    return doctest.testmod(M)   # idem

if __name__ == "__main__":
    _test()
```

Se si vuole testare il modulo come modulo principale non c'è bisogno di aggiungere M a `testmod()`; in questo caso testerà il modulo corrente.

Così, lanciando il modulo come uno script, si ottiene l'esecuzione e la verifica degli esempi contenuti nelle docstring:

```
python M.py
```

Questo non mostrerà nulla a meno che non fallisca un esempio, nel qual caso, l'esempio sbagliato e la causa/e dell'errore/i saranno stampati sullo stdout e l'ultima linea dell'output sarà `'Test failed.'`.

Utilizzo con l'opzione **-v**:

```
python M.py -v
```

viene stampato un rapporto dettagliato di tutti gli esempi provati, insieme ad un riepilogo finale.

Si può forzare la modalità prolissa passando `verbose=1` a `testmod()`, o al contrario, disattivarla passando `verbose=0`. In entrambi i casi, `sys.argv` non viene esaminato da `testmod()`.

In ogni caso, `testmod()` restituisce una coppia di interi `(`*f*`, `*t*`)`, dove *f* è il numero della docstring di esempio che fallisce e *t* è il numero totale di doctring di esempio provate.

5.2.2 Quali docstring vengono esaminate?

Date un'occhiata alle docstring in 'doctest.py' per tutti i dettagli. Nessuna sorpresa: vengono ricercate la docstring del modulo, di tutte le funzioni, delle classi e dei metodi. Facoltativamente, il test può essere diretto per escludere le docstring attaccate ad oggetti con nomi privati. Gli oggetti importati nel modulo non sono cercati.

In aggiunta, se `M.__test__` esiste ed "è vero", esso deve essere un dizionario in cui ogni voce mappa un nome (stringa) in un oggetto funzione, un oggetto classe o stringa. Le docstring degli oggetti funzione e classe trovati da `M.__test__` vengono ricercati anche se il test è impostato per non considerare i nomi privati nel resto del modulo.In output, compare una chiave `K` in `M.__test__` con nome

```
<name of M>.__test__.K
```

Qualunque classe trovata viene esaminata ricorsivamente allo stesso modo, per testare le docstring nei metodi contenuti e nelle classi annidate. Mentre i nomi privati raggiunti dai vari attributi globali di `M`, possono essere facoltativamente scartati, tutti i nomi raggiunti tramite `M.__test__` vengono ricercati.

5.2.3 Quale è il contesto di esecuzione?

Ogni volta che `testmod()` trova una docstring da testare, utilizza, in modo predefinito, una *copia* degli attributi globali di `M`, preservando i suoi veri attributi globali, cosicchè un test in `M` non possa lasciarsi dietro pezzetti che consentirebbero accidentalmente ad un altro test di funzionare. Questo significa che gli esempi possono liberamente utilizzare qualunque nome definito nel livello più alto in `M` insieme ai nomi definiti in precedenza nella docstring in esecuzione.

Si può forzare l'utilizzo di un proprio dizionario come contesto di esecuzione, passando `globs=your_dict` a `testmod()`. Presumibilmente questo sarà una copia di `M.__dict__` combinata con gli attributi globali di altri moduli importati.

5.2.4 Riguardo le eccezioni?

Nessun problema, finchè l'unico output generato dall'esempio è la traceback stessa. Per esempio.

```
>>> [1, 2, 3].remove(42)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ValueError: list.remove(x): x not in list
>>>
```

Notate che viene confrontato solo il tipo ed il valore eccezione (nello specifico, solo l'ultima riga della traceback). Le varie righe "File" della traceback, in mezzo, possono essere tralasciate (a meno che non aggiungano significativamente un valore documentativo all'esempio).

5.2.5 Uso avanzato

Sono disponibili diverse funzioni a livello di modulo per controllare il funzionamenteo dei doctest.

debug(*module*, *name*)

Esegue il debug di una singola docstring contenente il doctest.

Passate il *module* (o il nome del modulo in notazione puntata) contenete la docstring su cui eseguire il debug ed il *name* (all'interno del modulo) dell'oggetto con la docstring su cui eseguire il debug.

Gli esempi doctest vengono estratti (vedete la funzione `testsource()`), e scritti in un file temporaneo. Viene poi eseguito su quel file il debugger di Python, pdb. Nuovo nella versione 2.3.

testmod()
Questa funzione fornisce l'interfaccia base sui doctest. Crea un'istanza locale della classe `Tester`, esegue i metodi appropriati di quella classe e combina i risultati nell'istanza globale `Tester`, `master`.

Per disporre di un controllo migliore di quanto non venga fornito da `testmod()`, è possibile creare un'istanza di `Tester` secondo politiche personalizzate, oppure si possono eseguire i metodi di `master` direttamente. Per i dettagli vedete `Tester.__doc__`.

testsource(*module*, *name*)
Estrae gli esempi doctest da una docstring.

Fornite il *module* (o il nome del modulo in notazione puntata) contenente i test da estrarre ed il *name* (all'interno del modulo) dell'oggetto con le docstring contenenti i test da estrarre.

Gli esempi doctest vengono restituiti come una stringa contenente codice Python. L'output previsto negli esempi viene convertito come commento Python. Nuovo nella versione 2.3.

DocTestSuite([*module*])
Converte i test di doctest per un modulo in `unittest.TestSuite`.

La classe `TestSuite` restituita deve essere messa in esecuzione all'interno del framework unittest ed ha il compito di eseguire ogni doctest del modulo. Se uno dei doctest fallisce allora l'unità di test sintattica fallisce e viene sollevata un'eccezione `DocTestTestFailure` che mostra il nome del file contenente il test e (qualche volta in maniera approssimativa) il numero di riga.

L'argomento opzionale *module* specifica il modulo da testare. Può essere un oggetto modulo oppure il nome (possibilmente puntato) di un modulo. Se non specificato, viene utilizzato il modulo che chiama la funzione.

Esempio che mostra uno dei molti modi in cui il modulo `unittest` può usare `TestSuite`:

```
import unittest
import doctest
import my_module_with_doctests

suite = doctest.DocTestSuite(my_module_with_doctests)
runner = unittest.TextTestRunner()
runner.run(suite)
```

Nuovo nella versione 2.3. **Avvertenze:** Questa funzione non cerca `M.__test__` e le sue tecniche di ricerca non sono esattamente identiche a `testmod()` in ogni dettaglio. Le future versioni porteranno i due metodi a convergere.

5.2.6 Come vengono riconosciuti gli esempi nelle stringhe di documentazione?

Nella maggior parte dei casi copiare ed incollare l'output di una sessione interattiva è sufficiente — accertatevi solamente che gli spazi di inizio riga siano rigidamente consistenti (potreste essere troppo pigri per farlo in maniera coerente, potreste usare sia tabulazioni che spazi, ma `doctest` non è in grado di capire qual'è il significato che voi date alle tabulazioni).

```
>>> # i commenti vengono ignorati
>>> x = 12
>>> x
12
>>> if x == 13:
...     print "si"
... else:
...     print "no"
...     print "NO"
...     print "NO!!!"
...
no
NO
NO!!!
>>>
```

Ogni output atteso deve seguire immediatamente i `'>>> '` o `'... '` successivi alla riga finale contenente il codice, e l'output atteso (se è previsto) si estende fino al successivo `'>>> '` o a tutte le righe vuote.

Per una stampa corretta:

- L'output atteso non può contenere una riga di spazi vuoti, dato che una riga di questo tipo è usata come segnale per la fine dell'output atteso.
- Viene catturato l'output sullo standard output, ma non quello verso lo standard error (le traceback delle eccezioni sono catturate in maniera diversa).
- Se continuate una riga con un backslash in una sessione interattiva, o per qualsiasi ragione usate un backslash, nelle docstring dovete raddoppiarlo. Questo è dovuto semplicemente al fatto che siete in una stringa, e quindi deve essere anteposto il carattere di escape a fronte di un backslash per renderlo riconoscibile. Come:

  ```
  >>> if "si" == \\
  ...     "s" +   \\
  ...     "i":
  ...     print 'si'
  si
  ```

- La colonna iniziale non conta:

  ```
    >>> assert "Easy!"
          >>> import math
              >>> math.floor(1.9)
              1.0
  ```

  e viene eliminato dall'output atteso un numero di spazi vuoti iniziali pari a quelli presenti nella prima riga con `'>>> '` che ha scatenato il processo.

5.2.7 Avvertenze

1. `doctest` è esigente nel richiedere un'esatta corrispondenza nell'output atteso. Se anche solo un singolo carattere non è corretto, il test fallisce. Questo potrebbe qualche volta sorprendervi, mentre imparate che cosa Python garantisce e cosa no circa l'output. Per esempio, stampando un dizionario, Python non garantisce che una coppia chiave-valore sia stampata in un ordine particolare, così un test come:

```
>>> foo()
{"Hermione": "hippogryph", "Harry": "broomstick"}
>>>
```

è vulnerabile! Invece, la soluzione giusta è

```
>>> foo() == {"Hermione": "hippogryph", "Harry": "broomstick"}
True
>>>
```

Un altro metodo è

```
>>> d = foo().items()
>>> d.sort()
>>> d
[('Harry', 'broomstick'), ('Hermione', 'hippogryph')]
```

Ci sono ancora altri metodi ma ormai vi sarete fatti un'idea.

Un'altra cattiva idea è quella di stampare cose che incorporano un indirizzo di un oggetto, come per esempio

```
>>> id(1.0) # alcune volte potrà fallire
7948648
>>>
```

I numeri in virgola mobile sono inoltre soggetti a piccole variazioni di output in piattaforme diverse, perché Python differisce dalle librerie C relativamente alla formattazione in virgola mobile e le stesse librerie C differiscono ampiamente in qualità riguardo a questo argomento.

```
>>> 1./7  # rischiosa
0.14285714285714285
>>> print 1./7 # sicura
0.142857142857
>>> print round(1./7, 6) # molto sicura
0.142857
```

I numeri nella forma `I/2.**J` sono sicuri per tutte le piattaforme e spesso faccio in modo che gli esempi di doctest producano numeri in quella forma.

```
>>> 3./4  # totalmente sicuro
0.75
```

Inoltre le frazioni semplici sono più facili da capire e sono quindi più indicate per la documentazione.

2. Fate attenzione se avete del codice da eseguire una sola volta.

   Se avete codice a livello di modulo che deve essere eseguito una sola volta, una definizione molto semplice di `_test()` è

```
def _test():
    import doctest, sys
    doctest.testmod()
```

3. Iniziando con Python 2.3, non sempre ciò che si vede corrisponde a quello che otterremo (WYSIWYG). In Python 2.3, la stringa booleana `'0'` e `'1'` risulterà essere `'False'` e `'True'`. Di conseguenza risulterà goffo scrivere una doctest che visualizzi risultati booleani che superino le molte versioni di Python. In Python 2.3, in modo predefinito e come caso speciale, se un blocco di output previsto consiste esclusivamente di `'0'` ed il blocco di output attuale consiste solamente di `'False'`, il tutto verrà accettato come una corrispondenza esatta e nello stesso modo anche per `'1'` nei confronti di `'True'`. Questo comportamento può essere eliminato passando come argomento opzionale optionflags di `testmod()` la costante `DONT_ACCEPT_TRUE_FOR_1` del nuovo modulo (in 2.3). Fra alcuni anni, quando la rappresentazione dei boleani con interi sarà storia, questa forzatura verrà probabilmente di nuovo rimossa.

5.2.8 Soapbox

La prima parola in "doctest" è "doc" e per questo motivo l'autore ha scritto `doctest`: lo scopo era quello di fornire una documentazione aggiornata. É successo che `doctest` risulti un piacevole ambiente per l'unit testing (NdT: test per le unità di codice), ma non era questo l'intento.

Scegliete gli esempi con attenzione. Esiste un'arte necessaria da imparare che all'inizio non risulterà molto naturale. Gli esempi dovrebbero valorizzare la documentazione. Un buon esempio può sostituire molte parole. Se possibile il manuale espone alcuni esempi normali, casi limiti, finezze ed esempi per ogni tipo di eccezione. Probabilmente proverete i casi limiti e le finezze in una shell: `doctest` cercherà di rendere il più semplice possibile la cattura della vostra sessione e verificherà continuamente il lavoro svolto ed il suo design.

Se gli esempi sono stati seguiti con cura, saranno di fondamentale importanza e vi ripagheranno per il tempo perso. Rimango ancora stupito nel vedere come i miei esempi di `doctest` perdano efficacia a seguito di piccoli cambiamenti.

Per test esaustivi, o elaborati casi che non aggiungono valore ai documenti, definite invece un dizionario `__test__`.

5.3 `unittest` — Ambiente per il test delle unità di codice (unittest)

Nuovo nella versione 2.1.

L'ambiente per il test delle unità di codice di Python, spesso indicato come "PyUnit", è una derivazione di Junit per il linguaggio Python, ideata da Kent Beck ed Eric Gamma. A sua volta, JUnit è una versione di Java dell'ambiente di Kent Smalltalk. Ognuna rappresenta di fatto lo standard per il test delle unità di codice per il relativo linguaggio.

PyUnit supporta l'automazione dei test, la condivisione del setup e dello shutdown del codice per i test, l'aggregazione dei test in raccolte e l'indipendenza dei test dall'ambiente relativo. Il modulo `unittest` fornisce classi che rendono facile il supporto di queste caratteristiche per un insieme di test.

Per ottenere ciò, PyUnit supporta alcuni concetti importanti:

test fixture
: Il *test fixture* (NdT: impianto di test) rappresenta la preparazione necessaria per effettuare uno o più test ed ogni azione di cleanup relativa. Ciò può comportare, per esempio, la creazione di database proxy o temporanei, di directory oppure l'avvio di processi server.

test case
: Il *test case* è l'unità più piccola di codice da testare. Controlla una specifica risposta ad un insieme di input. PyUnit fornisce una classe di base, `TestCase`, che può essere utilizzata per creare dei nuovi test di unità di codice.

test suite
: Una *test suite* è una raccolta di test cases, test suites, o entrambi. Viene usata per aggregare test che devono essere eseguiti assieme.

test runner
: Un *test runner* (NdT: "esecutore del test") è un componente che dirige l'esecuzione dei vari test e forni-

sce l'esito all'utente. Il runner può far uso di un'interfaccia grafica, testuale, oppure restituire un valore particolare per indicare i risultati dell'esecuzione dei test.

I concetti di test case e fixture vengono supportati tramite le classi `TestCase` e `FunctionTestCase`; la prima dovrebbe essere usata quando si creano nuovi test mentre la seconda può essere utilizzata quando si integra codice di test esistente con un ambiente guidato da PyUnit. Quando si creano fixture di test usando `TestCase`, i metodi `setUp()` e `tearDown()` possono essere ridefiniti al fine di fornire l'inizializzazione e la terminazione della fixture. Con la classe `FunctionTestCase`, possono essere passate al costruttore, per questi scopi, delle funzioni esistenti. Quando si avvia il test, l'inizializzazione della fixture del test viene eseguita per prima; in caso di successo, il metodo di pulitura viene avviato dopo l'esecuzione del test, senza riguardo al suo esito. Ogni istanza di `TestCase` verrà usata solo per avviare un singolo metodo del test, perciò una nuova fixture verrà creata per ogni singolo test.

Le test suite vengono implementate dalla classe `TestSuite`. Tale classe consente l'aggregazione di test individuali e test suite; quando la suite viene eseguita, tutti i test vengono aggiunti direttamente alla suite ed le sue suite "figlie" vengono eseguite.

Un test runner è un oggetto che fornisce un singolo metodo, `run()`, il quale accetta come parametro un oggetto `TestCase` o `TestSuite`, e restituisce un oggetto per il risultato. La classe `TestResult` è stata adibita a questo scopo PyUnit fornisce la classe `TextTestRunner` come test runner di esempio il quale segnala in modo predefinito i risultati del test sullo standard error. Runner alternativi possono essere implementati per altri ambienti (come quelli grafici) senza alcun bisogno di derivarli da una classe specifica.

Vedete anche:

Modulo `doctest` (sezione 5.2):
Un altro modulo per il supporto dei test che permette di farne uso in modo differente.

Sito web di PyUnit
(http://pyunit.sourceforge.net/)
La sorgente per ulteriori informazioni su PyUnit.

Simple Smalltalk Testing: With Patterns
(http://www.XProgramming.com/testfram.htm)
Lo scritto originale di Kent Beck sulle strutture per i test, che usano i modelli condivisi da `unittest`.

5.3.1 Un semplice esempio

Il modulo `unittest` fornisce un ricco insieme di strumenti per costruire ed eseguire dei test. Questa sezione mostra come un piccolo sotto insieme di tali strumenti sia sufficiente a venire incontro alle necessità della maggior parte degli utenti.

Ecco un breve script per testare tre funzioni del modulo `random`:

```
import random
import unittest

class TestSequenceFunctions(unittest.TestCase):

    def setUp(self):
        self.seq = range(10)

    def testshuffle(self):
        # assicura che la sequenza rimescolata non perda degli elementi
        random.shuffle(self.seq)
        self.seq.sort()
        self.assertEqual(self.seq, range(10))

    def testchoice(self):
        element = random.choice(self.seq)
        self.assert_(element in self.seq)

    def testsample(self):
        self.assertRaises(ValueError, random.sample, self.seq, 20)
        for element in random.sample(self.seq, 5):
            self.assert_(element in self.seq)

if __name__ == '__main__':
    unittest.main()
```

Una testcase viene creata derivandola da una classe `unittest.TestCase`. I tre test singoli sono definiti tramite dei metodi il cui nome inizia con le lettere '`test`'. Questa convenzione sul nome informa il test runner di quali sono i metodi che rappresentano dei test.

Il punto cruciale di ogni test è una chiamata ad `assertEqual()` per controllare l'esattezza del risultato; al metodo `assert_()` per verificare una condizione, oppure ad `assertRaises()` per verificare che un'eccezione venga sollevata come ci si aspetta. Questi metodi sono usati in luogo di un'istruzione `assert` così che il test runner possa accumulare tutti i risultati dei vari metodi e produrre un resoconto finale.

Quando viene definito un metodo `setUp()`, il test runner avvierà questo metodo prima di ogni test. Allo stesso modo, se viene definito un metodo `tearDown()`, il test runner invocherà tale metodo dopo l'esecuzione di ogni test. Nell'esempio precedente, `setUp()` veniva usato per creare, per ogni test, una nuova sequenza.

Il blocco finale mostra un modo semplice per eseguire i test. `unittest.main()` mette a disposizione un'interfaccia a riga di comando per testare lo script. Quando viene eseguito da riga di comando, lo script precedente produce un output di questo tipo:

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Invece di `unittest.main()`, esistono altri modi per eseguire i test con un livello di controllo più raffinato, un output meno conciso e senza la necessità di eseguirli da riga di comando. Per esempio, le ultime due righe possono essere sostituite con:

```
suite = unittest.makeSuite(TestSequenceFunctions)
unittest.TextTestRunner(verbosity=2).run(suite)
```

Eseguendo il nuovo script dall'interprete o da un altro script, si ottiene il seguente output:

```
testchoice (__main__.TestSequenceFunctions) ... ok
testsample (__main__.TestSequenceFunctions) ... ok
testshuffle (__main__.TestSequenceFunctions) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.110s

OK
```

Gli esempi precedenti mostrano le funzionalità più comunemente usate di `unittest`, sufficienti ai bisogni di tutti i giorni. Il resto della documentazione approfondisce l'insieme completo delle funzionalità a partire dai principi di base.

5.3.2 Organizzare il codice di test

I componenti di base che costituiscono gli unit testing sono i *test case* (NdT: I "casi da testare") — singoli scenari che devono essere costruiti e testati per provarne la correttezza. In PyUnit, i test case sono rappresentati da istanze della classe `TestCase` nel modulo `unittest`. Per costruire dei vostri test case dovete scrivere delle classi derivate da `TestCase` o usare `FunctionTestCase`.

Un'istanza di una classe derivata da `TestCase` è un oggetto che può eseguire completamente un singolo metodo di test, insieme a codice facoltativo per l'ordinamento e l'impostazione.

Il codice da testare di una istanza `TestCase` dovrebbe esservi interamente contenuto, così che possa essere eseguito sia da solo che in una combinazione arbitraria con un numero qualsiasi di altri test case.

La più semplice sotto classe di test case semplicemente ridefinisce il metodo `runTest()` per permettere di eseguire del codice di test specifico:

```
import unittest

class DefaultWidgetSizeTestCase(unittest.TestCase):
    def runTest(self):
        widget = Widget("The widget")
        self.failUnless(widget.size() == (50,50), 'dimensione predefinita errata')
```

Notate che per testare qualcosa, si usa uno dei metodi `assert*()` o `fail*()`, messi a disposizione dalla classe base `TestCase`. Se il test fallisce quando il caso da testare viene eseguito, viene sollevata un'eccezione e l'ambiente di test identifica il caso da testare come *failure*. Altre eccezioni che non vengono sollevate dai controlli fatti attraverso i metodi `assert*()` e `fail*()` sono identificate come *errors* dall'ambiente di test.

Il modo per eseguire un caso da testare sarà descritto in seguito. Per ora, notate che per costruire un'istanza di questi test case si chiama il suo costruttore senza argomenti:

```
testCase = DefaultWidgetSizeTestCase()
```

Tali test case possono essere numerosi e la loro impostazione può essere ripetitiva. Nel caso precedente, la costruzione di un "Widget" in ognuno dei 100 Widget della classe derivata dei test case porta ad una sgradevole duplicazione.

Fortunatamente, possiamo automatizzare questo istanziamento implementando un metodo chiamato `setUp()` che il nostro ambiente di test chiamerà automaticamente quando verrà eseguito il test:

```
import unittest

class SimpleWidgetTestCase(unittest.TestCase):
    def setUp(self):
        self.widget = Widget("Il widget")

class DefaultWidgetSizeTestCase(SimpleWidgetTestCase):
    def runTest(self):
        self.failUnless(self.widget.size() == (50,50),
                        'dimensione predefinita errata')

class WidgetResizeTestCase(SimpleWidgetTestCase):
    def runTest(self):
        self.widget.resize(100,150)
        self.failUnless(self.widget.size() == (100,150),
                        'dimensione errata dopo il ridimensionamento')
```

Se il metodo `setUp()` riscontra una eccezione durante l'esecuzione del test, l'ambiente considererà il test come non riuscito e non sarà eseguito il metodo `runTest()`.

Analogamente, possiamo prevedere un metodo `tearDown()` che si avvia dopo l'esecuzione del metodo `runTest()`:

```
import unittest

class SimpleWidgetTestCase(unittest.TestCase):
    def setUp(self):
        self.widget = Widget("Il widget")

    def tearDown(self):
        self.widget.dispose()
        self.widget = None
```

Se `setUp()` va a buon fine, il metodo `tearDown()` sarà eseguito senza preoccuparsi se `runTest()` abbia avuto o meno successo.

Un tale ambiente di lavoro per testare il codice è chiamato *fixture*.

Spesso, molti piccoli test case useranno la medesima fixture. In questo caso, faremmo sfociare la classe derivata `SimpleWidgetTestCase` in piccole classi monometodo tipo la `DefaultWidgetSizeTestCase`. Questa pratica è una perdita di tempo ed è deprecata, visto che è sulla falsariga di JUnit, PyUnit fornisce un meccanismo più semplice:

```
import unittest

class WidgetTestCase(unittest.TestCase):
    def setUp(self):
        self.widget = Widget("Il widget")

    def tearDown(self):
        self.widget.dispose()
        self.widget = None

    def testDefaultSize(self):
        self.failUnless(self.widget.size() == (50,50),
                        'dimensione predefinita errata')

    def testResize(self):
        self.widget.resize(100,150)
        self.failUnless(self.widget.size() == (100,150),
                        'dimensione errata dopo il ridimensionamento')
```

Qui non abbiamo fornito un metodo `runTest()`, ma abbiamo invece fornito due metodi di test diversi. Le istanze di classe ora eseguiranno ciascuna uno dei metodi `test*()`, con `self.widget` che verrà creato e distrutto separatamente per ciascuna istanza. Quando creiamo un'istanza, dobbiamo specificare quale metodo di test eseguire. Questo si fa dichiarando il nome del metodo nel costruttore:

```
defaultSizeTestCase = WidgetTestCase("testDefaultSize")
resizeTestCase = WidgetTestCase("testResize")
```

Le istanze dei test case sono raggruppate assieme tenendo conto delle caratteristiche che vanno a testare. PyUnit fornisce un meccanismo per questo: la `test suite` (NdT: "insieme di test"), rappresentata dalla classe `TestSuite` nel modulo `unittest`:

```
widgetTestSuite = unittest.TestSuite()
widgetTestSuite.addTest(WidgetTestCase("testDefaultSize"))
widgetTestSuite.addTest(WidgetTestCase("testResize"))
```

Per facilitare l'esecuzione del test, come si vedrà più avanti, è buona idea inserire in ciascun modulo di test un oggetto richiamabile che restituisca una test suite preconfezionata:

```
def suite():
    suite = unittest.TestSuite()
    suite.addTest(WidgetTestCase("testDefaultSize"))
    suite.addTest(WidgetTestCase("testResize"))
    return suite
```

o addirittura:

```
class WidgetTestSuite(unittest.TestSuite):
    def __init__(self):
        unittest.TestSuite.__init__(self,map(WidgetTestCase,
                                              ("testDefaultSize",
                                               "testResize")))
```

(Il secondo è ammesso per coloro che non siano deboli di cuore!)

Visto che è un comportamento comune creare una classe derivata di `TestCase` con molte funzioni di test dal

nome simile, esiste una utile funzione, chiamata `makeSuite()` che costruisce una test suite contenente tutti i test case presenti in una classe test case:

```
suite = unittest.makeSuite(WidgetTestCase)
```

—-

Notate che quando si utilizza la funzione `makeSuite()`, l'ordine di esecuzione dei vari test case verrà determinato ordinando i nomi delle funzioni tramite la funzione built-in `cmp()`.

Spesso è desiderabile raggruppare insieme le test suite, in modo da eseguire in una volta i test per l'intero sistema. Questo è facile, visto che le istanze di `TestSuite` possono essere aggiunte a `TestSuite` proprio come istanze di `TestCase` possono essere aggiunte a `TestSuite`:

```
suite1 = module1.TheTestSuite()
suite2 = module2.TheTestSuite()
alltests = unittest.TestSuite((suite1, suite2))
```

Potete mettere le definizioni dei test case e delle test suite nello stesso modulo in cui si trova il codice che essi andranno a testare (per esempio 'widget.py'), ma mettere il codice da testare in un modulo separato, come 'widgettests.py', porta diversi vantaggi:

- Il modulo dei test può essere avviato da sé, dalla riga di comando.
- Il codice realizzato per i test può essere facilmente separato dal codice portante.
- Si è meno tentati di cambiare il codice realizzato per i test in modo da adattarlo al codice da testare senza una buona ragione.
- Il codice realizzato per i test dovrebbe essere modificato molto meno di frequente rispetto al codice che esso testa.
- Il codice testato può essere più facilmente rifattorizzato.
- I test per moduli di codice scritto in C dovrebbero comunque essere tenuti in moduli separati, quindi perché non essere coerenti?
- Se cambiasse la strategia da applicare ai test, non sarebbe necessario cambiare il codice sorgente.

5.3.3 Riutilizzare il vecchio codice realizzato per i test

Alcuni utenti si troveranno nella situazione di avere del codice di test esistente che vorrebbero poter lanciare da PyUnit, senza dover convertire ogni vecchia funzione di test in una sotto classe di `TestCase`.

Per questo motivo, PyUnit fornisce una classe `FunctionTestCase`. Questa sotto classe di `TestCase` può essere usata come un involucro per una funzione test esistente. Facoltativamente, possono essere creati involucri per funzioni di set-up e tear-down.

Data la seguente funzione di test:

```
def testSomething():
    something = makeSomething()
    assert something.name is not None
    # ...
```

si può creare una istanza test case equivalente come segue:

```
testcase = unittest.FunctionTestCase(testSomething)
```

Se ci fossero ulteriori metodi di set-up e tear-down che dovrebbero essere richiamati come parte dell'operazione test case, potrebbero anch'essi vernire forniti:

```
testcase = unittest.FunctionTestCase(testSomething,
                                     setUp=makeSomethingDB,
                                     tearDown=deleteSomethingDB)
```

Note: PyUnit supporta l'uso di `AssertionError` come indicatore di fallimento del test, ma questo non è consigliabile. Versioni future potrebbero trattare `AssertionError` in maniera differente.

5.3.4 Classi e funzioni

class `TestCase`()
Le istanze della classe `TestCase` rappresentano le più piccole unità testabili in un insieme di test. Questa classe è pensata per essere usata come classe di base, con specifici test implementati da sotto classi reali. Questa classe implementa l'interfaccia necessaria al test runner per permettergli di guidare il test, con metodi che il codice di test può utilizzare per controllo e per riportare vari tipi di fallimenti.

class `FunctionTestCase`(*testFunc*[, *setUp*[, *tearDown*[, *description*]]])
Questa classe implementa la porzione dell'interfaccia `TestCase` che permette al test runner di guidare il test, ma non fornisce i metodi che il codice approntato può utilizzare per controllarlo e riportare errori. La classe viene usata per creare dei test case usando codice di test ereditato, permettendogli di integrarsi in un framewrok di test basato su `unittest`.

class `TestSuite`([*tests*])
Questa classe rapprenta un'aggregazione di singoli test case e test suite. La classe presenta l'interfaccia necessaria al test runner per consentirgli di funzionare come ogni altro test case, ma eseguendo tutti i test tutte le test suite contenute. Vengono forniti ulteriori metodi per aggiungere test case e test suite all'aggregazione. Se viene passato *tests*, allora esso dev'essere una sequenza di test individuali che andranno aggiunti alla suite.

class `TestLoader`()
Questa classe è responsabile del caricamento dei test in accordo a diversi criteri e li riporta in un involucro costituito da una `TestSuite`. Può caricare tutti i test all'interno di un dato modulo o di una classe `TestCase`. Quando viene caricata da un modulo, essa considera tutte le classi derivate da `TestCase`. Per ognuna di queste classi, essa crea un'istanza per ogni metodo il cui nome inizia con la stringa '`test`'.

`defaultTestLoader`
Istanza della classe `TestLoader` che può essere condivisa. Se non sono necessarie personalizzazioni di `TestLoader`, questa istanza può essere sempre utilizzata invece che crearne di nuove.

class `TextTestRunner`([*stream*[, *descriptions*[, *verbosity*]]])
Un'implementazione di base del test runner che stampa i risultati sullo standard output. Possiede pochi parametri configurabili, ma essenzialmente è molto semplice. Le applicazioni grafiche che eseguono le test suite devono fornire implementazioni alternative.

`main`([*module*[, *defaultTest*[, *argv*[, *testRunner*[, *testRunner*]]]]])
Un programma a riga di comando che esegue un insieme di test; è previsto principalmente per la creazione di moduli di test facilmente eseguibili. L'uso più semplice di questa funzione è:

```
if __name__ == '__main__':
    unittest.main()
```

In alcuni casi, i test esistenti possono essere stati scritti utilizzando il modulo `doctest`. Se è questo il caso, quel modulo fornisce una classe `DocTestSuite` che può costruire automaticamente istanze `unittest.TestSuite` dal codice esistente da testare. Nuovo nella versione 2.3.

5.3.5 Oggetti TestCase

Ogni istanza di `TestCase` rappresenta un singolo test, ma ogni sotto classe concreta può essere utilizzata per definire test multipli — la classe concreta rappresenta una singola test fixture. La fixture viene creata e ripulita per ogni caso da testare.

Le istanze `TestCase` forniscono tre gruppi di metodi: un gruppo è utilizzato per eseguire il test, un altro viene utilizzato dall'implementazione del test per verificare le condizioni e riportare gli errori, mentre alcuni metodi di analisi permettono la raccolta di informazioni riguardanti il test stesso.

I metodi nel primo gruppo sono:

`setUp``()`
: Metodo chiamato per preparare la test fixture. Viene chiamato immediatamente prima di chiamare il metodo di test; ogni eccezione sollevata da questo metodo sarà considerata un errore e non un fallimento del test. L'implementazione predefinita non fa nulla.

`tearDown``()`
: Metodo chiamato immediatamente dopo che il metodo di test è stato chiamato ed il risultato registrato. Viene chiamato anche se il metodo di test solleva un'eccezione, perciò l'implementazione in sotto classi può necessitare di essere particolarmente attenta circa il controllo dello stato interno. Ogni eccezione sollevata da questo metodo sarà considerata un errore e non un fallimento del test. Questo metodo verrà chiamato soltanto nel caso in cui `setUp()` abbia successo, senza riguardo al risultato del metodo di test. L'implementazione predefinita non fa nulla.

`run``(`[*result*]`)`
: Esegue il test, inserendo il risultato nell'oggetto test result passato come *result*. Se *result* viene omesso o è `None`, viene creato ed usato un oggetto result temporaneo, ma non è reso disponibile al chiamante. Questo equivale a chiamare semplicemente l'istanza `TestCase`.

`debug``()`
: Esegue il test senza raccogliere il risultato. Questo permette alle eccezioni sollevate dal test di essere propagate al chiamante, e può essere usato per supportare i test all'interno di un debugger.

Il codice del test può utilizzare uno dei seguenti metodi per verificare e riportare fallimenti.

`assert_``(`*expr*[`,` *msg*]`)`
`failUnless``(`*expr*[`,` *msg*]`)`
: Segnala il fallimento del test se *expr* è falsa; la spiegazione dell'errore si troverà in *msg*, se fornito, altrimenti sarà `None`.

`assertEqual``(`*first*`,` *second*[`,` *msg*]`)`
`failUnlessEqual``(`*first*`,` *second*[`,` *msg*]`)`
: Verifica che *first* e *second* siano uguali. Se il valore risulta diverso, il test fallirà con la spiegazione data da *msg* o `None`. Notate che usando `failUnlessEqual()` si ottiene un miglior risultato che effettuando la comparazione come primo parametro di `failUnless()`: il valore predefinito per *msg* può essere computato per includere le rappresentazioni sia di *first* che di *second*.

`assertNotEqual``(`*first*`,` *second*[`,` *msg*]`)`
`failIfEqual``(`*first*`,` *second*[`,` *msg*]`)`
: Verifica che il *first* e *second* siano diversi. Se il valore risulta uguale, il test fallirà con la spiegazione inserita in *msg* o `None`. Notate che usando `failIfEqual()` si ottiene un risultato migliore che effettuando la comparazione come primo parametro di `failUnless()`: il valore predefinito per *msg* può essere computato per includere le rappresentazioni sia di *first* che di *second*.

`assertAlmostEqual``(`*first*`,` *second*[`,` *places*[`,` *msg*]]`)`
`failUnlessAlmostEqual``(`*first*`,` *second*[`,` *places*[`,` *msg*]]`)`
: Verifica che *first* e *second* siano approssimativamente uguali, calcolandone la differenza, arrotondandola al numero di posizioni decimali(NdT: *places*) passate e confrontandola con zero. Notate che comparare un dato numero di posizioni decimali non è la stessa cosa di comparare un numero che abbia un dato numero di cifre significative. Se i valori non risultano uguali, il test fallisce con la spiegazione data da *msg*, oppure `None`.

`assertNotAlmostEqual``(`*first*`,` *second*[`,` *places*[`,` *msg*]]`)`

failIfAlmostEqual(*first, second*[, *places*[, *msg*]])
Verifica che *first* e *second* non siano approssimativamente uguali calcolandone la differenza, arrotondandola al numero di posizioni decimali(NdT: *places*) passate e confrontandola con zero. Notate che comparare un dato numero di posizioni decimali non è la stessa cosa di comparare un numero che abbia un dato numero di cifre significative. Se i valori non risultano uguali, il test fallisce con la spiegazione data da *msg*, oppure `None`.

assertRaises(*exception, callable, ...*)
failUnlessRaises(*exception, callable, ...*)
Verifica che un'eccezione venga sollevata quando *callable* viene chiamata con un qualsiasi argomento posizionale o a parola chiave che vengono passati anche a `assertRaises()`. Il test passa se viene sollevata *exception*, restituisce errore nel caso l'eccezione sollevata sia un'altra, oppure fallisce se non viene sollevata alcuna eccezione. Per poter catturare insiemi di eccezioni, può essere passato come argomento *exception* una tupla contenente le classi delle eccezioni in questione.

failIf(*expr*[, *msg*])
Il contrario del metodo `failUnless()` è il metodo `failIf()`. Questo metodo segnala un fallimento del test se *expr* è vera, con *msg* o `None` come messaggio d'errore.

fail([*msg*])
Segnala un fallimento del test incondizionatamente, con *msg* o `None` come messaggio d'errore.

failureException
Questo attributo di classe contiene l'eccezione sollevata dal metodo `test()`. Se un ambiente di test necessita l'uso di un'eccezione particolare, forse per trasportare maggiori informazioni, si deve creare una sotto classe di questa eccezione per restare consistenti con l'ambiente. Il valore iniziale di questo attributo è `AssertionError`.

Gli ambienti di test possono utilizzare i seguenti metodi per raccogliere informazioni riguardanti il test:

countTestCases()
Restituisce il numero di test rappresentati da questo oggetto test. Per le istanze di `TestCase`, il risultato sarà sempre `1`, ma questo metodo è implementato anche dalla classe `TestSuite` la quale può restituire valori maggiori.

defaultTestResult()
Restituisce il tipo predefinito dell'oggetto test result che deve essere usato per eseguire il test.

id()
Restituisce una stringa che identifica il test case specifico. Solitamente è il nome completo del metodo di test, incluso il modulo ed il nome della classe.

shortDescription()
Restituisce in un'unica riga la descrizione del test, oppure `None` se non è stata fornita alcuna descrizione. L'implementazione predefinita di questo modulo prevede la restituzione della prima riga della stringa di documentazione del metodo di test, se disponibile, oppure `None`.

5.3.6 Oggetti TestSuite

Gli oggetti `TestSuite` si comportano in maniera simile agli oggetti `TestCase`, tranne per il fatto che non implementano direttamente un test. Vengono usati invece per aggregare in gruppi dei test che devono essere eseguiti insieme. Sono disponibili alcuni metodi addizionali per aggiungere dei test alle istanze di `TestSuite`:

addTest(*test*)
Aggiunge un `TestCase` o una `TestSuite` all'insieme di test che compongono la suite.

addTests(*tests*)
Aggiunge tutti i test da una sequenza di istanze di `TestCase` e `TestSuite` alla test suite.

Anche il metodo `run()` è leggermente diverso:

run(*result*)
Esegue il test associato a questa suite, raccogliendo il risultato nell'oggetto test result passato come *result*. Notate che al contrario di `TestCase.run()`, `TestSuite.run()` richiede che vi sia passato l'oggetto

TestResult.

Nell'utilizzo tipico degli oggetti `TestSuite`, il metodo `run()` viene invocato da `TestRunner` piuttosto che dall'utente finale.

5.3.7 Oggetti TestResult

Un oggetto `TestResult` immagazzina i risultati di una serie di test. Le classi `TestCase` e `TestSuite` assicurano che i risultati siano immagazzinati propriamente; gli autori dei test non devono preoccuparsi di registrazione la conclusione del test.

Gli ambienti di test costruiti su `unittest` possono voler accedere all'oggetto TestResult generato eseguendo una serie di test per fini reportistici; a questo scopo il metodo `TestRunner.run()` restituisce un'istanza di `TestResult`.

Ogni istanza contiene il numero totale di test eseguiti e la raccolta di fallimenti ed erorri occorsi in questi test. La raccolta contiene tuple di (*testcase*, *traceback*), dove *traceback* è una stringa contenente una versione formattata del traceback di quella eccezione.

L'istanza `TestResult` possiede i seguenti attributi che saranno di interesse ispezionando i risultati dell'esecuzione di una serie di test:

`errors`
Una lista contenente coppie di istanze `TestCase` e il traceback formattato dei test che sollevano un'eccezione, ma non segnalano il fallimento del test. Modificato nella versione 2.2: Contiene traceback formattate anziché il risultato di `sys.exc_info()`.

`failures`
Una lista contenente coppie di istanze `TestCase` e il traceback formattato dei test che segnalano un fallimento del codice di test. Modificato nella versione 2.2: Contiene traceback formattate anziché il risultato di `sys.exc_info()`.

`testsRun`
Il numero di test che sono stati avviati.

`wasSuccessful``()`
Restituisce vero se tutti i test eseguiti sino ad ora hanno avuto successo, altrimenti restituisce falso.

I seguenti metodi della classe `TestResult` vengono utilizzati per contenere le strutture dati interne e possono essere estesi in sotto classi per supportare ulteriori richieste di reportistica. Ciò torna particolarmente utile nella costruzione di tool che necessitino una reportistina interattiva durante l'esecuzione dei test.

`startTest`(*test*)
Chiamato quando il test case *test* sta essere lanciato.

`stopTest`(*test*)
Chiamato quando il test case *test* è stato eseguito, senza considerare il risultato.

`addError`(*test, err*)
Chiamato quando il test case *test* solleva un'eccezione senza segnalare il fallimento del test. *err* è una tupla nella forma restituita da `sys.exc_info()`: (*type*, *value*, *traceback*).

`addFailure`(*test, err*)
Chiamato quando il test case *test* segnala un fallimento. *err* è una tupla nella forma restituita da `sys.exc_info()`: (*type*, *value*, *traceback*).

`addSuccess`(*test*)
Questo metodo viene chiamato per un test che non fallisce; *test* è l'oggetto test case.

É disponibile un metodo aggiuntivo per gli oggetti `TestResult`:

`stop``()`
Questo metodo può essere invocato per segnalare che l'insieme dei test in corso dovrebbe essere interrotto. Una volta chiamato, l'oggetto `TestRunner` ritorna al chiamante senza eseguire nessun ulteriore test. Questo metodo viene utilizzato dalla classe `TextTestRunner` per interrompere l'ambiente di test quan-

do l'utente segnala un'interruzione dalla tastiera. I tool interattivi che forniscono degli attivatori possono utilizzarlo in maniera simile.

5.3.8 Oggetti TestLoader

La classe `TestLoader` viene utilizzata per creare test suite da classi e moduli. Normalmente, non c'è bisogno di creare un'istanza di questa classe; il modulo `unittest` fornisce un'istanza che può essere condivisa tramite l'attributo di modulo `defaultTestLoader`. Usare una sotto classe o un'istanza permetterebbe la personalizzazione di alcune proprietà configurabili.

Gli oggetti `TestLoader` possiedono i seguenti metodi:

`loadTestsFromTestCase`(*testCaseClass*)
Restituisce una suite di tutti i test case contenuti nella classe `TestCaseClass` derivata da `testCase`.

`loadTestsFromModule`(*module*)
Restituisce una suite di tutti i test case contenuti nel modulo passato. Questo metodo cerca nel modulo *module* le classi derivate da `TestCase` e ne crea un'istanza per ogni metodo di test definito in quelle classi.

Avvertenze: Mentre può essere conveniente usare una gerarchia di classi derivate `Testcase` per condividere funzioni di aiuto e fixture, risulta scomodo usare questo metodo su classi base che non devono essere istanziate direttamente. Farlo, comunque, può essere utile quando le fixture sono diverse e definite in sotto classi.

`loadTestsFromName`(*name*[, *module*])
Restituisce una suite di tutti i test case data una specifica stringa.

Il nome specificato *name* è un nome puntato che può riferirsi ad un modulo, ad una classe test case, ad un metodo di test all'interno di una classe test case o ad un oggetto richiamabile che restituisca un'istanza `TestCase` o `TestSuite`. Per esempio, se avete un modulo `SampleTests` che contiene una classe `SmapleTestCase` derivata da `TestCase`, con tre metodi di test (`test_uno()`, `test_due()` e `test_tre()`), lo specificatore `'SampleTests.SampleTestCase'` farà in modo che questo metodo restituisca una suite che eseguirà tutti e tre i metodi di test. Usando lo specificatore `'SampleTests.SampleTestCase.test_due'` il metodo restituirà una test suite che eseguirà solo il metodo di test `test_due()`. Lo specificatore può riferirsi a moduli o package che non sono stati importati; verranno importati automaticamente prima dell'esecuzione.

Il metodo, facoltativamente, risolve *name* relativamente al modulo dato.

`loadTestsFromNames`(*names*[, *module*])
Simile a `loadTestsFromName()`, ma riceve come parametro una sequenza di nomi invece che un singolo nome. Il valore restituito è una test suite che supporta tutti i test definiti per ciascun nome.

`getTestCaseNames`(*testCaseClass*)
Restituisce una sequenza ordinata dei nomi dei metodi trovati all'interno di *testCaseClass*.

I seguenti attributi di un `TestLoader` possono essere configurati tramite una sotto classe con l'assegnazione ad un'istanza:

`testMethodPrefix`
Una stringa che fornisce il prefisso dei nomi dei metodi che verranno interpretati come metodi di test. Il valore predefinito è `'test'`.

`sortTestMethodsUsing`
Funzione da usare per confrontare i nomi dei metodi durante l'ordinamento con `getTestCaseNames()`. Il valore predefinito è la funzione built-in `cmp()`; può essere impostata a `None` per disabilitare l'ordinamento.

`suiteClass`
Un oggetto richiamabile che costruisce una test suite da una lista di test. Non è necessario alcun metodo sull'oggetto risultante. Il valore predefinito è la classe `TestSuite`.

5.4 `test` — Package dei test di regressione per Python

Il package `test` contiene tutti i test di regressione per Python insieme ai moduli `test.test_support` e `test.regrtest`. `test.test_support` viene usato per aiutare a migliorare i propri test mentre `test.regrtest` controlla la suite dei test.

Ogni modulo nel package `test`, il cui nome inizia per '`test_`' è una suite di test per un modulo specifico o per uno strumento particolare. Tutti i nuovi test dovrebbero essere scritti usando il modulo `unittest`; l'uso di `unittest` non è necessario ma rende i test più flessibili e la loro manutenzione più semplice. Qualche test più vecchio venne scritto in modo da essere usato con `doctest` usando uno stile di testing "tradizionale"; questi stili non verranno considerati.

Vedete anche:

Modulo `unittest` (sezione 5.3):
: Scrivere test di regressione PyUnit.

Modulo `doctest` (sezione 5.2):
: Test inseriti nelle strighe di documentazione.

5.4.1 Scrivere Unit Tests per il package `test`

È preferibile che i test per il package `test` utilizzino il modulo `unittest` e seguano alcune linee guida. Per prima cosa tutti i nomi dei metodi di test devono iniziare con '`test_`' come anche il nome del modulo. Questo è necessario affinchè il test driver li riconosca come metodi di test. Inoltre, non dovrebbe essere inclusa alcuna stringa di documentazione. Al loro posto si dovrebbe utilizzare un commento (come '`#Le funzioni di test restituiscono solo True o False`') per fornire documentazione al metodo di test. Questo viene fatto perché le stringhe di documentazione (NdT: docstring) se esistono vengono stampate e perciò non risulta chiaro quale test sia in esecuzione.

Spesso viene utilizzata una costruzione basilare tipo questa:

```
import unittest
from test import test_support

class MyTestCase1(unittest.TestCase):

    # Usate solo setUp() e tearDown() se necessario

    def setUp(self):
        ... codice da utilizzare per preparare i test ...

    def tearDown(self):
        ... codice da eseguire per la ripulitura dopo i test ...

    def test_feature_one(self):
        # Test feature one.
... codice di test ...

    def test_feature_two(self):
        # Test feature two.
        ... codice di test ...

    ... altri metodi di test ...

class MyTestCase2(unittest.TestCase):
    ... stessa struttura di MyTestCase1 ...

... altre classi di test

def test_main():
    test_support.run_unittest(MyTestCase1,
                              MyTestCase2,
                              ... elencate altri test ...
                             )

if __name__ == '__main__':
    test_main()
```

Questo codice permette alla suite di test di essere eseguita sia da `test.regrtest` sia autonomamente come script.

Lo scopo dei test di regressione è quello di provocare la "rottura" del codice, facendolo andare in errore. Per cui è necessario seguire alcune linee guida:

- La suite di test deve provare tutte le classi, funzioni e costanti. Questo include non solo l'API esterna da presentare al codice cliente, ma anche il codice privato.
- É preferibile la verifica whitebox(verificare il codice che viene esaminato al momento stesso in cui i test vengono scritti). I test blackbox (testare solo l'interfaccia utente da pubblicare) non sono abbastanza completi da assicurare la verifica di tutti i casi marginali e di contorno.
- Assicuratevi che tutti i possibili valori vengano provati, inclusi quelli non validi. Questo vi garantisce non solo che tutti i valori corretti vengano accettati, ma che anche i valori scorretti vengano gestiti correttamente.
- Esaurite tutti i percorsi possibili all'interno del codice. Testate le ramificazioni e quindi adattate l'input per assicurarvi che vengano scelti più percorsi possibili all'interno del codice.
- Aggiungete un test esplicito per ogni difetto scoperto nel codice testato. Questo assicura che l'errore non si verifichi di nuovo nel caso il codice venga modificato in futuro.
- Assicuratevi di far pulizia dopo i test (come chiudere e cancellare tutti i file temporanei).
- Importate solo i moduli necessari e fatelo il prima possibile. Questo minimizza le dipendenze esterne dei test e riduce i comportamenti anomali causati dagli effetti secondari dell'importazione dei moduli.

- Tentate di massimizzare il riutilizzo del codice. In alcune occasioni, i test varieranno per qualcosa piccolo come il tipo di input utilizzato. Minimizzate la duplicazione del codice derivando una classe di test di base con una classe che specifica l'input:

```
class TestFuncAcceptsSequences(unittest.TestCase):

    func = mySuperWhammyFunction

    def test_func(self):
        self.func(self.arg)

class AcceptLists(TestFuncAcceptsSequences):
    arg = [1,2,3]

class AcceptStrings(TestFuncAcceptsSequences):
    arg = 'abc'

class AcceptTuples(TestFuncAcceptsSequences):
    arg = (1,2,3)
```

Vedete anche:

Test Driven Development
Un libro di Kent Beck riguardante la scrittura dei test prima del codice.

5.4.2 Eseguire test usando `test.regrtest`

`test.regrtest` può essere utilizzato come script per guidare la suite di test di regressione di Python. Eseguendo lo script, automaticamente si avviano tutti i test di regressione nel package `test`. Questo viene fatto trovando tutti i moduli nel package il cui nome inizia con '`test_`', importandoli ed eseguento la funzione `test_main()`, se presente. Il nome dei test da eseguire può anche essere passato allo script. Specificare un singolo test di regressione (**python regrtest.py test_spam.py**) minimizzerà l'output notificando solo il successo od il fallimento del test.

Eseguire direttamente `test.regrtest` permette alle risorse utilizzabili dal test di essere impostate direttamente. Potete farlo utilizzando l'opzione da riga di comando **-u**. Eseguite **python regrtest.py -uall** per abilitare tutte le risorse; specificando **all** come opzione per **-u** vengono abilitate tutte le risorse possibili. Se volete utilizzare tutte le risorse meno una (un caso più comune), potete elencare dopo **all** una lista di tutte le risorse indesiderate separate da una virgola. Il comando **python regrtest.py -uall,-audio,-largefile** eseguirà `test.regrtest` con tutte le risorse ad eccezione di **audio** e **largefile**. Per un elenco di tutte le risorse ed altre opzioni da riga di comando, digitate **python regrtest.py -h**.

Alcuni altri modi di eseguire i test di regressione dipendono dalla piattaforma su cui vengono eseguiti. Su UNIX, potete eseguire **make test** dalla directory principale in cui Python è stato compilato. Su Windows, eseguendo **rt.bat** dalla directory 'PCBuild' avvierete tutti i test di regressione.

5.5 `test.test_support` — Funzioni di utilità per i test

Il modulo `test.test_support` fornisce un supporto ai test di regressione di Python.

Questo modulo definisce le seguenti eccezioni:

exception `TestFailed`
Eccezione sollevata quando un test fallisce.

exception `TestSkipped`
Sottoclasse di `TestFailed`. Sollevata quando un test viene saltato. Si presenta quando una risorsa richiesta (come una connessione di rete) non è disponibile al momento del test.

exception `ResourceDenied`
Sottoclasse di `TestSkipped`. Sollevata quando una risorsa (come una connessione di rete) non è disponibile. Viene sollevata dalla funzione `requires()`.

Il modulo `test.test_support` definisce le seguenti costanti:

`verbose`
`True` quando l'output è abilitato. Dovrebbe essere controllato quando sono richieste informazioni più dettagliate circa l'esecuzione di un test. *verbose* viene impostata da `test.regrtest`.

`have_unicode`
`True` quando è disponibile il supporto per Unicode.

`is_jython`
`True` se l'interprete è Jython.

`TESTFN`
Imposta il percorso in cui può essere creato un file temporaneo. Ogni file temporaneo che viene creato deve essere chiuso e delinkato (rimosso).

Il modulo `test.test_support` definisce le seguenti funzioni:

`forget`(*module_name*)
Rimuove il modulo chiamato *module_name* da `sys.modules` e rimuove ogni file del modulo compilato in bytecode.

`is_resource_enabled`(*resource*)
Restituisce `True` se la risorsa *resource* è abilitata e disponibile. L'elenco delle risorse disponibili viene impostata solo quando `test.regrtest` sta eseguendo il test.

`requires`(*resource*[, *msg*])
Solleva `ResourceDenied` se la risorsa *resource* non è disponibile. *msg* è l'argomento di `ResourceDenied` se questa viene sollevata. Restituisce sempre vero se chiamata da una funzione il cui `__name__` è `'__main__'`. Utilizzata quando i test sono eseguiti da `test.regrtest`.

`findfile`(*filename*)
Restituisce il percorso al file chiamato *filename*. Se non c'è corrispondenza, viene restituito *filename*. Questo non è equiparabile ad un fallimento poiché può essere il percorso del file.

`run_unittest`(**classes*)
Esegue le sotto classi di `unittest.TestCase` passate alla funzione. La funzione analizza le classi alla ricerca di metodi che iniziano con il prefisso '`test_`' ed esegue i test individualmente. Questo è il modo preferenziale per eseguire i test.

`run_suite`(*suite*[, *testclass*])
Esegue l'istanza *suite* di `unittest.TestSuite`. L'argomento facoltativo *testclass* accetta una delle classi di test presenti nella suite, così da stampare informazioni più dettagliate sulla provenienza della suite di test.

5.6 `math` — Funzioni matematiche

Questo modulo è sempre disponibile. Fornisce l'accesso alle funzioni matematiche definite dallo standard C.

Queste funzioni non possono essere usate con i numeri complessi; usate le funzioni con lo stesso nome dal modulo `cmath` se avete bisogno del supporto per i numeri complessi. Visto che molti utilizzatori non intendono acquisire le necessarie conoscenze matematiche per comprendere i numeri complessi, si è deciso di porre una distinzione tra le funzioni che li supportano e quelle che non lo fanno. Ricevere un'eccezione piuttosto che un risultato complesso consente di rilevare immediatamente la presenza di inaspettati numeri complessi usati come un parametro, cosìcchè il programmatore possa determinare come e perché l'eccezione venne inizialmente generata.

Le seguenti funzioni vengono fornite da questo modulo. Tranne quando esplicitamente dichiarato, tutti i valori restituiti sono numeri in virgola mobile.

`acos`(*x*)

Restituisce l'arcocoseno di *x*.

asin(*x*)
Restituisce l'arcoseno di *x*.

atan(*x*)
Restituisce l'arcotangente di *x*.

atan2(*y*, *x*)
Restituisce `atan(`*y* `/` *x*`)`.

ceil(*x*)
Restituisce l'approssimazione per eccesso di *x* come numero in virgola mobile.

cos(*x*)
Restituisce il coseno di *x*.

cosh(*x*)
Restituisce il coseno iperbolico di *x*.

degrees(*x*)
Converte l'angolo *x* da radianti a gradi.

exp(*x*)
Restituisce `e**`*x*.

fabs(*x*)
Restituisce il valore assoluto di *x*.

floor(*x*)
Restituisce l'approssimazione per difetto di *x* come numero in virgola mobile.

fmod(*x*, *y*)
Restituisce `fmod(`*x*`, `*y*`)`, come definito nella libreria C della piattaforma. Notate che l'espressione Python *x* `%` *y* può non restituire lo stesso lo stesso risultato.

frexp(*x*)
Restituisce la mantissa e l'esponente di *x* come una coppia `(`*m*`, `*e*`)`. *m* è un numero in virgola mobile ed *e* è un numero intero come nell'espressione *x* `==` *m* `*` `2**`*e*. Se *x* è zero, restituisce `(0.0, 0)`, altrimenti `0.5 <= abs(`*m*`) < 1`.

hypot(*x*, *y*)
Restituisce la distanza euclidea, `sqrt(`*x*`*`*x* `+` *y*`*`*y*`)`.

ldexp(*x*, *i*)
Restituisce *x* `*` `(2**`*i*`)`.

log(*x*[, *base*])
Restituisce il logaritmo di *x* in base *base*. Se la base *base* non è specificata, restituisce il logaritmo naturale di *x*. Modificato nella versione 2.3: Aggiunto l'argomento *base*.

log10(*x*)
Restituisce il logaritmo di *x* in base 10.

modf(*x*)
Restituisce la parte frazionaria ed intera di *x*. Entrambi i risultati riportano il segno di *x*. la parte intera è restituita come numero in virgola mobile.

pow(*x*, *y*)
Restituisce *x*`**`*y*.

radians(*x*)
Converte l'angolo *x* da gradi a radianti.

sin(*x*)
Restituisce il seno di *x*.

sinh(*x*)
Restituisce il seno iperbolico di *x*.

sqrt(*x*)
Restituisce la radice quadrata di *x*.

tan(*x*)
Restituisce la tangente di *x*.

tanh(*x*)
Restituisce la tangente iperbolica di *x*.

Notate che `frexp()` e `modf()` hanno un modello di chiamata/risultato diverso dalle equivalenti chiamate C: esse richiedono un singolo argomento e restituiscono una coppia di valori, invece che restituire il loro secondo risultato per mezzo di un 'parametro di output' (non esiste una cosa del genere in Python).

Il modulo definisce anche due costanti matematiche:

pi
La costante matematica *pi* (NdT: Pi greco).

e
La costante matematica *e*.

Note: Il modulo matematico `math` consiste per lo più di sottili involucri costruiti attorno alle funzioni matematiche della libreria C della piattaforma sottostante. Il comportamento in casi eccezionali è specificato in modo vago dagli standard del linguaggio C, e Python eredita gran parte del funzionamento dei messaggi di errore delle funzioni matematiche proprio dall'implementazione C della piattaforma sottostante. Come risultato, le eccezioni specifiche sollevate in caso di errore (e anche se alcuni argomenti sono considerati del tutto eccezionali) non vengono definite in una maniera utile che sia portabile tra le piattaforme o le release. Per esempio, non sappiamo se `math.log(0)` debba restituire `-Inf` o sollevare (`ValueError`) o (`OverflowError`), e in alcuni casi quando `math.log(0)` solleva `OverflowError`, `math.log(0L)` potrebbe sollevare `ValueError`.

Vedete anche:

Modulo `cmath` (sezione 5.7):
La versione per numeri complessi della gran parte di queste funzioni.

5.7 `cmath` — Funzioni matematiche per i numeri complessi

Questo modulo è sempre disponibile. Fornisce le funzioni matematiche per i numeri complessi. Le funzioni disponibili sono le seguenti:

acos(*x*)
Restituisce l'arcocoseno di *x*. Esistono due rami di funzione ("branch cuts"): uno si estende verso destra da 1 a ∞lungo l'asse reale, continuo dal basso. L'altro si estende verso sinistra da -1 a -∞, lungo l'asse reale, continuo dall'alto.

acosh(*x*)
Restituisce l'arcocoseno iperbolico di *x*. Esiste un ramo di funzione che si estende verso sinistra da 1 a -∞lungo l'asse reale, continuo dall'alto.

asin(*x*)
Restituisce l'arco seno di *x*. Possiede gli stessi rami di funzione di `acos()`.

asinh(*x*)
Restituisce l'arcoseno iperbolico di *x*. Esistono due rami di funzione che si estendono verso sinistra da $\pm$`1j` a $\pm$-∞`j`, entrambi continui dall'alto. Questi rami di funzione dovrebbero essere considerati un errore da correggere nelle prossime versioni. I rami di funzione corretti dovrebbero estendersi lungo l'asse immaginario, uno da `1j` fino a ∞`j` e continuo da destra e uno da `-1j` a -∞`j` continuo da sinistra.

atan(*x*)
Restituisce l'arcotangente di *x*. Esistono due rami di funzione: uno si estende lungo l'asse immaginario da `1j` a ∞`j`, continuo da sinistra. L'altro si estende lungo l'asse immaginario da `-1j` a -∞`j`, continuo da sinistra. (Questo metodo probabilmente dovrebbe venire modificato così che la diramazione superiore diventi continua dall'altro lato.)

atanh(*x*)
Restituisce l'arcotangente iperbolica di *x*. Esistono due rami di funzione: uno si estende lungo l'asse reale da 1 a ∞, continuo dall'alto. L'altro si estende lungo l'asse reale da -1 a -∞, continuo dall'alto. (Questo metodo probabilmente dovrebbe venire modificato così che la diramazione di destra diventi continua dall'altro lato.)

cos(*x*)
Restituisce il coseno di *x*.

cosh(*x*)
Restituisce il coseno iperbolico di *x*.

exp(*x*)
Restituisce il valore esponenziale `e**`*x*.

log(*x*)
Restituisce il logaritmo naturale di *x* Esiste un ramo di funzione da 0 a -∞lungo l'asse reale negativa, continuo dall'alto.

log10(*x*)
Restituisce il logaritmo in base 10 di *x*. Possiede lo stesso ramo di funzione di `log()`.

sin(*x*)
Restituisce il seno di *x*.

sinh(*x*)
Restituisce il seno iperbolico di *x*.

sqrt(*x*)
Restituisce la radice quadrata di *x*. Possiede lo stesso ramo di funzione di `log()`.

tan(*x*)
Restituisce la tangente di *x*.

tanh(*x*)
Restituisce la tangente iperbolica di *x*.

Il modulo definisce anche due costanti matermatiche:

pi
La costante matematica *pi* (NdT: pi greco), come un reale.

e
La costante matematica *e*, come un reale.

Notate che la selezione di funzioni è simile, ma non identica, a quella del modulo `math`. La ragione di avere due moduli è che molti utilizzatori non sono interessati ai numeri complessi, e magari non sanno neppure cosa siano. Essi preferiscono che `math.sqrt(-1)` sollevi un'eccezione piuttosto che restituire un numero complesso. Inoltre osservate che le funzioni definite in `cmath` restituiscono sempre un numero complesso anche se il risultato può essere espresso come numero reale (nel caso in cui il numero complesso abbia la parte immaginaria uguale a zero).

Una nota sui rami di funzione: Essi sono curve lungo le quali la funzione data smette di essere continua. Questa è una caratteristica necessaria in molte funzioni complesse. Si suppone che se voi avete la necessità di lavorare con le funzioni complesse, dovrete conoscere il significato dei rami di funzione. Consultate un qualunque testo (non troppo elementare) che tratti delle variabili complesse per saperne di più. Per informazioni sulla scelta adatta dei rami di funzione per gli usi numerici, un buon riferimento può essere il seguente:

Vedete anche:

Kahan, W: Branch cuts for complex elementary functions; or, Much ado about nothings's sign bit. In Iserles, A., and Powell, M. (eds.), *The state of the art in numerical analysis*. Clarendon Press (1987) pp165-211.

5.8 `random` — Genera numeri pseudo casuali

Questo modulo implementa generatori di numeri pseudo casuali per varie distribuzioni.

Per numeri interi, la selezione avviene uniformemente da un intervallo. Per le sequenze, si seleziona uniformemente un elemento casuale, vengono fornite anche una funzione per generare una permutazione casuale sul posto di una lista ed una funzione per la campionatura casuale senza sostituzione.

Nel campo dei numeri reali, esistono funzioni per calcolare distribuzioni uniformi, normali (gaussiane), normali logaritmiche, esponenziali negative, gamma e beta. Per generare distribuzioni di angoli, è disponibile la distribuzione von Mises.

Quasi tutte le funzioni di questo modulo dipendono dalla funzione fondamentale `random()`, che genera uniformemente un numero in virgola mobile casuale nell'intervallo semi aperto [0.0; 1.0]. Python utilizza il Mersenne Twister come generatore principale. Esso produce numeri in virgola mobile con la precisione di 53 bit e dispone di un periodo di 2**19937-1. L'implementazione sottostante in C è sia veloce che sicura per i thread. Il Mersenne Twister è uno dei generatori di numeri casuali esistente più testati. Nonostante ciò, essendo completamente deterministico, non è adatto a tutti gli usi, ed è del tutto inutilizzabile per scopi crittografici.

Le funzioni fornite da questo modulo sono in realtà metodi legati ad un'istanza nascosta della classe `random.Random`. Potete creare le vostre istanze personali di `Random` per ottenere generatori indipendenti. Questo è utile soprattutto per programmi multi thread, creando differenti istanze di `Random` per ogni thread e usando il metodo `jumpahead()` si assicura che le sequenze generate in ogni thread non si sovrappongano.

Dalla classe `Random` si possono anche derivare delle sotto classi, nel caso in cui vogliate utilizzare per i vostri scopi un generatore di base differente: in quel caso, ridefinite i metodi `random()`, `seed()`, `getstate()`, `setstate()` e `jumpahead()`. Facoltativamente, un nuovo generatore può sostituire il metodo `getrandombits()` — questo permette a `randrange()` di produrre selezioni in una sequenza arbitrariamente grande. Nuovo nella versione 2.4: il metodo `getrandombits()`.

Come esempio di derivazione, il modulo `random` fornisce la classe `WichmannHill` che implementa un generatore alternativo in puro Python. La classe fornisce un modo retrocompatibile per riprodurre i risultati delle vecchie versioni di Python che usavano l'algoritmo Wichmann-Hill come generatore principale. Modificato nella versione 2.3: Sostituito Wichmann-Hill con MersenneTwister.

Funzioni di calcolo:

`seed`([*x*])
: Inizializza il generatore di base dei numeri casuali. L'argomento facoltativo *x* può essere un qualunque oggetto di cui si possa calcolare l'hash (NdT: hashable). Se *x* viene omesso o è `None`, si utilizza il tempo di sistema corrente; il tempo di sistema corrente viene usato anche per inizializzare il generatore quando si importa il modulo per la prima volta. Se l'argomento non è ne `None` ne un int e neppure un long, viene utilizzato `hash(`*x*`)`. Se *x* è un int o un long, si usa direttamente *x*.

`getstate`()
: Restituisce un oggetto che contiene lo stato interno corrente del generatore. Questo oggetto può essere passato a `setstate()` per ristabilire lo stato. Nuovo nella versione 2.1.

`setstate`(*state*)
: *state* deve essere un oggetto ottenuto da una precedente chiamata a `getstate()`, e `setstate()` ripristina lo stato interno del generatore come era al momento della chiamata di `setstate()`. Nuovo nella versione 2.1.

`jumpahead`(*n*)
: Cambia lo stato interno in uno differente e probabilmente lontano dallo stato corrente. *n* è un intero non negativo che viene usato per rimescolare il vettore corrente degli stati. Questo è molto utile nei programmi multi thread, in combinazione con più istanze della classe `Random`: `setstate()` o `seed()` possono essere utilizzate per forzare tutte le instanze ad avere lo stesso stato, mentre `jumpahead()` ne forza l'allontanamento. Nuovo nella versione 2.1. Modificato nella versione 2.3: Invece di saltare ad uno specifico stato, *n* passi più avanti, `jumpahead(`*n*`)` salta ad un altro stato probabilmente separato da molti passi..

`getrandbits`(*k*)
: Restituisce un intero `long` Python con *k* bit casuali. Questo metodo viene fornito con il generatore MersenneTwister ed anche alcuni altri generatori possono fornirlo come parte opzionale dell'API. Quando disponibile, `getrandbits()` abilita `randrange()` a gestire sequenze arbitrariamente grandi. Nuovo nella versione 2.4.

Funzioni per gli interi:

randrange([*start,*] *stop*[, *step*])
: Restituisce un elemento casuale nell'intervallo generato da `range(`*start*`, `*stop*`, `*step*`)`. È equivalente a `choice(range(`*start*`, `*stop*`, `*step*`))`, ma non costruisce un oggetto range. Nuovo nella versione 1.5.2.

randint(*a*, *b*)
: Restituisce un intero casuale *N* tale che *a* `<=` *N* `<=` *b*.

Funzioni per le sequenze:

choice(*seq*)
: Restituisce un elemento casuale della sequenza non vuota *seq*.

shuffle(*x*[, *random*])
: Rimescola la sequenza *x* sul posto. L'argomento facoltativo *random* è una funzione priva argomenti che restituisce un numero casuale in virgola mobile nell'intervallo [0.0; 1.0]; il valore predefinito è la funzione `random()`.

  Notate che anche per `len(`*x*`)` piuttosto piccolo, il numero totale di permutazioni di *x* è più grande del periodo della maggior parte dei generatori di numeri casuali; questo implica che la maggior parte delle permutazioni di una lunga sequenza non potrà mai essere generata.

sample(*population, k*)
: Restituisce una lista di lunghezza *k* di elementi distinti, scelti dalla sequenza population. Viene usato per la campionatura casuale senza sostituzione. Nuovo nella versione 2.3.

  Restituisce una nuova lista contenente elementi presi da population lasciando immutata la population originale. La lista ottenuta viene restituita in ordine di selezione così che ogni sotto lista sia un campionamento casuale valido. Questo permette di individuare tra i vincitori (il campionamento) il primo ed i secondi posti (le sotto liste).

  Non è necessario che gli elementi di population siano utilizzabili da hash o che siano elementi distinti. Se population contiene elementi ripetuti, ogni occorrenza è una possibile selezione del campione.

  Per scegliere un campione da un intervallo di interi, usate `xrange` come argomento. Questo è un metodo particolarmente veloce ed economico per la memoria di ottenere un campione da un'ampia popolazione: `sample(xrange(10000000), 60)`.

Le seguenti funzioni generano specifiche distribuzioni di numeri reali. I parametri delle funzioni vengono ricalcati in base ai nomi delle variabili corrispondenti nelle equazioni delle distribuzioni, come si usa nella comune terminologia matematica; la maggior parte di queste equazioni si possono trovare in un qualunque testo di statistica.

random()
: Restituisce il prossimo numero in virgola mobile nell'intervallo [0.0; 1.0].

uniform(*a*, *b*)
: Restituisce un numero reale casuale *N* tale che *a* `<=` *N* `<` *b*.

betavariate(*alpha, beta*)
: Distribuzione Beta. Le condizioni per i parametri sono: *alpha* `>` `-1` e *beta* `>` `-1`. I valori restituiti si trovano nell'intervallo fra 0 e 1.

expovariate(*lambd*)
: Distribuzione esponenziale. *lambd* è 1.0 diviso la media desiderata. (Il parametro doveva chiamarsi "lambda", ma questa è una parola riservata in Python.) Il valore restituito è compreso tra 0 e l'infinito positivo.

gammavariate(*alpha, beta*)
: Distribuzione Gamma. (*Non* funzione gamma!) Le condizioni sui parametri sono *alpha* `>` `0` e *beta* `>` `0`.

gauss(*mu, sigma*)
: Distribuzione gaussiana. *mu* è la media e *sigma* è la deviazione standard. Questa è leggermente più veloce della funzione `normalvariate()` definita qui sotto.

lognormvariate(*mu, sigma*)
: Distribuzione normale logaritmica. Se prendete il logaritmo naturale di questa distribuzione, otterrete la

distribuzione normale con media *mu* e deviazione standard *sigma*. *mu* può avere ogni valore mentre *sigma* deve essere maggiore di zero.

normalvariate(*mu, sigma*)
Distribuzione normale. *mu* è la media e *sigma* è la deviazione standard.

vonmisesvariate(*mu, kappa*)
mu è l'angolo medio, espresso in radianti con un valore compreso tra 0 e 2**pi*, mentre *kappa* è il parametro di concentrazione, che deve essere maggiore o uguale a zero. Se *kappa* è uguale a zero, questa distribuzione si restringe ad un angolo casuale costante, compreso nell'intervallo tra 0 e 2**pi*.

paretovariate(*alpha*)
Distribuzione Pareto. *alpha* è il parametro di forma.

weibullvariate(*alpha, beta*)
Distribuzione Weibull. *alpha* è il parametro di scala e *beta* il parametro di forma.

Generatori alternativi

class WichmannHill([*seed*])
Classe che implementa l'algoritmo Wichmann-Hill come generatore principale. Possiede gli stessi metodi di `Random` più il metodo `whseed` descritto sotto. Poiché questa classe è implementata in puro Python, non è threadsafe e potrebbe richiedere lock tra le chiamate. Il periodo del generatore è 6.953.607.871.644 che è abbastanza piccolo da richiedere attenzione affichè due sequenze casuali indipendenti non si sovrappongano.

whseed([*x*])
Questo metodo è obsoleto, supportato per la compatibilità a livello di bit con le versioni di Python precedenti alla 2.1. Vedete `seed` per i dettagli. `whseed` non garantisce che diversi argomenti interi producano diversi stati interni e non può generare più di 2**24 stati interni distinti.

Vedete anche:

M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator", *ACM Transactions on Modeling and Computer Simulation* Vol. 8, No. 1, January pp.3-30 1998.

Wichmann, B. A. & Hill, I. D., "Algorithm AS 183: An efficient and portable pseudo-random number generator", *Applied Statistics* 31 (1982) 188-190.

5.9 whrandom — Generatore di numeri pseudo casuali

Deprecato dalla versione 2.1. Usate `random` al suo posto.

Note: Questo modulo era un componente del modulo `random` nelle versioni di Python antecedenti alla 2.1. Non viene più utilizzato. Non usate questo modulo direttamente; usate `random` al suo posto.

Questo modulo implementa una classe generatrice di numeri pseudo casuali Wichmann-Hill chiamata anche `whrandom`. Le istanze della classe `whrandom` sono conformi all'interfaccia Random Number Generator descritta nella sezione **??**. Esse offrono inoltre i seguenti metodi, specifici all'algoritmo Wichmann-Hill:

seed([*x, y, z*])
Inizializza il generatore di numeri casuali partendo dagli interi *x*, *y* and *z*. Quando il modulo è importato per la prima volta viene inizializzato con dei valori ricavati dall'orario corrente. Se *x*, *y* e *z* sono tutti `0` o vengono omessi, il seme verrà computato dall'orario corrente. Se uno o due dei parametri sono `0`, ma non tutti tre, i valori zero vengono sostituiti da valori uno. Come conseguenza alcuni semi di inizializzazione apparentemente differenti sono uguali, con le relative conseguenze nella serie di numeri pseudo casuali prodotti dal generatore.

choice(*seq*)
Sceglie un elemento casuale nalla sequenza non vuota *seq* e lo restituisce.

randint(*a, b*)
Restituisce un numero casuale intero *N* tale che *a*<=*N*<=*b*.

random()

Restituisce il prossimo numero casuale in virgola mobile nell'intervallo [0.0...1.0).

`seed`(*x*, *y*, *z*)
Inizializza il generatore di numeri casuali con gli interi *x*, *y* e *z*. Quando il modulo viene importato per la prima volta il numero casuale viene inizializzato usando valori derivati dall'orario corrente.

`uniform`(*a*, *b*)
Restituisce un numero reale casuale *N* tale che *a*<=*N*<*b*.

Quando viene importato, il modulo `whrandom` crea anche un'istanza della classe `whrandom`, e rende disponibili i metodi di questa istanza al livello del modulo. Quindi si può scrivere sia `N = whrandom.random()` che:

```
generator = whrandom.whrandom()
N = generator.random()
```

Notate che usare istanze separate del generatore conduce a sequenze di numeri pseudo casuali indipendenti.

Vedete anche:

Modulo `random` (sezione 5.8):
Generatori per varie distribuzioni casuali e documentazione dell'interfaccia del generatore di numeri casuali (Random Number Generator interface).

Wichmann, B. A. & Hill, I. D., "Algorithm AS 183: An efficient and portable pseudo-random number generator", *Applied Statistics* 31 (1982) 188-190.

5.10 `bisect` — Algoritmo di bisezione di array

Questo modulo da la possibilità di mantenere una lista sempre ordinata senza la necessita di effettuare l'ordinamento dopo ogni inserzione. Per lunghe liste di elementi con operazioni di confronto dispendiose, grazie a questo modulo si può ottenere un miglioramento rispetto all'approccio più comune. Il modulo è chiamato `bisect` poiché internamente utilizza un algoritmo di bisezione. Il codice sorgente può essere molto utile come esempio di funzionamento dell'algoritmo (le condizioni al contorno vengono già definite!).

Vengono fornite le seguenti funzioni:

`bisect_left`(*list*, *item*[, *lo*[, *hi*]])
Individua l'esatto punto di inserimento dell'elemento *item* nella lista *list* in modo da mantenere l'ordine. I parametri *lo* e *hi* possono venire utilizzati per specificare il sotto insieme della lista da considerare; se non specificati, viene utilizzata l'intera lista. Se *item* è già presente in *list*, il punto di inserimento si troverà prima (a sinistra) di ogni voce esistente di quell'elemento. Il valore restituito è utilizzabile come primo parametro del metodo *list*`.insert()`. Si presuppone che *list* sia già ordinata. Nuovo nella versione 2.1.

`bisect_right`(*list*, *item*[, *lo*[, *hi*]])
Simile a `bisect_left()`, ma restituisce un punto di inserzione dopo (a destra) ogni voce esistente di *item* in *list*. Nuovo nella versione 2.1.

`bisect`(...)
Sinonimo di `bisect_right()`.

`insort_left`(*list*, *item*[, *lo*[, *hi*]])
Inserisce *item* in *list* in modo da mantenere l'ordine. È equivalente a *list*`.insert(bisect.bisect_left(`*list*`,` *item*`,` *lo*`,` *hi*`),` *item*`)`. Si presuppone che *list* sia già ordinata. Nuovo nella versione 2.1.

`insort_right`(*list*, *item*[, *lo*[, *hi*]])
Simile a `insort_left()`, ma *item* viene inserito in *list* dopo tutte le occorrenze di *item* presenti. Nuovo nella versione 2.1.

`insort`(...)
Sinonimo di `insort_right()`.

5.10.1 Esempi

La funzione `bisect()` generalmente è utile per classificare dati numerici. Questo esempio utilizza `bisect()` per valutare con una scala in lettere un esame scolastico classificato con risultati numerici: 'A' significa oltre 85, 'B' da 75 a 84, ecc.

```
>>> grades = "FEDCBA"
>>> breakpoints = [30, 44, 66, 75, 85]
>>> from bisect import bisect
>>> def grade(total):
...           return grades[bisect(breakpoints, total)]
...
>>> grade(66)
'C'
>>> map(grade, [33, 99, 77, 44, 12, 88])
['E', 'A', 'B', 'D', 'F', 'A']
```

5.11 `collections` — Tipi di dato contenitore ad alte prestazioni

Nuovo nella versione 2.4.

Questo modulo implementa tipi di dato contenitore ad alte prestazioni. Al momento l'unico tipo di dato è una deque. Future integrazioni potranno comprendere B-tree ed heap di Fibonacci.

`deque`([*iterable*])
Restituisce un nuovo oggetto deque inizializzato da sinistra a destra (mediante `append()`) con i dati presi da *iterable*. Se *iterable* non viene specificato, il nuovo deque è vuoto.

I deque sono una generalizzazione delle pile e delle code (si pronuncia "deck" ed è l'abbreviazione di "double-ended queue"). I deque supportano il thread-safe ed i metodi append e pop sono efficienti nell'uso della della memoria da entrambe le estremità del deque con approssimativamente le stesse prestazioni `O(1)` in entrambe le direzioni.

Sebbene gli oggetti `list` supportino operazioni simili, i deque sono ottimizzati per operazioni veloci a lunghezza fissa e comportano un costo di `O(n)` spostamenti in memoria per le operazioni '`pop(0)`' e '`insert(0, v)`' che modificano sia la dimensione che la posizione della rappresentazione sottostante dei dati. Nuovo nella versione 2.4.

Gli oggetti deque supportano i seguenti metodi:

`append`(*x*)
aggiunge *x* all'estremità destra del deque.

`appendleft`(*x*)
Aggiunge *x* all'estremità sinistra del deque.

`clear`()
Rimuove tutti gli elementi dal deque riducendolo a lunghezza 0.

`extend`(*iterable*)
Estende l'estremità destra del deque aggiungendo gli elementi presi dall'argomento iterabile *iterable*.

`extendleft`(*iterable*)
Estende l'estremità sinistra del deque aggiungendo gli elementi presi dall'argomento iterabile *iterable*. Notate che la serie di inserimenti a sinistra del deque corrisponde ad invertire l'ordine degli elementi dell'argomento *iterable*.

`pop`()
Rimuove e restituisce un elemento dall'estremità destra del deque. Se non sono presenti elementi viene sollevata un'eccezione `IndexError`.

popleft()
: Rimuove e restituisce un elemento dall'estremità sinistra del deque. Se non sono presenti elementi viene sollevata un'eccezione `IndexError`.

rotate(*n*)
: Ruota il deque di *n* posizioni verso destra. Se *n* è negativo, ruota a sinistra. Ruotare di una posizione a destra equivale a: '`d.appendleft(d.pop())`'.

In aggiunta ai metodi citati, i deque supportano l'iterazione, il pickling, '`len(d)`', '`reversed(d)`', '`copy.copy(d)`', '`copy.deepcopy(d)`', il controllo di appartenenza con l'operatore `in` ed i riferimenti del tipo '`d[-1]`'.

Esempio:

```
>>> from collections import deque
>>> d = deque('ghi')                 # crea un nuovo deque con tre elementi
>>> for elem in d:                   # itera gli elementi del deque
...     print elem.upper()
G
H
I

>>> d.append('j')                    #  aggiunge un nuovo elemento
                                     #+ all'estremità destra
>>> d.appendleft('f')                #  aggiunge un nuovo elemento
                                     #+ all'estremità sinistra
>>> d                                #  mostra il contenuto del deque
deque(['f', 'g', 'h', 'i', 'j'])

>>> d.pop()                          #  restituisce e rimuove
                                     #+ l'elemento più a destra
'j'
>>> d.popleft()                      #  restituisce e rimuove
                                     #+ l'elemento più a sinistra
'f'
>>> list(d)                          #  elenca il contenuto del deque
['g', 'h', 'i']
>>> d[0]                             #  punta all'elemento più a sinistra
'g'
>>> d[-1]                            #  punta all'elemento più a destra
'i'

>>> list(reversed(d))                #  elenca il contenuto del deque
                                     #+ in ordine inverso
['i', 'h', 'g']
>>> 'h' in d                         #  cerca nel deque
True
>>> d.extend('jkl')                  #  aggiunge più elementi in una volta
>>> d
deque(['g', 'h', 'i', 'j', 'k', 'l'])
>>> d.rotate(1)                      #  rotazione a destra
>>> d
deque(['l', 'g', 'h', 'i', 'j', 'k'])
>>> d.rotate(-1)                     #  rotazione a sinistra
>>> d
deque(['g', 'h', 'i', 'j', 'k', 'l'])

>>> deque(reversed(d))               #  crea un nuovo deque rovesciato
deque(['l', 'k', 'j', 'i', 'h', 'g'])
>>> d.clear()                        #  svuota il deque
>>> d.pop()                          #  non si può eseguire pop su un
                                     #+ deque vuoto
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in -toplevel-
    d.pop()
IndexError: pop from an empty deque

>>> d.extendleft('abc')              #  extendleft() inverte l'ordine
                                     #+ di input
>>> d
deque(['c', 'b', 'a'])
```

5.11.1 Ricette

Questa sezione mostra vari approcci per lavorare con i deque.

Il metodo `rotate()` fornisce un modo per implementare l'affettamento e la cancellazione dei `deque`:

Quest'implementazione in puro python di `del d[n]` mostra l'utilizzo del metodo `rotate()` come base di partenza per implementare diverse operazioni di `deque`:

```
def delete_nth(d, n):
    d.rotate(-n)
    d.popleft()
    d.rotate(n)
```

Per implementare l'affettamento dei `deque`, si usa un approccio simile, applicando `rotate()` per portare l'elemento desiderato all'estremità sinistra del deque. Si rimuovono i vecchi elementi con `popleft()`, si aggiungono i nuovi elementi con `extend()`, quindi si inverte la rotazione.

Apportando variazioni minime a questo approccio è facile implementare manipolazioni di stack in stile Forth quali `dup`, `drop`, `swap`, `over`, `pick`, `rot` e `roll`.

Un task server roundrobin può essere costruito partendo da un `deque` ed usando `popleft()` per selezionare il task corrente ed `append()` per aggiungerlo in fondo alla tasklist se il flusso di input non si è esaurito:

```
def roundrobin(*iterables):
    pending = deque(iter(i) for i in iterables)
    while pending:
        task = pending.popleft()
        try:
            yield task.next()
        except StopIteration:
            continue
        pending.append(task)

>>> for value in roundrobin('abc', 'd', 'efgh'):
...     print value

a
d
e
b
f
c
g
h
```

Gli algoritmi di riduzione dei dati multi-pass possono essere espressi in maniera succinta e codificati efficentemente estraendo gli elementi mediante chiamate multiple a `popleft()`, applicando la funzione di riduzione ed usando `append()` per riporre il risultato nella queue.

Per esempio, la costruzione di un albero binario bilanciato di liste annidate si ottiene riducendo due nodi adiacenti ad uno solo raggruppandoli in una lista:

```
def maketree(iterable):
    d = deque(iterable)
    while len(d) > 1:
        pair = [d.popleft(), d.popleft()]
        d.append(pair)
    return list(d)

>>> print maketree('abcdefgh')
[[[['a', 'b'], ['c', 'd']], [['e', 'f'], ['g', 'h']]]]
```

5.12 heapq — Algoritmo heap queue

Nuovo nella versione 2.3.

Questo modulo fornisce un'implementazione dell'algoritmo heap queue, conosciuto anche come algoritmo di coda con priorità.

Gli Heap sono vettori tali che *heap*`[`*k*`] <= `*heap*`[2*`*k*`+1]` e *heap*`[`*k*`] <= `*heap*`[2*`*k*`+2]` per ogni *k*, contando gli elementi da zero. Per esigenze di confronto, gli elementi non presenti vengono considerati infiniti, quindi maggiori di qualunque altro. L'interessante proprietà dello heap è che *heap*`[0]` è sempre l'elemento più piccolo.

L'API seguente differisce dagli algoritmi di heap dei manuali informatici per due aspetti: (a) Noi usiamo indici che iniziano da zero. Questo rende le relazioni tra l'indice di un nodo e gli indici dei suoi figli leggermente meno ovvie, ma è più conveniente poiché anche Python usa indicizza da zero. (b) Il nostro metodo pop restituisce l'elemento più piccolo, non il più grande (chiamato il min heap nei manuali; un max heap è più comune nei testi grazie alla sua adattabilità all'ordinamento sul posto).

Questi due punti permettono di trattare l'heap come una normale lista Python senza sorprese: *heap*`[0]` è l'elemento più piccolo e *heap*`.sort()` mantiene l'heap invariato!

Per creare uno heap, potete utilizzare una lista inizializzata a `[]` o trasformare una lista non vuota in uno heap tramite la funzione `heapify()`.

Vengono fornite le seguenti funzioni :

heappush(*heap, item*)
Inserisce il valore *item* in *heap*, mantenendo invariato lo heap.

heappop(*heap*)
Estrae e restituisce il più piccolo elemento di *heap*, mantenendo lo heap invariato. Se lo heap è vuoto, viene sollevata l'eccezione `IndexError`.

heapify(*x*)
Trasforma la lista *x* in uno heap, sul posto, in un tempo lineare.

heapreplace(*heap, item*)
Estrae e restituisce il più piccolo elemento di *heap* e vi introduce un nuovo elemento, *item*. La dimensione dello heap non cambia. Se lo heap è vuoto, viene sollevata un'eccezione `IndexError`. Questa funzione è più efficiente di `heappop()` seguita da `heappush()` e può essere più adatta quando si usa uno heap di dimensione fissa. Notate che il valore restituito può essere più grande di *item*! Per cui utilizzate attentamente questa procedura.

Esempio di utilizzo:

```
>>> from heapq import heappush, heappop
>>> heap = []
>>> data = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]
>>> for item in data:
...     heappush(heap, item)
...
>>> sorted = []
>>> while heap:
...     sorted.append(heappop(heap))
...
>>> print sorted
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> data.sort()
>>> print data == sorted
True
>>>
```

5.12.1 Teoria

(Questa spiegazione è dovuta a François Pinard. Il codice Python per questo modulo è stato fornito da Kevin O'Connor.)

Gli heap sono vettori per i quali `a[`*k*`] <= a[2*`*k*`+1]` e `a[`*k*`] <= a[2*`*k*`+2]` per ogni *k*, contando gli elementi da zero. Per esigenze di confronto, gli elementi non presenti vengono considerati di valore infinito. L'interessante proprietà degli heap è che `a[0]` è sempre il suo elemento più piccolo.

La strana regola precedente può essere intesa come una rappresentazione di un torneo con poco dispendio per la memoria .I numeri seguenti sono *k*, non `a[`*k*`]`:

```
                                0

             1                                  2

      3              4                5              6

   7      8       9       10      11      12      13      14

 15 16  17 18  19 20  21 22  23 24  25 26  27 28  29 30
```

Nell'albero precedente, ogni cella *k* è padre delle celle `2*`*k*`+1` e `2*`*k*`+2`. In un qualunque torneo ad eliminazione, come vediamo negli sport, ogni cella contiene il vincitore delle celle sottostanti e possiamo seguire il vincitore lungo l'albero per vedere tutti gli avversari che ha incontrato. Tuttavia nelle applicazioni informatiche di tali tornei, non abbiamo bisogno di tracciare la storia del vincitore. Per avere una maggior efficienza nell'uso della memoria, quando un vincitore viene promosso, si cerca di sostituirlo con qualcuno del livello sottostante, la regola diventa quindi che una cella e le sue due celle sottostanti contengono tre differenti elementi, ma la cella superiore vince sulle altre due.

Se questa regola dello heap viene sempre rispettata, l'elemento di indice zero è chiaramente il vincitore assoluto. Il meccanismo più semplice per rimuoverlo e cercare il prossimo vincitore consiste nello spostare un perdente (diciamo la cella 30 nel diagramma precedente) nella posizione 0 e poi far scendere questo elemento lungo l'albero, scambiando i valori, finché la regola viene ristabilita. Questo algoritmo è chiaramente logaritmico sul numero totale degli elementi che compongono l'albero. Iterando su tutti gli elementi, ottenete un ordinamento di tempo O(n log n).

Una caratteristica utile di questo algoritmo è che potete inserire in modo efficiente nuovi elementi mentre l'ordinamento è in funzione, purché gli elementi inseriti non siano migliori dell'ultimo elemento di indice 0 che avete estratto. Questo torna particolarmente utile nei contesti di simulazione, dove l'albero contiene tutti gli eventi introdotti e la condizione di vincitore indica il minor tempo di schedulazione. Quando un evento schedula altri eventi per l'esecuzione, questi verrano schedulati nel futuro, così che possano essere facilmente inseriti nello heap.

Perciò lo heap è una buona struttura per implementare gli scheduler (e quello che ho usato per il mio sequenziatore MIDI :-).

Sono state studiate approfonditamente varie strutture per implementare gli scheduler e gli heap sono adatti a questo scopo, essendo ragionevolmente veloci, la velocità è pressoché costante ed il caso peggiore non è molto diverso dal caso medio. Ad ogni modo esistono altre rappresentazioni più efficienti, ma i casi peggiori potrebbero risultare terribili.

Gli heap sono molto utili anche per l'ordinamento di grandi dischi. Tutti voi probabilmente sapete che per realizzare grandi ordinamenti si producono delle runs (che sono sequenze preordinate, la cui dimensione è solitamente legata alla quantità di memoria della CPU), procedendo poi allo loro fusione, questa fusione è spesso organizzata molto ingegnosamente[1]. È molto importante che l'ordinamento iniziale produca sequenze più lunghe possibili. I tornei sono una buona soluzione. Se, usando tutta la memoria disponibile per contenere i tornei, sostituite e fate scendere gli elementi che si succedono per riempire la run corrente, produrrete run che sono due volte la dimensione della memoria per input casuali e ancora di più per input parzialmente ordinati.

Inoltre, se estraete l'elemento di indice 0 sul disco ed ottenete un input che non potrebbe essere inserito nell'attuale torneo (perché il valore vince sull'ultimo valore estratto), non potete inserirlo nello heap, percui la dimensione di quest'ultimo diminuisce. La memoria liberata potrebbe essere ragionevolmente riutilizzata subito per costruire progressivamente un secondo heap, che aumenta con la stessa rapidità con cui diminuisce il primo. Quando il primo heap è completamente svanito, scambiate gli heap e iniziate una nuova sequenza. Intelligente e abbastanza efficace!

In una parola, gli heap sono strutture di memorizzazione utili da conoscere. Io gli uso in diverse applicazioni e credo sia bene tenere a portata di mano un modulo 'heap'. :-)

5.13 `array` — Array efficienti di valori numerici

Questo modulo definisce un tipo di oggetto che può rappresentare in modo efficiente un array (NdT: vettore) di valori basilari: caratteri, interi, numeri in virgola mobile. Gli array sono tipi sequenza e si comportano in modo molto simile alle liste, tranne per il fatto che il tipo degli oggetti contenuti è vincolato. Il tipo viene specificato al momento della creazione dell'oggetto usando un singolo carattere chiamato *type code*. Vengono definiti i seguenti type code:

Type code	Tipo C	Tipo Python	Dimensione minima in byte
`'c'`	char	character	1
`'b'`	signed char	int	1
`'B'`	unsigned char	int	1
`'u'`	Py_UNICODE	Unicode character	2
`'h'`	signed short	int	2
`'H'`	unsigned short	int	2
`'i'`	signed int	int	2
`'I'`	unsigned int	long	2
`'l'`	signed long	int	4
`'L'`	unsigned long	long	4
`'f'`	float	float	4
`'d'`	double	float	8

La rappresentazione reale dei valori viene determinata dall'architettura della macchina (strettamente parlando, dall'implementazione C). La dimensione reale può essere ottenuta attraverso l'attributo `itemsize`. I valori memorizzati per gli elementi `'L'` e `'I'` verranno rappresentati come interi long Python al momento del loro recupero, poiché il tipo dei normali interi Python non può rappresentare l'intera estensione degli interi(long) senza segno del C.

[1] Gli algoritmi di bilanciamento del disco in uso al giorno d'oggi, sono più fastidiosi che intelligenti e questa è una conseguenza della capacità di lettura dei dischi. Nei dispositivi che non possono ricercare, come i grandi drive a nastro magnetico, la storia era leggermente differente, e si doveva essere parecchio furbi per assicurarsi (molto anticipatamente) che ogni movimento del nastro sarebbe stato il più efficiente possibile (il che significa partecipare al meglio alla progressione della fusione). Qualche nastro era persino capace di leggere all'indietro, e questo veniva anche usato per evitare i tempi di riavvolgimento. Credetemi, i veri ordinamente dei nastri erano proprio spettacolari da vedere! Dalla notte dei tempi l'ordinamento è sempre stata una Grande Arte! :-)

Il modulo definisce il seguente tipo:

`array`(*typecode*[, *initializer*])
: Restituisce un nuovo array i cui elementi vengono limitati al tipo indicato da *typecode* ed inizializzati dal valore facoltativo *initializer*, che deve essere una lista o una stringa. La lista o la stringa viene passata al metodo `fromlist()`, `fromstring()` o `fromunicode()` del nuovo array (vedi in seguito) per aggiungere elementi iniziali all'array.

`ArrayType`
: Sinonimo obsoleto di `array`.

Gli oggetti array supportano le ordinarie operazioni di indicizzazione, affettamento, concatenazione e moltiplicazione delle sequenze. Durante l'assegnazione di una fetta, il valore assegnato deve essere un oggetto array con lo stesso type code; in tutti gli altri casi, viene sollevata un'eccezione `TypeError`. Gli oggetti array implementano anche l'interfaccia buffer e possono essere utilizzati ovunque siano supportati gli oggetti buffer.

Vengono supportati anche i seguenti attributi e metodi:

`typecode`
: Il carattere che indica il typecode utilizzato per creare l'array.

`itemsize`
: La lunghezza in byte di un elemento dell'array nella rappresentazione interna.

`append`(*x*)
: Aggiunge un nuovo elemento con valore *x* alla fine dell'array.

`buffer_info`()
: Restituisce una tupla (*indirizzo*, *lunghezza*) che fornisce l'indirizzo di memoria corrente e la lunghezza in elementi del buffer usato per contenere l'array. La dimensione in byte del buffer di memoria può essere calcolato come *array*`.buffer_info()[1] *` *array*`.itemsize`. Questo torna utile talvolta, quando si lavora con interfacce I/O a basso livello (e quindi insicure) che richiedono indirizzi di memoria, come alcune `ioctl()`. I numeri restituiti sono validi a patto che l'array esista e non vengano applicate operazioni su di esso che ne modifichino la lunghezza.

  Note: Quando si usano oggetti array da codice scritto in C o C++ (il solo modo per fare effettivo utilizzo di questa informazione), ha più senso usare l'interfaccia buffer supportata dagli oggetti array. Questo metodo viene mantenuto per retrocompatibilità e bisognerebbe evitarlo nel nuovo codice. L'interfaccia buffer è documentata nel *Python/C API Reference Manual*.

`byteswap`()
: Esegue il "Byteswap" (NdT: lo scambio) dei byte di tutti gli elementi dell'array. Viene supportato solo per valori che hanno dimensione di 1, 2, 4 o 8 byte; per gli altri tipi di valori viene sollevata un'eccezione `RuntimeError`. Questo metodo è utile quando si leggono dati da un file scritto su una macchina che utilizza un ordine di byte differente.

`count`(*x*)
: Restituisce il numero di occorrenze di *x* nell'array.

`extend`(*iterable*)
: Aggiunge alla fine dell'array gli elementi di *iterable*. Se *iterable* è un altro array, dovrà avere *esattamente* lo stesso type code; altrimenti verrà sollevata l'eccezione `TypeError`. Se *iterable* non è un array, dovrà comunque essere iterabile ed i suoi elementi verranno aggiunti alla destra dell'array. Modificato nella versione 2.4: In passato, l'argomento poteva essere solamente un altro array.

`fromfile`(*f*, *n*)
: Legge *n* elementi (come valori macchina) dall'oggetto file *f* e li aggiunge alla fine dell'array. Se sono disponibili meno di *n* elementi, viene sollevata un'eccezione `EOFError`, ma gli elementi che erano disponibili vengono comunque inseriti nell'array. *f* deve essere un vero oggetto file built-in; un altro oggetto, anche se possiede un metodo `read()`, non è sufficiente.

`fromlist`(*list*)
: Aggiunge gli elementi della lista *list*. Equivale a '`for x in` *list*`: a.append(x)`' eccetto per il fatto che se viene rilevato un errore sul tipo, l'array rimane immutato.

`fromstring`(*s*)

Aggiunge elementi prendendoli dalla stringa *s*, interpretando la stringa come un array di valori macchina (come se fossero stati letti da un file usando il metodo `fromfile()`).

`fromunicode`(*s*)
Estende l'array con dati provenienti da una stringa unicode. L'array deve essere un array di tipo 'u'; altrimenti viene sollevata un'eccezione `ValueError`. Usate '`array.fromstring(ustr.decode(enc))`' per aggiungere dati Unicode ad un array di qualche altro tipo.

`index`(*x*)
Restituisce il più piccolo *i* tale che *i* sia l'indice della prima occorrenza di *x* nell'array.

`insert`(*i*, *x*)
Inserisce nell'array un nuovo elemento con valore *x* prima della posizione *i*. I valori negativi vengono trattati come relativi alla fine dell'array.

`pop`([*i*])
Rimuove l'elemento di indice *i* dall'array e lo restituisce. Il valore predefinito dell'argomento facoltativo è `-1`, così che, senza argomenti, `pop()` rimuova e restituisca l'ultimo elemento.

`read`(*f*, *n*)
Deprecato dalla versione 1.5.1. Usate il metodo `fromfile()`. Legge *n* elementi (come valori macchina) dall'oggetto file *f* e li aggiunge alla fine dell'array. Se sono disponibili meno di *n* elementi, viene sollevata un'eccezione `EOFError`, ma gli elementi disponibili vengono comunque inseriti nell'array. *f* deve essere un vero oggetto file built-in; un altro oggetto, anche se possiede un metodo `read()`, non è sufficiente.

`remove`(*x*)
Rimuove la prima occorrenza di *x* dall'array.

`reverse`()
Inverte l'ordine degli elementi dell'array.

`tofile`(*f*)
Scrive tutti gli elementi (come valori macchina) nell'oggetto file *f*.

`tolist`()
Converte l'array in una normale lista con gli stessi elementi.

`tostring`()
Converte l'array in un array di valori macchina e restituisce la rappresentazione sotto forma di stringa (la stessa sequenza di byte che verrebbero scritti in un file dal metodo `tofile()`).

`tounicode`()
Converte l'array in una stringa unicode. L'array deve essere di tipo 'u'; altrimenti viene sollevata un'eccezione `ValueError`. Usate *array*`.tostring().decode(enc)` per ottenere una stringa unicode da un array di altro tipo.

`write`(*f*)
Deprecato dalla versione 1.5.1. Usate il metodo `tofile()`. Scrive tutti gli elementi (come valori macchina) nell'oggetto file *f*.

Quando un oggetto array viene stampato o convertito in una stringa, viene rappresentato come `array(`*typecode*`,` *initializer*`)`. Il parametro *initializer* viene omesso se l'array è vuoto, è una stringa se il *type code* è `'c'`, altrimenti è una lista di numeri. Viene garantito che la stringa possa essere riconvertita in un array dello stesso tipo e valore usando gli apici inversi (‘), a condizione che la funzione `array()` sia stata importata usando l'istruzione `from array import array`. Esempi:

```
array('l')
array('c', 'hello world')
array('u', u'hello \textbackslash u2641')
array('l', [1, 2, 3, 4, 5])
array('d', [1.0, 2.0, 3.14])
```

Vedete anche:

Modulo `struct` (sezione 4.3):
: Impacchettamento e spacchettamento di dati binari eterogenei.

Modulo `xdrlib` (sezione 12.17):
: Impacchettamento e spacchettamento di External Data Representation (NdT: rappresentazione di dati esterni) (XDR) come avviene in alcune chiamate di sistema di procedure remote.

Il manuale di Numerical Python
(http://numpy.sourceforge.net/numdoc/HTML/numdoc.htm)
: Le estensioni Numeric Python (NumPy) definiscono un altro tipo di array; vedete http://numpy.sourceforge.net/ per ulteriori informazioni circa Numerical Python. Una versione in PDF del manuale di NumPy è disponibile presso http://numpy.sourceforge.net/numdoc/numdoc.pdf.

5.14 `sets` — Raccolte non ordinate di elementi distinti (Insiemi)

Nuovo nella versione 2.3.

Il modulo `sets` fornisce classi per la costruzione e la manipolazione di raccolte non ordinate di elementi distinti. Si possono usare per test di appartenenza, per rimuovere duplicati da una sequenza e per calcolare le operazioni matematiche standard sugli insiemi come l'intersezione, l'unione, la differenza e la differenza simmetrica.

Come le altre raccolte, sets supporta *x* `in` *set*, `len(`*set*`)` e `for` *x* `in` *set*. Essendo una raccolta non ordinata, sets non memorizza la posizione degli elementi o l'ordine di inserimento. Di conseguenza, sets non supporta l'indicizzazione, l'affettamento o altre caratteristiche proprie delle sequenze.

La maggior parte delle applicazioni insiemistiche utilizza la classe `Set` che fornisce tutti i metodi per la gestione degli insiemi eccetto `__hash__()`. Per applicazioni avanzate che richiedono un metodo hash, la classe `ImmutableSet` aggiunge un metodo `__hash__()`, ma non possiede i metodi che alterano il contenuto dell'insieme. Sia `Set` che `ImmutableSet` derivano da `BaseSet`, una classe astratta utile per definire che cosa sia un insieme: `isinstance(`*obj*`, BaseSet)`.

Le classi set vengono implementate usando i dizionari. Di conseguenza, gli insiemi non possono contenere elementi mutabili come liste o dizionari. In ogni caso possono contenere raccolte immutabili come tuple o istanze di `ImmutableSet`. Per praticità nell'implementazione di insiemi di insiemi, gli insiemi nidificati vengono automaticamente convertiti nella loro forma immutabile, per esempio: `Set([Set(['dog'])])` viene trasformato in `Set([ImmutableSet(['dog'])])`.

class `Set``(`[*iterable*]`)`
: Costruisce un nuovo oggetto `Set` vuoto. Se è presente il parametro facoltativo *iterable*, l'insieme viene riempito con gli elementi ottenuti dall'iterazione. Tutti gli elementi di *iterable* dovrebbero essere immutabili o poter essere trasformati in immutabili utilizzando il protocollo descritto nella sezione 5.14.3.

class `ImmutableSet``(`[*iterable*]`)`
: Costruisce un nuovo oggetto `ImmutableSet` vuoto. Se è presente il parametro facoltativo *iterable*, l'insieme viene riempito con gli elementi ottenuti dall'iterazione. Tutti gli elementi di *iterable* dovrebbero essere immutabili o poter essere trasformati in immutabili utilizzando il protocollo descritto nella sezione 5.14.3.

  Poiché gli oggetti `ImmutableSet` forniscono un metodo `__hash__()`, essi possono essere utilizzati come elementi di un insieme o come chiavi di un dizionario. Gli oggetti `ImmutableSet` non possiedono metodi per aggiungere o rimuovere elementi, perciò tutti gli elementi devono essere noti al momento della chiamata al costruttore.

5.14.1 Oggetti Set

Le istanze sia di `Set` che di `ImmutableSet` forniscono le seguenti operazioni:

Operazione	Equivalente	Risultato
`len(`*s*`)`		cardinalità dell'insieme *s*
x `in` *s*		controlla se *x* appartiene a *s*
x `not in` *s*		controlla se *x* non appartiene a *s*
s`.issubset(`*t*`)`	*s* `<=` *t*	controlla se ogni elemento di *s* appartiene a *t*
s`.issuperset(`*t*`)`	*s* `>=` *t*	controlla se ogni elemento di *t* appartiene a *s*
s`.union(`*t*`)`	*s* | *t*	nuovo insieme formato dagli elementi che appartengono sia a *s* che a *t*
s`.intersection(`*t*`)`	*s* & *t*	nuovo insieme formato dagli elementi comuni a *s* e a *t*
s`.difference(`*t*`)`	*s* - *t*	nuovo insieme con gli elementi che appartengono a *s* ma non a *t*
s`.symmetric_difference(`*t*`)`	*s* ^ *t*	nuovo insieme con gli elementi che appartengono a *s* o a *t* ma non a entram
s`.copy()`		nuovo insieme copiato da *s*

Notate, la versione non operatore di `union()`, `intersection()`, `difference()` e `symmetric_difference()` accetteranno ogni oggetto iterabile come argomento. Viceversa, i loro corrispettivi basati su operatori richiedono che gli argomenti siano insiemi. Questo evita costruzioni ambigue come `Set('abc') & 'cbs'` in favore della più leggibile `Set('abc').intersection('cbs')`. Modificato nella versione 2.3.1: Prima tutti gli argomenti dovevano essere insiemi.

Inoltre, sia `Set` che `ImmutableSet` supportano il confronto tra insiemi. Due insiemi sono uguali se e solo se ogni elemento di un insieme è contenuto nell'altro (ognuno è sotto insieme dell'altro). Un insieme è minore di un altro se e solo se il primo è un sotto insieme proprio del secondo (è un sotto insieme, ma non è uguale). Un insieme è maggiore di un altro se e solo se il secondo insieme è un sotto insieme proprio del primo (è un superinsieme, ma non è uguale).

Le relazioni essere sotto insieme ed essere uguali non possono venire generalizzate in una relazione d'ordine completa. Per esempio due insiemi disgiunti non sono uguali e neppure sono uno sotto insieme dell'altro, così ognuna delle seguenti espressioni restituisce `False`: *a*<*b*, *a*==*b*, o *a*>*b*. Quindi, negli insiemi non viene implementato il metodo `__cmp__`.

Poiché negli insiemi viene definito solo un ordine parziale (la relazione sotto insieme), il risultato del metodo `list.sort()` rimane indefinito per liste di insiemi.

La seguente tabella elenca le operazioni disponibili in `ImmutableSet` ma che non si trovano in `Set`:

Operazione	Risultato
`hash(`*s*`)`	restituisce un valore hash per *s*

La seguente tabella elenca le operazioni disponibili in `Set` ma che non si trovano in `ImmutableSet`:

Operazione	Equivalente	Risultato
s`.union_update(`*t*`)`	*s* |= *t*	restituisce *s* con gli elementi aggiunti da *t*
s`.intersection_update(`*t*`)`	*s* &= *t*	restituisce *s* lasciando solo gli elementi che si trovano anche in *t*
s`.difference_update(`*t*`)`	*s* -= *t*	restituisce *s* dopo aver rimosso gli elementi trovati in *t*
s`.symmetric_difference_update(`*t*`)`	*s* ^= *t*	restituisce *s* con gli elementi che si trovano in *s* o in *t* ma non in
s`.add(`*x*`)`		aggiunge l'elemento *x* all'insieme *s*
s`.remove(`*x*`)`		rimuove l'elemento *x* dall'insieme *s*; solleva l'eccezione KeyErr
s`.discard(`*x*`)`		rimuove *x* dall'insieme *s* se è presente
s`.pop()`		rimuove e restituisce un elemento qualsiasi dell'insieme *s*; sollev
s`.clear()`		rimuove tutti gli elementi dall'insieme *s*

Notate, le versioni non operatore di `union_update()`, `intersection_update()`, `difference_update()` e `symmetric_difference_update()` accetteranno ogni oggetto iterabile come argomento. Modificato nella versione 2.3.1: Prima tutti gli argomenti dovevano essere insiemi.

5.14.2 Esempio

```
>>> from sets import Set
>>> engineers = Set(['John', 'Jane', 'Jack', 'Janice'])
>>> programmers = Set(['Jack', 'Sam', 'Susan', 'Janice'])
>>> managers = Set(['Jane', 'Jack', 'Susan', 'Zack'])
>>> employees = engineers | programmers | managers           # unione
>>> engineering_management = engineers & managers            # intersezione
>>> fulltime_management = managers - engineers - programmers # differenza
>>> engineers.add('Marvin')                                  # aggiunge un elemento
>>> print engineers
Set(['Jane', 'Marvin', 'Janice', 'John', 'Jack'])
>>> employees.issuperset(engineers)           # controllo superinsieme
False
>>> employees.union_update(engineers)         # unione con un nuovo insieme
>>> employees.issuperset(engineers)
True
>>> for group in [engineers, programmers, managers, employees]:
...     group.discard('Susan')        # rimozione incondizionata di un elemento
...     print group
...
Set(['Jane', 'Marvin', 'Janice', 'John', 'Jack'])
Set(['Janice', 'Jack', 'Sam'])
Set(['Jane', 'Zack', 'Jack'])
Set(['Jack', 'Sam', 'Jane', 'Marvin', 'Janice', 'John', 'Zack'])
```

5.14.3 Protocollo per la conversione automatica in oggetti immutabili

Gli insiemi possono contenere soltanto elementi immutabili. Per comodità, gli oggetti `Set` mutabili vengono automaticamente copiati in `ImmutableSet` prima di essere inseriti come elementi di un insieme.

Il meccanismo è quello di aggiungere sempre un elemento hashable o, se non è hashable, si controlla se possieda un metodo `__as_immutable__()` che restituisca un oggetto immutabile equivalente.

Poiché gli oggetti `Set` possiedono un metodo `__as_immutable__()` che restituisce un'istanza di `ImmutableSet`, è possibile costruire insiemi di insiemi.

Un meccanismo simile viene richiesto dai metodi `__contains__()` e `remove()` che richiedono di applicare la funzione di hash su un elemento per controllare se è presente in un insieme. Questi metodi controllano se è possibile applicare la funzione di hash su un elemento, se non è possibile, controllano la presenza del metodo `__as_temporarily_immutable__()` il quale restituisce l'elemento incapsulato in una classe che fornisce metodi temporanei `__hash__()`, `__eq__()` e `__ne__()`.

Questo metodo alternativo risparmia la necessità di costruire una copia separata dell'oggetto mutabile originale.

Gli oggetti `Set` implementano il metodo `__as_temporarily_immutable__()` il quale restituisce l'oggetto `Set` incapsulato in una nuova classe `_TemporarilyImmutableSet`.

I due meccanismi per aggiungere la possibilità di calcolare il valore di hash in genere sono invisibili all'utente; tuttavia possono sorgere dei conflitti negli ambienti multi-thread in cui un thread può modificare un insieme mentre un altro lo ha temporaneamente incapsulato in `_TemporarilyImmutableSet`. In altre parole gli insiemi composti da insiemi mutabili non sono sicuri per i thread.

5.15 `itertools` — Funzioni che creano iteratori per cicli efficienti

Nuovo nella versione 2.3.

Questo modulo implementa un certo numero di elementi base per la costruzione degli iteratori ispirati ai costrutti dei linguaggi di programmazione Haskell e SML. Entrambi sono stati riadattati ad una forma adeguata per Python.

Il modulo standardizza un nucleo di strumenti veloci ed efficienti nell'uso della memoria, che sono utili separatamente o in combinazione. La standardizzazione aiuta ad evitare i problemi di leggibilità ed affidabilità che sorgono quando differenti individui creano le loro personali implementazioni, che differiscono lievemente fra loro, ognuna con le proprie variazioni e convenzioni sui nomi.

Gli strumenti sono stati progettati per poter essere utilizzati in combinazione. Questo semplifica la costruzione di strumenti più specializzati in modo sintetico ed efficiente in puro Python.

Per esempio, SML fornisce uno strumento di tabulazione: `tabulate(f)` che produce una sequenza di `f(0), f(1), ...`. Questo modulo fornisce `imap()` e `count()` che possono essere combinati per formare `imap(f, count())` e produrre un risultato equivalente.

In maniera simile, gli strumenti funzionali sono stati progettati per cooperare in modo adeguato con le funzioni ad alta velocità fornite dal modulo `operator`.

L'autore del modulo accetta volentieri suggerimenti per altri semplice elementi base da inserire nelle future versioni del modulo.

Che siano realizzati in puro Python o in codice C, gli strumenti che sfruttano gli iteratori sono più efficienti nell'uso della memoria (e più veloci) rispetto ai corrispondenti basati sulle liste. Adottando il principio della realizzazione in tempo reale, creano dati dove e quando necessario, evitando di tenere occupata la memoria del computer come se fosse un "magazzino".

Il vantaggio prestazionale degli iteratori diventa sempre più evidente man mano che aumenta il numero degli elementi – in alcuni casi, le liste divengono talmente grandi da provocare un pesante impatto sulle prestazioni della memoria cache e causarne il rallentamento.

Vedete anche:

The Standard ML Basis Library, *The Standard ML Basis Library*.

Haskell, A Purely Functional Language, *Definition of Haskell and the Standard Libraries*.

5.15.1 Funzioni di itertools

Tutte le seguenti funzioni del modulo costruiscono e restituiscono iteratori. Alcune forniscono flussi di lunghezza infinita, percui dovrebbero essere utilizzate solo da funzioni o cicli che troncano il flusso.

`chain`(**iterables*)
Crea un iteratore che restituisce elementi dalla prima sequenza iterabile passata per parametro fino al suo esaurimento, poi passa alla successiva, fino a che ogni sequenza iterabile viene esaurita. Questo metodo viene utilizzaro per trattare sequenze consecutive come se fossero una singola sequenza. Equivalente a:

```
def chain(*iterables):
    for it in iterables:
        for element in it:
            yield element
```

`count`([*n*])
Crea un iteratore che restituisce interi consecutivi a partire da *n*. Se non specificato il valore predefinito di *n* è zero. Attualmente gli interi long di Python non vengono supportati. Questo metodo viene spesso usato come argomento di `imap()` per generare punti di dati consecutivi. Utile anche con `izip()` per aggiungere sequenze di numeri. Equivalente a:

```
def count(n=0):
    while True:
        yield n
        n += 1
```

Notate, `count()` non esegue controlli di overflow e restituisce numeri negativi dopo aver superato `sys.maxint`. Questo comportamento potrebbe cambiare in futuro.

cycle(*iterable*)

Crea un iteratore che restituisce elementi da *iterable* salvando una copia di ciascuno. Quando la sequenza viene terminata, restituisce gli elementi dalla copia salvata. Si ripete indefinitamente. Equivalente a:

```
def cycle(iterable):
    saved = []
    for element in iterable:
        yield element
        saved.append(element)
    while saved:
        for element in saved:
              yield element
```

Notate, questo è l'unico membro del toolkit che può richiedere un notevole spazio di memoria (in funzione della lunghezza della sequenza).

dropwhile(*predicate*, *iterable*)

Crea un iteratore che scorre gli elementi di *iterable* fin quando il predicato è vero; dopodichè restituisce ogni elemento. Notate, l'iteratore non produce *alcun* risultato finché il predicato è vero, perciò può avere un lungo tempo di avvio. Equivalente a:

```
def dropwhile(predicate, iterable):
    iterable = iter(iterable)
    for x in iterable:
        if not predicate(x):
            yield x
            break
    for x in iterable:
        yield x
```

groupby(*iterable*[, *key*])

Crea un iteratore che restituisce chiavi e gruppi consecutivi da *iterable*. *key* è una funzione che calcola un valore chiave per ogni elemento. Se non viene specificata o è `None`, il valore predefinito di *key* è la funzione identica che restituisce l'elemento immutato. Generalmente, l'iterabile deve già essere ordinato con la stessa funzione chiave.

Il gruppo restituito è esso stesso un iteratore che condivide l'iterabile sottostante con `groupby()`. Poiché la sorgente è condivisa, quando l'oggetto `groupby` viene fatto avanzare, il gruppo precedente non è più visibile. Quindi, se i dati sono necessari in seguito, devono essere memorizzati in una lista:

```
groups = []
uniquekeys = []
for k, g in groupby(data, keyfunc):
    groups.append(list(g))      # Salva l'iteratore di gruppo come lista.
    uniquekeys.append(k)
```

`groupby()` è equivalente a:

```
class groupby(object):
    def __init__(self, iterable, key=None):
        if key is None:
            key = lambda x: x
        self.keyfunc = key
        self.it = iter(iterable)
        self.tgtkey = self.currkey = self.currvalue = xrange(0)
    def __iter__(self):
        return self
    def next(self):
        while self.currkey == self.tgtkey:
            self.currvalue = self.it.next() # Esce da StopIteration
            self.currkey = self.keyfunc(self.currvalue)
        self.tgtkey = self.currkey
        return (self.currkey, self._grouper(self.tgtkey))
    def _grouper(self, tgtkey):
        while self.currkey == tgtkey:
            yield self.currvalue
            self.currvalue = self.it.next() # Esce da StopIteration
            self.currkey = self.keyfunc(self.currvalue)
```

Nuovo nella versione 2.4.

`ifilter`(*predicate, iterable*)

Crea un iteratore che filtra gli elementi dall'iterabile *iterable* restituendo solo quelli per i quali *predicate* è `True`. Se il predicato, *predicate*, è `None`, restituisce gli elementi che sono veri. Equivalente a:

```
def ifilter(predicate, iterable):
    if predicate is None:
        predicate = bool
    for x in iterable:
        if predicate(x):
            yield x
```

`ifilterfalse`(*predicate, iterable*)

Crea un iteratore che filtra gli elementi dall'iterabile *iterable* restituendo solo quelli per i quali *predicate* è `False`. Se *predicate* è `None`, restituisce gli elementi che sono falsi. Equivalente a:

```
def ifilterfalse(predicate, iterable):
    if predicate is None:
        predicate = bool
    for x in iterable:
        if not predicate(x):
            yield x
```

`imap`(*function, *iterables*)

Crea un iteratore che calcola la funzione *function* usando argomenti da ognuno degli iterabili *iterables*. Se la funzione è posta a `None`, allora `imap()` restituisce gli argomenti come una tupla. È simile a `map()` ma si interrompe quando l'iterabile più breve si è esaurito, invece che riempire di `None` gli iterabili più corti. Il motivo della differenza è che argomenti di iteratori infiniti in genere mandano in errore `map()` (poiché l'output viene valutato completamente) ma rappresenta una via comune e comoda per fornire argomenti a `imap()`. Equivalente a:

```
def imap(function, *iterables):
    iterables = map(iter, iterables)
    while True:
        args = [i.next() for i in iterables]
        if function is None:
            yield tuple(args)
        else:
            yield function(*args)
```

islice(*iterable*, [*start*,] *stop* [, *step*])

Crea un iteratore che restituisce gli elementi selezionati da *iterable*. Se *start* è diverso da zero, gli elementi di *iterable* vengono saltati fino a raggiungere l'indice *start*. In seguito, gli elementi vengono restituiti in successione a meno che *step*, che indica il numero di elementi da saltare ad ogni passaggio, non sia più grande di uno. Se *stop* è `None`, l'iterazione continua fino all'esaurimento dell'iteratore, altrimenti si ferma alla posizione specificata. Diversamente dall'affettamento regolare, `islice()` non supporta valori negativi per *start*, *stop* o *step*. Questo metodo può venire usato per estrarre campi specifici da dati che presentano una struttura interna appiattita (per esempio, un rapporto multi-linea può presentare un campo nome ogni tre linee). Equivalente a:

```
def islice(iterable, *args):
    s = slice(*args)
    next, stop, step = s.start or 0, s.stop, s.step or 1
    for cnt, element in enumerate(iterable):
        if cnt < next:
            continue
        if stop is not None and cnt >= stop:
            break
        yield element
        next += step
```

izip(**iterables*)

Crea un iteratore che aggrega elementi da ognuno degli iterabili *iterables*. Somiglia a `zip()` eccetto per il fatto che restituisce un iteratore invece di una lista. Questo metodo viene utilizzato per iterazioni lock-step su diversi iterabili contemporaneamente. Equivalente a:

```
def izip(*iterables):
    iterables = map(iter, iterables)
    while iterables:
        result = [i.next() for i in iterables]
        yield tuple(result)
```

Modificato nella versione 2.4: Se non vengono specificati *iterables*, il metodo restituisce un iteratore di lunghezza zero invece di sollevare un'eccezione TypeError.

repeat(*object*[, *times*])

Crea un iteratore che restituisce l'oggetto *object* in continuazione. Viene eseguito indefinitamente a meno che non venga specificato l'argomento *times*. Questo metodo viene utilizzato come argomento di `imap()` per passare sempre gli stessi parametri alla funzione chiamata. Usato anche con `izip()` per creare una parte invariante di un record di tuple. Equivalente a:

```
def repeat(object, times=None):
    if times is None:
        while True:
            yield object
    else:
        for i in xrange(times):
            yield object
```

`starmap`(*function, iterable*)

Crea un iteratore che calcola la funzione *function* usando tuple di argomenti ottenuti da *iterable*. Questo metodo viene usato al posto di `imap()` quando i parametri di argomento sono già stati raggruppati in tuple da un singolo iterabile (i dati sono stati "pre-zippati"). La differenza tra `imap()` e `starmap()` equivale alla distinzione tra `function(a,b)` e `function(*c)`. Equivalente a:

```
def starmap(function, iterable):
    iterable = iter(iterable)
    while True:
        yield function(*iterable.next())
```

`takewhile`(*predicate, iterable*)

Crea un iteratore che restituisce gli elementi dell'iterabile *iterable* finché *predicate* è vero. Equivalente a:

```
def takewhile(predicate, iterable):
    for x in iterable:
        if predicate(x):
            yield x
        else:
            break
```

`tee`(*iterable*[, *n=2*])

Restituisce *n* iteratori indipendenti da un singolo iterabile *iterable*. Nel caso in cui *n* sia due è equivalente a:

```
def tee(iterable):
    def gen(next, data={}, cnt=[0]):
        for i in count():
            if i == cnt[0]:
                item = data[i] = next()
                cnt[0] += 1
            else:
                item = data.pop(i)
            yield item
    it = iter(iterable)
    return (gen(it.next), gen(it.next))
```

Notate, una volta che `tee()` ha fatto una suddivisione, l'iterabile *iterable* originale non dovrebbe essere più usato altrove; altrimenti *iterable* potrebbe venire fatto avanzare senza che gli oggetti di `tee()` ne siano informati.

Notate, questo membro del toolkit potrebbe richiedere un notevole spazio di memoria (a seconda della quantità di dati temporanei che è necessario immagazzinare). In genere, se un iteratore dovrà utilizzare tutti o gran parte i dati prima dell'altro iteratore, è più veloce l'utilizzo di `list()` piuttosto che di `tee()`. Nuovo nella versione 2.4.

5.15.2 Esempi

Gli esempi seguenti mostrano gli utilizzi più comuni di ogni tool e illustrano la maniera in cui si possono combinare.

```
>>> amounts = [120.15, 764.05, 823.14]
>>> for checknum, amount in izip(count(1200), amounts):
...     print 'Check %d is for $%.2f' % (checknum, amount)
...
Check 1200 is for $120.15
Check 1201 is for $764.05
Check 1202 is for $823.14

>>> import operator
>>> for cube in imap(operator.pow, xrange(1,5), repeat(3)):
...    print cube
...
1
8
27
64

>>> reportlines = ['EuroPython', 'Roster', '', 'alex', '', 'laura',
                   '', 'martin', '', 'walter', '', 'mark']
>>> for name in islice(reportlines, 3, None, 2):
...    print name.title()
...
Alex
Laura
Martin
Walter
Mark

# Mostra il dizionario ordinato per valori di gruppo
>>> from operator import itemgetter
>>> d = dict(a=1, b=2, c=1, d=2, e=1, f=2, g=3)
>>> di = sorted(d.iteritems(), key=itemgetter(1))
>>> for k, g in groupby(di, key=itemgetter(1)):
...     print k, map(itemgetter(0), g)
...
1 ['a', 'c', 'e']
2 ['b', 'd', 'f']
3 ['g']

# Trova gli insiemi di numeri consecutivi usando groupby.  La chiave
# della soluzione sta nel differenziare attraverso un intervallo cosìcché tutti i
# numeri consecutivi appaiano nel medesimo gruppo.
>>> data = [ 1,  4,5,6, 10, 15,16,17,18, 22, 25,26,27,28]
>>> for k, g in groupby(enumerate(data), lambda (i,x):i-x):
...     print map(operator.itemgetter(1), g)
...
[1]
[4, 5, 6]
[10]
[15, 16, 17, 18]
[22]
[25, 26, 27, 28]
```

5.15.3 Ricette

Questa sezione mostra delle ricette per creare ed estendere i toolset usando gli itertools esistenti come elementi base.

I tool estesi offrono le medesime alte prestazioni del toolset sottostante. Le prestazioni superiori nell'utilizzo della memoria vengono mantenute elaborando un singolo elemento alla volta piuttosto che immettendo in un sol colpo in memoria l'intero iterabile . Il volume del codice viene mantenuto piccolo collegando i tool fra loro usando uno stile funzionale che aiuta ad eliminare le variabili temporanee. L'alta velocità viene ottenuta preferendo elementi base "vettorizzati", al posto dell'utilizzo di cicli for e generatori che causano l'overhead dell'interprete.

```
def take(n, seq):
    return list(islice(seq, n))

def enumerate(iterable):
    return izip(count(), iterable)

def tabulate(function):
    "Restituisce function(0), function(1), ..."
    return imap(function, count())

def iteritems(mapping):
    return izip(mapping.iterkeys(), mapping.itervalues())

def nth(iterable, n):
    "Restituisce l'n-esimo elemento"
    return list(islice(iterable, n, n+1))

def all(seq, pred=bool):
    "Restituisce True se pred(x) è vero per ogni elemento di iterable"
    return False not in imap(pred, seq)

def any(seq, pred=bool):
    "Restituisce True se pred(x) è vero per almeno un elemento di iterable"
    return True in imap(pred, seq)

def no(seq, pred=bool):
    "Restituisce True se pred(x) è falso per ogni elemento di iterable"
    return True not in imap(pred, seq)

def quantify(seq, pred=bool):
    "Conta quante volte il predicato è vero nella sequenza"
    return sum(imap(pred, seq))

def padnone(seq):
    """Restituisce gli elementi della sequenza e poi None indefinitamente

    Utile per emulare il comportamento della funzione built-in map().

    """
    return chain(seq, repeat(None))

def ncycles(seq, n):
    "Restituisce gli elementi della sequenza ripetuti n volte"
    return chain(*repeat(seq, n))

def dotproduct(vec1, vec2):
    return sum(imap(operator.mul, vec1, vec2))

def flatten(listOfLists):
    return list(chain(*listOfLists))

def repeatfunc(func, times=None, *args):
    """Ripete le chiamate a func con gli argomenti specificati.

    Esempio:  repeatfunc(random.random)

    """
    if times is None:
        return starmap(func, repeat(args))
    else:
        return starmap(func, repeat(args, times))

def pairwise(iterable):
    "s -> (s0,s1), (s1,s2), (s2, s3), ..."
    a, b = tee(iterable)
    try:
        b.next()
    except StopIteration:
        pass
```

5.16 `ConfigParser` — Analizzatore dei file di configurazione

Questo modulo definisce la classe `ConfigParser`. La classe `ConfigParser` implementa un semplice linguaggio di analisi dei file di configurazione il quale presenta una struttura simile a quella che trovereste nei file INI di Microsoft Windows. Potete utilizzare questa classe per scrivere programmi facilmente personalizzabili dall'utente finale.

Questa libreria *non* interpreta o scrive i prefissi dei tipi-valori usati nella sintassi INI della versione estesa del Windows Registry. .

Il file di configurazione è composto da sezioni, ciascuna introdotta da una intestazione '`[sezione]`' seguita da dichiarazioni '`nome: valore`', con prolungamenti nello stile di RFC 822; viene accettato anche '`nome=valore`'. Notate che gli spazi vuoti iniziali dei valori vengono rimossi. I valori facoltativi possono contenere stringhe di formato che si riferiscono ad altri valori nella stessa sezione o a valori in una sezione speciale `DEFAULT`. I valori predefiniti possono essere forniti nell'inizializzazione e nel ripristino. Le righe che iniziano con '#' o con '`;`' vengono ignorate e possono essere utilizzate per realizzare commenti.

Per esempio:

```
[My Section]
foodir: %(dir)s/whatever
dir=frob
```

risolverebbe '`%(dir)s`' nel valore di '`dir`' ('`frob`' in questo caso). Tutte le espansioni dei riferimenti vengono eseguite solo su richiesta.

I valori predefiniti possono essere specificati nel costruttore di `ConfigParser` sotto forma di dizionario. Altri valori predefiniti possono venir passati al metodo `get()` che sovrascriverà tutti gli altri.

class `RawConfigParser`([*defaults*])
L'oggetto di configurazione di base. Quando viene passato l'argomento *defaults*, esso viene inizializzato nel dizionario degli valori predefiniti intrinseci. Questa classe non supporta la magica funzionalità di interpolazione. Nuovo nella versione 2.3.

class `ConfigParser`([*defaults*])
Classe derivata da `RawConfigParser` che implementa la funzionalità di interpolazione magica ed aggiunge argomenti facoltativi ai metodi `get()` ed `items()`. I valori in *defaults* devono essere adatti all'interpolazione della stringa '`%()s`'. Notate che *__name__* è un elemento predefinito intrinseco; il suo valore è il nome della sezione e sovrascriverà ogni valore fornito in *defaults*.

Tutti i nomi di opzione usati nell'interpolazioni passeranno attraverso il metodo `optionxform()` allo stesso modo di ogni altro riferimento ad un nome di opzione. Per esempio, usando l'implementazione predefinita di `optionxform()` (che converte i nomi di opzione in minuscolo), il valori '`foo %(bar)s`' e '`foo %(BAR)s`' sono da considerarsi equivalenti.

class `SafeConfigParser`([*defaults*])
Classe derivata da `ConfigParser` che implementa una variante più sensata della funzionalità di interpolazione magica. Questa implementazione è anche più prevedibile. Le nuove applicazioni dovrebbero preferire questa versione se non hanno la necessità di essere compatibili con vecchie versioni di Python. Nuovo nella versione 2.3.

exception `NoSectionError`
Eccezione sollevata quando non viene trovata una sezione specifica.

exception `DuplicateSectionError`
Eccezione sollevata se `add_section()` viene chiamato con il nome di una sezione già presente.

exception `NoOptionError`
Eccezione sollevata quando non viene trovata un'opzione specifica nella sezione specificata.

exception `InterpolationError`
Classe base per le eccezioni sollevate quando intervengono dei problemi durante l'interpolazione delle stringhe.

exception `InterpolationDepthError`
Eccezione sollevata quando l'interpolazione di una stringa non può essere completata perché il numero delle iterazioni eccede `MAX_INTERPOLATION_DEPTH`. Classe derivata da `InterpolationError`.

exception `InterpolationMissingOptionError`
Eccezione sollevata quando un'opzione referenziata da un valore non esiste. Classe derivata da `InterpolationError`. Nuovo nella versione 2.3.

exception `InterpolationSyntaxError`
Eccezione sollevata quando il testo sorgente nel quale vengono fatte le sostituzioni non è conforme alla sintassi richiesta. Classe derivata da `InterpolationError`. Nuovo nella versione 2.3.

exception `MissingSectionHeaderError`
Eccezione sollevata quando si tenta di analizzare un file che non ha intestazioni di sezione.

exception `ParsingError`
Eccezione sollevata quando avvengono degli errori nell'analisi di un file.

`MAX_INTERPOLATION_DEPTH`
La massima profondità di interpolazione ricorsiva del metodo `get()` quando il parametro *raw* è falso. É significativo solo per la classe `ConfigParser`.

Vedete anche:

Modulo `shlex` (sezione 5.21):
Supporto alla creazione di mini-linguaggi simili alla shell di UNIX che possono essere usati come formato alternativo nei file di configurazione delle applicazioni.

5.16.1 Gli oggetti RawConfigParser

Le istanze di `RawConfigParser` possiedono i seguenti metodi:

`defaults``()`
Restituisce un dizionario contenente i valori predefiniti a livello di istanza.

`sections``()`
Restituisce una lista delle sezioni disponibili; `DEFAULT` non viene incluso nella lista.

`add_section``(`*section*`)`
Aggiunge una sezione chiamata *section* all'istanza. Se già esiste una sezione con quel nome, viene sollevata l'eccezione `DuplicateSectionError`.

`has_section``(`*section*`)`
Indica se nella configurazione è presente la sezione richiesta. La sezione `DEFAULT` non viene riconosciuta.

`options``(`*section*`)`
Restituisce una lista delle opzioni disponibili nella sezione *section* specificata.

`has_option``(`*section, option*`)`
Se la sezione *section* esiste e contiene l'opzione *option*, restituisce `True`; altrimenti restituisce `False`. Nuovo nella versione 1.6.

`read``(`*filenames*`)`
Tenta di leggere ed analizzare una lista di nomi di file, restituendo l'elenco di quelli analizzati con successo. Se *filenames* è una stringa o una stringa Unicode, viene trattata come un singolo nome di file. Se un file elencato in *filenames* non può essere aperto, quel file verrà ignorato. In questo modo è possibile specificare una lista di possibili locazioni di file di configurazione (per esempio, la directory corrente, l'home directory dell'utente ed alcune directory di sistema) ed ogni file di configurazione presente nella lista verrà letto. Se non esiste nessuno dei file nominati, l'istanza di `ConfigParser` conterrà un insieme di dati vuoto. Un'applicazione che richiede che dei valori iniziali siano caricati da un file, dovrebbe caricare il file o i file richiesti usando `readfp()` prima di chiamare `read()` per ogni file facoltativo:

```
import ConfigParser, os

config = ConfigParser.ConfigParser()
config.readfp(open('defaults.cfg'))
config.read(['site.cfg', os.path.expanduser('~/.myapp.cfg')])
```

Modificato nella versione 2.4: Restituisce una lista dei file analizzati con successo.

readfp(*fp*[, *filename*])
Legge ed analizza dati di configurazione dall'oggetto file o simile a file in *fp* (viene usato solo il metodo `readline()`). Se *filename* viene omesso ed *fp* possiede un attributo `name`, viene usato questo come *filename*; il valore predefinito è '`<???>`'.

get(*section, option*)
Restituisce un valore *option* della sezione *section*.

getint(*section, option*)
Un metodo comodo che converte in un intero l'opzione *option* nella sezione *section* specificata.

getfloat(*section, option*)
Un metodo comodo che converte in un numero in virgola mobile l'opzione *option* nella sezione *section* specificata.

getboolean(*section, option*)
Un metodo comodo che converte in un valore booleano l'opzione *option* nella sezione *section* specificata. Notate che i valori accettati per le opzioni sono `1`, `yes`, `true` e `on`, che fa sì che il metodo restituisca il valore `True` e `0`, `no`, `false` e `off`, che fa sì che ritorni `False`. Questi valori di stringa possono essere sia maiuscoli che minuscoli. Ogni altro valore solleverà l'eccezione `ValueError`.

items(*section*)
Restituisce una lista di coppie (*nome*, *valore*) per ogni opzione nella sezione *section*.

set(*section, option, value*)
Se la sezione *section* esiste, assegna all'opzione *option* il valore *value*; altrimenti solleva un'eccezione `NoSectionError`. *value* deve essere una stringa (`str` o `unicode`); altrimenti viene sollevata un'eccezione `TypeError`. Nuovo nella versione 1.6.

write(*fileobject*)
Scrive la rappresentazione della configurazione nell'oggetto file specificato da *fileobject*. Questa configurazione potrà essere analizzata da una futura chiamata `read()`. Nuovo nella versione 1.6.

remove_option(*section, option*)
Rimuove l'opzione *option* dalla sezione *section* specificata. Se la sezione non esiste, viene sollevata un'eccezione `NoSectionError`. Se l'opzione esiste viene rimossa e restituito `True`; altrimenti viene restituito `False`. Nuovo nella versione 1.6.

remove_section(*section*)
Rimuove la sezione *section* dalla configurazione. Se la sezione esiste viene restituito `True`. Altrimenti viene restituito `False`.

optionxform(*option*)
Trasforma il nome dell'opzione *option* come trovato in un file di input o come passato da un programma client nella forma che dovrebbe venire usata nelle strutture interne. L'implementazione predefinita restituisce una versione di *option* in lettere minuscole; eventuali modifiche a questo comportamento possono essere operate dalle sotto classi oppure attraverso un programma client che assegna un attributo di questo nome alle istanze. Assegnando questo a `str()`, per esempio, renderebbe i nomi di opzione sensibili alle differenze tra maiuscole e minuscole.

5.16.2 Oggetti ConfigParser

La classe `ConfigParser` estende alcuni metodi dell'interfaccia `RawConfigParser`, aggiungendo alcuni argomenti facoltativi. La classe `SafeConfigParser` implementa la medesima interfaccia estesa.

get(*section, option*[, *raw*[, *vars*]])
Acquisisce il valore *option* per la sezione di nome *section*. Tutte le interpolazioni '%' vengono espanse nei valori restituiti, in base a quelli predefiniti passati al costruttore, come anche le opzioni fornite tramite *vars*, a meno che l'argomento *raw* non sia true.

items(*section*[, *raw*[, *vars*]])
Restituisce una lista di coppie (*nome, valore*) per ogni opzione presente nella sezione *section*. Gli argomenti facoltativi hanno lo stesso significato di quelli del metodo `get()`. Nuovo nella versione 2.3.

5.17 `fileinput` — Itera su righe provenienti da più flussi di input

Questo modulo implementa una classe ed alcune funzioni utili per scrivere rapidamente un ciclo che operi sullo standard input o su una lista di file.

L'utilizzo tipico è il seguente:

```
import fileinput
for line in fileinput.input():
    process(line)
```

In questo esempio, si itera sulle righe di tutti i file elencati in `sys.argv[1:]`, utilizzando `sys.stdin` se la lista degli argomenti è vuota. Se il nome di un file passato è `'-'`, viene anch'esso rimpiazzato da `sys.stdin`. Per specificare una lista alternativa di nomi di file, passatela come primo argomento di `input()`. È permesso usare anche un singolo nome di file.

Tutti i file vengono aperti in modalità testo. Se si verifica un errore di I/O durante l'apertura o la lettura di un file, viene sollevata un'eccezione `IOError`.

Se `sys.stdin` viene utilizzato più di una volta, dopo il primo utilizzo non verrà più restituita alcuna riga, eccetto forse per la modalità interattiva, o se è stato esplicitamente reimpostato (es.: usando `sys.stdin.seek(0)`).

I file vuoti vengono aperti ed immediatamente chiusi; si può constatare la presenza di questi file nella lista solo quando l'ultimo file aperto è vuoto.

È possibile che l'ultima riga di un file non termini con il carattere di fine riga; le righe vengono restituite insieme al carattere di fine riga terminale, se presente.

La seguente funzione è l'interfaccia base di questo modulo:

input([*files*[, *inplace*[, *backup*]]])
Crea un'istanza della classe `FileInput`. L'istanza verrà poi usata come stato globale dalle funzioni di questo modulo, e viene anche restituita per essere usata durante l'iterazione. I parametri di questa funzione verranno passati al costruttore della classe `FileInput`.

Le seguenti funzioni utilizzano lo stato globale creato da `input()`; se non è presente uno stato attivo, viene sollevata un'eccezione `RuntimeError`.

filename()
Restituisce il nome del file che è attualmente in fase di lettura. Prima della lettura della prima riga del file, viene restituito `None`.

lineno()
Restituisce il numero totale di righe appena lette. Prima della lettura di qualsiasi riga restituisce `0`. Dopo che è stata letta l'ultima riga dell'ultimo file, restituisce il numero di quella riga.

filelineno()
Restituisce il numero di righe lette nel file corrente. Prima della lettura della prima riga, restituisce `0`. Dopo la lettura dell'ultima riga dell'ultimo file, restituisce il numero di quella riga all'interno di quel file.

isfirstline()
Restituisce true se la riga appena letta è la prima del relativo file, altrimenti restituisce false.

isstdin()

Restituisce true se l'ultima riga è stata letta da `sys.stdin`, altrimenti restituisce false.

`nextfile``()`
Chiude il file corrente cosicché la successiva iterazione leggerà la prima riga del file successivo (se presente); le righe non lette dal file appena chiuso non verranno aggiunte al conteggio generale di tutte le righe. Il nome del file non sarà aggiornato fino a che non viene letta la prima riga del file successivo. Questa funzione non ha effetto prima della lettura della prima riga del file; quindi non può essere usata per saltare il primo file. Anche dopo la lettura dell'ultima riga dell'ultimo file questa funzione non ha effetto.

`close``()`
Chiude la sequenza.

Anche la classe che implementa il comportamento sequenziale fornito dal modulo è disponibile per poter essere derivata:

class `FileInput``([`*files*`[`, *inplace*`[`, *backup*`]]])`
La classe `FileInput` è l'implementazione; i suoi metodi `filename()`, `lineno()`, `fileline()`, `isfirstline()`, `isstdin()`, `nextfile()` e `close()` corrispondono alle funzioni con lo stesso nome nel modulo. In aggiunta possiede però un metodo `readline()` che restituisce la successiva riga di input, ed un metodo `__getitem__()` che implementa effettivamente il comportamento sequenziale. Si deve accedere alla sequenza in ordine strettamente sequenziale; non si può mescolare l'uso di `readline()` con un accesso casuale.

Filtro opzionale sul posto: se alla funzione `input()` o al costruttore di `FileInput` viene passato l'argomento a parola chiave *inplace*`=1`, il file viene spostato in un altro file di backup e lo standard output viene diretto al file di input (se un file dello stesso nome del file di backup già esiste, verrà sovrascritto senza sollevare eccezioni). Questo rende possibile scrivere un filtro che riscrive il suo file di input sul posto. Passando anche l'argomento a parola chiave *backup*`='.<qualche estensione>'`, si specifica l'estensione che deve avere il file di backup, facendo sì che questo file di backup non venga cancellato; infatti l'estensione predefinita dei file di backup è `'.bak'`, ed essi vengono cancellati dopo la chiusura del file di output. Il filtraggio sul posto viene disabilitate durante la lettura da standard input.

Avvertenze: L'implementazione corrente non funziona con il filesystem 8+3 di MS-DOS.

5.18 `xreadlines` — Iterazione efficiente su un file

Nuovo nella versione 2.1.

Deprecato dalla versione 2.3. Usate '`for` *line* `in` *file*'.

Questo modulo definisce un nuovo tipo di oggetto che può iterare in maniera efficiente sulle righe di un file. Un oggetto xreadlines è un tipo sequenza che implementa una semplice indicizzazione ordinata a partire da `0`, come richiesto dall'istruzione `for` o dalla funzione `filter()`.

Percui il codice

```
import xreadlines, sys

for line in xreadlines.xreadlines(sys.stdin):
    pass
```

ha più o meno la stessa velocità di esecuzione e lo stesso utilizzo di memoria di

```
while 1:
    lines = sys.stdin.readlines(8*1024)
    if not lines: break
    for line in lines:
        pass
```

anche se nel primo caso viene mantenuta la chiarezza dell'istruzione `for`.

`xreadlines``(`*fileobj*`)`

Restituisce un nuovo oggetto xreadlines che itererà sul contenuto di *fileobj*. *fileobj* deve possedere un metodo `readlines()` che supporti il parametro *sizehint*. **Note:** Poiché il metodo `readlines()` bufferizza i dati, di fatto verranno ignorati gli effetti causati dall'impostazione non bufferizzata dell'oggetto file.

Un oggetto xreadlines *s* supporta la seguente operazione sulle sequenze:

Operazione	Risultato
s[*i*]	*i*-esima riga di *s*

Se valori successivi di *i* non sono sequenziali a partire da `0`, questo codice solleverà un'eccezione `RuntimeError`.

Dopo aver letto l'ultima riga del file, questo codice solleverà un'eccezione `IndexError`.

5.19 `calendar` — Funzioni generali per la gestione del calendario

Questo modulo vi permette di generare calendari in modo simile al comando **cal** di UNIX, ed offre utili funzioni aggiuntive per la gestione del calendario. Di predefinito, questi calendari hanno lunedì come primo giorno della settimana, e domenica come ultimo (la convenzione Europea). Usate `setfirstweekday()` per impostare il primo giorno della settimana a Domenica (6) o ad ogni altro giorno della settimana. I parametri che specificano le date vengono trattati come interi.

Molte di queste funzioni si basano sul modulo `datetime` che utilizza un calendario idealizzato, quello Gregoriano, esteso indefinitamente nel passato e nel futuro. Questo corrisponde alla definizione data nel libro di Dershowitz e Reingold Calendrical Calculations, di calendario Gregoriano proleptico, che è il calendario base per tutti i calcoli computazionali.

`setfirstweekday`(*weekday*)
Imposta quale giorno (`0` è lunedì, il `6` è la domenica) debba essere considerato il primo della settimana. I valori `MONDAY`, `TUESDAY`, `WEDNESDAY`, `THURSDAY`, `FRIDAY`, `SATURDAY` e `SUNDAY` vengono forniti per convenienza. Per esempio, per impostare il primo giorno della settimana a domenica:

```
import calendar
calendar.setfirstweekday(calendar.SUNDAY)
```

Nuovo nella versione 2.0.

`firstweekday`()
Restituisce il primo giorno della settimana impostato correntemente. Nuovo nella versione 2.0.

`isleap`(*year*)
Restituisce `True` se *year* è un anno bisestile, altrimenti `False`.

`leapdays`(*y1*, *y2*)
Restituisce il numero di anni bisestili nell'intervallo [*y1*...*y2*], dove *y1* e *y2* sono anni. Modificato nella versione 2.0: Questa funzione non funziona bene per intervalli a cavallo del cambiamento di secolo nella versione di Python 1.5.2.

`weekday`(*year, month, day*)
Restituisce il giorno della settimana (`0` è Lunedì) per l'anno *year* (`1970`–...), mese *month* (`1`–`12`), giorno *day* (`1`–`31`).

`monthrange`(*year, month*)
Restituisce il giorno della settimana del primo giorno del mese ed il numero di giorni nel mese, avendo specificato l'anno (*year*) ed il mese (*month*).

`monthcalendar`(*year, month*)
Restituisce una matrice che rappresenta il calendario del mese. Ogni riga indica una settimana; i giorni fuori del mese vengono segnati da uno zero. Ogni settimana inizia con il Lunedì, tranne se diversamente impostato da `setfirstweekday()`.

`prmonth`(*theyear, themonth*[, *w*[, *l*]])
Stampa il calendario del mese come restituito da `month()`.

month(*theyear, themonth*[, *w*[, *l*]])
Restituisce il calendario del mese in una stringa multiriga. Il parametro *w*, se presente, specifica la larghezza delle colonne della data, che sono centrate. Il parametro *l*, se presente, specifica il numero di righe usate da ogni settimana. Dipende dal primo giorno della settimana impostato da `setfirstweekday()`. Nuovo nella versione 2.0.

prcal(*year*[, *w*[, *l*[*c*]]])
Stampa il calendario di un intero anno, come restituito da `calendar()`.

calendar(*year*[, *w*[, *l*[*c*]]])
Restituisce il calendario, su 3 colonne, di un intero anno sotto forma di stringa multiriga. I parametri opzionali *w*, *l* e *c* impostano rispettivamente la larghezza delle colonne delle date, le righe per settimana ed il numero di spazi tra le colonne dei mesi. Dipende dal primo giorno della settimana impostato da `setfirstweekday()`. L'anno più antico per il quale può essere generato il calendario dipende dalla piattaforma. Nuovo nella versione 2.0.

timegm(*tuple*)
Una funzione isolata ma utile che richiede come parametro una tupla come quella restituita dalla funzione `gmtime()` del modulo `time`, e restituisce il valore corrispondente al timestamp dei sistemi UNIX, che assume come inizio l'anno 1970, e la codifica POSIX. Infatti, `time.gmtime()` e `timegm()` sono una l'inverso dell'altra. Nuovo nella versione 2.0.

Vedete anche:

Modulo `datetime` (sezione 6.9):
Interfaccia orientata agli oggetti per date e orari con funzionalità simili al modulo `time`.

Modulo `time` (sezione 6.10):
Funzioni di basso livello per la gestione del tempo.

5.20 cmd — Supporto per interpreti a riga di comando

La classe `Cmd` offre un semplice ambiente per la scrittura di interpreti a riga di comando. Questi interpreti tornano spesso utili per effettuare dei test, per creare strumenti di amministrazione e per realizzare prototipi che verranno poi incapsulati in un'interfaccia più sofisticata.

class Cmd([*completekey*[, *stdin*[, *stdout*]]])
Un'istanza di `Cmd` o di una sua classe derivata è un inteprete a riga di comando. Non esistono valide ragioni per istanziare `Cmd` direttamente; piuttosto, è utile come base di una classe interprete definita da voi al fine di ereditare i metodi di `Cmd` ed incapsulare metodi d'azione.

L'argomento opzionale *completekey* è il nome `readline` del tasto per il completamento automatico; il tasto `Tab` è quello predefinito. Se *completekey* non è `None`, ed è disponibile il modulo `readline`, il completamento dei comandi viene effettuato automaticamente.

Gli argomenti facoltativi *stdin* e *stdout* specificano gli oggetti file di input e di output che verranno utilizzati dall'istanza di Cmd o da una sua sotto classe. Se non specificati, i valori predefiniti saranno *sys.stdin* e *sys.stdout*.

Modificato nella versione 2.3: Aggiunti i parametri *stdin* e *stdout*..

5.20.1 Oggetti Cmd

Un'istanza di `Cmd` possiede i seguenti metodi:

cmdloop([*intro*])
Mostra ripetutamente un prompt, accetta l'input, analizza la parte iniziale dell'input ricevuto e richiama i metodi d'azione, passando loro il resto della riga come argomento.

L'argomento facoltativo è un titolo o una stringa di introduzione che viene mostrata prima del primo prompt (questo sovrascrive il membro di classe `intro`).

Se viene caricato il modulo `readline`, l'input erediterà automaticamente la gestione dello storico tipica delle shell simili alla **bash** (per es.: `Control-P` mostra il comando precedente, `Control-N` quello suc-

cessivo, `Control-F` muove il cursore a destra in maniera non distruttiva, `Control-B` muove il cursore a sinistra in maniera non ditruttiva, etc.).

In fondo all'input viene inserita la fine del file tramite la stringa `'EOF'`.

Un'istanza dell'interprete individuerà un comando di nome '`foo`' se e solo se possiede un metodo di nome `do_foo()`. Come caso speciale, viene inviata al metodo `do_help()` una riga che inizia con il carattere '`?`'. Come altro caso speciale, viene inviato al metodo `do_shell()`(sempre che sia stato definito) una riga che inizia con il carattere '`!`'.

Se il completamento è abilitato, verrà eseguito automaticamente il completamento dei comandi, e in particolare avverrà chiamando la funzione `complete_foo()` con gli argomenti *text*, *line*, *begidx* ed *endidx*. *text* è il prefisso della stringa che stiamo cercando di completare: tutte le corrispondenze devono iniziare con *text*. *line* è la riga di input corrente senza lo spazio bianco iniziale, mentre *begidx* ed *endidx* sono gli indici iniziale e finale del testo prefissato, che possono essere usati per ottenere diversi tipi di completamento in base alla posizione occupata dal testo all'interno della stringa che si vuole ottenere.

Tutte le sotto classi di `Cmd` ereditano un metodo predefinito `do_help()`. Questo, chiamato con un argomento `'bar'`, invoca il corrispondente metodo `help_bar()`. Senza argomenti, `do_help()` elenca tutte le possibili richieste di aiuto (cioè, tutti i comandi con il metodo `help_*()` corrispondente), oltre a tutti i comandi non documentati.

`onecmd`(*str*)
Interpreta l'argomento come se fosse stato digitato in risposta al prompt. Può essere sovrascritto, ma normalmente non dovrebbe essere necessario; si considerino in proposito i metodi `precmd()` e `postcmd()` per delle tipologie utili di esecuzione. Il valore restituito è un'opzione che indica se l'interpretazione dei comandi si debba o meno fermare.

`emptyline`()
Questo metodo viene chiamato quando viene passata una riga vuota in risposta al prompt. Se questo metodo non viene sovrascritto, ripete l'ultimo comando non vuoto appena inserito.

`default`(*line*)
Questo metodo viene chiamato quando il comando su una riga di input non viene riconosciuto. Se questo metodo non viene sovrascritto, stampa un messaggio di errore ed esce.

`completedefault`(*text, line, begidx, endidx*)
Questo metodo viene chiamato per completare una riga di input quando non esiste un metodo `complete_*()` specifico. Di predefinito restituisce una lista vuota.

`precmd`(*line*)
Questo metodo di aggancio viene eseguito appena prima di interpretare la riga *line*, ma subito dopo che il prompt è stato generato ed emesso. In `Cmd` è un metodo inattivo; esiste per essere sovrascritto dalle sotto classi. Il valore restituito viene usato come comando da inviare al metodo `onecmd()`; l'implementazione di `precmd()` può riscrivere il comando o semplicemente restituire la riga *line* senza cambiamenti.

`postcmd`(*stop, line*)
Questo metodo di aggancio viene eseguito subito dopo il termine dell'esecuzione di un comando. In `Cmd` è inattivo; esiste per essere sovrascritto dalle sotto classi. *line* è la riga di comando che è stata eseguita, e *stop* è un'opzione che indica se l'esecuzione debba terminare dopo la chiamata a `postcmd()`; questo sarà il valore restituito dal metodo `onecmd()`. Il valore restituito da questo metodo verrà usato come nuovo valore dell'opzione interna corrispondente a *stop*; l'interpretazione andrà avanti nel caso venga restituito un valore false.

`preloop`()
Questo metodo di aggancio viene eseguito una sola volta al momento della chiamata a `cmdloop()`. È un metodo inattivo in `Cmd`; esiste per essere sovrascritto dalle sotto classi.

`postloop`()
Questo metodo di aggancio viene eseguito una sola volta quando `cmdloop()` sta per terminare. È un metodo inattivo in `Cmd`; esiste per essere sovrascritto dalle sotto classi.

Le istanze delle sotto classi di `Cmd` possiedono alcune variabili d'istanza pubbliche:

`prompt`
Il prompt mostrato per sollecitare l'input.

identchars
La stringa di caratteri valida come prefisso dei comandi.

lastcmd
L'ultimo comando non vuoto inserito.

intro
La stringa da mostrare come avviso o messaggio introduttivo. Può essere sovrascritta passando un argomento al metodo `cmdloop()`.

doc_header
L'intestazione da mostrare se l'aiuto visualizzato possiede una sezione per i comandi documentati.

misc_header
L'intestazione da mostrare se l'aiuto visualizzato possiede una sezione per le voci relative a comandi particolari (cioè, esistono metodi `help_*()` senza i relativi metodi `do_*()`).

undoc_header
L'intestazione da mostrare se l'aiuto visualizzato possiede una sezione per i comandi non documentati (cioè, esistono metodi `do_*()` senza i corrispondenti metodi `help_*()`).

ruler
Il carattere usato per disegnare le linee separatrici sotto le intestazioni dei messaggi d'aiuto. Se vuoto, non viene disegnata nessuna linea. Il valore predefinito e il carattere '`=`'.

use_rawinput
È un'opzione che ha come valore predefinito vero. Se è vera, `cmdloop()` utilizza `raw_input()` per mostrare il prompt e leggere il comando successivo; se invece è falsa, vengono utilizzati `sys.stdout.write()` e `sys.stdin.readline()`. (Ciò significa che importando `readline`, su sistemi che lo supportano, l'interprete riconoscerà automaticamente i comandi per la gestione dello storico e il modo di gestire il testo tipici di **Emacs**).

5.21 `shlex` — Semplice analizzatore lessicale

Nuovo nella versione 1.5.2.

La classe `shlex` rende semplice la scrittura di analizzatori lessicali per sintassi elementari somiglianti a quella della shell di UNIX. Questo risulterà spesso utile nella scrittura di minilinguaggi, (per esempio nell'esecuzione di file di controllo per le applicazioni Python) o per analizzare stringhe racchiuse tra virgolette.

Il modulo `shlex` definisce le seguenti funzioni:

split(*s*[, *comments*])
Suddivide la stringa *s* usando una sintassi simile alla shell. Se *comments* ha valore `False` (predefinito), l'analisi dei commenti nella stringa fornita verrà disabilitata (impostando il membro `commenters` dell'istanza di `shlex` ad una stringa vuota). Questa funzione opera in modalità POSIX. Nuovo nella versione 2.3.

Il modulo `shlex` definisce inoltre le seguenti classi:

class shlex([*instream*[, *infile*[, *posix*]]])
Un'istanza di `shlex` o di una sua sotto classe è un oggetto che rappresenta un analizzatore lessicale. Il primo argomento, se presente, specifica da dove leggere i caratteri. Deve essere un oggetto file/flusso che disponga dei metodi `read()` e `readline()`, oppure anche una stringa (possibilità, questa, introdotta dalla versione 2.3). Se non viene passato alcun argomento, verrà utilizzato l'input proveniente da `sys.stdin`. Il secondo argomento opzionale è una stringa indicante il nome di un file, che imposta il valore iniziale del membro `infile`. Se l'argomento *instream* viene omesso oppure è uguale a `sys.stdin`, questo secondo argomento verrà impostato al valore predefinito "stdin". L'argomento *posix* è stato introdotto nella versione 2.3 di Python, e definisce la modalità operativa. Quando *posix* non è vero(predefinito), l'istanza di `shlex` funzionerà in modalità compatibile. Quando invece funziona in modalità POSIX, `shlex` cercherà di essere il più possibile conforme alle convenzioni di analisi di una shell POSIX. Vedete in proposito la sezione 5.21.1.

Vedete anche:

Modulo `ConfigParser` (sezione 5.16):
Analizzatore dei file di configurazione simili ai file '.ini' di Windows.

5.21.1 Oggetti shlex

Un'istanza di `shlex` possiede i seguenti metodi:

`get_token``()`
Restituisce un token. Se i token sono stati inseriti nello stack usando `push_token()`, questo metodo estrae l'ultimo elemento inserito nello stack. Altrimenti lo legge dal flusso dell'input. Se in fase di lettura viene incontrato un carattere di fine file, restituisce `self.eof` (una stringa vuota (`''`) in modalità non POSIX, e `None` in modalità POSIX).

`push_token``(`*str*`)`
Inserisce l'argomento in cima allo stack.

`read_token``()`
Legge un token indipendentemente dalla sua posizione nello stack e non interpreta le richieste della sorgente. (Questo metodo di solito non è molto utile, viene qui documentato solo per ragioni di completezza).

`sourcehook``(`*filename*`)`
Quando `shlex` rileva una richiesta da una sorgente (vedete `source` sotto), questo metodo viene richiamato con il token successivo come argomento, e restituisce una tupla consistente di un nome di file ed un oggetto simile a file aperto.

Normalmente, questo metodo toglie subito tutte le virgolette dall'argomento. Se il risultato è un percorso assoluto, o non c'è stata una precedente richiesta di una sorgente, o la sorgente precedente era un flusso (ad esempio `sys.stdin`), il risultato rimane inalterato. Altrimenti, se il risultato è un percorso relativo, viene prefissato con la parte che si riferisce alla directory nel nome del file che si trova immediatamente prima nello stack di inclusione della sorgente (questo comportamento è simile a quello del preprocessore C quando deve gestire `#include file.h`).

Il risultato delle varie manipolazioni viene trattato come un nome di file e restituito come primo componente della tupla, con `open()` chiamato su di esso per ottenere il secondo componente (Notate: questo è l'inverso dell'ordine degli argomenti nell'inizializzazione dell'istanza!).

Questo aggancio è stato presentato in modo che possiate usarlo per implementare percorsi di ricerca nelle directory, aggiungere estensioni di file ed altre manipolazioni sul namespace. Non c'è un corrispondente aggancio 'close', ma un'istanza di shlex richiamerà il metodo `close()` del flusso in input sorgente quando esso restituirà EOF.

Per avere un controllo più esplicito dello stack delle sorgenti, usate i metodi `push_source()` e `pop_source()`.

`push_source``(`*stream*`[`*, filename*`])`
Inserisce un flusso sorgente di input in cima allo stack dell'input. Se viene specificato l'argomento filename, esso verrà utilizzato per i messaggi di errore. Questo è lo stesso metodo usato internamente dal metodo `sourcehook`. Nuovo nella versione 2.1.

`pop_source``()`
Estrae la sorgente di input dalla cima dello stack di input. Questo è lo stesso metodo usato internamente dall'analizzatore quando incontra un EOF su un flusso di input nello stack. Nuovo nella versione 2.1.

`error_leader``([`*file*`[`*, line*`]])`
Questo metodo genera un messaggio di errore nel formato del compilatore C UNIX; il formato è `'%s, line %d: '`, dove '`%s`' viene rimpiazzato dal nome del file sorgente corrente e '`%d`' con il numero della riga di input corrente (gli argomenti facoltativi possono essere utilizzati per sovrascrivere questo comportamento predefinito).

Questa utilità viene fornita allo scopo di incoraggiare gli utenti di `shlex` a generare messaggi di errore nel formato standard riconosciuto da Emacs e da altri strumenti UNIX.

Le istanze delle sotto classi di `shlex` possiedono alcune variabili d'istanza pubbliche che permettono di controllare l'analisi lessicale oppure utilizzabili per il debugging:

commenters
La stringa dei caratteri che vengono considerati come inizio di commento. Tutti i caratteri da quello iniziale alla fine della riga vengono ignorati. Il predefinito è solamente il carattere '#'.

wordchars
La stringa dei caratteri che verranno accumulati nei token multi-carattere. Di predefinito include tutti i caratteri ASCII alfanumerici ed il trattino basso.

whitespace
I caratteri che verranno considerati spazi vuoti e quindi saltati. Gli spazi vuoti delimitano i token. Di predefinito vengono inclusi lo spazio vuoto, la tabulazione, il codice di controllo dell'avanzamento di riga e quello di ritorno carrello.

escape
I caratteri che verranno considerati caratteri di protezione. Questi potranno essere usati solamente in modalità POSIX, ed di predefinito viene incluso solo '\'. Nuovo nella versione 2.3.

quotes
Caratteri che verranno considerati come quotatori di stringa. Il token si accumula finché non viene incontrato il medesimo carattere di quotatura (quindi, caratteri differenti di quotatura si proteggono tra loro come nella shell). Di predefinito, include i caratteri di virgolette ASCIIsingole e doppie.

escapedquotes
I caratteri in `quotes` che interpreteranno i caratteri di protezione definiti in `escape`. Questo viene utilizzato solamente in modalità POSIXPOSIX e di predefinito include solamente ''. Nuovo nella versione 2.3.

whitespace_split
Se `True`, i token verranno suddivisi solamente quando incontrano spazi vuoti. Questo è utile, per esempio, per esaminare le righe di comando con `shlex`, ottenendo i token in modo simile agli argomenti passati alla shell. Nuovo nella versione 2.3.

infile
Il nome del file di input corrente, come inizialmente impostato nel momento di istanziazione della classe o inserito nello stack da richieste successive. Può essere utile controllarlo quando si costruiscono messaggi di errore.

instream
Il flusso di input da cui questa istanza `shlex` sta leggendo i caratteri.

source
Il valore predefinito per questo membro è `None`. Se viene assegnata ad esso una stringa, quella stringa verrà riconosciuta come una richiesta di inclusione a livello lessicale allo stesso modo della parola chiave '`source`' in varie shell. Quindi, il token che segue immediatamente verrà aperto come un nome di file e l'input catturato da quel flusso fino ad EOF, momento in cui il metodo `close()` di quel flusso verrà chiamato ed la sorgente di input diventerà nuovamente il flusso originario di input. Richieste di sorgente possono essere accumulate a qualunque numero di livelli di profondità.

debug
Se questo membro è numerico e vale `1` o più, un'istanza `shlex` stamperà in modo prolisso e progressivo l'output riguardo al suo andamento. Se c'è la necessità di utilizzarlo, potete trovare i dettagli leggendo il codice sorgente del modulo.

lineno
Numero di riga della sorgente (conteggio dei caratteri di fine riga trovati finora più uno).

token
Il token buffer. Può essere utile esaminarlo quando si catturano le eccezioni.

eof
Il token usato per determinare la fine del file. Verrà impostato alla stringa vuota (''), in modalità non POSIX, e `None` in modalità POSIX. Nuovo nella versione 2.3.

5.21.2 Regole di analisi

Quando si lavora in modalità non POSIX, `shlex` cercherà di rispettare le seguenti regole.

- I caratteri di quotatura non vengono riconosciuti all'interno delle parole (`Nonvoglioseparare` viene scandito come la singola parola `Nonvoglioseparare`);
- I caratteri di protezione non vengono riconosciuti;
- I caratteri racchiusi all'interno dei caratteri di quotatura conservano il loro valore letterale;
- I caratteri di quotatura suddividono le parole (`DeviSeparare` viene scandito come `Devi` e `Separare`);
- Se `whitespace_split` viene impostata a `False`, qualunque carattere che non venga dichiarato come un carattere parola, uno spazio vuoto o una quotatura, verrà restituito come un token di carattere singolo. Se invece viene impostato a `True`, `shlex` si limiterà a spezzare le parole in spazi vuoti;
- EOFviene segnalato con una stringa vuota (″);
- Non è possibile analizzare una stringa vuota, anche se quotata.

Quando si opera in modalità POSIX, `shlex` cercherà di rispettare le seguenti regole:

- I caratteri di quotatura vengono eliminati, e non separano le parole (`NonSiSepara` viene scandito come la singola parola `NonSiSepara`);
- I caratteri di protezione e non di quotatura (come '\'), preservano il valore letterale del prossimo carattere che li segue;
- Racchiudere fra caratteri quotatura dei caratteri che non fanno parte di `escapedquotes` (come ''), preserva il loro valore letterale.
- Racchiudere fra caratteri quotatura dei caratteri che fanno parte di `escapedquotes` (come ''), preserva il loro valore letterale,con l'eccezione dei caratteri menzionati in `escape`. I caratteri di protezione mantengono il loro significato speciale solo quando sono seguiti dalla quotatura in uso, o dallo stesso carattere di protezione. Altrimenti il carattere di protezione verrà considerato come un carattere normale.
- EOFviene indicato con un valore `None`;
- È consentito quotare stringhe vuote (″);

CAPITOLO

SEI

Servizi comuni ai Sistemi Operativi

I moduli descritti in questo capitolo forniscono interfacce per le caratteristiche presenti su (quasi) tutti i sistemi operativi, come i file o l'orologio di sistema. Queste interfacce vengono generalmente progettate dopo le interfacce C o UNIX, ma sono disponibili anche sugli altri sistemi. Eccone una panoramica:

os	Interfacce per vari sistemi operativi.
os.path	Tipiche manipolazioni dei nomi di percorso.
dircache	Restituisce listati di directory memorizzati con il meccanismo della memoria cache.
stat	Programma utile per Interpretare i risultati di `os.stat()`, `os.lstat()` e `os.fstat()`.
statcache	Esegue lo stat dei file, memorizzandone i risultati.
statvfs	Costanti usate per l'interpretazione del risultato di `os.statvfs()`.
filecmp	Confronta i file efficientemente.
popen2	Sotto processi con flussi standard di I/O accessibili.
datetime	Tipi di base per data e tempo.
time	Accesso al tempo e conversioni.
sched	Schedulatore di eventi per uso generale.
mutex	Lock e coda per mutue esclusioni.
getpass	Lettura portabile delle password e recupero dello userid.
curses	Un'interfaccia alla libreria curses, fornisce una gestione portabile dei terminali.
curses.textpad	Input di testo in finestre curses in stile Emacs.
curses.wrapper	Wrapper di gestione dei terminali per i programmi curses.
curses.ascii	Costanti e funzioni per i caratteri ASCII.
curses.panel	Un'estensione panel stack (NdT: pila dei pannelli) che aggiunge profondità alle finestre curses.
getopt	Parser portabile per le opzioni da riga di comando; supporta nomi delle opzioni sia lunghi che brevi.
optparse	Una libreria potente, flessibile, ed estensibile, facile da usare, da linea di comando, per analizzare le opzio
tempfile	Generare file e directory temporanei.
errno	Sistema standard dei simboli di errore.
glob	Modello di espansione del percorso nello stile della shell UNIX.
fnmatch	Modello di corrispondenza dei nomi di file nello stile della shell UNIX.
shutil	Operazioni di alto livello sui file, inclusa la copia.
locale	Servizi per l'internazionalizzazione.
gettext	Multilingual internationalization services.
logging	Modulo logging per Python basato su PEP 282.
platform	Recupera quante più informazioni possibili sulla piattaforma.

6.1 os — Interfacce per vari sistemi operativi

Questo modulo fornisce un metodo portabile per sfruttare le funzionalità specifiche di un sistema operativo, confrontato all'utilizzo diretto dei moduli built-in specifici di un sistema, come `posix` o `nt`.

Questo modulo verifica la presenza di moduli built-in come `mac` o `posix`, dipendenti dal sistema operativo, ed esporta gli stessi dati e funzioni individuati in tali moduli. Tutti i moduli built-in di Python dipendenti dal sistema operativo vengono progettati in modo tale che se una certa funzione è disponibile, usi la stessa interfaccia in tutti i moduli; per esempio la funzione `os.stat(`*path*`)` restituisce l'informazione richiesta sul *path* (NdT: percorso)

nello stesso formato (generato dall'interfaccia POSIX).

Le estensioni specifiche di un determinato sistema operativo sono disponibili anche attraverso il modulo `os`, ma usarle ovviamente pregiudica la portabilita'!

Notate che successivamente al primo import del modulo `os`, *non* c'è alcuna riduzione di prestazioni nell'uso delle funzioni di `os` invece che di quelle del modulo built-in dipendente dal sistema operativo, per cui non dovrebbe esistere alcuna ragione per *non* usare `os`!

exception `error`
: Questa eccezione viene sollevata quando una funzione restituisce un errore dipendente dal sistema (non per argomenti di tipo non corretto o per altri errori accidentali). L'eccezione è anche conosciuta come eccezione built-in `OSError`. Il valore che la contraddistingue è una coppia, contenente il valore numerico corrente della variabile C `errno`, e la stringa corrispondente così come verrebbe restituita dalla funzione C `perror()`. Vedete il modulo `errno`, che contiene i nomi per i codici di errore definiti dal sistema operativo sottostante.

  Quando le eccezioni sono gestite come classi, l'eccezione ha due attributi, `errno` e `strerror`. Il primo contiene il valore dell'omonima variabile C, il secondo contiene il corrispondente messaggio di errore restituito da `strerror()`. Per eccezioni che comportano un percorso di accesso ad un file (come quelle che possono venire sollevate da `chdir()` o `unlink()`), l'istanza di eccezione conterrà un terzo attributo, `filename`, che è il nome del file passato alla funzione.

`name`
: Il nome del modulo importato, dipendente dal sistema operativo. I seguenti nomi sono stati finora registrati: `'posix'`, `'nt'`, `'mac'`, `'os2'`, `'ce'`, `'java'`, `'riscos'`.

`path`
: Il corrispondente modulo standard, dipendente da sistema operativo, per le operazioni sui nomi dei percorsi, come `posixpath` o `macpath`. Quindi, avendo importato i moduli giusti, l'espressione `os.path.split(`*file*`)` è equivalente ma più portabile dell'espressione `posixpath.split(`*file*`)`. Notate che questo è anche un modulo importabile: può venire importato direttamente come `os.path`.

6.1.1 Parametri di processo

Le seguenti funzioni e dati forniscono informazioni ed agiscono sul processo e sull'utente corrente.

`environ`
: Un oggetto mappa che rappresenta le variabili di ambiente. Per esempio, `environ['HOME']` è il percorso della vostra home directory (su alcune piattaforme), ed è equivalente a chiamare `getenv(HOME)` in C.

  Se la piattaforma supporta la funzione `putenv()`, questa mappa può venire usata per modificare le variabili di ambiente, oltre che per interrogarle. La funzione `putenv()` verrà chiamata automaticamente ad ogni modifica della mappa. **Note:** Su alcune piattaforme, incluse FreeBSD e MAc OS X, impostare `environ` (NdT: variabili d'ambiente) può causare spreco di memoria. Fate riferimento alla documentazione di sistema per la funzione putenv.

  Se la funzione `putenv()` non viene fornita, questa mappa si può comunque modificare per passarla alle funzioni appropriate di creazione di processo, per fare in modo che i processi figli usino variabili di ambiente modificate.

`chdir`(*path*)
`fchdir`(*fd*)
`getcwd`()
: Queste funzioni vengono descritte in "File e Directory" (sezione6.1.4).

`ctermid`()
: Restituisce il nome del file corrispondente al terminale che controlla il processo. Disponibilità: UNIX.

`getegid`()
: Restituisce l'effettivo ID del gruppo a cui appartiene il processo corrente. Questo corrisponde al bit 'set id' del file che si sta eseguendo nel processo corrente. Disponibilità: UNIX.

`geteuid`()
: Restituisce l'identificativo utente effettivo per il processo corrente. Disponibilità: UNIX.

getgid()
Restituisce l'identificativo di gruppo reale per il processo corrente. Disponibilità: UNIX.

getgroups()
Restituisce la lista degli identificativi di gruppo aggiuntivi associati al processo corrente. Disponibilità: UNIX.

getlogin()
Restituisce il nome dell'utente loggato al terminale che controlla il processo corrente. Per molti usi, è più utile usare la variabile di ambiente LOGNAME per scoprire chi è l'utente, oppure `pwd.getpwuid(os.getuid())[0]` per ottenere il nome di login corrispondente al corrente ed effettivo identificativo dell'utente. Disponibilità: UNIX.

getpgid(*pid*)
Restituisce l'identificativo di gruppo del processo con identificativo uguale al parametro *pid*. Se *pid* è zero, viene restituito l'identificativo di gruppo del processo corrente. Disponibilità: UNIX. Nuovo nella versione 2.3.

getpgrp()
Restituisce l'identificativo di gruppo del processo corrente. Disponibilità: UNIX.

getpid()
Restituisce l'identificativo del processo corrente. Disponibilità: UNIX, Windows.

getppid()
Restituisce l'identificativo del processo padre. Disponibilità: UNIX.

getuid()
Restituisce l'identificativo utente del processo corrente. Disponibilità: UNIX.

getenv(*varname*[, *value*])
Restituisce il valore della variabile di ambiente indicata dal parametro *varname* se esiste, altrimenti il valore del parametro *value*. *value* ha valore predefinito `None`. Disponibilità: quasi tutte le varietà di UNIX, Windows.

putenv(*varname, value*)
Imposta il valore della variabile ambientale indicata dal parametro *varname* uguale alla stringa contenuta dal parametro *value*. Questi cambiamenti delle variabili ambientali influiscono sui processi deirvati da `os.system()`, `popen()` oppure `fork()` e `execv()`. Disponibilità: quasi tutte le varietà di UNIX, Windows.

Note: Su alcune piattaforme, incluso FreeBSD e MAc OS X, cambiare il valore di `environ` può causare spreco di memoria. Consultate la documentazione di sistema per putenv.

Quando `putenv()` viene supportata, le assegnazioni agli elementi di `os.environ` vengono automaticamente tradotte nelle corrispondenti chiamate alla funzione `putenv()`; tuttavia le chiamate a `putenv()` non aggiornano `os.environ`, per cui è attualmente preferibile cambiare valore agli elementi di `os.environ`.

setegid(*egid*)
Imposta l'identificativo effettivo di gruppo per il processo corrente. Disponibilità: UNIX.

seteuid(*euid*)
Imposta l'identificativo effettivo di utente per il processo corrente. Disponibilità: UNIX.

setgid(*gid*)
Imposta l'identificativo di gruppo per il processo corrente. Disponibilità: UNIX.

setgroups(*groups*)
Imposta la lista degli identificativi di gruppo aggiuntivi associati al processo corrente, ponendolo uguale al valore del parametro *groups*. *groups* deve essere una sequenza, e ciascun elemento deve essere un intero che identifichi un gruppo. Questa operazione è solitamente consentita solo al superutente. Disponibilità: UNIX. Nuovo nella versione 2.2.

setpgrp()
Chiama la funzione di sistema `setpgrp()` o `setpgrp(0, 0)` a seconda della versione implementata (se ne esiste una). Consultate un manuale UNIX per la semantica di questa funzione. Disponibilità: UNIX.

setpgid(*pid, pgrp*)
Chiama la funzione di sistema `setpgid()` per impostare l'identificativo di gruppo del processo con identificativo *pid*, al valore *pgrp*. Disponibilità: UNIX.

setreuid(*ruid, euid*)
Imposta l'identificativo utente reale e quello effettivo per il processo corrente. Disponibilità: UNIX.

setregid(*rgid, egid*)
Imposta l'identificativo di gruppo reale e quello effettivo per il processo corrente. Disponibilità: UNIX.

getsid(*pid*)
Chiama la funzione di sistema operativo `getsid()`. Vedete un manuale UNIX per la semantica di questa funzione. Disponibilità: UNIX. Nuovo nella versione 2.4.

setsid()
Chiama la funzione di sistema operativo `setsid()`. Vedete un manuale UNIX per la semantica di questa funzione. Disponibilità: UNIX.

setuid(*uid*)
Imposta l'identificativo utente del processo corrente. Disponibilità: UNIX.

strerror(*code*)
Restituisce il messaggio di errore corrispondente al codice di errore contenuto in *code*. Disponibilità: UNIX, Windows.

umask(*mask*)
Imposta la corrente umask numerica, e restituisce il valore precedente della stessa. Disponibilità : UNIX, Windows.

uname()
Restituisce una tupla di cinque elementi contenente informazioni che identificano il sistema operativo. La tupla contiene cinque stringhe: (*sysname*, *nodename*, *release*, *version*, *machine*). Alcuni sistemi troncano *nodename* a otto caratteri oppure alla prima componente del nome del nodo; un modo migliore di ottenere l'hostname è `socket.gethostname()` o persino `socket.gethostbyaddr(socket.gethostname())`. Disponibilità: versioni recenti di UNIX.

6.1.2 Creazioni di oggetti file

Queste funzioni creano nuovi oggetti file.

fdopen(*fd*[, *mode*[, *bufsize*]])
Restituisce un oggetto file aperto, collegato al descrittore di file *fd*. Gli argomenti *mode* e *bufsize* hanno lo stesso significato dei corrispondenti argomenti della funzione built-in `open()`. Disponibilità: Macintosh, UNIX, Windows.

Modificato nella versione 2.3: Quando specificato, l'argomento *mode* deve cominciare con una delle tre lettere '`r`', '`w`', o '`a`', altrimenti viene sollevata un'eccezione `ValueError`.

popen(*command*[, *mode*[, *bufsize*]])
Lancia un sotto processo che esegue *command* ed apre una pipe da o verso quel processo. Il valore restituito è un oggetto file collegato alla pipe, che può venire letto o scritto a seconda che l'argomento *mode* sia `'r'` (predefinito) o `'w'`. L'argomento *bufsize* ha lo stesso significato del corrispondente argomento della funzione built-in `open()`. Lo stato di uscita del comando eseguito (codificato nel formato specificato per `wait()`) è disponibile come valore restituito dal metodo `close()` quando applicato all'oggetto file, eccetto il caso in cui lo stato di uscita sia zero (comando completato senza errori), nel qual caso il metodo `close()` restituisce `None`. Disponibilità: UNIX, Windows.

Modificato nella versione 2.0: Questa funzione non era affidabile in Windows in versioni precedenti di Python. Questo era dovuto all'uso della funzione `_popen()` fornita dalle librerie di Windows. Le nuove versioni di Python non usano l'implementazione `_popen()` difettosa fornita dalle librerie di Windows.

tmpfile()
Restituisce un nuovo oggetto file aperto in modalità aggiornamento ('`w+b`'). Il file non viene associato ad alcun elemento di directory, e viene cancellato una volta che non ci siano più descrittori che fanno riferimento ad esso. Disponibilità: UNIX, Windows.

Per ciascuna di queste varianti di `popen()`, se l'argomento *bufsize* viene usato, specifica la dimensione del buffer per le pipe I/O. Se fornito, l'argomento *mode* dovrebbe essere o la stringa `'b'` oppure la stringa `'t'`; in Windows questo è necessario per determinare se gli oggetti file debbano venire aperti in modalità binaria o in modalità testuale. Il valore predefinito per l'argomento *mode* è `'t'`.

Questi metodi non danno modo di ottenere il codice di ritorno dal processo figlio. L'unico modo di controllare i flussi di input ed output, ed in più, di ottenere i codici restituiti, è quello di usare le classi `Popen3` e `Popen4` del modulo `popen2`; queste classi sono disponibili solo su UNIX.

Per una discussione delle possibili condizioni di deadlock collegate all'uso di queste funzioni, vedete "Problemi con il controllo di flusso" (sezione 6.8.2).

`popen2`(*cmd*[, *mode*[, *bufsize*]])
Esegue *cmd* come processo derivato. Restituisce gli oggetti file (*child_stdin*, *child_stdout*). Disponibilità: UNIX, Windows. Nuovo nella versione 2.0.

`popen3`(*cmd*[, *mode*[, *bufsize*]])
Esegue *cmd* come processo derivato. Restituisce gli oggetti file (*child_stdin*, *child_stdout*, *child_stderr*). Disponibilità: UNIX, Windows. Nuovo nella versione 2.0.

`popen4`(*cmd*[, *mode*[, *bufsize*]])
Esegue *cmd* come processo derivato. Restituisce gli oggetti file (*child_stdin*, *child_stdout_and_stderr*). Dispoinibilità: UNIX, Windows. Nuovo nella versione 2.0.

Questa funzionalità è anche disponibile nel modulo `popen2`, utilizzando funzioni con gli stessi nomi, ma i valori restituiti da queste hanno diverso ordinamento.

6.1.3 Operazioni sui descrittori di file

Queste funzioni operano sui flussi I/O a cui si fa riferimento tramite descrittori di file.

`close`(*fd*)
Chiude il descrittore di file *fd*. Disponibilità: Macintosh, UNIX, Windows.

Note: Questa funzione è concepita per I/O di basso livello e va applicata ad un descrittore di file come quello restituito da `open()` o `pipe()`. Per chiudere un "oggetto file" restituito dalla funzione built-in `open()` o da `popen()` o da `fdopen()`, usate il metodo `close()` dell'oggetto file.

`dup`(*fd*)
Restituisce un duplicato del descrittore di file *fd*. Disponibilità: Macintosh, UNIX, Windows.

`dup2`(*fd, fd2*)
Duplica il descrittore di file *fd* copiandone il contenuto in *fd2*, chiudendo prima quest'ultimo se necessario. Disponibilità: UNIX, Windows.

`fdatasync`(*fd*)
Forza la scrittura su disco del file con descrittore *fd*. Non forza l'aggiornamento dei metadata. Disponibilità: UNIX.

`fpathconf`(*fd, name*)
Restituisce le informazioni di configurazione di sistema relative ad un file aperto. *name* specifica il parametro di configurazione da cercare; può essere una stringa corrispondente al nome di un predefinito parametro di sistema; questi nomi vengono specificati in vari standard (POSIX.1, UNIX 95, UNIX 98 e altri). Alcune piattaforme definiscono inoltre parametri *name* addizionali. I nomi noti per il sistema ospite vengono forniti dal dizionario `pathconf_names`. Per consentire l'accesso a parametri di configurazione non inclusi nel suddetto dizionario, viene anche consentito di passare un intero come parametro *name*. Disponibilità: UNIX.

Se *nome* è una stringa e non è noto, viene sollevata un'eccezione `ValueError`. Se uno specifico valore di *name* non viene supportato dal sistema ospite, anche se incluso in `pathconf_names`, viene sollevata un'eccezione `OSError`, con codice di errore numerico `errno.EINVAL`.

`fstat`(*fd*)
Restituisce lo stato del descrittore di file *fd*, in modo simile alla funzione `stat()`. Disponibilità: UNIX, Windows.

fstatvfs(*fd*)
Restituisce le informazioni sul file system contenente il file descritto con il descrittore *fd*, in modo simile a `statvfs()`. Disponibilità: UNIX.

fsync(*fd*)
Forza la scrittura su disco del file descritto da *fd*. In UNIX, questa funzione chiama la funzione nativa `fsync()`; in Windows, viene chiamata la funzione `_commit()`.

Partendo da un oggetto file di Python *f*, dovete prima eseguire *f*`.flush()` e poi `os.fsync(`*f*`.fileno())`, per fare in modo che tutti i buffer interni associati ad *f* vengano scritti su disco. Disponibilità: UNIX, e Windows a partire da Python in 2.2.3.

ftruncate(*fd, length*)
Tronca il file corrispondente al descrittore *fd*, in modo che sia al più di dimensione pari a *length* byte. Disponibilità: UNIX.

isatty(*fd*)
Restituisce `True` se il descrittore di file *fd* è aperto e connesso al dispositivo di tipo tty, altrimenti restituisce `False`. Disponibilità:UNIX.

lseek(*fd, pos, how*)
Imposta la posizione corrente del descrittore di file *fd* alla posizione *pos*, modificata da *how*: `0` per impostare la posizione relativamente all'inizio del file; `1` per impostarla alla posizione corrente; `2` per impostare la posizione raltivamente alla fine del file. Disponibilità: Macintosh, UNIX, Windows.

open(*file, flags*[, *mode*])
Apre il file *file* e imposta la varie opzioni secondo quanto indicato da *flags*, e possibilmente imposta il modo di accesso del file, secondo in parametro *mode*. Il modo predefinito è `0777` (ottale), ed il valore corrente della umask viene applicato per primo. Restituisce il descrittore per il file appena aperto. Disponibilità: Macintosh, UNIX, Windows.

Per una descrizione dei valori delle opzioni e del modo, vedete la documentazione del run-time del linguaggio C; anche le opzioni costanti (come `O_RDONLY` e `O_WRONLY`) vengono definite in questo modulo (vedete più avanti).

Note: Questa funzione viene utilizzata per I/O di basso livello. Per l'uso normale, usate la funzione built-in `open()`, che restituisce un "oggetto file" con metodi `read()` e `write()` (e molti altri).

openpty()
Apre una nuova coppia pseudo terminale. Restituisce una coppia di descrittori di file (*master, slave*) per la pty e la tty, rispettivamente. Per un approccio (leggermente) più portabile, usate il modulo `pty`. Disponibilità: alcune varietà di UNIX.

pipe()
Crea una pipe. Restituisce una coppia di descrittori di file (*r, w*) utilizzabile per leggere e scrivere, rispettivamente. Disponibilità: UNIX, Windows.

read(*fd, n*)
Legge fino a *n* byte dal descrittore di file *fd*. Restituisce una stringa contenente i byte letti. Se la fine del file a cui *fd* fa riferimento è stata raggiunta, viene restituita una stringa vuota. Disponibilità: Macintosh, UNIX, Windows.

Note: Questa funzione viene intesa per I/O di basso livello e deve venire applicata al descrittore di file restituito da `open()` e `pipe()`. Per leggere un "oggetto file" restituito dalle funzioni built-in `open()`, `popen()`, `fdopen()` o `sys.stdin`, usate i suoi metodi `read()` o `readline()`.

tcgetpgrp(*fd*)
Restituisce il gruppo associato al terminale definito da *fd* (un descrittore di file come quelli restituiti da `open()`). Disponibilità: UNIX.

tcsetpgrp(*fd, pg*)
Imposta il gruppo associato al terminale descritto da *fd* (un descrittore di file aperto come quello restituito da `open()`), al valore di *pg*. Disponibilità: UNIX.

ttyname(*fd*)
Restituisce una stringa che specifica il dispositivo di tipo terminale associato al descrittore di file *fd*. Se *fd* non è associato ad un dispositivo terminale, viene sollevata un'eccezione. Disponibilità: UNIX.

write(*fd*, *str*)
Scrive la stringa *str* nel descrittore di file *fd*. Restituisce il numero di bytes effettivamente scritti. Disponibilità: Macintosh, UNIX, Windows.

Note: Questa funzione viene intesa per I/O di basso livello e deve essere applicato al descrittore di file restituito da `open()` o da `pipe()`. Per scrivere in un "oggetto file" restituito dalle funzioni built-in `open()`, `popen()`, `fdopen()`, `sys.stdout` o `sys.stderr`, usate il suo metodo `write()`.

I seguenti elementi dato sono disponibili per la creazione del parametro *flags* della funzione `open()`.

O_RDONLY
O_WRONLY
O_RDWR
O_NDELAY
O_NONBLOCK
O_APPEND
O_DSYNC
O_RSYNC
O_SYNC
O_NOCTTY
O_CREAT
O_EXCL
O_TRUNC
Opzioni per l'argomento *flag* della funzione `open()`. Possono venire combinate con un OR bit per bit. Disponibiltà: Macintosh, UNIX, Windows.

O_BINARY
Opzione per l'argomento *flag* della funzione `open()`. Può venire combinata con le opzioni elencate sopra tramite un OR bit per bit. Disponibiltà: Macintosh, Windows.

O_NOINHERIT
O_SHORT_LIVED
O_TEMPORARY
O_RANDOM
O_SEQUENTIAL
O_TEXT
Opzioni per l'argomento *flag* della funzione `open()`. Puossono venire combinate tramite un OR bit per bit. Disponibilità: Windows.

6.1.4 File e directory

access(*path*, *mode*)
Usate gli uid/gid reali del processo corrente per verificare i permessi di accesso a *path*. Notate che la maggior parte delle operazioni useranno gli uid/gid effettivi, per cui questa funzione può venire usata in un ambiente suid/sgid, per verificare se l'utente che chiama la funzione può accedere a *path*. *mode* dovrebbe essere `F_OK` per verificare l'esistenza di *path*, oppure può essere l'OR inclusivo di uno o più tra i seguenti: `R_OK`, `W_OK` e `X_OK` per la verifica dei permessi. La funzione restituisce `True` se l'accesso è permesso, `False` altrimenti. Vedete la pagina di manuale della funzione UNIX *access*(2) per maggiori informazioni. Disponibilità: UNIX, Windows.

F_OK
Valore da passare al parametro *mode* della funzione `access()`, per verificare l'esistenza di *path*.

R_OK
Valore da includere nel parametro *mode* della funzione `access()`, per verificare la leggibilità di *path*.

W_OK
Valore da includere nel parametro *mode* della funzione `access()`, per verificare la scrivibilità di *path*.

X_OK
Valore da includere nel parametro *mode* della funzione `access()`, per determinare l'eseguibilità di *path*.

chdir(*path*)
Cambia la directory di lavoro corrente nel valore *path*. Disponibilità: Macintosh, UNIX, Windows.

fchdir(*fd*)
Cambia la directory di lavoro corrente ponendola uguale alla directory rappresentata dal descrittore di file *fd*. Il descrittore deve riferirsi ad una directory aperta, non ad un file. Disponibilità: UNIX. Nuovo nella versione 2.3.

getcwd()
Restituisce una stringa rappresentante la directory di lavoro corrente. Disponibilità: Macintosh, UNIX, Windows.

getcwdu()
Restituisce un oggetto Unicode rappresentante la directory di lavoro corrente. Disponibilità: UNIX, Windows. Nuovo nella versione 2.3.

chroot(*path*)
Cambia la directory radice del processo corrente nel valore *path*. Disponibilità: UNIX. Nuovo nella versione 2.2.

chmod(*path, mode*)
Cambia il modo di *path* nel valore numerico *mode*. *mode* può prendere uno dei valori seguenti (come definiti nel modulo `stat`):

- `S_ISUID`
- `S_ISGID`
- `S_ENFMT`
- `S_ISVTX`
- `S_IREAD`
- `S_IWRITE`
- `S_IEXEC`
- `S_IRWXU`
- `S_IRUSR`
- `S_IWUSR`
- `S_IXUSR`
- `S_IRWXG`
- `S_IRGRP`
- `S_IWGRP`
- `S_IXGRP`
- `S_IRWXO`
- `S_IROTH`
- `S_IWOTH`
- `S_IXOTH`

Disponibilità: UNIX, Windows.

chown(*path, uid, gid*)
Cambia l'identificativo del proprietario e del gruppo di *path*, ponendoli uguali ai valori numerici *uid* e *gid*. Disponibilità: UNIX.

lchown(*path, uid, gid*)
Cambia l'identificativo del proprietario e del gruppo di *path*, ponendoli uguali ai valori numerici *uid* e *gid*. Questa funzione non segue i link simbolici. Disponibilità: UNIX. Nuovo nella versione 2.3.

link(*src, dst*)
Crea un link fisico che punta a '*src*, nominandolo *dst*. Disponibilità: UNIX.

listdir(*path*)
Restituisce una lista contenente i nomi degli elementi della directory. La lista è in ordine arbitrario. Non include gli elementi speciali `'.'` e `'..'` benchè presenti nella directory. Disponibilità: Macintosh, UNIX, Windows.

Modificato nella versione 2.3: Su Windows NT/2k/XP ed in Unix, se *path* è un oggetto Unicode, anche il risultato sarà una lista di oggetti Unicode..

lstat(*path*)
Come la funzione `stat()`, ma non segue i link simbolici. Disponibilità: UNIX.

mkfifo(*path*[, *mode*])
Crea una FIFO (una pipe con un nome) chiamata come il valore del parametro *path* e avente come modo il valore numerico del parametro *mode*. Il valore predefinito di *mode* è `0666` (ottale). Il valore corrente della umask viene prima applicato come maschera a *mode*. Disponibilità: UNIX.

Le FIFO sono pipe a cui si può accedere come a dei file regolari. Le FIFO continuano ad esistere finché non vengono cancellate (ad esempio con `os.unlink()`). Generalmente le FIFO vengono usate come sincronizzazione tra processi "client" e processi "server": il server apre la FIFO in lettura ed il client la apre in scrittura. Notate che `mkfifo()` non apre la FIFO — crea solamente il punto di sincronizzzione.

mknod(*path*[, *mode=0600*, *device*])
Crea un nodo di filesystem (file regolare, file di dispositivo speciale, o una pipe con nome) chiamato come il valore del parametro filename. Il parametro *mode* specifica sia i permessi da usare che il tipo di nodo da creare, essendo una combinazione (con OR bit per bit) delle opzioni di permesso con uno tra: S_IFREG, S_IFCHR, S_IFBLK e S_IFIFO (queste costanti vengono definite nel modulo `stat`). Per S_IFCHR e S_IFBLK, il parametro *device* definisce il dispositivo speciale appena creato (probabilmente con `os.makedev()`), altrimenti viene ignorato. Nuovo nella versione 2.3.

major(*device*)
Estrae il numero maggiore di dispositivo da un numero di dispositivo grezzo. Nuovo nella versione 2.3.

minor(*device*)
Estrae il numero minore di dispositivo da un numero di dispositivo grezzo. Nuovo nella versione 2.3.

makedev(*major*, *minor*)
Compone un numero di dispositivo grezzo dai numeri maggiore e minore di dispositivo. Nuovo nella versione 2.3.

mkdir(*path*[, *mode*])
Crea una directory chiamata come il valore del parametro *path* con il modo numerico specificato dal parametro *mode*. Il modo predefinito è `0777` (ottale). In alcuni sistemi, *mode* viene ignorato. Dove usato, il valore corrente di umask viene prima applicato come maschera. Disponibilità: Macintosh, UNIX, Windows.

makedirs(*path*[, *mode*])
Funzione di creazione di directory ricorsiva. Come `mkdir()`, ma tutte le directory intermedie conterranno quella finale. Solleva un'eccezione `error` se la directory finale esiste già o non può essere creata. Il modo predefinito è `0777` (ottale). Questa funzione non gestisce in modo corretto i percorsi UNC (di rilievo solo per sistemi Windows; i percorsi Universal Naming Convention) sono quelli che usano la sintassi '`\\host\path`'. Nuovo nella versione 1.5.2.

pathconf(*path*, *name*)
Restituisce i dati di configurazione del sistema relativi al file nominato. Il parametro *name* specifica il valore di configurazione da ottenere; può essere una stringa corrispondente al nome di un definito parametro di sistema; questi parametri vengono specificati in un certo numero di standard (POSIX.1, UNIX 95, UNIX 98 e altri). Alcune piattaforme definiscono nomi aggiuntivi. I nomi noti al sistema operativo ospite sono disponibili nel dizionario `pathconf_names`. Per le variabili di configurazione non incluse in questo dizionario, è accettabile passare un numero intero come valore di *name*. Disponibilità: UNIX.

Se il parametro *name* è una stringa e non è nota, viene sollevata un'eccezione `ValueError`. Se un valore specifico per *name* non viene supportato dal sistema ospite anche se presente in `pathconf_names`, viene sollevata un'eccezione `OSError` con un valore pari a `errno.EINVAL` come codice di errore.

pathconf_names
Dizionario che mappa i nomi accettati da `pathconf()` e `fpathconf()` nei corrispondenti valori interi

definiti dal sistema operativo ospite. Può venire usato per determinare l'insieme dei nomi conosciuti dal sistema. Disponibilità: UNIX.

readlink(*path*)
Restituisce una stringa rappresentante il percorso a cui punta il link simbolico. Il risultato può essere un percorso relativo o assoluto; se è relativo, lo si può convertire in assoluto usando `os.path.join(os.path.dirname(`*path*`),` *result*`)`. Disponibilità: UNIX.

remove(*path*)
Rimuove il file indicato dal parametro *path*. Se *path* è una directory, viene sollevata un'eccezione `OSError`; vedete la funzione `rmdir()`, più avanti, per rimuovere una directory. Questa funzione è identica a functionunlink(), documentata più avanti. In Windows, cercare di rimuovere un file in uso causa il sollevamento di un'eccezione; in UNIX, viene rimosso il riferimento al file nella directory, ma lo spazio allocato al file non viene reso disponibile fino al momento in cui il file non sia più in uso. Disponibilità: Macintosh, UNIX, Windows.

removedirs(*path*)
Rimuove ricorsivamente le directory. Funziona come `rmdir()`, tranne per il fatto che, se la directory finale viene rimossa con successo, le directory via via corrispondenti al segmento più a destra del percorso specificato verrano "potate", fino a che l'intero percorso non sia stato cancellato, oppure non venga generato un errore (che viene ignorato, in quanto di solito questo significa che una directory genitrice non è vuota). Genera un'eccezione `error` se non è possibile rimuovere la directory finale. Nuovo nella versione 1.5.2.

rename(*src, dst*)
Rinomina il file o la directory *src* in *dst*. Se *dst* è una directory, viene sollevata un'eccezione `OSError`. Su UNIX, se *dst* esiste ed è un file, viene rimosso senza avviso se l'utente ne ha il permesso. L'operazione può fallire su alcuni tipi di UNIX se *src* e *dst* sono su filesystem diversi. In caso di successo, il cambio di nome sarà un'operazione atomica (questo è un requisito POSIX). In Windows, se *dst* esiste già, viene sollevata un'eccezione `OSError` anche se si tratta di un file; potrebbe non esistere modo di implementare una rinominazione atomica quando *dst* fa riferimento ad un file esistente. Disponibilità: Macintosh, UNIX, Windows.

renames(*old, new*)
Funzione ricorsiva per il cambio di nome a directory e file. Funziona come `rename()`, eccetto che prima cerca di creare tutte le directory intermedie necessarie perché *new* sia un percorso valido. Dopo il cambio di nome, le directory via via corrispondenti alla parte più a destra del percorso *old* verranno rimosse utilizzando `removedirs()`. Nuovo nella versione 1.5.2.

Note: Questa funzione può fallire dopo aver creato la nuova struttura di directory, se l'utente non ha i permessi necessari per rimuovere la directory finale del percorso o il file. Nuovo nella versione 1.5.2.

rmdir(*path*)
Rimuove la directory indicata da *path*. Disponibilità: Macintosh, UNIX, Windows.

stat(*path*)
Esegue una chiamata alla funzione di sistema `stat()` usando come argomento *path*. Il valore restituito è un oggetto i cui attributi corrispondono ai membri della struttura `stat`, e cioè: `st_mode` (bit di protezione), `st_ino` (numero di inode), `st_dev` (dispositivo), `st_nlink` (numero di link fisici), `st_uid` (identificativo utente del proprietario), `st_gid` (identificativo di gruppo del proprietario), `st_size` (dimensione del file in byte), `st_atime` (tempo dell'accesso più recente), `st_mtime` (tempo della modifica dei contenuti più recente), `st_ctime` (dipende dalla piattaforma; tempo indicante il più recente cambiamento dei metadata su UNIX, tempo di creazione su Windows).

Modificato nella versione 2.3: Se `stat_float_times` restituisce vero, i tempi sono dei numeri in virgola mobile che indicano secondi. Le frazioni di secondo possono venire indicate se il sistema lo supporta. Su Mac OS, i tempi sono sempre numeri in virgola mobile. Vedete `stat_float_times` per ulteriori dettagli. .

Su qualche tipo di Unix (ad esempio Linux), possono anche essere disponibili i seguenti attributi: `st_blocks` (numero di blocchi allocati per file), `st_blksize` (dimensione di un blocco usata dal filesystem), `st_rdev` (tipo di dispositivo, se si tratta di un inode che rappresenta un dispositivo).

Su sistemi Mac OS, possono anche essere disponibili i seguenti attributi: `st_rsize`, `st_creator`, `st_type`.

Su sistemi RISCOS, sono anche disponibili i seguenti attributi: `st_ftype` (tipo di file), `st_attrs` (attributi), `st_obtype` (tipo di oggetto).

Per compatibilità con le versioni precedenti, il valore restituito da `stat()` è anche accessibile come una tupla di almeno 10 numeri interi, che raccoglie i più importanti (e portabili) elementi della struttura `stat`, nell'ordine: `st_mode`, `st_ino`, `st_dev`, `st_nlink`, `st_uid`, `st_gid`, `st_size`, `st_atime`, `st_mtime`, `st_ctime`. Altri elementi possono venire aggiunti tramite alcune implementazioni. Il modulo standard `stat` definisce le funzioni e le costanti utili per estrarre informazioni da una struttura `stat`. (Su Windows, ad alcuni elementi vengono assegnati valori fittizi). Disponibilità: Macintosh, UNIX, Windows.

Modificato nella versione 2.2: Aggiunta la possibilità di accedere ai valori mediante gli attributi dell'oggetto restituito.

`stat_float_times`([*newvalue*])

Determina se `stat_result` deve rappresentare i tempi come numeri in virgola mobile. Se *newvalue* vale True, chiamate successive a `stat()` restituiranno i valori in virgola mobile; se vale False, successive chiamate a `stat()` restituiranno i valori come interi. Se *newvalue* viene omesso, la funzione restituisce l'impostazione corrente.

Per compatibilità con precedenti versioni di Python, utilizzando `stat_result` come tupla si ottengono sempre interi. Per compatibilità con Python 2.2, anche accedendo ai valori dei tempi attraverso i nomi dei campi, si ottengono interi. Le applicazioni che vogliano ottenere le frazioni di secondo dei tempi possono usare questa funzione per ottenere i tempi espressi in numeri in virgola mobile. Se riusciranno o meno ad ottenere effettivamente frazioni diverse da zero dipenderà dal sistema sottostante.

Future versioni di Python cambieranno il valore predefinito di questa impostazione; applicazioni che non possono gestire i tempi espressi come numeri in virgola mobile potranno usare questa funzione per disattivare la caratteristica.

Si raccomanda di cambiare questa impostazione solo alla partenza del programma nel modulo *__main__*; le librerie non dovrebbero mai cambiare questa impostazione. Se un'applicazione usa una libreria che non funziona correttamente nell'elaborare i tempi dei file in forma di numeri in virgola mobile, questa applicazione dovrebbe disattivare la caratteristica fino a che la libreria non sia stata corretta.

`statvfs`(*path*)

Esegue una chiamata di sistema `statvfs()` sul percorso indicato. Il valore restituito è un oggetto i cui attributi descrivono il file system del percorso indicato, e corrispondono ai componenti della struttura `statvfs` e cioè: `f_frsize`, `f_blocks`, `f_bfree`, `f_bavail`, `f_files`, `f_ffree`, `f_favail`, `f_flag`, `f_namemax`. Disponibilità: UNIX.

Per compatibilità con le versioni precedenti, il valore restituito è anche accessibile come tupla i cui elementi corispondono agli attributi dell'oggetto, nell'ordine descritto precedentemente. Il modulo standard `statvfs` definisce costanti che sono utili per estrarre informazioni da una struttura `statvfs` quando vi si accede come ad una sequenza; ciò è ancora utile qualora si scriva codice che deve funzionare con versioni di Python che non supportano l'accesso ai campi di `statvfs` in termini di attributi.

Modificato nella versione 2.2: Aggiunta la possibilità di accedere agli elementi di `statvfs` come attributi dell'oggetto restituito.

`symlink`(*src*, *dst*)

Crea un link simbolico che punta a *src*, chiamandolo *dst*. Disponibilità: UNIX.

`tempnam`([*dir*[, *prefix*]])

Restituisce un percorso unico che sia usabile per la creazione di un file temporaneo. Il valore restituito sarà un percorso assoluto che indica un potenziale file nella directory *'dir* o in una directory comunemente usata per i file temporanei se *dir* non viene specificata o vale `None`. Se *dir* viene specificata e non è `None`, allora *prefix* viene usato per fornire un prefisso breve al nome del file. Le applicazioni hanno la responsabilità di creare e gestire correttamente i file creati usando i percorsi restituiti da `tempnam()`; non viene effettuata nessuna pulizia automatica dei file temporanei. In Unix, la variabile ambientale TMPDIR ha precedenza sul parametro *dir*, mentre in Windows viene usata la variabile TMP. Il comportamento specifico di questa funzione dipende dall'implementazione della sottostante libreria C; alcuni aspetti sono specificati in modo insufficiente nella documentazione di sistema. **Avvertenze:** L'uso di `tempnam()` rende il programma vulnerabile ad attacchi di tipo symlink; considerate come alternativa l'uso di `tmpfile()`. Disponibilità: UNIX, Windows.

tmpnam()
Restituisce un percorso unico utilizzabile per creare un file temporaneo. Si tratterà di un percorso assoluto che corrisponde ad un potenziale file all'interno di una directory comunemente utilizzata per file temporanei. Le applicazioni sono responsabili di creare e gestire in modo appropriato i file creati utilizzando percorsi restituiti da `tmpnam()`; non viene fatta alcuna pulizia automatica. **Avvertenze:** L'uso di `tmpnam()` rende il programma vulnerabile ad attacchi di tipo symlink; considerate come alternativa l'uso di `tmpfile()`. Disponibilità: UNIX, Windows. Tuttavia questa funzione non dovrebbe essere usata in Windows, probabilmente: l'implementazione Microsoft di `tmpnam()` crea sempre un nome di file localizzato nella directory radice dell'unità corrente, e questa è generalmente una scelta infelice per un file temporaneo (a seconda dei privilegi, potreste non essere neanche in grado di aprire un file usando questo nome).

TMP_MAX
Il massimo numero di nomi unici che `tmpnam()` può generare prima di riusare nomi già generati.

unlink(*path*)
Rimuove il file indicato da *path*. Questa è la stessa funzione `remove()` con un altro nome; il nome `unlink()` è nella tradizione UNIX. Disponibilità: Macintosh, UNIX, Windows.

utime(*path, times*)
Imposta il tempo d'accesso e di modifica del file specificato da *path*. Se *times* vale `None`, allora i tempi di accesso e di modifica vengono impostati sul tempo corrente. Altrimenti *times* deve essere una coppia di numeri, della forma `(`*atime*`,` *mtime*`)`, usati rispettivamente come tempo di accesso e di modifica. Modificato nella versione 2.0: Aggiunto il supporto per *times* uguale a `None`. Disponibilità: Macintosh, UNIX, Windows.

walk(*top*[, *topdown*=*True* [, *onerror*=*None*]])
`walk()` genera i nomi dei file in un albero di directory, percorrendo l'albero dall'alto in basso o viceversa. Per ogni directory dell'albero con la radice in *top* (incluso lo stesso *top*), produce la tupla di 3 elementi `(`*dirpath*`,` *dirnames*`,` *filenames*`)`.

dirpath è una stringa, corrispondente al percorso della directory. *dirnames* è una lista dei nomi delle sotto directory di *dirpath* (escluse `'.'` e `'..'`). *filenames* è una lista dei nomi dei file di tipo non directory contenuti in *dirpath*. Notate che i nomi in queste liste non contengono i percorsi. Per ottenere un percorso completo (relativo a *top*) per un file o una directory in *dirpath*, eseguite `os.path.join(`*dirpath*`,` *name*`)`.

Se il parametro facoltativo *topdown* vale vero o non viene specificato, la tripla relativa ad una specifica directory viene sempre generata prima delle triple per ciascuna delle sue sotto directory. (le directory vengono generate dall'alto in basso). Se *topdown* vale falso, la tripla di una specifica directory viene generata dopo le triple di tutte le sue sotto directory (le directory vengono generate dal basso in alto).

Quando *topdown* vale vero, il chiamante può modificare la lista *dirnames* direttamente (ad esempio usando `del` o l'assegnamento di una fetta della lista), e `walk()` processerà solamente quelle directory i cui nomi sono ancora in *dirnames*; questo sistema può venire usato per limitare la ricerca, imporre un particolare ordine con cui le directory devono essere visitate o persino per comunicare a `walk()` i nomi di directory che create o rinominate dal chiamante prima che il controllo venga passato di nuovo a `walk()`. Modificare *dirnames* quando *topdown* vale falso non ha nessun effetto, perché procedendo dal basso verso l'alto, le directory in *dirnames* vengono analizzate prima di generare la stessa lista *dirnames*.

In modo predefinito, gli errori generati dalla chiamata a `os.listdir()` vengono ignorati. Se il parametro facoltativo *onerror* viene specificato, dovrebbe essere una funzione; tale funzione verrà chiamata con un argomento, che è un'istanza di os.error. La funzione può riportare l'errore e far procedere la scansione dell'albero, oppure può sollevare un'eccezione e far abortire la scansione. Notate che il nome del file relativo all'errore è disponibile come attributo `filename` dell'oggetto eccezione.

Note: Se passate alla funzione un percorso relativo, non cambiate la directory corrente di lavoro tra due continuazioni di `walk()`. `walk()` non cambia mai la directory corrente ed assume che il chiamante faccia lo stesso.

Note: Su sistemi che supportano i link simbolici, i link a sotto directory appaiono nelle liste *dirnames*, ma `walk()` non analizzerà queste directory (è difficile evitare dei loop infiniti quando si seguono link simbolici). Per analizare le directory indicate da link simbolici, potete identificarle usando `os.path.islink(`*path*`)` e quindi chiamare direttamente `walk(`*path*`)` su ciascuna di esse.

Questo esempio stampa il numero di byte usato da ciascun file, esclusi quelli di tipo directory, che si trovi in una sotto directory di quella di partenza, escluse le directory CVS e le loro sotto directory.

```
import os
from os.path import join, getsize
for root, dirs, files in os.walk('python/Lib/email'):
    print root, "usa",
    print sum([getsize(join(root, name)) for name in files]),
    print "byte in", len(files), "file di tipo non directory"
    if 'CVS' in dirs:
        dirs.remove('CVS')  # non visitare directory CVS
```

Nel prossimo esempio, il fatto di percorrere l'albero dal basso in alto è essenziale: `rmdir()` non permette di cancellare una directory se questa non è vuota.

```
import os
from os.path import join
# Cancella tutto ciò che è raqggiungibile dalla directory
# indicata in 'top'
# ATTENZIONE: è pericoloso! Per esempio se top == '/'
# potrebbe cancellare tutti i vostri file sul disco!!
for root, dirs, files in os.walk(top, topdown=False):
    for name in files:
        os.remove(join(root, name))
    for name in dirs:
        os.rmdir(join(root, name))
```

Nuovo nella versione 2.3.

6.1.5 Gestione dei processi

Queste funzioni possono venire usate per creare e gestire i processi.

Le diverse funzioni `exec*()` prendono in input una lista di argomenti, che vengono passati al programma caricato nel nuovo processo. In ciascun caso, il primo di questi argomenti passati al programma è il nome del programma stesso, piuttosto che un argomento digitato dall'utente sulla riga di comando. Per i programmatori in C, questo è il corrispondente dell'argomento `argv[0]` passato alla funzione `main()` di un programma. Per esempio, '`os.execv('/bin/echo', ['foo', 'bar'])`' stamperà solamente '`bar`' sullo standard output; '`foo`' verrà apparentemente ignorato.

`abort``()`
Genera un segnale `SIGABRT` per il processo corrente. In UNIX, il comportamento predefinito è quello di produrre un "core dump"; in Windows, il processo termina immediatamente restituendo un codice di uscita uguale a `3`. Fatte attenzione al fatto che i programmi che usano `signal.signal()` per registrare una funzione di gestione del segnale `SIGABRT`, si comporteranno diversamente. Disponibilità: UNIX, Windows.

`execl`(*path, arg0, arg1, ...*)
`execle`(*path, arg0, arg1, ..., env*)
`execlp`(*file, arg0, arg1, ...*)
`execlpe`(*file, arg0, arg1, ..., env*)
`execv`(*path, args*)
`execve`(*path, args, env*)
`execvp`(*file, args*)
`execvpe`(*file, args, env*)
Queste funzioni eseguono tutte un nuovo programma, rimpiazzando il processo corrente; pertanto esse non restituiscono mai il controllo al chiamante. In UNIX, il nuovo eseguibile viene caricato nel processo corrente, e avrà lo stesso identificativo del processo in cui viene eseguita la chiamata. Eventuali errori verranno riportati come eccezioni `OSError`.

Le varianti '`l`' e '`v`' delle funzioni `exec*()` si differenziano per il modo in cui i parametri sulla riga di

comando vengono passati. Le varianti '`l`' sono forse le più facili con cui lavorare se il numero di parametri non varia una volta scritto il codice; i singoli parametri semplicemente diventano argomenti addizionali delle funzioni `execl*()`. Le varianti '`v`' sono utili quando il numero di parametri è variabile, dato che vengono passati come lista o tupla, corrispondente all'argomento *args*. In ogni caso, i parametri sulla riga di comando per il processo figlio devono iniziare con il nome del comando che si sta per eseguire.

Le varianti che includono una '`p`' verso la fine del nome (`execlp()`, `execlpe()`, `execvp()` e `execvpe()`) faranno uso della variabile di ambiente PATH per localizzare il programma *file*. Quando le variabili di ambiente vengono sostituite (usando una delle varianti `exec*e()` discusse nel prossimo paragrafo), il nuovo ambiente viene utilizzato come sorgente per la variabile PATH. Le altre varianti, `execl()`, `execle()`, `execv()` ed `execve()` non faranno uso della variabile di ambiente PATH per localizzare l'eseguibile; in questo caso l'argomento *path* deve contenere un percorso appropriato, relativo o assoluto, per arrivare ad esso.

Per `execle()`, `execlpe()`, `execve()` e `execvpe()` (notate come tutte abbiano un nome che termina per '`e`'), l'argomento *env* deve essere un oggetto di tipo mappa, usato per definire le variabili di ambiente per il nuovo processo; le funzioni `execl()`, `execlp()`, `execv()` e `execvp()` fanno tutte in modo che il nuovo processo erediti l'ambiente del processo corrente. Disponibilità: UNIX, Windows.

`_exit`(*n*)
Termina immediatamente il programma con un codice di uscita pari a *n*, senza chiamare prima funzioni di pulizia finale, senza scaricare i buffer di 'stdio', o altro. Disponibilità: UNIX, Windows.

Note: Il modo standard di terminare un programma è `sys.exit(`*n*`)`. `_exit()` dovrebbe venire usata normalmente solo da processi figli creati da `fork()`.

Vengono definiti i seguenti codici di uscita utilizzabili con `_exit()`, sebbene non richiesti. Questi codici vengono di solito usati per programmi di sistema scritti in Python, come ad esempio un server di posta che per la consegna dei messaggi.

EX_OK
Codice di uscita indicante che non è stato rilevato nessun errore. Disponibilità: UNIX. Nuovo nella versione 2.3.

EX_USAGE
Codice di uscita indicante che il comando non è stato usato in modo corretto, come quando viene passato un numero errato di argomenti. Disponibilità: UNIX. Nuovo nella versione 2.3.

EX_DATAERR
Codice di uscita indicante che i dati in input erano errati. Disponibilità: UNIX. Nuovo nella versione 2.3.

EX_NOINPUT
Codice di uscita indicante che un file di input non esisteva o non era leggibile. Disponibilità: UNIX. Nuovo nella versione 2.3.

EX_NOUSER
Codice di uscita indicante che un utente specificato non esisteva. Disponibilità: UNIX. Nuovo nella versione 2.3.

EX_NOHOST
Codice di uscita indicante che un host specificato non esisteva. Disponibilità: UNIX. Nuovo nella versione 2.3.

EX_UNAVAILABLE
Codice di uscita indicante che un servizio richiesto non è disponbile. Disponibilità: UNIX. Nuovo nella versione 2.3.

EX_SOFTWARE
Codice di uscita indicante che è stato individuato un errore interno del software. Disponibilità: UNIX. Nuovo nella versione 2.3.

EX_OSERR
Codice di uscita indicante che è stato individuato un errore di sistema operativo, come ad esempio l'impossibilità di eseguire l'operazione di 'fork' o di creare una 'pipe'. Disponibilità: UNIX. Nuovo nella versione 2.3.

EX_OSFILE

Codice di uscita indicante che uno o più file di sistem non esistevano, non è stato possibile aprirli, o simili condizioni di errore. Disponibilità: UNIX. Nuovo nella versione 2.3.

EX_CANTCREAT
Codice di uscita indicante che non è stato possibile creare un file di output specificato dall'utente. Disponibilità: UNIX. Nuovo nella versione 2.3.

EX_IOERR
Codice di uscita indicante che si è verificato un errore durante l'esecuzione di operazioni I/O su qualche file. Disponibilità: UNIX. Nuovo nella versione 2.3.

EX_TEMPFAIL
Codice di uscita indicante che si è verificato un errore di tipo temporaneo. Questo codice indica il verificarsi di condizioni che potrebbero non essere degli errori, come l'impossibilità di effettuare una connessione di rete durante un'operazione che può comunque venire tentata di nuovo. Disponibilità: UNIX. Nuovo nella versione 2.3.

EX_PROTOCOL
Codice di uscita indicante che un protocollo di comunicazione era non permesso, non valido o comunque non comprensibile. Disponibilità: UNIX. Nuovo nella versione 2.3.

EX_NOPERM
Codice di uscita indicante che non vi erano i permessi necessari per eseguire l'operazione (ma questo non indica problemi sul file system). Disponibilità: UNIX. Nuovo nella versione 2.3.

EX_CONFIG
Codice di uscita indicante che si è verificato un qualche errore di configurazione. Disponibilità: UNIX. Nuovo nella versione 2.3.

EX_NOTFOUND
Codice di uscita indicante condizioni del tipo "elemento non trovato". Disponibilità: UNIX. Nuovo nella versione 2.3.

fork()
Sdoppia il processo corrente, creando un processo figlioi (NdT: fork). Restituisce `0` nel flusso di controllo del processo figlio, mentre in quello del processo padre restituisce l'identificativo del processo figlio. Disponibilità: UNIX.

forkpty()
Crea un processo figlio, usando un nuovo pseudo terminale come suo terminale di controllo. Restituisce una coppia del tipo (*pid*, *fd*), dove *pid* vale `0` nel processo figlio ed è uguale all'identificativo del processo figlio nel processo padre, mentre *fd* è il descrittore di file della terminazione master dello pseudo terminale. Per un approccio più portabile, usate il modulo `pty`. Disponibilità: alcune varianti di UNIX.

kill(*pid*, *sig*)
Uccide il processo *pid* mediante il segnale *sig*. Nel modulo `signal` vengono definite delle costanti che rappresentano i segnali disponibili sul sistema ospite. Disponibilità: UNIX.

killpg(*pgid*, *sig*)
Uccide il gruppo *pgid* mediante il segnale *sig*. Disponibilità: UNIX. Nuovo nella versione 2.3.

nice(*increment*)
Aggiunge il valore di *increment* al valore "nice" del processo. Restituisce il nuovo valore di "nice". Disponibilità: UNIX.

plock(*op*)
Blocca in memoria alcuni segmenti del programma. Il valore di *op* (definito in `<sys/lock.h>`) determina quali segmenti vengono bloccati. Disponibilità: UNIX.

popen(...)
popen2(...)
popen3(...)
popen4(...)
Esegue dei processi figli, restituendo delle pipe aperte che servono per comunicare con essi. Queste funzioni vengono descritte nella sezione 6.1.2.

spawnl(*mode, path, ...*)
spawnle(*mode, path, ..., env*)
spawnlp(*mode, file, ...*)
spawnlpe(*mode, file, ..., env*)
spawnv(*mode, path, args*)
spawnve(*mode, path, args, env*)
spawnvp(*mode, file, args*)
spawnvpe(*mode, file, args, env*)

Esegue il programma indicato da *path* in un nuovo processo. Se l'argomento *mode* vale `P_NOWAIT`, questa funzione restituisce l'identificativo del nuovo processo; se *mode* vale `P_WAIT`, restituisce il codice di uscita del processo se questo termina normalmente, oppure `-`*signal*, dove *signal* indica il segnale che ha ucciso il processo. In windows, l'identificativo del processo sarà in effetti l'handle dello stesso, in modo da poter venire usato con la funzione `waitpid()`.

Le varianti '`l`' e '`v`' della funzione `spawn*()` si differenziano nel modo in cui vengono passati i parametri da linea di comando vengono passati. Le varianti '`l`' sono forse le più facili con le quli lavorare se il numero di parametri viene fissato, una volta scritto il codice; i parametri individuali semplicemente diventano parametri aggiuntivi delle funzioni `spawnl*()`. Le varianti '`v`' sono adatte quando il numero dei parametri è variabile, dato che essi vengono passati come lista o tupla corrispondente all'argomento *args*. In entrambi i casi, gli argomenti del processo figlio devono cominciare con il nome del programma che tale processo sta per eseguire.

Le varianti che includono una seconda '`p`' verso la fine del nome (`spawnlp()`, `spawnlpe()`, `spawnvp()` e `spawnvpe()`) useranno la variabile di ambiente PATH per localizzare il programma *file*. Quando le variabili di ambiente vengono sostituite (usando una delle varianti `spawn*e()` discusse nel prossimo paragrafo), il nuovo ambiente viene utilizzato come sorgente della nuova variabile PATH. Le altre varianti, `spawnl()`, `spawnle()`, `spawnv()` e `spawnve()`, non useranno la variabile PATH per localizzare l'eseguibile; *path* deve contenere un percorso appropriato, assoluto o relativo.

Per quanto riguarda le funzioni `spawnle()`, `spawnlpe()`, `spawnve()`, e `spawnvpe()` (notate che tutte hanno nomi che terminano in '`e`'), l'argomento *env* deve essere un oggetto di tipo mappa usato per definire le variabili di ambiente per il nuovo processo; le funzioni `spawnl()`, `spawnlp()`, `spawnv()` e `spawnvp()` fanno tutte in modo che il nuovo processo erediti l'ambiente del processo corrente.

Come esempio, considerate come le seguenti chiamate a `spawnlp()` e `spawnvpe()` siano equivalenti.

```
import os
os.spawnlp(os.P_WAIT, 'cp', 'cp', 'index.html', '/dev/null')

L = ['cp', 'index.html', '/dev/null']
os.spawnvpe(os.P_WAIT, 'cp', L, os.environ)
```

Disponibilità: UNIX, Windows. `spawnlp()`, `spawnlpe()`, `spawnvp()` and `spawnvpe()` non sono disponibili in Windows. Nuovo nella versione 1.6.

P_NOWAIT
P_NOWAITO

Possibili valori per il parametro *mode* della famiglia di funzioni `spawn*()`. Se viene passato uno di questi valori, le funzioni di tipo `spawn*()` terminano non appena il nuovo processo viene creato, usando l'identificativo del nuovo processo come valore restituito. Disponibilità: UNIX, Windows. Nuovo nella versione 1.6.

P_WAIT

Possibile valore per il parametro *mode* della famiglia di funzioni `spawn*()`. Se viene passato questo valore come *mode*, le funzioni di tipo `spawn*()` non termineranno fino a che il nuovo processo non sia terminato, e restituiranno come valore di ritorno il codice di uscita del processo, se è stato possibile eseguirlo con successo, oppure `-`*signal*, se un segnale ha ucciso il processo. Disponibilità: UNIX, Windows. Nuovo nella versione 1.6.

P_DETACH
P_OVERLAY

Possibili valori per il parametro *mode* della famiglia funzioni `spawn*()`. Questi valori sono meno portabili rispetto a quelli precedentemente elencati. `P_DETACH` è simile a `P_NOWAIT`, ma il nuovo processo

viene staccato dalla console del processo padre. Se viene usato `P_OVERLAY`, il processo corrente viene rimpiazzato dal nuovo processo; in questo caso la funzione `spawn*()` non restituirà mai il controllo. Disponibilità: Windows. Nuovo nella versione 1.6.

`startfile`(*path*)
Avvia un file mediante l'applicazione ad essa associata. L'effetto è lo stesso che fare un doppio click sul file in Windows Explorer, o passare il nome del file al comando di **start** dalla shell interattiva: il file viene aperto con qualsiasi applicazione (se esiste) associata all'estensione del file.

`startfile()` restituisce il controllo non appena l'applicazione associata viene lanciata. Non ci sono opzioni per attendere che l'applicazione si chiuda, e non c'è modo di recuperare il valore di uscita della funzione. Il parametro *path* è relativo alla directory corrente. Se volete usare un percorso assoluto, assicuratevi che il primo carattere non sia uno slash ('`/`'); la sottostante funzione Win32 `ShellExecute()` non funzionerebbe in questo caso. Usate la funzione `os.path.normpath()` per assicurarvi che il percorso venga codificato correttamente per Win32. Disponibilità: Windows. Nuovo nella versione 2.0.

`system`(*command*)
Esegue il comando *command* (una stringa) in una shell di comandi eseguita da un sotto processo. Questa funzione viene implementata usando la funzione `system()` dello Standard C, e ha le sue stesse limitazioni. Cambi a `posix.environ`, `sys.stdin`, ecc., non vengono riflessi nell'ambiente del comando eseguito.

Su UNIX, il valore restituito è lo stato di uscita del processo codificato nel formato usato per `wait()`. Notate che POSIX non specifica il significato del valore di ritorno della funzione C `system()`, per cui il valore restituito della funzione Python dipende dal sistema su cui sta girando.

Su Windows, il valore restituito è quello restituito dalla shell di sistema dopo aver eseguito il comando *command*, ottenuto dalla variabile di ambiente di Windows COMSPEC: su sistemi con **command.com** (Windows 95, 98 e ME) questo valore è sempre `0`; in sistemi con **cmd.exe** (Windows NT, 2000 e XP) questo valore corrisponde allo stato di uscita del comando eseguito; per sistemi che usano una shell non nativa, consultate la documentazione della shell.

Disponibilità: UNIX, Windows.

`times`()
Restituisce una quintupla di numeri in virgola mobile che indicano i tempi cumulativi (di uso del processore ed altro), in secondi. Gli elementi della tupla sono, nell'ordine: tempo utente, tempo sistema, tempo utente usato dai sotto processi, tempo sistema usato dai sotto processi, e tempo reale trascorso da un punto di riferimento fisso nel passato. Vedete la pagina di manuale UNIX *times*(2) o la corrispondente documentazione della API della piattaforma Windows. Disponibilità: UNIX, Windows.

`wait`()
Attende il completamento di un processo figlio, e restituisce una tupla contenente il suo identificativo e l'indicazione del suo stato di uscita: quest'ultimo è un numero a sedici bit, il cui byte meno significativo corrisponde al codice di segnale che ha terminato il processo, mentre il suo byte più signifcativo corrisponde allo stato di uscita (se il codice del segnale è zero); il bit più significativo del byte meno significativo viene posto ad 1 se è stato prodotto un core file. Disponibilità: UNIX.

`waitpid`(*pid, options*)
I dettagli di questa funzione sono diversi in Windows e UNIX.

Su UNIX: Attende il completamento di un processo figlio con identificativo *pid*, e restituisce una tupla contenente il suo identificativo e l'indicazione del suo stato di uscita (codificata come per `wait()`). La semantica della chiamata viene influenzata dal valore del parametro intero *options*, che per operazioni normali dovrebbe valere `0`.

Se *pid* è più grande di `0`, `waitpid()` richiede informazioni di stato per il processo specifico. Se *pid* vale `0`, la richiesta viene effettuata per un qualunque sotto processo dello stesso gruppo del processo corrente. Se *pid* vale `-1`, la richiesta riguarda un qualsiasi processo figlio del processo corrente. Se *pid* è inferiore a `-1`, viene richiesto lo stato per un qualsiasi processo con identificativo di gruppo pari a `-`*pid* (valore assoluto di *pid*).

Su Windows: Attende il completamento di un processo identificato dall'handle *pid* e restituisce una tupla contenente *pid* e lo stato di uscita del processo spostato a sinistra di 8 bit (questo rende più semplice l'utilizzo multipiattaforma della funzione). Un valore del *pid* minore o uguale a `0` non ha significato particolare in Windows, e causa il sollevamento di un'eccezione. Il valore del parametro intero *options* non ha effetto. *pid* può riferirsi a qualunque processo di cui si conosca l'dentificativo, non necessariamente ad un processo

figlio. La funzione `spawn()` chiamata con `P_NOWAIT` restituisce degli handle adatti per venire usati con questa funzione.

`WNOHANG`
Questa è l'opzione per `waitpid()` che fa in modo che la funzione non si blocchi nel caso non sia immediatamente disponibile lo stato di alcun processo figlio. Disponibilità: UNIX.

`WCONTINUED`
Questa opzione fa in modo che venga riportato lo stato di un processo figlio se questo viene fatto continuare dopo essere stato fermato con un comando di controllo del job, all'ultimo controllo di stato. Disponibilità: alcuni sistemi UNIX. Nuovo nella versione 2.3.

`WUNTRACED`
Questa opzione fa in modo che lo stato dei processi figli venga riportato, qualora questi vengano bloccati ma il loro stato corrente non sia più stato riportato da quando sono stati bloccati. Disponibilità: UNIX. Nuovo nella versione 2.3.

Le seguenti funzioni accettano come argomento un codice di uscita di un processo come restituito da `system()`, `wait()`, o `waitpid()`. Possono venire usate per determinare il modo in cui è terminato il processo.

`WCOREDUMP`(*status*)
Restituisce `True` se viene generato un core dump per il processo, altrimenti restituisce `False`. Disponibilità: UNIX. Nuovo nella versione 2.3.

`WIFCONTINUED`(*status*)
Restituisce `True` se il processo viene fatto continuare dopo essere stato bloccato, altrimenti restituisce `False`. Disponibilità: UNIX. Nuovo nella versione 2.3.

`WIFSTOPPED`(*status*)
Restituisce `True` se il processo viene bloccato, altrimenti restituisce `False`. Disponibilità: UNIX.

`WIFSIGNALED`(*status*)
Restituisce `True` se il processo è terminato a causa di un segnale, altrimenti restituisce `False`. Disponibilità: UNIX.

`WIFEXITED`(*status*)
Restituisce `True` se il processo è terminato usando la funzione di sistema *exit*(2), altrimenti restituisce `False`. Disponibilità: UNIX.

`WEXITSTATUS`(*status*)
Se `WIFEXITED`(*status*) vale `True`, restistuisce il parametro intero passato alla funzione di sistema *exit*(2). Altrimenti, il valore restituito è privo di significato. Disponibilità: UNIX.

`WSTOPSIG`(*status*)
Restituisce il codice del segnale che ha bloccato il processo. Disponibilità: UNIX.

`WTERMSIG`(*status*)
Restituisce il codice del segnale che ha causato la terminazione del processo. Disponibilità: UNIX.

6.1.6 Informazioni di sistema di vario tipo

`confstr`(*name*)
Restituisce delle stringhe contenenti i valori dei parametri di configurazione del sistema. *name* specifica quale parametro di configurazione viene richiesto; può essere una stringa corrispondente al nome di un parametro d sistema definito; questi nomi vengono specificati in vari standard (POSIX, UNIX 95, UNIX 98, e altri). Alcune piattaforme definiscono nomi aggiuntivi rispetto agli standard. I nomi noti al sistema operativo ospite vengono contenuti nel dizionario `confstr_names`. Per variabili di configurazione non incluse nel dizionario, è consentito anche passare un intero per *name*. Disponibilità: UNIX.

Se un parametro di configurazione specificato da *name* non viene definito, viene restituita la stringa vuota.

Se *name* è una stringa e non è un parametro noto, viene sollevata l'eccezione `ValueError`. Se uno specifico valore non viene supportato dal sistema, anche se incluso in `confstr_names`, viene sollevata l'eccezione `OSError` con un codice di errore `errno.EINVAL`.

`confstr_names`

Dizionario che collega i nomi dei parametri di sistema accettati da `confstr()` ai codici interi definiti per tali parametri dal sistema operativo ospite. Può venire usato per definire l'insieme di nomi noti al sistema. Disponibilità: UNIX.

`getloadavg``()`
Restituisce la media del numero di processi nella coda di esecuzione del sistema, calcolata negli ultimi 1, 5 e 15 minuti, oppure solleva l'eccezione OSError se questo dato non è ottenibile.

Nuovo nella versione 2.3.

`sysconf``(`*name*`)`
Restituice il valore intero dei parametri di configurazione del sistema. Se il parametro di configurazione specificato da *name* non viene definito, viene restituito il valore `-1`. Quanto detto per l'argomento *name* di `confstr()` si applica anche in questo caso; il dizionario che fornisce le informazioni sui nomi noti è `sysconf_names`. Disponibilità: UNIX.

`sysconf_names`
Dizionario che collega i nomi di parametri di sistema accettati da `sysconf()` con i codici interi corrispondenti a tali parametri definiti dal sistema ospite. Può venire usato per determinare l'insieme dei nomi noti al sistema. Disponibilità: UNIX.

Le variabili seguenti hanno valori che possono venire usati a supporto delle operazoni di manipolazione dei nomi di percorso. Vengono definite per tutte le piattaforme.

Nel modulo `os.path` vengono definite operazioni di più alto livello sui nomi dei percorsi.

`curdir`
Corrisponde alla stringa usata dal sistema operativo per indicare la directory corrente. Per esempio: `'.'` per POSIX oppure `':'` per Macintosh. È anche disponibile nel modulo `os.path`.

`pardir`
Corrisponde alla stringa usata dal sistema operativo per riferirsi alla directory genitrice. Per esempio: `'..'` per POSIX oppure `'::'` per Macintosh. È anche disponibile nel modulo `os.path`.

`sep`
Il carattere usato dal sistema operativo per separare le componenti del nome di un percorso, per esempio '`/`' per POSIX oppure '`:`' per Macintosh. Notate che conoscere questo non basta per essere in grado di analizzare o concatenare nomi di percorso — meglio usare `os.path.split()` e `os.path.join()` — ma è comunque utile occasionalmente. È anche disponibile nel modulo `os.path`.

`altsep`
Un carattere alternativo usato per separare le componenti del nome di un percorso, oppure `None` se esiste un solo tipo di carattere separatore. Questa varisabile è posta uguale a '`/`' nei sistemi Windows, in cui la variabile `sep` vale backslash ('\'). È anche disponibile nel modulo `os.path`.

`extsep`
Il carattere che separa il nome base di un file dalla sua estensione; per esempio, il carattere '`.`' nel nome 'os.py'. È anche disponibile nel modulo `os.path`. Nuovo nella versione 2.2.

`pathsep`
Il carattere convenzionalmente usato dal sistema operativo per separare le componenti nelle liste dei percorsi di ricerca (come nel valore da assegnare alla variabile ambientale PATH), come ad esempio '`:`' per POSIX oppure '`;`' per Windows. È anche disponibile nel modulo `os.path`.

`defpath`
La lista di percorsi di ricerca usata in modo predefinito dalle funzioni di tipo `exec*p*()` e `spawn*p*()`, nel caso in cui l'ambiente non abbia una chiave `'PATH'`. È anche disponibile nel modulo `os.path`.

`linesep`
La stringa usata sulla piattaforma ospite per separare (o meglio, per terminare) le righe. Può essere un singolo carattere, come `'\n'` per POSIX oppure `'\r'` per Mac OS, o più caratteri, come per esempio `'\r\n'` per Windows.

6.2 `os.path` — Tipiche manipolazioni dei nomi di percorso

Questo modulo implementa alcune utili funzioni di manipolazione dei nomi di percorso.

Avvertenze: In Winidows, molte di queste funzioni non supportano propriamente i nomi di percorso di tipo UNC. Le funzioni `splitunc()` e `ismount()` li gestiscono correttamente.

`abspath`(*path*)
Restituisce la versione assolutizzata e normalizzata di un percorso *path*. In molte piattaforme, questo è equivalente a `normpath(join(os.getcwd(), path))`. Nuovo nella versione 1.5.2.

`basename`(*path*)
Restituisce la base del nome di percorso *path*. Ciò corrisponde alla seconda metà della coppia restituita da `split(path)`. Notate che il risultato di questa funzione è diverso da quello del programma UNIX **basename**; dove **basename** avendo in input `'/foo/bar/'` restituisce `'bar'`, la funzione `basename()` restituisce una stringa vuota (″).

`commonprefix`(*list*)
Restituisce la più lunga stringa possibile (considerata carattere per carattere) che costistuisce il prefisso comune a tutti i percorsi nella lista *list*. Se la lista è vuota, restituisce la stringa vuota (″). Notate che questa funzione può restituire un percorso non valido, considerato che lavora un carattere per volta.

`dirname`(*path*)
Restituisce il nome di directory contenuto nel percorso *path*. Corrisponde alla prima metà della coppia restituita da `split(path)`.

`exists`(*path*)
Restituisce `True` se *path* fa riferimento ad un percorso esistente.

`expanduser`(*path*)
Restituisce la stringa usata come argomento, sostituendo la sua componente iniziale uguale a '`~`' o '*~user*', con la directory home dell'utente *user*. Un carattere iniziale come '`~`' viene sostituito dal valore della variabile di ambiente HOME; un inizio della stringa come '*~user*' viene sostituito con la directory associata all'utente, ottenuta usando il modulo built-in `pwd`. Se l'espansione dell'argomento di input fallisce, o se l'argomento non comincia con un carattere tilde ('`~`'), viene restituito l'argomento di input non modificato. Su Macintosh, viene sembre restituito *path* non modificato.

`expandvars`(*path*)
Restituisce l'argomento, espandendo le variabili ambientali. Sottostringhe del tipo '$*name*' o '${*name*}' vengono sostituite dal valore della variabile ambientale *name*. Nomi specificati non correttamente o che si riferiscono a variabili non esistenti non vengono modificati. Su Machintosh, viene sempre restituito *path* non modificato.

`getatime`(*path*)
Restituisce il tempo dell'ultimo accesso a *path*. Il valore restituito è un numero che corrisponde al numero di secondi dall'epoca di riferimento (vedete il modulo `time`). Solleva l'eccezione `os.error` se il file non esiste o non è accessibile. Nuovo nella versione 1.5.2. Modificato nella versione 2.3: Se `os.stat_float_times()` restituisce True, restituisce True, il risultato è un numero in virgola mobile.

`getmtime`(*path*)
Restituisce il tempo dall'ultima modifica a *path*. Il valore restituito è un numero che corrisponde al numero di secondi dall'epoca di riferimento (vedete il modulo `time`). Solleva l'eccezione `os.error` se il file non esiste o non è accessibile. Nuovo nella versione 1.5.2. Modificato nella versione 2.3: Se `os.stat_float_times()` restituisce True, restituisce True, il risultato è un numero in virgola mobile.

`getctime`(*path*)
Restituisce il "ctime" del sistema, che, in qualche sistema (come UNIX) corrisponde al tempo dell'ultima modifica, mentre in altri (come Windows) corrisponde al tempo di creazione di *path*. Il valore restituito è un numero che corrisponde al numero di secondi dall'epoca di riferimento (vedete il modulo `time`). Se il file indicato da *path* non esiste o non è accessibile, viene sollevata l'eccezione `os.error`. Nuovo nella versione 2.3.

`getsize`(*path*)
Restituisce la dimensione, in byte, di *path*. Solleva l'eccezione `os.error` se il file non esiste o non è

accessibile. Nuovo nella versione 1.5.2.

isabs(*path*)
Restituisce `True` se *path* indica un percorso assoluto (comincia con un carattere slash ('`/`')).

isfile(*path*)
Restituisce `True` se *path* indica un file regolare esistente. Questa funzione segue i link simbolici, quindi sia `islink()` che `isfile()` possono restituire vero per uno stesso valore di *path*.

isdir(*path*)
Restituisce `True` se *path* rappresenta una directory esistente. Questa funzione segue i link simbolici, quindi sia `islink()` che `isdir()` possono restituire vero per uno stesso valore di *path*.

islink(*path*)
Restituisce `True` se *path* si riferisce ad un elemento di directory corrispondente ad un link simbolico. Restituisce sempre `False` su piattaforme in cui i link simbolici non vengono supportati.

ismount(*path*)
Restituisce `True` se il percorso indicato da *path* è un *mount point* (NdT: punto di mount): vale a dire un punto nel filesystem dove viene collegato un altro file system. La funzione controlla se la directory genitrice di *path*, ovvero '*path*`/..`', si trova su un dispositivo diverso da *path*, oppure se '*path*`/..`' e *path* puntano allo stesso i-node sullo stesso dispositivo — questo metodo dovrebbe poter identificare i punti di mount su tutte le varianti UNIX e POSIX.

join(*path1*[, *path2*[, ...]])
Unisce in modo intelligente uno o più componenti di un percorso. Se una qualsiasi delle componenti è un percorso assoluto, tutte quelle precedenti vengono scartate, e l'unione delle componenti continua. Il valore restituito è la concatenazione di *path1* e facoltativamente *path2*, eccetera, con esattamante un separatore di directory (`os.sep`) inserito tra le componenti, a meno che *path2* non sia vuoto. Notate che in Windows, dato che esiste una directory corrente per ogni drive, `os.path.join(c:, foo)` rappresenta un percorso relativo alla directory corrente del drive 'C:' ('c:foo'), e non 'c:\\foo'.

normcase(*path*)
Normalizza le maiuscole/minuscole all'interno del nome di un percorso. In UNIX, restituisce *path* non modificato; su file system non sensibili alla differenza tra maiuscole e minuscole, converte in minuscole tutte le lettere di *path*. In Windows, converte inoltre le barre '`/`' in barre rovesciate '`\`'.

normpath(*path*)
Normalizza il nome di un percorso. Questa funzione elimina i separatori ridondanti ed i riferimenti a directory di livello superiore, per esempio `A//B`, `A/./B` e `A/foo/../B` diventano tutti `A/B`. Non trasforma maiuscole in minuscole (usate `normcase()` per questo). In Windows, converte le barre '`/`' in barre rovesciate '`\`'.

realpath(*path*)
Restituisce il percorso canonico di uno specifico nome di file, eliminando i link simbolici presenti nel percorso. Disponibilità: UNIX. Nuovo nella versione 2.2.

samefile(*path1, path2*)
Restituisce `True` se entrambi gli argomenti, rappresentanti nomi di percorso, fanno riferimento allo stesso file o alla stessa directory (come indicato dal numero di dispositivo e dal numero di i-node). Solleva un'eccezione se la chiamata ad `os.stat()` su ciascuno degli argomenti fallisce. Disponibilità: Macintosh, UNIX.

sameopenfile(*fp1, fp2*)
Restituisce `True` se gli oggetti file *fp1* e *fp2* fanno riferimento allo stesso file. I due oggetti file possono rappresentare differenti descrittori di file. Disponibilità: Macintosh, UNIX.

samestat(*stat1, stat2*)
Restituisce `True` se le tuple di tipo stat, *stat1* e *stat2* si riferiscono allo stesso file. Tali tuple possono essere state restituite da `fstat()`, `lstat()`, o `stat()`. Questa funzione implementa il codice di confronto usato internamente da `samefile()` e `sameopenfile()`. Disponibilità: Macintosh, UNIX.

split(*path*)
Suddivide il percorso *path* in una coppia, (*head, tail*), dove *tail* è l'ultima componente del nome del percorso e *head* è tutto ciò che la precede. La parte *tail* non contiene mai una barra '`/`'; se l'argomento

path termina con una barra, *tail* sarà una stringa vuota. Se non c'è nessuna barra in *path*, *head* sarà una stringa vuota. Se *path* è una stringa vuota, sia *head* che *tail* saranno vuote. Eventuali barre aggiuntive vengono rimosse da *head*, a meno che questi non rappresenti la directory radice (composta solo da una o più barre). In quasi tutti i casi, `join(`*head*`, `*tail*`)` ricostruisce *path* (l'unica eccezione si verifica quando barre multiple separano *head* da *tail*).

`splitdrive``(`*path*`)`
Separa il nome di percorso *path* in una coppia `(`*drive* *tail*`)`, dove *drive* corrisponde al drive specificato nel percorso oppure è una stringa vuota. Sui sistemi che non usano specificare il drive nel percorso, *drive* sarà sempre una stringa vuota. In tutti i casi, *drive* `+` *tail* sarà sempre equivalente a *path*. Nuovo nella versione 1.3.

`splitext``(`*path*`)`
Separa il nome di percorso *path* in una coppia `(`*root*`, `*ext*`)`, tale che *radice* `+` *estensione* `==` *path*, e con *ext* vuoto oppure iniziante con un punto e contenente al massimo un punto.

`walk``(`*path, visit, arg*`)`
Chiama la funzione argomento *visit* con gli argomenti `(`*arg*`, `*dirname*`, `*nomes*`)`, per ciascuna directory contenuta nell'albero delle directory con radice in *path* (includendo lo stesso *path*, se è una directory). L'argomento *dirname* specifica la directory che si sta visitando, l'argomento *names* elenca i file nella suddetta directory (ottenuta chiamando `os.listdir(`*dirname*`)`). La funzione *visit* può modificare *names* per influenzare l'insieme delle directory visitate al di sotto di *dirname*, e quindi per evitare di visitare alcune sezioni dell'albero delle directory (l'oggetto a cui si riferisce *names* va modificato direttamente, usando `del` oppure l'assegnamento di una fetta).

Note: Link simbolici che puntano ad altre directory non vengono trattati come sotto directory, e quindi `walk()` non li visiterà. Per visitare una directory puntata da un link, dovete identificarla con `os.path.islink(`*file*`)` e `os.path.isdir(`*file*`)`, quindi chiamare `walk()` a seconda delle esigenze.

Note: Il più recente generatore `os.walk()` fornisce funzionalità simili e può essere più facile da usare.

`supports_unicode_filenames`
Vale True se una stringa Unicode arbitraria può venire usata come nome di file (nelle limitazioni imposte dal file system), e se `os.listdir()` restituisce stringhe Unicode in corrispondenza di un argomento Unicode. Nuovo nella versione 2.3.

6.3 `dircache` — Listati di directory memorizzati nella memoria cache

Il modulo `dircache` definisce una funzione per leggere i listati delle directory con l'aiuto di una memoria cache, che viene invalidata utilizzando il tempo di modifica (*mtime*) della directory. In aggiunta, il modulo definisce una funzione per marcare le directory, aggiungendovi una barra '`/`' alla fine.

Il modulo `dircache` definisce le seguenti funzioni:

`listdir``(`*path*`)`
Restituisce la lista del contenuto della directory *path*, così come ottenuta da `os.listdir()`. Notate che finché *path* non viene modificato, ulteriori chiamate a `listdir()` non riesamineranno più la struttura della directory.

Notate che la lista restituita dovrebbe essere trattata come in sola lettura. (Forse è opportuno che una versione futura restituisca una tupla?)

`opendir``(`*path*`)`
La stessa cosa di `listdir()`. Definita per compatibilità con le versioni precedenti.

`annotate``(`*head, list*`)`
Assume che *list* sia una lista di percorsi relativi a *head* ed aggiunge, modificandone gli elementi, una barra '`/`' ad ogni percorso corrispondente ad una directory.

```
>>> import dircache
>>> a = dircache.listdir('/')
>>> a = a[:] # Copia il valore restituito, in modo da poter cambiare 'a'
>>> a
['bin', 'boot', 'cdrom', 'dev', 'etc', 'floppy', 'home', 'initrd', 'lib', 'lost+
found', 'mnt', 'proc', 'root', 'sbin', 'tmp', 'usr', 'var', 'vmlinuz']
>>> dircache.annotate('/', a)
>>> a
['bin/', 'boot/', 'cdrom/', 'dev/', 'etc/', 'floppy/', 'home/', 'initrd/', 'lib/
', 'lost+found/', 'mnt/', 'proc/', 'root/', 'sbin/', 'tmp/', 'usr/', 'var/', 'vm
linuz']
```

6.4 `stat` — Interpretare i risultati di `stat()`

Il modulo `stat` definisce costanti e funzioni per interpretare i risultati di `os.stat()`, `os.fstat()` e `os.lstat()` (se esistono). Per dettagli più esaustivi sulle funzioni di sistema `stat()`, `fstat()` e `lstat()`, consultate la documentazione del vostro sistema.

Il modulo `stat` definisce le seguenti funzioni per fare verifiche su specifici tipi di file.

`S_ISDIR`(*mode*)
: Restituisce un valore non zero se *mode* proviene da una directory.

`S_ISCHR`(*mode*)
: Restituisce un valore non zero, se *mode* proviene da un file speciale che rappresenta un dipositivo con interfaccia a carattere.

`S_ISBLK`(*mode*)
: Restituisce un valore non zero se *mode* proviene da un file speciale che rappresenta un dispositivo con interfaccia a blocchi.

`S_ISREG`(*mode*)
: Restituisce un valore non zero se *mode* proviene da un file regolare.

`S_ISFIFO`(*mode*)
: Restituisce un valore non zero se *mode* proviene da una FIFO (un canale "pipe" associato ad un nome).

`S_ISLNK`(*mode*)
: Restituisce un valore non zero se *mode* proviene da un link simbolico.

`S_ISSOCK`(*mode*)
: Restituisce un valore non zero se *mode* proviene da un socket.

Sono inoltre definite due funzioni che manipolano in modo più generale il "modo" del file:

`S_IMODE`(*mode*)
: Restituisce quella parte del "modo" del file che può essere cambiata con `os.chmod()`—vale a dire, i bit dei permessi del file più il bit 'sticky', i bit di 'set-user-id' e di 'set-group-id' (per i sistemi che li supportano).

`S_IFMT`(*mode*)
: Restituisce quella parte del "modo" del file che descrive il tipo di file (usato dalle funzioni `S_IS*()` di cui sopra).

Normalmente, si usano le le funzion `os.path.is*()` per verificare il tipo di file; le funzioni descritte in questa sezione sono utili quando si stanno facendo controlli multipli sullo stesso file e si vuole evitare il lavoro aggiuntivo che si otterrebbe dall'esecuzione di `stat()` per ogni singolo controllo. Queste funzioni sono anche utili quando si vogliono controllare le informazioni su file non gestiti da `os.path`, come ad esempio i file che rappresentano dispositivi con interfaccia a carattere o a blocchi.

Tutte le variabili elencate in seguito sono semplicemente indici simbolici nella decupla restituita da `os.stat()`, `os.fstat()` o `os.lstat()`.

ST_MODE
Modo di protezione dell'inode.

ST_INO
Numero dell'inode.

ST_DEV
Dispositivo su cui risiede l'inode.

ST_NLINK
Numero di link che puntano all'inode.

ST_UID
Identificativo utente del proprietario.

ST_GID
Identificativo di gruppo del proprietario.

ST_SIZE
Per un file regolare, dimensione in byte; per alcun i tipi di file che rappresentano dispositivi, la quantità di dati in attesa.

ST_ATIME
Tempo dall'ultimo accesso.

ST_MTIME
Tempo dall'ultima modifica.

ST_CTIME
Il "ctime" come riportato dal Sistema Operativo. Su alcuni sistemi (simili a UNIX) è il tempo trascorso dall'ultimo cambiamento dei metadati e, su altri sistemi (simili a Windows) rappresenta il tempo trascorso dalla creazione (vedete la documentazione della piattaforma che volete usare per i dettagli).

L'interpretazione del termine "dimensione del file" cambia a secondo del tipo di file. Per file di tipo regolare, rappresenta la dimensione effettiva in byte. Per FIFO e socket, in molte varianti di UNIX (incluso ed in particolar modo Linux), la "dimensione" corrisponde al numero di byte in attesa di essere letti al momento in cui si è chiamato `os.stat()`, `os.fstat()`, oppure `os.lstat()`; questo può qualche volta essere utile, specialmente per interrogare periodicamente questi tipi di file speciali dopo averli aperti in modo non bloccante. Il significato del campo 'size', per altri file rappresentanti dispositivi a carattere e a blocchi, varia in misura maggiore e dipende molto dall'implementazione della funzione di sistema operativo sottostante.

Esempio:

```
import os, sys
from stat import *

def walktree(top, callback):
    '''discende ricorsivamente un albero di directory con radice in
       top, chiamando la funzione callback per ogni file di tipo
       regolare incontrato'''

    for f in os.listdir(top):
        pathname = os.path.join(top, f)
        mode = os.stat(pathname)[ST_MODE]
        if S_ISDIR(mode):
            # E' una directory, va analizzata ricorsivamente
            walktree(pathname, callback)
        elif S_ISREG(mode):
            # E' un file, chiama la funzione callback
            callback(pathname)
        else:
            # E' un file di tipo sconosciuto, stampa un messaggio
            print 'Escludo %s' % pathname

def visitfile(file):
    print 'Sto analizzando ', file

if __name__ == '__main__':
    walktree(sys.argv[1], visitfile)
```

6.5 `statcache` — Un'ottimizzazione per `os.stat()`

Deprecato dalla versione 2.2. Usate direttamente la funzione `os.stat()` invece di impiegare la memoria cache; questa introduce un livello molto alto di fragilità nelle applicazioni che ne fanno uso e ne complica il codice, con la necessità di avere un supporto per la gestione della memoria cache.

Il modulo `statcache` fornisce una semplice ottimizzazione per `os.stat()`: permette infatti di memorizzare i valori ottenuti da chiamate precedenti.

Il modulo `statcache` definisce le seguenti funzioni:

`stat`(*path*)
: Questa è l'interfaccia principale del modulo. È identica a `os.stat()`, con l'eccezione che ne memorizza il risultato per usi futuri.

Il resto delle funzioni vengono usate per svuotare la cache, o parte di essa.

`reset`()
: Svuota la memoria cache: rimuove dalla memoria tutti i risultati di precedenti chiamate a `stat()`.

`forget`(*path*)
: Rimuove dalla memoria cache il risultato di `stat(`*path*`)`, se esiste.

`forget_prefix`(*prefix*)
: Rimuove dalla memoria cache tutti i risultati di `stat(`*path*`)`, cominciando *path* con *prefix*.

`forget_dir`(*prefix*)
: Rimuove dalla memoria cache tutti i risultati di `stat(`*path*`)`, essendo *path* un file della directory *prefix*, includendo `stat(`*prefix*`)`.

`forget_except_prefix`(*prefix*)
: Simile a `forget_prefix()`, ma rimuove tutti i risultati *eccetto* quelli che cominciano con *prefix*.

Esempio:

```
>>> import os, statcache
>>> statcache.stat('.')
(16893, 2049, 772, 18, 1000, 1000, 2048, 929609777, 929609777, 929609777)
>>> os.stat('.')
(16893, 2049, 772, 18, 1000, 1000, 2048, 929609777, 929609777, 929609777)
```

6.6 `statvfs` — Costanti usate con la funzione `os.statvfs()`

Il modulo `statvfs` definisce delle costanti che consentono l'interpretazione dei risultati di `os.statvfs()`, che restituisce una tupla, senza doversi ricordare "numeri magici". Ciascuna delle costanti definite in questo modulo corrisponde all'*indice* di un elemento della tupla restituita da `os.statvfs()` che contiene uno specifico tipo di dato.

F_BSIZE
: Dimensione preferita per i blocchi del file system.

F_FRSIZE
: Dimensione di base per i blocchi del file system.

F_BLOCKS
: Numero totale dei blocchi del file system.

F_BFREE
: Numero totale dei blocchi liberi.

F_BAVAIL
: Numero dei blocchi disponibili per un utente non amministatore del sistema.

F_FILES
: Numero totale dei file node.

F_FFREE
: Numero totale dei file node liberi.

F_FAVAIL
: Numero dei file node disponibili per un utente non amministatore di sistema.

F_FLAG
: Opzioni. Dipendenti dal sistema. Vedete la pagina di manuale di `statvfs()`.

F_NAMEMAX
: Massima lunghezza del nome di un file.

6.7 `filecmp` — Confronti tra file e directory

Il modulo `filecmp` definisce funzioni che permettono di confrontare file e directory, con vari livelli facoltativi di compromesso tra tempo di calcolo e correttezza del risultato.

Il modulo `filecmp` definisce le seguenti funzioni:

`cmp`(*f1*, *f2*[, *shallow*[, *use_statcache*]])
: Confronta i file con nome *f1* e *f2*, restituendo `True` se appaiono uguali, altrimenti `False`.

  A meno che l'argomento *shallow* non abbia un valore falso, i file da cui si ottiene un identico risultato con `os.stat()` vengono considerati uguali. Modificato nella versione 2.3: L'argomento *use_statcache* viene considerato obsoleto e quindi ignorato..

  I file già confrontati usando questa funzione non vengono più confrontati finchè non cambia il loro risultato di `os.stat()`.

Notate che nessun programma esterno viene chiamato da questa funzione, che così risulta portabile ed efficiente.

cmpfiles(*dir1, dir2, common*[, *shallow*[, *use_statcache*]])
Restituisce tre liste di nomi di file: *match*, *mismatch* e *errors*. La lista *match* contiene la lista dei file risultanti uguali sia nella directory *dir1* che nella directory *dir2*; la lista *mismatch* include i nomi dei file che risultano diversi; la lista *errors* contiene i nomi dei file non confrontabili. Dei file possono venire inclusi nella lista *errors* perché l'utente può non avere i permessi per leggerli o per molte altre ragioni, ma sempre perché per qualche motivo non è stato possibile effettuare il confronto.

Il parametro *common* è una lista di nomi di file individuati in entrambe le directory (NdT: il confronto viene effettuato su questi file). I parametri *shallow* e *use_statcache* hanno gli stessi significati e gli stessi valori predefiniti degli argomenti omonimi della funzione `filecmp.cmp()`.

Esempio:

```
>>> import filecmp
>>> filecmp.cmp('libundoc.tex', 'libundoc.tex')
True
>>> filecmp.cmp('libundoc.tex', 'lib.tex')
False
```

6.7.1 La classe `dircmp`

Le istanze della classe `dircmp` vengono create usando questo costruttore:

class dircmp(*a, b*[, *ignore*[, *hide*]])
Crea un nuovo oggetto per il confronto di directory, utilizzabile per confrontare le directory *a* and *b*. L'argomento *ignore* è una lista di nomi da ignorare e ha come valore predefinito la lista `['RCS', 'CVS', 'tags']`. L'argomento *hide* è una lista di nomi da nascondere e ha come valore predefinito la lista `[os.curdir, os.pardir]`.

La classe `dircmp` fornisce i seguenti metodi:

report()
Stampa (su `sys.stdout`) il risultato del confronto tra le directory *a* e *b*.

report_partial_closure()
Stampa il risultato del confronto tra le immediate sotto directory comuni alle directory *a* e *b*.

report_full_closure()
Stampa il risultato del confronto tra *a* and *b* e le loro sotto directory comuni (ricorsivamente).

La classe `dircmp` offre un insieme di interessanti attributi che possono venire usati per ottenere informazioni varie sugli alberi di directory che vengono confrontati.

Notate che attraverso le estensioni del metodo `__getattr__()`, tutti gli attributi vengono calcolati in modo opportuno, in modo tale che non si presenti alcuna penalizzazione di velocità nel caso in cui vengano usati solo quegli attributi calcolabili velocemente.

left_list
Lista dei file e delle sotto directory di *a*, filtrati attraverso le liste *hide* e *ignore*.

right_list
Lista dei file e delle sotto directory di *b*, filtrati attraverso le liste *hide* e *ignore*.

common
Lista dei file e delle sotto directory presenti sia in *a* che in *b*.

left_only
Lista dei file e delle sotto directory presenti solo in *a*.

right_only
Lista dei file e delle sotto directory presenti solo in *b*.

common_dirs
Lista delle sotto directory sia in *a* che in *b*.

common_files
Lista dei file presenti sia in *a* che in *b*.

common_funny
Lista di nomi degli elementi comuni ad *a* e *b*, tali però da risultare di tipo diverso tra le directory, o aventi nomi per i quali `os.stat()` restituisce un errore.

same_files
Lista di file identici presenti sia in *a* che in *b*.

diff_files
Lista di file presenti sia in *a* che in *b*, ma con diverso contenuto.

funny_files
Lista di file presenti sia in *a* and *b* ma che non è stato possibile confrontare.

subdirs
Un dizionario che mappa i nomi in `common_dirs` con altrettante istanze della classe `dircmp`.

6.8 popen2 — Sotto processi con flussi di I/O accessibili

Questo modulo permette di creare sotto processi, di connettersi ai loro canali input/output/error e di ottenere i loro codici di uscita in UNIX e Windows.

Notate che a partire da Python 2.0, questa funzionalità è disponibile utilizzando le funzioni del modulo `os` che hanno lo stesso nome delle funzioni factory qui descritte, ma l'ordine dei valori restituiti è più intuitivo rispetto al quello delle varianti del modulo `os`.

L'interfaccia primaria offerta da questo modulo è composta da una terna di funzioni factory. Per ciascuna di queste, se viene specificato l'argomento *bufsize*, esso indica la dimensione del buffer per le pipe I/O. Il parametro *mode*, se fornito, dovrebbe esere la stringa `'b'` o `'t'`; in Windows questo è necessario per determinare se gli oggetti file devono venire aperti in modo binario o in modo testo. Il valore predefinito per *mode* è `'t'`.

L'unico modo di recuperare i codici restituiti per i processi figli è quello di utilizzare i metodi `poll()` o `wait()` sulle classi `Popen3` e `Popen4`; queste sono disponibili solo su UNIX. Questa informazione non è disponibile usando le funzioni `popen2()`, `popen3()` e `popen4()`, o le funzioni equivalenti nel modulo `os`.

popen2(*cmd*[, *bufsize*[, *mode*]])
Esegue *cmd* come processo derivato. Restituisce la coppia di oggetti file (*child_stdout*, *child_stdin*).

popen3(*cmd*[, *bufsize*[, *mode*]])
Esegue *cmd* come processo derivato. Restituisce la tripla di oggetti file (*child_stdout*, *child_stdin*, *child_stderr*).

popen4(*cmd*[, *bufsize*[, *mode*]])
Esegue *cmd* come processo derivato. Restituisce gli oggetti file (*child_stdout_and_stderr*, *child_stdin*). Nuovo nella versione 2.0.

Su UNIX, è anche disponibile una classe che definisce gli oggetti restituiti dalle funzioni factory. Queste non vengono usate nell'implementazione per Windows e non sono quindi disponibili per questa piattaforma.

class Popen3(*cmd*[, *capturestderr*[, *bufsize*]])
Questa classe rappresenta un processo derivato. Normalmente, istanze di `Popen3` vengono create utilizzando le funzioni factory `popen2()` e `popen3()` descritte in precedenza.

Se non si sta usando una delle funzioni dell'helper per creare oggetti `Popen3`, il parametro *cmd* è il comando di shell da eseguire nel processo derivato. L'opzione *capturestderr*, se è *True*, specifica che l'oggetto dovrebbe anche catturare i dati emessi sullo standard error del processo figlio. Il valore predefinito per questa opzione è *False*. Se il parametro *bufsize* viene specificato, esso specifica la dimensione dei buffer di I/O verso e dal sotto processo.

class Popen4(*cmd*[, *bufsize*])

Simile a Popen3, ma cattura sempre i dati emessi sullo standard error nello stesso oggetto file di quelli emessi sullo standard output. Istanze di questa classe vengono tipicamente generate usando popen4(). Nuovo nella versione 2.0.

6.8.1 Oggetti delle classi Popen3 e Popen4

Le istanze delle classi Popen3 e Popen4 hanno i seguenti metodi:

poll()
Restituisce -1 se il processo figlio non è ancora terminato, altrimenti restituisce il suo codice di uscita.

wait()
Attende che il processo figlio termini, e ne restituisce il suo codice di stato. Il codice di stato include sia il codice di uscita del processo che l'informazione sul fatto che il processo sia terminato eseguendo una exit() piuttosto che a causa di un segnale. Il modulo os definisce delle funzioni che aiutano ad interpretare il codice di stato; vedete la sezione 6.1.5 per la famiglia di funzioni W*().

Sono disponibili anche i seguenti attributi:

fromchild
Un file oggetto che permette di accedere ai dati emessi dal processo figlio sullo standard output. Per istanze di Popen4, questo oggetto fornirà accesso sia ai dati emessi sullo standard output che a quelli emessi sullo standard error.

tochild
Un file oggetto che fornisce accesso allo standard input del processo figlio.

childerr
Un file oggetto che fornisce l'output dell'errore da un processo figlio, se *capturestderr* ha valore vero per il costruttore, altrimenti None. Per instanze della classe Popen4, vale sempre None.

pid
L'identificativo ID del processo figlio.

6.8.2 Problemi con il controllo di flusso

Ogni volta che lavorate con qualche forma di comunicazione tra i processi, il controllo di flusso deve venire attentamente considerato. Questo è anche il caso con i file oggetto forniti da questo modulo (o con i suoi equivalenti nel modulo os).

Quando viene letto l'output di un sotto processo che scrive molti dati sullo standard error mentre il processo padre è occupato a leggerne i dati sullo standard output, si può verificare una situazione di blocco reciproco (deadlock). Una situazione simile può accadere con altre combinazioni di lettura/scrittura. Le cause essenziali di queste situazioni sono che da un lato un processo tenta di scrivere, in modo bloccante, più bytes di quanti ne può contenere il buffer (_PC_PIPE_BUF), mentre dall'altro lato, l'altro processo sta tentanto di leggere dati su un'altro canale, sempre in modo bloccante.

Esistono diversi metodi per gestire questa situazione.

Il modo più semplice di adattare l'applicazione, in molti casi, sarà quello di seguire il seguente modello nel processo padre:

```
import popen2

r, w, e = popen2.popen3('python slave.py')
e.readlines()
r.readlines()
r.close()
e.close()
w.close()
```

con codice come questo nel processo figlio:

```
import os
import sys

# notate che ciascuna delle istruzioni di stampa
# scrive una singola lunga linea

print >>sys.stderr, 400 * 'this is a test\n'
os.close(sys.stderr.fileno())
print >>sys.stdout, 400 * 'this is another test\n'
```

In particolare notate che `sys.stderr` va chiuso dopo aver scritto tutti i dati, oppure `readlines()` non terminerà. Notate anche che va usato `os.close()`, in quanto `sys.stderr.close()` non chiude il canale dello `stderr` (altrimenti riassegnare `sys.stderr` ne provocherebbe l'implicita chiusura, impedendo la stampa di ulteriori errori).

Applicazioni che hanno la necessità di supportare un approccio più generale, dovrebbero integrare nel loro ciclo di `select()`, l'input/output sui canali input/output/error dei processi figli, oppure usare thread separati per leggere ciascuno dei file forniti da qualsivoglia funzione `popen*()` o classe `Popen*` essi usino.

6.9 `datetime` — Tipi di base per data e tempo

Nuovo nella versione 2.3.

Il modulo `datetime` fornisce delle classi per manipolare date e tempi sia in modalità semplici che complesse. Anche se viene supportata l'aritmetica di tempi e date, l'implementazione del modulo si focalizza sopratutto su un'efficiente estrazione delle componenti, per la manipolazione e la formattazione dell'output.

Esistono due categorie di oggetti rappresentanti date e tempi: "semplici" e "complessi". Questa distinzione si riferisce al fatto che l'oggetto abbia la nozione del fuso orario, dell'ora legale o di altri tipi di correzione dei tempi dovuti a ragioni politiche o di calcolo. Dipende esclusivamente dal programma il fatto che un oggetto semplice di tipo `datetime` rappresenti il Tempo Coordinato Universale (UTC), il tempo locale o il tempo di qualche altro fuso orario, esattamente come dipende dal programma il fatto che un particolare numero rappresenti miglia, metri o massa. Oggetti semplici di tipo `datetime` sono facili da capire e da utilizzare, a costo però di ignorare alcuni aspetti della realtà rappresentata.

Per applicazioni che richiedano di più, gli oggetti di tipo `time` e `datetime` hanno un attributo facoltativo per l'informazione sul fuso orario, `tzinfo`, che può contenere un'istanza di una classe derivata della classe astratta `tzinfo`. Questi oggetti di tipo `tzinfo` includono informazioni circa la differenza tra tempo locale ed UTC, il nome del fuso orario locale, ed il fatto che sia attiva o meno l'ora legale. Notate che il modulo `datetime` non fornisce nessuna classe concreta di tipo `tzinfo`. Viene lasciato all'applicazione il compito di supportare la nozione di fuso orario, con il livello di dettaglio desiderato. Le regole per le correzioni del tempo applicate nel mondo sono di natura più politica che logica, e non esiste uno standard valido per qualunque applicazione.

Il modulo `datetime` esporta le seguenti costanti:

`MINYEAR`
Il più piccolo valore per l'anno consentito in un oggetto di tipo `date` o `datetime`. `MINYEAR` vale `1`.

`MAXYEAR`
Il più grande valore per l'anno consentito in un oggetto di tipo `date` o `datetime`. `MAXYEAR` vale `9999`.

Vedete anche:

Modulo `calendar` (sezione 5.19):
Funzioni generali collegate al tempo di calendario.

Modulo `time` (sezione 6.10):
Lettura e conversione dei tempi.

6.9.1 Tipi disponibili nel modulo `datetime`

class `date`
Una classe idealizzata e semplice per rappresentare le date, assumendo che il corrente calendario gregoriano sia sempre stato usato e che sempre lo sarà. Attributi: `year`, `month` e `day`.

class `time`
Una classe idealizzata per rappresentare il tempo, indipendente da un giorno particolare, che assume che ogni giorno sia esattamente 24*60*60 secondi (non vi è qui alcuna nozione dei leap seconds (NdT: sbalzi di secondi)). Attributi: `hour`, `minute`, `second`, `microsecond` e `tzinfo`.

class `datetime`
Una combinazione di una data e di un tempo. Attributi: `year`, `month`, `day`, `hour`, `minute`, `second`, `microsecond` e `tzinfo`.

class `timedelta`
Un intervallo temporale che rappresenta la differenza tra due tempi espressi da istanze di `date`, `time` o `datetime`, con risoluzione al microsecondo.

class `tzinfo`
Una classe base astratta per rappresentare gli oggetti informativi sul fuso orario. Questi oggetti vengono usati dalle classi `datetime` e `time` come strumento personalizzato di calcolo degli aggiustamenti del tempo (ad esempio, per tener conto del fuso orario e/o dell'ora legale).

Gli oggetti di ciascuno di questi tipi sono immutabili.

Gli oggetti di tipo `date` sono sempre di tipo semplice.

Un oggetto *d* di tipo `time` o `datetime` può essere sia semplice che complesso. *d* è complesso se l'attributo *d*`.tzinfo` ha un valore diverso da `None` e se la chiamata *d*`.tzinfo.utcoffset(`*d*`)` restituisce un valore diverso da `None`. Se *d*`.tzinfo` vale `None`, oppure se *d*`.tzinfo` non vale `None` ma *d*`.tzinfo.utcoffset(`*d*`)` restituisce `None`, allora *d*) è di tipo semplice.

La distinzione tra oggetti semplici e complessi non si applica agli oggetti di tipo `timedelta`.

Relazioni tra le classi derivate:

```
object
    timedelta
    tzinfo
    time
    date
        datetime
```

6.9.2 Oggetti di tipo `timedelta`

Un oggetto di tipo `timedelta` rappresenta una durata, la differenza tra due date o tempi.

class `timedelta`([*days*[, *seconds*[, *microseconds*[, *milliseconds*[, *minutes*[, *hours*[, *weeks*]]]]]]])
Tutti gli argomenti sono facoltativi ed il loro valore predefinito è `0`. Gli argomenti possono essere interi, long, o numeri in virgola mobile, e possono essere sia positivi che negativi.

Solo *days*, *seconds* e *microseconds* vengono memorizzati internamente. Gli altri argomenti del costruttore vengono convertiti in queste unità:

- Un millisecondo viene convertito in 1000 microsecondi.
- Un minuto viene convertito in 60 secondi.
- Un'ora viene convertita in 3600 secondi.
- Una settimana viene convertita in 7 giorni.

e giorni, secondi e millisecondi vengono quindi normalizzati in modo tale che la rappresentazione della durata sia unica e con queste caratteristiche:

- `0 <=` *microseconds* `< 1000000`
- `0 <=` *seconds* `< 3600*24` (il numero di secondi in un giorno)
- `-999999999 <=` *days* `<= 999999999`

Se uno degli argomenti è un numero in virgola mobile e vi sono frazioni di microsecondo, le frazioni avute come resto nella conversione di ogni argomento vengono sommate tra di loro e la somma viene arrotondata al microsecondo più vicino. Se nessun argomento è un numero in virgola mobile, la conversione e la normalizzazione sono esatte (non si perde alcuna informazione).

Se il valore normalizzato del numero di giorni è al di fuori dell'intervallo indicato, viene sollevata l'eccezione `OverflowError`.

Notate che la normalizzazione di valori negativi può essere soprendente di primo acchito. Per esempio:

```
>>> d = timedelta(microseconds=-1)
>>> (d.days, d.seconds, d.microseconds)
(-1, 86399, 999999)
```

Gli attributi di classe sono:

`min`
L'oggetto di tipo `timedelta` con il massimo valore negativo `timedelta(-999999999)`.

`max`
L'oggetto di tipo `timedelta` con il massimo valore positivo `timedelta(giorni=999999999, ore=23, minuti=59, secondi=59, microsecondi=999999)`.

`resolution`
La più piccola differenza possibile tra oggetti di tipo `timedelta` considerati non uguali, cioè `timedelta(microseconds=1)`.

Notate che, a causa della normalizzazione, si verifica che `timedelta.max` >`-timedelta.min`. Il valore `-timedelta.max` non è rappresentabile come un oggetto di tipo `timedelta`.

Attributi delle istanze (in sola lettura):

Attributo	Valore
`days`	Compreso tra -999999999 e 999999999, limiti inclusi
`seconds`	Compreso tra 0 e 86399, limiti inclusi
`microseconds`	Compreso tra 0 e 999999 limiti inclusi

Operazioni supportate:

Operazione	Risultato
t1 `=` *t2* `+` *t3*	Somma di *t2* e *t3*. Dopo l'operazione sono vere le espressioni *t1*-*t2* == *t3* e *t1*-*t3* == *t2*. (1)
t1 `=` *t2* `-` *t3*	Differenza fra *t2* e *t3*. Dopo l'operazione sono vere le espressioni *t1* == *t2* - *t3* e *t2* == *t1* + *t3*. (
t1 `=` *t2* `*` *i* o *t1* `=` *i* `*` *t2*	Delta moltiplicato per un intero o un intero long. Dopo l'operazione è vera l'espressione *t1* // i =
	In generale, l'espressione *t1* * i == *t1* * (i-1) + *t1* è vera. (1)
t1 `=` *t2* `//` *i*	Viene calcolato il valore approssimato per difetto ed il resto (se esiste) viene scartato. (3)
+*t1*	Restituisce un oggetto `timedelta` con lo stesso valore di *t1*. (2)
-*t1*	Equivalente a `timedelta`(-*t1.days*, -*t1.seconds*, -*t1.microseconds*), come pure a *t1** -1. (1)(4)
`abs(`*t*`)`	Equivalente a +*t* quando `t.days >= 0`, oppure equivalente a -*t* quando `t.days < 0`. (2)

Note:

(1) L'espressione è esatta, ma può causare un overflow.

(2) L'espressione è esatta e non può causare un overflow.

(3) Una divisione per zero solleva l'eccezione `ZeroDivisionError`.

(4) -*timedelta.max* non è rappresentabile come oggetto di tipo `timedelta`.

In aggiunta alle operazioni elencate sopra, gli oggetti di tipo `timedelta` supportano alcune addizioni o sottrazioni con oggetti di tipo `date` o `datetime` (vedete più avanti).

Confronti tra oggetti di tipo `timedelta` vengono supportati utilizzando l'oggetto di tipo `timedelta` rappresentante la più piccola durata, che viene considerato come il più piccolo oggetto di tipo `timedelta`. Allo scopo di evitare che venga usato il metodo predefinito di confronto tra gli indirizzi degli oggetti in caso di confronto tra tipi diversi, quando un oggetto di tipo `timedelta` viene confrontato con un oggetto di un tipo differente, viene sollevata l'eccezione `TypeError`, a meno che il confronto non sia `==` o `!=`, nei quali casi il vengono restituiti rispettivamente `False` o `True`.

Gli oggetti di tipo `timedelta` possono venire usati come chiavi dei dizionari (sono cioè "ashable"), supportano la serializzazione ("pickling") in modo efficiente e, in contesti booleani, un oggetto di tipo `timedelta` viene considerato vero se e solo se non è uguale a `timedelta(0)`.

6.9.3 Oggetti di tipo `date`

Un oggetto di tipo `date` rappresenta una data (anno, mese e giorno) in un calendario idealizzato corrispondente al corrente calendario Gregoriano, esteso indefinitamente in entrambe le direzioni. Il primo Gennaio dell'anno 1 viene definito come il giorno numero uno, il 2 Gennaio dell'anno 1 è il giorno due, e così via. Questo corrisponde alla definizione di calendario Gregoriano prolettico nel libro di Dershowitz and Reingold *Calendrical Calculations*, in cui tale calendario viene usato come base per tutti i calcoli. Vedete questo libro per conoscere gli algoritmi di conversione tra il calendario Gregoriano prolettico e molti altri sistemi di calendarizzazione.

class `date`(*year, month, day*)
Tutti gli argomenti sono obbligatori. Gli argomenti possono essere interi o interi long, all'interno dei seguenti intervalli:

- `MINYEAR <=` *year* `<= MAXYEAR`
- `1 <=` *month* `<= 12`
- `1 <=` *day* `<= numero dei giorni nel dato mese ed anno`

Se viene passato un argomento al di fuori di questi intervalli, viene sollevata l'eccezione `ValueError`.

Altri costruttori, tutti definiti come metodi di classe:

`today`()
Restituisce la data corrente in tempo locale. È equivalente a `date.fromtimestamp(time.time())`.

`fromtimestamp`(*timestamp*)
Restitusce la data in tempo locale corrispondente al valore di riferimento temporale in standard POSIX specificato come input, ad esempio quello restituito da `time.time()`. Questo metodo può sollevare l'eccezione `ValueError` se il riferimento temporale è fuori dall'intervallo dei valori supportati dall'implementazione disponibile della funzione C `localtime()`. Di solito i valori vengono ristretti agli anni compresi tra il 1970 e il 2038. Notate che su sistemi non POSIX che includono i leap seconds nella loro rappresentazione di riferimento temporale, tali secondi vengono ignorati dalla funzione `fromtimestamp()`.

`fromordinal`(*ordinal*)
Restituisce la data corrispondente all'ordinale del calendario Gregoriano prolettico, dove il primo Gennaio dell'anno 1 corrisponde all'ordinale 1. A meno che `1 <=` *ordinal* `<= date.max.toordinal()`, viene sollevata un'eccezione `ValueError`. Per ogni oggetto *d* di tipo date, è vera la condizione: `date.fromordinal(`*d*`.toordinal()) ==` *d*.

Attributi delle classi:

`min`
La data rappresentabile più lontana nel passato, `date(MINYEAR, 1, 1)`.

`max`
La data rappresentabile più lontana nel futuro, `date(MAXYEAR, 12, 31)`.

`resolution`
La più piccola differenza possibile tra oggetti di tipo date considerati non uguali, equivalente a `timedelta(days=1)`.

Attributi delle istanze (in sola lettura):

`year`
Anno. Compreso tra `MINYEAR` e `MAXYEAR`, limiti inclusi.

`month`
Mese. Compreso tra 1 e 12, limiti inclusi.

`day`
Giorno. Compreso tra 1 ed il numero di giorni del mese specificato nell'anno specificato.

Operazioni supportate:

Operazione	Risultato
date2 `=` *date1* `+` *timedelta*	*date2* rappresenta una data che precede *date1* di un numero di giorni pari a *timedelta*`.days`. (1)
date2 `=` *date1* `-` *timedelta*	Calcola *date2* tale che *date2* `+` *timedelta* `==` *date1*. (2)
timedelta `=` *date1* `-` *date2*	(3)
date1 `<` *date2*	Deve essere vero. *date1* viene considerato minore di *date2* quando rappresenta una data preceden

Note:

(1) *date2* viene spostata in avanti nel tempo se *timedelta*`.days > 0`, viene spostata all'indietro se *timedelta*`.days < 0`. Dopo l'operazione risulta che *date2* `-` *date1* `==` *timedelta*`.days`. Vengono ignorati *timedelta*`.seconds` e *timedelta*`.microseconds`. Se il valore da assegnare a *date2*`.year` risulta minore di `MINYEAR` o maggiore di `MAXYEAR`, viene sollevata l'eccezione `OverflowError`.

(2) Questo non è esattamente equivalente a date1 + (-timedelta), l'espressione -timedelta presa separatamente può provocare un overflow, mentre date1 - timedelta no. *timedelta*`.seconds` e *timedelta*`.microseconds` vengono ignorati.

(3) Questa espressione è esatta e non può provocare overflow. *timedelta*`.seconds` e *timedelta*`.microseconds` sono posti uguali a zero e dopo il calcolo è vera l'espressione *date2* `+` *timedelta* `== time1`.

(4) In altre parole, `date1 < date2` è vera se e solo se è vera *date1*`.toordinal() <` *date2*`.toordinal()`. Per impedire che il confronto ricada nel caso predefinito del confronto tra gli indirizzi degli oggetti, il confronto fra oggetti di tipo `date` solitamente solleva l'eccezione `TypeError` se l'altro termine di paragone non è anch'esso un oggetto di tipo `date`. Tuttavia, viene invece restituito `NotImplemented` nel caso in cui l'altro termine di paragone abbia un attributo `timetuple`. Questa estensione al meccanismo di confronto consente che generi di oggetti `date` diversi possano avere la possibilità di implementare dei confronti, pur non essendo dello stesso tipo. Ad eccezione di questo caso, se un oggetto di tipo `date` viene confrontato con un oggetto di altro tipo, viene sollevata l'eccezione `TypeError`, a meno che il confronto non sia `==` oppure `!=`. In questi ultimi vengono restituiti rispettivamente `False` o `True`.

Gli oggetti di tipo `date` possono venire usati come chiavi di dizionario. In contesti booleani, tutti gli oggetti di tipo `date` vengono considerati valori veri.

Metodi delle istanze:

`replace`(*year, month, day*)
Restituisce una data con lo stesso valore dell'istanza a cui si applica, tranne che per quei membri per cui vengono specificati nuovi valori attraverso gli argomenti specificati. Ad esempio, se `d == date(2002, 12, 31)`, allora `d.replace(day=26)` corrisponde a `date(2000, 12, 26)`.

`timetuple`()
Restituisce un oggetto di tipo `time.struct_time` come quelli restituiti da `time.localtime()`. Le ore, i minuti ed i secondi vengono posti a zero e l'opzione DST è -1. *d*`.timetuple()` è equivalente a `time.struct_time((`*d*`.year,` *d*`.month,` *d*`.day, 0, 0, 0,` *d*`.weekday(),` *d*`.toordinal() - date(`*d*`.year, 1, 1).toordinal() + 1, -1))`

toordinal()
Restituisce l'ordinale prolettico Gregoriano della data, dove il primo Gennaio dell'anno 1 ha ordinale 1. Per ogni oggetto *d* di tipo `date`, vale la relazione `date.fromordinal(`*d*`.toordinal()) == `*d*.

weekday()
Restituisce il giorno della settimana come intero, dove Lunedì è 0 e Domenica è 6. Per esempio l'espressione `date(2002, 12, 4).weekday()` corrisponde a `2`, ovvero un Mercoledì. Vedete anche `isoweekday()`.

isoweekday()
Restituisce il giorno della settimana in forma di intero, dove Lunedì corrisponde a 1 e Domenica a 7. Per esempio, l'espressione `date(2002, 12, 4).isoweekday()` corrisponde a `3`, ovvero un Mercoledì. Vedete anche `weekday()` e `isocalendar()`.

isocalendar()
Restituisce una tripla (anno ISO, numero ISO della settimana, giorno ISO della settimana).

Il calendario ISO è una variante largamente usata del calendario Gregoriano. Vedete il sito http://www.phys.uu.nl/~vgent/calendar/isocalendar.htm per una buona spiegazione in materia.

Il calendario ISO consiste di 52 o 53 settimane piene, nel quale una settimana comincia di Lunedì e termina di Domenica. La prima settimana di un anno ISO è la prima settimana calendariale dell'anno Gregoriano che contiene un Giovedì. Questa viene definita settimana numero 1, e l'anno ISO in cui si trova quel Giovedì coincide con il suo anno Gregoriano.

Per esempio, l'anno 2004 comincia con un Giovedì, per cui la prima settimana dell'anno ISO 2004 comincia di Lunedì 29 Dicembre 2003 e termina con Domenica, 4 Gennaio 2004, e quindi l'espressione `date(2003, 12, 29).isocalendar()` corrisponde a `(2004, 1, 1)`, e l'espressione `date(2004, 1, 4).isocalendar()` corrisponde a `(2004, 1, 7)`.

isoformat()
Restituisce una stringa che rappresenta la data in formato ISO 8601, cioè *'YYYY-MM-DD'*. Per esempio, l'espressione `date(2002, 12, 4).isoformat()` corrisponde a `'2002-12-04'`.

__str__()
Dato un oggetto *d* di tipo date, `str(`*d*`)` è equivalente a *d*`.isoformat()`.

ctime()
Restituisce una stringa che rappresenta la data, per esempio *date(2002,12, 4).ctime()* restituisce la stringa *'Wed Dec 4 00:00:00 2002'*. L'espressione *d*`.ctime()` è equivalente a `time.ctime(time.mktime(`*d*`.timetuple()))` su piattaforme dove la funzione nativa C `ctime()` (che è chiamata da `time.ctime()` ma non da `date.ctime()`) è conforme allo standard C.

strftime(*format*)
Restituisce una stringa rappresentante la data, secondo quanto specificato da un esplicito formato stringa. I codici di formato relativo alle ore, ai minuti ed ai secondi risulteranno in valori zero. Vedete anche la sezione che descrive il comportamento di `strftime()`.

6.9.4 Oggetti di tipo `datetime`

Un oggetto di tipo `datetime` contiene in una singola istanza le informazioni di un oggetto di tipo `date` e di un oggetto di tipo `time`. Come oggetto di tipo `date`, un oggetto di tipo `datetime` usa il calendario Gregoriano corrente esteso in entrambe le direzioni; come oggetto di tipo *time*, un oggetto di tipo `datetime` assume che vi siano esattamente 3600*24 secondi per ogni giorno.

Costruttore:

class datetime(*year, month, day*[, *hour*[, *minute*[, *second*[, *microsecond*[, *tzinfo*]]]]])
Gli argomenti year (anno), month (mese) e day (giorno) sono obbligatori. Il parametro *tzinfo* può essere `None` o un'istanza di una classe derivata di `tzinfo`. Gli altri argomenti possono essere interi o interi long, nei seguenti intervalli:

- `MINYEAR <=` *year* `<= MAXYEAR`

- `1 <=` *month* `<= 12`
- `1 <=` *day* `<= numero di giorni del mese e dell'anno specificati`
- `0 <=` *hour* `< 24`
- `0 <=` *minute* `< 60`
- `0 <=` *second* `< 60`
- `0 <=` *microsecond* `< 1000000`

Se viene passato un argomento esterno a questi intervalli, viene sollevata l'eccezione `ValueError`.

Altri costruttori, tutti metodi di classe:

`today``()`
: Restituisce un oggetto datetime che rappresenta la data e l'ora locale correnti, con `tzinfo` uguale `None`. Tale oggetto è equivalente a quello calcolato con l'espressione `datetime.fromtimestamp(time.time())`. Vedete anche le funzioni `now()` e `fromtimestamp()`.

`now``(`[*tz*]`)`
: Restituisce la data ed il tempo locali correnti. Se l'argomento facoltativo *tz* vale `None` o non viene specificato, il metodo è equivalente a `today()`, ma, se possibile, offre una maggiore precisione di quella che può ottenersi attraverso il valore di tempo restituito da `time.time()` (questo può ad esempio verificarsi su piattaforme che forniscano una implementazione della funzione C `gettimeofday()`).

  Se non vale `None`, allora l'argomento *tz* deve essere una istanza di una classe derivata di `tzinfo`, e data e tempo correnti vengono convertiti nel fuso orario espresso da *tz*. In questo caso, il risultato equivale all'espressione *tz*`.fromutc(datetime.utcnow().replace(tzinfo=`*tz*`))`. Vedete anche i metodi `today()`, `utcnow()`.

`utcnow``()`
: Restituisce la data ed il tempo corrente con riferimento UTC, con l'attributo `tzinfo` uguale a `None`. Questo metodo è simile a `now()`, ma restituisce la data ed il tempo correnti con riferimento UTC, come un oggetto `datetime` di tipo semplice. Vedete anche il metodo `now()`.

`fromtimestamp``(`*timestamp*[, *tz*]`)`
: Restituisce il tempo e la data locali corrispondenti al tempo di riferimento POSIX specificato dall'argomento *timestamp*, come restituito da `time.time()`. Se l'argomento facoltativo *tz* vale `None` o non viene specificato, allora il tempo di riferimento passato in input viene convertito nel tempo locale della piattaforma, e l'oggetto di tipo `datetime` restituito è di tipo semplice.

  Se non vale `None`, l'argomento *tz* deve essere una istanza di una classe derivata di `tzinfo`, ed il valore di tempo in input viene convertito nel fuso orario espresso da *tz*. In questo caso il risultato è equivalente all'espressione *tz*`.fromutc(datetime.utcfromtimestamp(`*timestamp*`).replace(tzinfo=`*tz*`))`.

  Il metodo `fromtimestamp()` può sollevare un'eccezione di tipo `ValueError`, nel caso che il tempo di riferimento sia esterno all'intervallo di valori supportati dall'implementazione delle funzioni C `localtime()` o `gmtime()` fornite dalla piattaforma ospite. È comune che il valore del tempo venga ristretto agli anni tra il 1970 ed il 2038. Notate che, su sistemi non POSIX che includano i leap seconds nella loro nozione di valori del tempo, tali secondi vengono ignorati dal metodo `fromtimestamp()`, ed è quindi possibile che due valori di tempo differenti per un secondo conducano a valori identici di `datetime`. Vedete anche il metodo `utcfromtimestamp()`.

`utcfromtimestamp``(`*timestamp*`)`
: Restituisce l'oggetto `datetime` rappresentante il tempo e la data UTC corrispondenti al valore di tempo POSIX passato come input, con l'attributo `tzinfo` posto uguale a `None`. Questo metodo può sollevare l'eccezione `ValueError`, nel caso che l'input sia esterno all'intervallo di valori supportati dalla funzione C `gmtime()` offerta dalla piattaforma ospite. È comune che questa funzione venga ristretta agli anni tra il 1970 ed il 2038. Vedete anche `fromtimestamp()`.

`fromordinal``(`*ordinal*`)`
: Restituisce l'oggetto di tipo `datetime` rappresentante l'ordinale del calendario prolettico Gregoriano, dove il primo Gennaio dell'anno 1 ha ordinale 1. Nel caso che non sia vera la condizione `1 <= ordinal`

`<= datetime.max.toordinal()`, viene sollevata l'eccezione `ValueError`. Gli attributi dell'oggetto corrispondenti all'ora, minuto, secondo e microsecondo vengono posti a zero, mentre `tzinfo` viene impostato a `None`.

`combine`(*date*, *time*)
Restituisce un nuovo oggetto di tipo `datetime` i cui membri vengono impostati agli stessi valori di quelli di `date`, mentre i membri `tzinfo` vengono impostati ai valori di quelli dell'argomento `time`. Per ogni oggetto *d* di tipo `datetime`, è vera l'espressione *d* `== datetime.combine(`*d*`.date(), `*d*`.timetz())`. Nel caso in cui l'argomento `date` sia un oggetto di tipo `datetime`, sia i suoi attributi rappresentanti il tempo che `tzinfo`, vengono ignorati.

Attributi di classe:

`min`
L'oggetto `datetime` corrispondente al più lontano tempo passato rappresentabile, cioè `datetime(MINYEAR, 1, 1, tzinfo=None)`.

`max`
L'oggetto `datetime` corrispondente al più lontano tempo futuro rappresentabile, cioè `datetime`, `datetime(MAXYEAR, 12, 31, 23, 59, 59, 999999, tzinfo=None)`.

`resolution`
La più piccola differenza possibile tra oggetti `datetime` non uguali, cioè `timedelta(microseconds=1)`.

Attributi d'istanza (in sola lettura):

`year`
Anno. Compreso tra `MINYEAR` e `MAXYEAR`, limiti inclusi.

`month`
Mese. Compreso tra 1 e 12, limiti inclusi.

`day`
Giorno. Compreso tra 1 ed il numero di giorni del mese e dell'anno specificato.

`hour`
Ora. Compreso nell'intervallo dato da `range(24)`.

`minute`
Minuto. Compreso nell'intervallo dato da `range(60)`.

`second`
Secondo. Compreso nell'intervallo dato da `range(60)`.

`microsecond`
Microsecondo. Compreso nell'intervallo dato da `range(1000000)`.

`tzinfo`
L'oggetto passato come argomento *tzinfo* al costruttore `datetime`, oppure `None` se l'argomento non viene specificato.

Operationi supportate:

Operazione	Risultato
datetime2 `=` *datetime1* `+` *timedelta*	(1)
datetime2 `=` *datetime1* `-` *timedelta*	(2)
timedelta `=` *datetime1* `-` *datetime2*	(3)
datetime1 `<` *datetime2*	Confronta due oggetti `datetime`. (4)

(1) *datetime2* rappresenta un tempo distante timedelta rispetto a *datetime1*, spostato nel futuro se *timedelta*`.days` > 0, o nel passato se *timedelta*`.days` < 0. Il risultato ha lo stesso membro `tzinfo` dell'oggetto `datetime` di input, e dopo l'operazione è vera l'espressione *datetime2* `-` *datetime1* `==` *timedelta*. Viene sollevata un'eccezione `OverflowError` se *datetime2.year* risulta minore di `MINYEAR` o maggiore di `MAXYEAR`. Notate che non viene effettuato alcun aggiustamento di fuso orario, anche se il primo operando è un oggetto di tipo complesso.

(2) Calcola datetime2 tale che sia vera l'espressione *datetime2* `+` *timedelta* `==` *datetime1*. Come nel caso dell'addizione, il risultato ha lo stesso attributo `tzinfo` del primo operando, e nessun aggiustamento di tempo dovuto al fuso orario viene effettuato, anche se il primo operando è un oggetto di tipo complesso. Questa operazione non è esattamente equivalente a *datetime1* `+ (-`*timedelta*`)`, perché il valore *-timedelta*, considerato isolatamente, può andare in overflow nei casi in cui con *datetime1* `-` *timedelta* ciò non accade.

(3) La sottrazione tra due oggetti di tipo `datetime` viene definita solo se entrambi gli operandi sono semplici o complessi. Se uno è complesso e l'altro è semplice, viene sollevata un'eccezione di tipo `TypeError`.

Se entrambi gli operandi sono semplici oppure sono complessi, ma hanno lo stesso valore per l'attributo `tzinfo`, tali valori vengono ignorati ed il risultato è un oggetto *t* di tipo `timedelta`, tale che *datetime2* `+` *t* `==` *datetime1*. In questo caso non viene effettuato alcun aggiustamento di fuso orario.

Se entrambi gli operandi sono complessi ed hanno attributi `tzinfo` non uguali, allora `a-b` funziona convertendo prima *a* e *b* in oggetti semplici con riferimendto ad UTC. Il risultato è equivalente a quello dell'espressione `(`*a*`.replace(tzinfo=None) -` *a*`.utcoffset()) -` `(`*b*`.replace(tzinfo=None) -` *b*`.utcoffset())`, eccetto per il fatto che l'implementazione reale non va mai in overflow.

(4) *datetime1* viene considerato inferiore a *datetime2* se lo precede nel tempo.

Se un termine di paragone è un oggetto semplice e l'altro un oggetto complesso, viene sollevata un'eccezione `TypeError`. Se entrambi i termini sono oggetti complessi ed hanno lo stesso valore per l'attributo `tzinfo`, allora questo viene ignorato e vengono semplicemente confrontati i tempi base. Se entrambi i termini sono complessi ed hanno diversi valori per l'attributo `tzinfo`, allora entrambi vengono prima corretti sottraendone le differenze di fuso orario rispetto al riferimento UTC (ottenuto da `self.utcoffset()`). **Note:** Per impedire che si ricada nel comportamento predefinito per il confronto di oggetti, che è quello di confrontarne gli indirizzi, il metodo di confronto della classe `datetime` normalmente solleva un'eccezione `TypeError` se uno dei due termini non è un oggetto `datetime`. Tuttavia, nel caso invece che tale oggetto abbia un attributo `timetuple`, viene restituito il valore `NotImplemented`. Questa estensione al meccanismo di confronto consente che generi di oggetti `date` diversi possano avere la possibilità di implementare dei confronti, pur non essendo dello stesso tipo. Se l'oggetto non ha un attributo `timetuple`, quando lo si confronta con un oggetto `datetime` viene sollevata un'eccezione `TypeError`, a meno che il confronto non sia `==` o `!=`. In questo caso, il confronto restituisce rispettivamente `False` e `True`.

Oggetti `datetime` possono venire usati come chiavi di dizionario. In un contesto booleano, tutti gli oggetti `datetime` vengono considerati veri.

Metodi di istanza:

`date``()`
Restituisce un oggetto `date` con lo stesso anno, mese e giorno.

`time``()`
Restituisce un oggetto `time` con gli stessi valori per ora, minuto, secondo e microsecondo. L'attributo `tzinfo` vale `None`. Vedete anche il metodo `timetz()`.

`timetz``()`
Restituisce un oggetto `time` con gli stessi valori di attributi per ora, minuto, secondo, microsecondo ed il fuso orario (tzinfo). Vedete anche il metodo `time()`.

`replace``(`[*year*[, *month*[, *day*[, *hour*[, *minute*[, *second*[, *microsecond*[, *tzinfo*]]]]]]]]`)`
Restituisce un oggetto `datetime` con gli stessi valori di attributo, eccetto per quelli per cui vengono forniti nuovi valori tramite gli argomenti passati per nome. Notate che si può specificare `tzinfo=None` per creare un oggetto semplice partendo da uno complessi, senza che venga effettuata alcuna conversione dei valori degli attributi di tempo e data.

`astimezone``(`*tz*`)`
Restituisce un oggetto `datetime` con l'attributo `tzinfo` che assume il valore dell'argomento *tz*, provocando una correzione dei valori degli attributi di tempo e data tale da ottenere lo stesso tempo dell'oggetto di partenza (*self*), ma nel fuso orario locale *tz*.

L'argomento *tz* deve essere un'istanza di una classe derivata di `tzinfo`, ed i sui metodi `utcoffset()` e `dst()` non devono restituire `None`. L'oggetto di partenza (*self*) deve essere di tipo complesso, (*self*`.tzinfo` deve essere diverso da `None` e *self*`.utcoffset()` non deve restituire `None`).

Nel caso in cui *self*`.tzinfo` sia uguale a *tz*, l'espressione *self*`.astimezone(`*tz*`)` risulta uguale a *self*: non viene effettuata alcuna correzione dei valori degli attributi di tempo e data. In caso contrario il risultato rappresenta il tempo, espresso nel fuso orario di *tz*, corripondente allo stesso UTC rappresentato da *self*: dopo aver eseguito l'istruzione *astz* `=` *dt*`.astimezone(`*tz*`)`, generalmente risulta vero che l'espressione *astz* `-` *astz*`.utcoffset()` abbia gli stessi membri di data e tempo corrispondenti a *dt* `-` *dt*`.utcoffset()`. La spiegazione a proposito della classe `tzinfo` tratta i casi limite, riguardanti i passaggi da e verso l'ora legale, in cui non è possibile ottenere l'egualianza di cui sopra.

Se volete semplicemente aggiungere un oggetto *tz* che rappresenti un fuso orario ad un oggetto *dt* di tipo `datetime`, senza correggere i valori di data e tempo di quest'ultimo, allora usate *dt*`.replace(tzinfo=`*tz*`)`. Se volete semplicemente rimuovere l'informazione sul fuso orario da un oggetto *dt* di tipo complesso, senza correzione dei valori di data e tempo, allora usate *dt*`.replace(tzinfo=None)`.

Notate che l'implementazione predefinita del metodo `tzinfo.fromutc()` può venire sostituita in una classe derivata di `tzinfo` allo scopo di influenzare il risultato restituito da `astimezone()`. Ignorando i casi di errore, `astimezone()` funziona così:

```
def astimezone(self, tz):
    if self.tzinfo is tz:
        return self
    # Converte self in UTC, e aggunge il nuovo oggetto rappresentante
    # il fuso orario.
    utc = (self - self.utcoffset()).replace(tzinfo=tz)
    # Converte da UTC al fuso orario espresso da tz.
    return tz.fromutc(utc)
```

`utcoffset``()`
Se l'attributo `tzinfo` vale `None`, restituisce `None`, altrimenti restituisce *self*`.tzinfo.utcoffset(`*self*`)` e solleva un'eccezione se quest'ultimo restituisce un valore diverso da `None` o da un oggetto `timedelta` che rappresenti un numero totale di minuti inferiore a quelli di un giorno.

`dst``()`
Se l'attributo `tzinfo` vale `None` restituisce `None`, altrimenti restituisce *self*`.tzinfo.dst(`*self*`)` e solleva un'eccezione se quest'ultimo restituisce un valore diverso da `None` e da un oggetto `timedelta` che rappresenti un numero totale di minuti inferiore a quelli di un giorno.

`tzname``()`
Se l'attributo `tzinfo` vale `None`, restituisce `None`, altrimenti restituisce *self*`.tzinfo.tzname(`*self*`)` e solleva un'eccezione se quest'ultimo non restituisce `None` o un oggetto stringa.

`timetuple``()`
Restituisce un oggetto `time.struct_time` così come restituito da `time.localtime()`. L'espressione *d*`.timetuple()` è equivalente a `time.struct_time((`*d*`.year, `*d*`.month, `*d*`.day, `*d*`.hour, `*d*`.minute, `*d*`.second, `*d*`.weekday(), `*d*`.toordinal() - date(`*d*`.year, 1, 1).toordinal() + 1, dst))`. L'opzione `tm_isdst` del risultato viene impostata in accordo col metodo `dst()`: se l'attributo `tzinfo` vale `None` oppure se `dst()` restituisce `None`, allora `tm_isdst` viene posto uguale a `-1`; se invece `dst()` restituisce un valore non nullo, `tm_isdst` viene posto uguale ad `1`; altrimenti `tm_dst` viene posto uguale a `0`.

`utctimetuple``()`
Se l'istanza *d* di `datetime` è di tipo semplice, allora questo metodo è equivalente a *d*`.timetuple()`, eccetto per il fatto che l'opzione `tm_isdst` viene forzata a `0`, indipendentemente dal valore restituito da `d.dst()`. Il DST (NdT: Daylight Saving Time, ora locale) non viene mai applicato ai tempi con riferimento UTC.

Se *d* è un oggetto di tipo complesso, allora viene normalizzato in tempo con riferimento UTC sottraendone *d*`.utcoffset()`, e viene quindi restituita un'istanza di `time.struct_time` per il tempo normalizzato. `tm_isdst` viene impostato a `0`. Notate che il risultato del valore dell'attributo `tm_year` potrebbe essere `MINYEAR`-1 o `MAXYEAR`+1, se *d*.year era `MINYEAR` o `MAXYEAR`, e la correzione UTC potrebbe andare oltre i limiti imposti agli anni.

toordinal()
Restituisce l'ordinale del calendario Gregoriano prolettico corrispondente alla data rappresentata dall'oggetto. È equivalente a `self.date().toordinal()`.

weekday()
Restituisce il giorno della settimana in forma di intero, dove Lunedì corrisponde a `0` e Sabato a `6`. È equivalente a `self.date().weekday()`. Vedete anche `isoweekday()`.

isoweekday()
Restituisce il giorno della settimana come intero, dove Lunedì corrisponde ad `1` e Sabato a `7`. È equivalente a `self.date().isoweekday()`. Vedete anche `weekday()` e `isocalendar()`.

isocalendar()
Restituisce una tupla di 3 elementi (anno ISO, numero ISO della settimana, giorno ISO della settimana). È equivalente a `self.date().isocalendar()`.

isoformat([*sep*])
Restituisce una stringa rappresentante la data ed il tempo in formato ISO 8601, YYYY-MM-DDTHH:MM:SS.mmmmmm, oppure, se `microsecond` vale `0`, YYYY-MM-DDTHH:MM:SS.

Se `utcoffset()` non restituisce `None`, viene aggiunta una stringa di 6 caratteri, indicante la differenza rispetto all'UTC in ore e minuti (con segno): YYYY-MM-DDTHH:MM:SS.mmmmmm+HH:MM oppure, se `microsecond` vale `0`, YYYY-MM-DDTHH:MM:SS+HH:MM

L'argomento facoltativo *sep* (il suo valore predefinito è `'T'`) è un separatore di un singolo carattere, piazzato tra la porzione data e la porzione tempo del risultato. Per esempio:

```
>>> from datetime import tzinfo, timedelta, datetime
>>> class TZ(tzinfo):
...     def utcoffset(self, dt): return timedelta(minutes=-399)
...
>>> datetime(2002, 12, 25, tzinfo=TZ()).isoformat(' ')
'2002-12-25 00:00:00-06:39'
```

__str__()
Per un'istanza *d* della classe `datetime`, `str(`*d*`)` è equivalente a *d*`.isoformat(' ')`.

ctime()
Restituisce una stringa rappresentante la data ed il tempo, ad esempio `datetime(2002, 12, 4, 20, 30, 40).ctime()` restituisce `'Wed Dec 4 20:30:40 2002'`. L'espressione `d.ctime()` è equivalente all'espressione `time.ctime(time.mktime(d.timetuple()))` sulle piattaforme in cui la funzione C `ctime()` (che viene chiamata da `time.ctime()` ma non da `datetime.ctime()`) risulti conforme allo standard del linguaggio C.

strftime(*format*)
Restituisce una stringa rappresentante la data ed il tempo, formattata secondo una specifica stringa di formato. Vedete la sezione sul comportamento di `strftime()`.

6.9.5 Oggetti di tipo `time`

Un oggetto di tipo time rappresenta il tempo (locale) all'interno di una giornata, indipendentemente da un giorno specifico, e suscettibile di correzioni attraverso un oggetto di tipo `tzinfo`.

class time(*hour*[, *minute*[, *second*[, *microsecond*[, *tzinfo*]]]])
Tutti gli argomenti sono facoltativi. L'argomento *tzinfo* può essere `None`, oppure un'istanza di una classe derivata di `tzinfo`. Gli altri argomenti possono essere sia interi che interi long, all'interno dei seguenti intervalli:

- `0 <=` *ora* `< 24`
- `0 <=` *minuto* `< 60`
- `0 <=` *secondo* `< 60`
- `0 <=` *microsecondo* `< 1000000`.

Se viene fornito un argomento al di fuori di questi intervalli, viene sollevata l'eccezione `ValueError`. Il valore predefinito per tutti è `0`, tranne che per *tzinfo*, il cui valore predefinito è `None`.

Attributi di classe:

`min`
Il minimo tempo rappresentabile, `time`, `time(0, 0, 0, 0)`.

`max`
Il massimo tempo rappresentabile, `time`, `time(23, 59, 59, 999999)`.

`resolution`
La più piccola differenza possibile tra oggetti di tipo `time` considerati non uguali, `timedelta(microseconds=1)`, ma notate che l'aritmetica tra oggetti di tipo `time` non viene supportata.

Attributi di istanza (in sola lettura):

`hour`
Ora, nell'intervallo `range(24)`.

`minute`
Minuto, nell'intervallo `range(60)`.

`second`
Secondo, nell'intervallo `range(60)`.

`microsecond`
Microsecondo, nell'intervallo `range(1000000)`.

`tzinfo`
L'oggetto passato come argomento `tzinfo` al costruttore `time`, oppure `None` se non viene passato alcun oggetto.

Operazioni supportate:

- confronto tra oggetti `time`, in cui *a* viene considerato inferiore a *b* se *a* precede nel tempo *b*. Se uno dei termini di paragone è "semplice" e l'altro è "complesso", viene sollevata un'eccezione di tipo `TypeError`. Se entrambi i termini di paragone sono "complessi", ed hanno lo stesso valore di attributo `tzinfo`, allora questo viene ignorato e vengono confrontati i tempi di base rappresentati. Se entrambi i termini sono "complessi" e hanno attributi `tzinfo` differenti, essi vengono prima corretti, sottraendo al tempo da loro rappresentato le rispettive differenze orarie da UTC (ottenuto da `self.utcoffset()`). Per evitare che il confronto tra tipi diversi ricada nel comportamento predefinito, cioè quello di confrontare gli indirizzi degli oggetti, quando viene confrontato un oggetto di tipo `time` con un oggetto di tipo diverso, viene sollevata un'eccezione `TypeError`, a meno che il confronto non sia `==` o `!=`. In questi ultimi casi il confronto restituisce rispettivamente `False` o `True`.
- hash, cioè la possibilità di venire usato come chiave di dizionario
- un'efficiente serializzazione
- in contesti booleani, un oggetto di tipo `time` viene considerato vero se e solo se, dopo essere stato convertito in minuti e aver sottratto `utcoffset()` (nel caso non sia `None`), il risultato è non nullo.

Metodi di istanza:

`replace``([`*hour*`[`*, minute*`[`*, second*`[`*, microsecond*`[`*, tzinfo*`]]]]])`
Restituisce un oggetto `time` con gli stessi valori, eccetto per quegli attributi per i quali vengono specificati nuovi valori. Notate che può venire specificato `tzinfo=None` per creare un oggetto `time` "semplice" a partire da un'oggetto `time` "complesso", senza dover convertire i membri del tempo rappresentato.

`isoformat``()`
Restituisce una stringa rappresentante il tempo in formato ISO 8601, HH:MM:SS.mmmmmm oppure, se `self.microsecond` è `0`, HH:MM:SS Se `utcoffset()` non restituisce `None`, allora viene aggiunta una stringa di sei caratteri, rappresentante la differenza oraria da UTC in ore e minuti (con segno): HH:MM:SS.mmmmmm+HH:MM oppure, se `self.microsecond` è `0`, HH:MM:SS+HH:MM

__str__()
Per un oggetto *t* di tipo `time`, `str(`*t*`)` è equivalente a *t*`.isoformat()`.

strftime(*format*)
Restituisce una stringa rappresentante il tempo, formattata secondo un'esplicita stringa di formato. Vedete la sezione sul comportamento di `strftime()`.

utcoffset()
Se `tzinfo` vale `None`, restituisce `None`, altrimenti restituisce *self*`.tzinfo.utcoffset(None)` e solleva un'eccezione nei casi in cui quest'ultimo restituisca qualcosa di diverso da `None`, oppure un oggetto `timedelta` rappresentante un numero di minuti inferiore a quello di un giorno.

dst()
Se `tzinfo` vale `None`, restituisce `None`, altrimenti restituisce *self*`.tzinfo.utcoffset(None)` e solleva un'eccezione nei casi in cui quest'ultimo restituisca qualcosa di diverso da `None`, oppure da un oggetto `timedelta` rappresentante un numero di minuti inferiore a quello di un giorno.

tzname()
Se `tzinfo` vale `None`, restituisce `None`, altrimenti restituisce *self*`.tzinfo.dst(None)`, oppure solleva un'eccezione se quest'ultimo restituisce qualcosa di diverso da `None` o da una stringa.

6.9.6 Oggetti di tipo `tzinfo`

La classe `tzinfo` è una classe di base astratta, vale a dire che questa classe non dovrebbe venire istanziata direttamente. Occorre invece derivarne una classe concreta e, come minimo, fornire un'implementazione dei metodi standard di `tzinfo` necessari per i metodi della classe `datetime` che si intende utilizzare. Il modulo `datetime` non fornisce nessuna classe derivata concreta di `tzinfo`.

Un'istanza di (una classe concreta di) `tzinfo` può venire passata al costruttore per ogetti di tipo `datetime` e `time`. Questi ultimi considerano il tempo espresso dai loro attributi come tempo locale, e l'oggetto `tzinfo` associato ad essi fornisce metodi per determinare la distanza tra il tempo locale e l'UTC, il nome del fuso orario, l'offset DST (NdT: la differenza di ora legale), tutti relativi ad un oggetto di tipo `datetime` o `time` passato come argomento.

Requisiti speciali per effettuare la serializzazione: una classe derivata di `tzinfo` deve avere un metodo `__init__` che possa venire chiamato senza argomenti, altrimenti tale oggetto potrà comunque venire serializzato, ma potrebbe succedere di non poterlo più deserializzare. Questo è un requisito tecnico che potrebbe essere reso meno rigido in futuro.

Una classe derivata concreta di `tzinfo` può dover implementare i seguenti metodi. Quali metodi esattamente siano necessari dipende dall'uso che si è fatto degli oggetti `datetime` di tipo "complesso". Se avete dei dubbi in proposito, semplicemente implementateli tutti.

utcoffset(*self, dt*)
Restituisce la distanza del tempo locale dall'UTC, in minuti e andando verso est rispetto all'UTC. Se il tempo locale è ad ovest dell'UTC, il risultato dovrebbe essere negativo. Notate che con questo si intende rappresentare la distanza totale dall'UTC; per esempio, se un oggetto `tzinfo` rappresenta sia il fuso orario che la correzione dovuta al DST (ora legale), allora `utcoffset()` dovrebbe restituire la somma dei due. Se l'offset UTC non è noto, `utcoffset()` restituisce `None`. Altrimenti, il valore restituito deve essere un oggetto `timedelta` che specifichi il numero complessivo di minuti nell'intervallo da -1439 a 1439, limiti inclusi (1440 = 24*60; la grandezza della distanza deve essere inferiore ad un giorno). La maggior parte delle implementazioni di `utcoffset()` saranno probabilmente simili ad una di queste due:

```
return CONSTANT                 #  classe con distanza fissa da UTC
return CONSTANT + self.dst(dt)  #  classe consapevole delle
                                #+ correzioni per l'ora legale
```

Se `utcoffset()` non restituisce `None`, allora non dovrebbe farlo neanche `dst()`.

L'implementazione predefinita di `utcoffset()` solleva l'eccezione `NotImplementedError`.

dst(*self, dt*)
Restituisce la correzione per l'ora legale (NdT: DST=Daylight Saving Time), in minuti orientati ad est ri-

spetto all'UTC, oppure restituisce None se quest'informazione non è nota. Restituisce timedelta(0) se l'ora legale non è applicata. Se l'ora è applicata, restituisce la correzione di tempo come un oggetto timedelta (vedete utcoffset() per i dettagli). Notate che la differenza per l'ora legale, se applicabile, è già stata aggiunta alla differenza di fuso orario restituita da utcoffset(), per cui non vi è necessità di chiamare dst(), a meno che non siate interessati ad ottenere l'informazione sull'ora legale separatamente. Per esempio, datetime.timetuple() chiama il metodo dst() dell'oggetto referenziato dal suo attributo tzinfo per stabilire come l'opzione tm_isdst dovrebbe venire impostata, e tzinfo.fromutc() chiama dst() per tener conto dei cambi di ora legale quado si attraversano i fusi orari.

Un'istanza *tz* di una classe derivata di tzinfo che modellizzi sia l'ora solare che quella legale deve essere consistente, nel senso che l'espressione:

tz.utcoffset(*dt*) - *tz*.dst(*dt*)

deve restituire lo stesso risultato per ogni oggetto *dt* di tipo datetime tale che *dt*.tzinfo == *tz*. Per classi derivate di tzinfo bene implementate, questa espressione corrisponde alla "differenza standard", che non dovrebbe dipendere dalla data o dal tempo, ma solo dalla posizione geografica. L'implementazione di datetime.astimezone() fa affidamento su questa assunzione, ma non è in grado di individuarne le violazioni; è responsabilità del programmatore fare in modo che non ve ne siano. Se una classe derivata di tzinfo non può garantire tale condizione, può alternativamente tentare di sostituire l'implementazione predefinita di tzinfo.fromutc() in modo da funzionare correttamente con astimezone(), indipendentemente dal fatto che la condizione sia vera.

La maggior parte delle iplementazioni di dst() probabilmente somiglieranno ad una di queste due:

```
def dst(self):
    # una classe con distanza fissa: non tiene conto del DST
    return timedelta(0)
```

oppure

```
def dst(self):
    # Codice per porre il valore di "dston" e "dstoff" ai tempi di
    # transizione dell'ora legale per il fuso orario considerato,
    # basandosi su dt.year ed esprimendo tali tempi in tempo locale.
    # Quindi:

    if dston <= dt.replace(tzinfo=None) < dstoff:
        return timedelta(hours=1)
    else:
        return timedelta(0)
```

L'implementazione predefinita di dst() solleva l'eccezione NotImplementedError.

tzname(*self, dt*)

Restituisce una stringa corrispondente al nome della zona di fuso orario corrispondente all'argomento *dt*, di tipo datetime. Nel modulo datetime non viene definito niente riguardo a tali nomi, e non vi sono requisiti che abbiano un qualche significato specifico. Per esempio, GMT, UTC, -500, -5:00, EDT, US/Eastern, America/New York sono tutte risposte valide. Questo metodo deve restituire None se il nome di una stringa non è noto. Notate che la ragione principale per cui questo è un metodo piuttosto che una stringa di valore costante, consiste nel fatto che è possibile che una classe derivata di tzinfo voglia restituire nomi differenti per diversi valori dell'argomento, specialmente nel caso in cui la classe tzinfo tenga conto dell'ora legale.

L'implementazione predefinita di tzname() solleva l'eccezione NotImplementedError.

Questi metodi vengono chiamati da un oggetto di tipo datetime o time, in risposta a chiamate dei loro metodi con lo stesso nome. Un oggetto datetime passa se stesso come argomento mentre un oggetto time passa None come argomento. I metodi di una classe derivata di tzinfo devono dunque essere preparati ad accettare come argomento sia un'istanza di datetime, ossia *dt*, che None.

Quando None viene passato come argomento, il compito di decidere la risposta migliore viene lasciato al progettista della classe derivata di tzinfo. Per esempio, restituire None è appropriato se nell'implementazione della

classe si vuole specificare che gli oggetti di tipo `time` non sono interessati alla gestione del fuso orario. Può però essere più utile che `utcoffset(None)` restituisca la distanza standard dall'UTC, visto che non vi è altro modo per stabilire la distanza standard.

Quando viene passato un oggetto `datetime` come risultato di una chiamata ad un metodo di `datetime`, allora `dt.tzinfo` sarà uguale a *self*. I metodi di `tzinfo` possono fare affidamento su questo, a meno che il codice utente non chiami direttamente i metodi della classe `tzinfo`. La ragione di ciò è di consentire che i metodi di `tzinfo` considerino l'argomento *dt* come rappresentante il tempo locale, senza preoccuparsi di oggetti rappresentanti un tempo espresso in fusi orari differenti.

Vi è un altro metodo della classe `tzinfo` che le classi derivate possono voler sostituire:

`fromutc`(*self, dt*)

Questo metodo viene chiamato nella implementazione predefinita di `datetime.astimezone()`. Usato in questo modo, *dt*`.tzinfo` corrisponde a *self*, e gli attributi di data e tempo di *dt* vanno considerati come rappresentanti un tempo con riferimento UTC. Lo scopo di `fromutc()` è quello di correggere gli attributi di data e tempo, resituendo un oggetto datetime equivalente ma rappresentante il tempo nel fuso orario di *self*.

La maggior parte delle classi derivate di `tzinfo` dovrebbero essere in grado di ereditare l'implementazione predefinita di `fromutc()` senza problemi. Tale implementazione è abbastanza robusta da gestire fusi orari con differenza oraria costante e fusi orari che tengano conto sia dell'ora solare (standard) che dell'ora legale, ed in quest'ultimo caso può persino gestire i casi in cui i tempi di passaggio all'ora legale cambino da un anno all'altro. Un esempio di una situazione che l'implementazione predefinita di `fromutc()` potrebbe non gestire sempre correttamente si ha quando la differenza oraria standard (rispetto all'UTC) dipende da una specifica data e tempo passato, situazione che si può verificare per motivi politici. L'implementazione predefinita di `astimezone()` e `fromutc()` può non produrre il risultato voluto, se il risultato è una delle ore successive al momento in cui la differenza oraria è cambiata.

Ignorando il codice necessario per gestire i casi di errore, l'implementazione di default di `fromutc()` funziona così:

```
def fromutc(self, dt):
    # solleva l'eccezione ValueError se dt.tzinfo è diverso da self
    dtoff = dt.utcoffset()
    dtdst = dt.dst()
    # solleva ValueError se dtoff o dtdts sono uguali a None
    delta = dtoff - dtdst  #  questa è la differenza oraria
                           #+ standard di self
    if delta:
        dt += delta   # converte nel tempo locale standard
        dtdst = dt.dst()
        # solleva l'eccezione ValueError se dtdst vale None
    if dtdst:
        return dt + dtdst
    else:
        return dt
```

Esempi di classi derivate di `tzinfo`:

```
from datetime import tzinfo, timedelta, datetime

ZERO = timedelta(0)
HOUR = timedelta(hours=1)

# Una classe UTC.

class UTC(tzinfo):
    """UTC"""

    def utcoffset(self, dt):
        return ZERO
```

```
    def tzname(self, dt):
        return "UTC"

    def dst(self, dt):
        return ZERO

utc = UTC()

# Una classe che crea oggetti tzinfo per fusi orari con distanza
# oraria fissa dall'UTC.  Notate che FixedOffset(0, "UTC") è un
# altro modo per creare un oggetto tzinfo rappresentante l'UTC.

class FixedOffset(tzinfo):
    """Distanza fissa in minuti ad est dell'UTC."""

    def __init__(self, offset, name):
        self.__offset = timedelta(minutes = offset)
        self.__name = name

    def utcoffset(self, dt):
        return self.__offset

    def tzname(self, dt):
        return self.__name

    def dst(self, dt):
        return ZERO

#  Una classe che cattura il concetto di tempo locale
#+ supportato dalla piattaforma

import time as _time

STDOFFSET = timedelta(seconds = -_time.timezone)
if _time.daylight:
    DSTOFFSET = timedelta(seconds = -_time.altzone)
else:
    DSTOFFSET = STDOFFSET

DSTDIFF = DSTOFFSET - STDOFFSET

class LocalTimezone(tzinfo):

    def utcoffset(self, dt):
        if self._isdst(dt):
            return DSTOFFSET
        else:
            return STDOFFSET

    def dst(self, dt):
        if self._isdst(dt):
            return DSTDIFF
        else:
            return ZERO

    def tzname(self, dt):
        return _time.tzname[self._isdst(dt)]

    def _isdst(self, dt):
        tt = (dt.year, dt.month, dt.day,
              dt.hour, dt.minute, dt.second,
              dt.weekday(), 0, -1)
```

```
        stamp = _time.mktime(tt)
        tt = _time.localtime(stamp)
        return tt.tm_isdst > 0

Local = LocalTimezone()


# Una comleta implementazioni delle regole correnti di DST (ora legale)
# per i principali fusi orari degli Stati Uniti.

def first_sunday_on_or_after(dt):
    days_to_go = 6 - dt.weekday()
    if days_to_go:
        dt += timedelta(days_to_go)
    return dt

# Negli Stati Uniti, il DST comincia alle 2 AM (tempo standard) della
# prima domenica di Aprile.
DSTSTART = datetime(1, 4, 1, 2)
# e termina alle 2 AM (tempo DST: 1 AM tempo standard) dell'ultima
# domenica di ottobre, cioè la prima domenica a partire dal 25 Ottobre,
# incluso questo giorno.
DSTEND = datetime(1, 10, 25, 1)

class USTimeZone(tzinfo):

    def __init__(self, hours, reprname, stdname, dstname):
        self.stdoffset = timedelta(hours=hours)
        self.reprname = reprname
        self.stdname = stdname
        self.dstname = dstname

    def __repr__(self):
        return self.reprname

    def tzname(self, dt):
        if self.dst(dt):
            return self.dstname
        else:
            return self.stdname

    def utcoffset(self, dt):
        return self.stdoffset + self.dst(dt)

    def dst(self, dt):
        if dt is None or dt.tzinfo is None:
            # Un'eccezione può essere una buona idea qui, in uno od
    # entrambi i casi. Dipende da come li si vuole trattare.
    # L'implementazione di default di fromutc() ( chiamata dalla
    # implementazione di default di astimezone() ) passa un oggetto
    # datetime "dt" in cui dt.tzinfo è uguale a self.
            return ZERO
        assert dt.tzinfo is self

        # Trova la prima domenica di Aprile e l'ultima di Ottobre.
        start = first_sunday_on_or_after(DSTSTART.replace(year=dt.year))
        end = first_sunday_on_or_after(DSTEND.replace(year=dt.year))

        # Non è possibile confrontare oggetti consapevoli con altri
# semplicistici, per cui è meglio prima rimuovere da dt
# l'informazione sul fuso orario.
        if start <= dt.replace(tzinfo=None) < end:
            return HOUR
```

```
            else:
                return ZERO

    Eastern  = USTimeZone(-5, "Eastern",  "EST", "EDT")
    Central  = USTimeZone(-6, "Central",  "CST", "CDT")
    Mountain = USTimeZone(-7, "Mountain", "MST", "MDT")
    Pacific  = USTimeZone(-8, "Pacific",  "PST", "PDT")
```

Notate le inevitabili sottigliezze necessarie per implementare una classe derivata di `tzinfo` che tenga conto sia del tempo standard che dell'ora legale, in corrispondenza dei punti di transizione dell'ora legale, due volte all'anno. Per concretezza, considerate il fuso orario Est degli Stati Uniti, dove l' EDT (NdT: East Daylight Time, ora legale per il fuso Est) comincia il minuto successivo alle 1:59 (EST) della prima domenica di Aprile e termina il minuto successivo alle 1:59 (EDT) dell'ultima domenica di Ottobre:

```
      UTC   3:MM  4:MM  5:MM  6:MM  7:MM  8:MM
      EST  22:MM 23:MM  0:MM  1:MM  2:MM  3:MM
      EDT  23:MM  0:MM  1:MM  2:MM  3:MM  4:MM

    start  22:MM 23:MM  0:MM  1:MM  3:MM  4:MM

      end  23:MM  0:MM  1:MM  1:MM  2:MM  3:MM
```

Quando l'ora legale inizia (la riga start), le lancette degli orologi locali saltano dalle 1:59 alle 3:00. Un tempo di orologio del tipo 2:MM non ha proprio senso in quel giorno, per cui `astimezone(Eastern)` non restituirà alcun risultato quando l'attributo `hour` vale `2` nel giorno di inizio del DST. Perché sia possibile che il metodo `astimezone()` rispetti questa garanzia, il metodo `rzinfo.dst()` deve considerare l'intervallo di tempo compreso nell'ora mancante (2:MM per la zona Est) come espresso in ora legale.

Quando l'ora legale termina (la riga end), vi è un problema potenzialmente peggiore: vi è infatti un'ora che non può venire indicata in modo non ambiguo con un tempo di orologio locale, vale a dire l'ultima ora del periodo di ora legale. Nella zona Est, questi sono i tempi del tipo 5:MM UTC del giorno in cui l'ora legale finisce. Le lancette degli orologi locali saltano all'indietro dalle 1:59 (ora legale) fino a tornare alle 1:00 (ora solare). Tempi locali espressi nella forma 1:MM sono ambigui. Il metodo `astimezone()` imita il comportamento del tempo di un orologio locale, facendo quindi corrispondere due ore UTC adiacenti alla stessa ora locale. Nell'esempio della zona Est, i tempi UTC del tipo 5:MM e 6:MM corrispondono entrambi a tempi del tipo 1:MM, una volta convertiti in ora locale. Allo scopo di consentire a `astimezone()` di garantire questo comportamento, il metodo `tzinfo.dst()` deve considerare i tempi nella ora ripetuta come espressi in ora solare standard. Questo viene facilmente ottenuto, come nell'esempio, rappresentando i tempi DST corrispondenti al passaggio all'ora legale e viceversa nel tempo solare standard del fuso orario interessato.

Applicazioni che non sono in grado di gestire questo tipo di ambiguità dovrebbero evitare di usare classi derivate ibride di `tzinfo`; non vi sono ambiguità quando si usa l'UTC, o ogni altra classe derivata di `tzinfo` con una differenza oraria fissa rispetto all'UTC (come ad esempio una classe rappresentante solo l'EST (differenza costante -5 ore) o solo l'EDT (differenza costante -4 ore)).

6.9.7 Il comportamento di `strftime()`

Gli oggetti `date`, `datetime` e `time` implementano tutti un metodo `strftime(`*format*`)`, per creare una stringa rappresentante il tempo, secondo una specifica stringa di formato. In termini generali, l'istruzione `d.strftime(fmt)` agisce come `time.strftime(fmt, d.timetuple())`, sebbene non tutti gli oggetti implementino il metodo `timetuple()`.

Per oggetti di tipo `time`, i codici di formato per anno mese e giorno non dovrebbero venire usati, dato che oggetti `time` non hanno tali attributi. Se vengono usati ugualmente, per l'anno viene usato il valore `1900`, mentre per il mese ed il giorno viene usato il valore `0`.

Per oggetti di tipo `date`, i codici di formato per ore, minuti e secondi non dovrebbero venire usati, dato che oggetti `date` non hanno tali attributi. Se vengono usati ugualmente, viene usato il valore `0` per tutti loro.

Per un oggetto "semplice", i codici di formato `%z` e `%Z` vengono convertiti in stringhe vuote.

Per un oggetto “complesso”:

`%z` `utcoffset()` viene trasformato in una stringa di cinque caratteri nella forma +HHMM oppure -HHMM, dove HH è una stringa di due caratteri che esprime le ore di differenza oraria e MM è una stringa di due caratteri che esprime i minuti di differenza oraria. Per esempio, se `utcoffset()` restituisce `timedelta(hours=-3, minutes=-30)`, allora `%z` viene sostituito con la stringa `-0330`.

`%Z` Se `tzname()` restituisce `None`, `%Z` viene sostituito con una stringa vuota. Altrimenti, `%Z` viene sostituito dal valore di ritorno di `tzname()`, che deve essere una stringa.

L’insieme completo dei codici di formato varia a seconda della piattaforma, poiché Python chiama la funzione `strftime()` messa a disposizione dalla libreria C della piattaforma, e variazioni di piattaforma sono comuni. La documentazione del modulo Python `time` elenca i codici di formato richiesti dallo standard del linguaggio C (versione 1989), e questi funzionano su tutte le piattaforme con implementazione C standard. Notate che la versione del 1999 dello standard C ha aggiunto altri codici di formato.

Anche l’intervallo di anni per cui `strftime()` funziona correttamente varia a seconda della piattaforma. Ad ogni modo, gli anni precedenti il 1900 non possono venire usati su nessuna piattaforma.

6.10 `time` — Accesso al tempo e conversioni

Questo modulo fornisce varie funzioni relative al tempo. È sempre disponibile, ma non tutte le funzioni sono disponibili su tutte le piattaforme. La maggior parte delle funzioni definite in questo modulo chiamano le funzioni della libreria C della piattaforma con lo stesso nome. Qualche volta può essere utile consultare la documentazione della piattaforma specifica, poiché la semantica di queste funzioni varia tra le piattaforme.

Una spiegazione di alcune terminologie e convenzioni viene riportata di seguito:

- *epoch* è l’istante da cui inizia in conteggio del tempo. Il 1ř gennaio di quell’anno, all’ora 0, il “tempo trascorso da epoch” è zero. Per i sistemi UNIX, epoch è il 1970. Per scoprire qual’è l’epoch, vedete `gmtime(0)`.
- Le funzioni in questo modulo non gestiscono i tempi e le date prima di epoch o lontane nel futuro. Il punto di interruzione nel futuro viene determinato dalla libreria C; per UNIX, tale valore viene solitamente indicato nel 2038.
- **Problema dell’anno 2000 (Y2K)**: Python dipende dalla libreria C della piattaforma sottostante, che generalmente non viene condizionata dal problema dell’anno 2000, in quanto tutte le date ed i tempi vengono rappresentati internamente come il numero di secondi trascorsi da epoch. Le funzioni che accettano uno `struct_time` (vedete più avanti) generalmente richiedono un anno di 4 cifre. Per compatibilità all’indietro, vengono supportati anni con valori di 2 cifre se la variabile di modulo `accept2dyear` è un intero diverso da zero; questa variabile viene inizializzata a `1`, a meno che la variabile d’ambiente PYTHONY2K non sia stata impostata ad una stringa non vuota, nel qual caso viene inizializzata a `0`. Perciò potete impostare PYTHONY2K ad una stringa non vuota nell’ambiente, per fare in modo che venga richiesto per tutti gli input un anno di 4 cifre. Quando vengono accettati anni con 2 cifre, questi vengono convertiti secondo gli standard POSIX o X/Open: i valori 69-99 vengono mappati a 1969-1999, e i valori 0-68 vengono mappati a 2000-2068. I valori 100-1899 sono sempre non accettabili. Notate che questa è una novità introdotta in Python 1.5.2(a2); le versioni precedenti, fino a Python 1.5.1 e 1.5.2a1, avrebbero aggiunto 1900 ai valori di anno inferiori a 1900.
- UTC è l’acronimo di Coordinated Universal Time (NdT: Tempo Universale Coordinato), precedentemente conosciuto come Greenwich Mean Time, (NdT: Tempo Medio di Greenwich) o GMT. L’acronimo UTC non è un errore grammaticale, ma deriva da un compromesso tra lingua Inglese e lingua Francese.
- DST è l’acronimo di Daylight Saving Time, (NdT: Ora Legale), una correzione del fuso orario di (generalmente) un’ora, per una parte dell’anno. Le regole per il DST sono magiche (determinate dalle leggi locali) e possono cambiare di anno in anno. La libreria C ha una tabella contenente le regole locali (generalmente letta da un file di sistema, per flessibilità) ed è la sola fonte della “Vera Saggezza” a riguardo.

- La precisione delle varie funzioni che restituiscono il tempo reale può essere inferiore rispetto a quanto suggerito dalle unità in cui il loro valore o gli argomenti vengono espressi. Per esempio, sulla maggior parte dei sistemi UNIX, l'orologio fa "tic-tac" soltanto 50 o 100 volte al secondo, e su Mac, i tempi sono accurati soltanto all'intero secondo.
- D'altra parte, la precisione delle funzioni `time()` e `sleep()` è migliore rispetto alle equivalenti UNIX: i tempi vengono rappresentati mediante un numero in virgola mobile, `time()` restituisce il tempo disponibile più accurato (utilizzando la funzione UNIX `gettimeofday()` dove presente), e `sleep()` accetterà un tempo con una frazione non nulla (la funzione UNIX `select()`, dove disponibile, viene utilizzata per implementare questo comportamento).
- Il valore di tempo restituito da `gmtime()`, `localtime()` e `strptime()`, e accettato da `asctime()`, `mktime()` e `strftime()`, è una sequenza di 9 interi. I valori restituiti da `gmtime()`, `localtime()`, e `strptime()`, forniscono anche i nomi degli attributi per i singoli campi.

Indice	Attributo	Valore
0	`tm_year`	(per esempio, 1993)
1	`tm_mon`	intervallo [1,12]
2	`tm_mday`	intervallo [1,31]
3	`tm_hour`	intervallo [0,23]
4	`tm_min`	intervallo [0,59]
5	`tm_sec`	intervallo [0,61]; vedete **(1)** nella descrizione di `strftime()`
6	`tm_wday`	intervallo [0,6], Lunedì è 0
7	`tm_yday`	intervallo [1,366]
8	`tm_isdst`	0, 1 or -1; vedete sotto

Notate che diversamente dalla struttura C, il valore del mese viene compreso nell'intervallo 1-12, non 0-11. Un valore per l'anno verrà gestito come descritto sopra, in "Problema dell'anno 2000 (Y2K)". Un argomento `-1` come opzione per l'ora legale, passato a `mktime()` generalmente genererà il corretto stato per l'ora legale da inserire.

Quando una tupla di lunghezza non corretta viene passata ad una funzione che si aspetta uno `struct_time`, o quando sono presenti alcuni elementi del tipo errato, viene sollevata un'eccezione `TypeError`.

Modificato nella versione 2.2: La sequenza dei valori del tempo è stata cambiata da una tupla in una classe `struct_time`, con l'aggiunta dei nomi degli attributi per i campi..

Il modulo definisce i seguenti elementi di funzioni e dati:

`accept2dyear`
Un valore booleano che indica se gli anni a 2 cifre verranno accettati. Il valore predefinito è Vero, ma verrà impostato a Falso se la variabile d'ambiente PYTHONY2K è stata impostata ad una stringa non vuota. Può anche venire modificato a runtime.

`altzone`
La distanza del fuso orario DST locale, in secondi a ovest di UTC, se definita. Assume valore negativo se il fuso orario DST locale è ad est di UTC (come nell'Europa Ovest, incluso il Regno Unito). Utilizzatelo solamente se `daylight` è non nullo.

`asctime`([*t*])
Converte una tupla o una classe `struct_time` rappresentante un tempo, come restituita da `gmtime()` o da `localtime()`, in una stringa di 24 caratteri, nella forma seguente: `'Sun Jun 20 23:21:05 1993'`. Se *t* non viene fornito, viene utilizzato il tempo corrente restituito da `localtime()`. Le informazioni sulla localizzazione non vengono utilizzate da `asctime()`. **Note:** Diversamente dalla funzione C con lo stesso nome, non c'è qui un carattere di fine riga. Modificato nella versione 2.1: È consentito omettere *t*.

`clock`()
Su UNIX, restituisce il tempo di processore corrente come un numero in virgola mobile espresso in secondi. La precisione, ed in effetti la vera definizione del significato di "tempo di processore", dipende dalla funzione C con lo stesso nome, ma in ogni caso, questa è la funzione da utilizzare per effettuare le verifiche di performance su Python o sugli algoritmi di tempo.

Su Windows, questa funzione restituisce il tempo, in secondi, trascorso dalla prima chiamata a questa funzione, come un numero in virgola mobile, basato sulla funzione Win32 `QueryPerformanceCounter()`. La risoluzione tipicamente è migliore di un microsecondo.

`ctime`([*secs*])
Converte un tempo espresso in secondi da epoch, in una stringa rappresentante il tempo locale. Se *secs* non viene fornito, viene utilizzato il valore corrente del tempo restituito dalla funzione `time()`. `ctime(`*secs*`)` è equivalente a `asctime(localtime(`*secs*`))`. Le informazioni sulla localizzazione non vengono utilizzate da `ctime()`. Modificato nella versione 2.1: È consentito omettere *secs*.

`daylight`
Un valore diverso da zero se viene definita una zona DST.

`gmtime`([*secs*])
Converte un tempo espresso in secondi da epoch in una classe `struct_time` in UTC, in cui l'opzione DST è sempre zero. Se *secs* non viene fornito, viene utilizzato il tempo corrente restituito da `time()`. Le frazioni di secondo vengono ignorate. Vedete sopra per una descrizione dell'oggetto `struct_time`. Modificato nella versione 2.1: È consentito omettere *secs*.

`localtime`([*secs*])
Come `gmtime()`, ma converte nel tempo locale. L'opzione dst viene impostata a `1` quando DST viene si applica al tempo corrente. Modificato nella versione 2.1: È consentito omettere *secs*.

`mktime`(*t*)
È la funzione inversa di `localtime()`. Come argomento accetta una classe `struct_time` o una tupla di 9 elementi pieni (poiché l'opzione dst è necessaria; utilizzate `-1` come valore per l'opzione dst se questa non è nota) che esprimone il tempo nel formato *locale*, non UTC. Restituisce un numero in virgola mobile, per compatibilità con `time()`. Se l'input non può venire rappresentato come un tempo valido, possono venire sollevate le eccezioni `OverflowError` o `ValueError` (quale delle due dipende da chi intercetta prima il valore errato, se Python o la libreria C sottostante). La data generabile più indietro nel temp dipende dalla piattaforma.

`sleep`(*secs*)
Sospende l'esecuzione per il dato numero di secondi. L'argomento può essere un numero in virgola mobile, per indicare un tempo di sleep più preciso. Il reale tempo di sospensione può essere inferiore di quello richiesto poiché ogni segnale intercettato termina la funzione `sleep()`, continuando con l'esecuzione della routine di intercettazione del segnale. Inoltre, il tempo di sospensione potrebbe essere maggiore di quello richiesto di una quantità arbitraria, a seguito della schedulazione di altre attività nel sistema.

`strftime`(*format*[, *t*])
Converte una tupla o una classe `struct_time`, rappresentante un tempo come restituito da `gmtime()` o `localtime()`, in una stringa come specificato dall'argomento *format*. Se *t* non viene fornito, viene utilizzato il valore corrente di tempo come restituito da `localtime()`. *format* deve essere una stringa. Viene sollevata l'eccezione `ValueError` se uno dei campi in *t* risulti esterno all'intervallo consentito. Modificato nella versione 2.1: È consentito omettere *t*. Modificato nella versione 2.4: L'eccezione `ValueError` viene sollevata se un campo in *t* è esterno all'intervallo..

Le seguenti direttive possono venire inserite nella stringa *format*. Vengono mostrate senza specificare il campo facoltativo di ampiezza e precisione, e vengono sostituite dai caratteri indicati nel risultato di `strftime()`.

Direttiva	Significato	Note
%a	Nome abbreviato del giorno della settimana, secondo il locale corrente.	
%A	Nome completo del giorno delle settimana, secondo il locale corrente.	
%b	Nome abbreviato del mese, secondo il locale corrente.	
%B	Nome completo del mese, secondo il locale corrente.	
%c	Rappresentazione appropriata della data e dell'ora correnti, secondo il locale corrente.	
%d	Giorno del mese, come numero decimale [01,31].	
%H	Ora (orologio a 24 ore) come numero decimale [00,23].	
%I	Ora (orologio a 12 ore) come numero decimale [01,12].	
%j	Giorno del'anno come numero decimale [001,366].	
%m	Mese come numero decimale [01,12].	
%M	Minuto come numero decimale [00,59].	
%p	Equivalente di AM o PM, secondo il locale corrente.	(1)
%S	Secondo come numero decimale [00,61].	(2)
%U	Numero della settimana dell'anno (Domenica come primo giorno della settimana) come numero decimale [00,53]. Tutti i giorni di un nuovo anno precedendi la prima Domenica vengono considerati come appartenenti alla settimana 0.	
%w	Giorno della settimana come numero decimale [0(Domenica),6].	
%W	Numero della settimana dell'anno (Lunedì come primo giorno della settimana) come numero decimale [00,53]. Tutti i giorni di un nuovo anno precedendi il primo Lunedì vengono considerati come appartenenti alla settimana 0.	
%x	Rappresentazione della data appropriata secondo il locale corrente.	
%X	Rappresentazione dell'ora appropriata secondo il locale corrente.	
%y	Anno senza il secolo come numero decimale [00,99].	
%Y	Anno con il secolo come numero decimale.	
%Z	Nome del fuso oraio (nessun carattere se non esiste un fuso orario).	
%%	Un carattere costante '%'.	

Note:

(1) Quando usata con la funzione `strptime()`, la direttiva `%p` influisce solo l'output del campo dell'ora, se la direttiva `%I` viene usata per analizzare l'ora.

(2)L'intervallo reale è da `0` a `61`; questo spiega gli sbalzi di secondi e i (molto rari) doppi sbalzi di secondi.

Ecco un esempio, un formato per le date compatibile con quello specificato nella RFC 2822 Internet Email Standard. [1]

```
>>> from time import gmtime, strftime
>>> strftime("%a, %d %b %Y %H:%M:%S +0000", gmtime())
'Thu, 28 Jun 2001 14:17:15 +0000'
```

Ulteriori direttive possono venire supportate su certe piattaforme, ma solo quelle indicate sopra hanno un significato standardizzato dall'ANSI C.

Su alcune piattaforme, uno campo facoltativo, di ampiezza e precisione specificate, può immediatamente seguire il carattere '%' iniziale di una direttiva, nell'ordine che segue; anche questo non è portabile. L'ampiezza per il campo è normalmente 2, ad eccezione di `%j` in cui è 3.

[1] L'utilizzo di `%Z` viene adesso deprecato, ma l'escape `%z` che espande la distanza oraria ore/minuti preferita non viene supportato da tutte le librerie ANSI C. Inoltre, una lettura rigorosa dell'originale standard RFC 822 del 1992 richiede un anno di 2 cifre (%y invece che %Y), ma per pratica l'anno di 4 cifre è stato adottato prima dell'anno 2000. L'anno a 4 cifre è stato ufficializzato dalla RFC 2822, che ha reso obsoleta la RFC 822.

strptime(*string*[, *format*])

Analizza una stringa rappresentante un tempo, in accordo con la formattazione fornita. Il valore restituito è una `struct_time`, come restituita da `gmtime()` o da `localtime()`. Il parametro *format* utilizza le stesse direttive usate da `strftime()`; viene impostato in modo predefinito a `%a %b %d %H:%M:%S %Y`, che corrisponde al formato restituito da `ctime()`. Se *string* non può essere analizzata in accordo con *format*, viene sollevata l'eccezione `ValueError`. Se la stringa da analizzare ha dati in eccesso dopo l'analisi, viene sollevata l'eccezione `ValueError`. I valori predefiniti usati per riempire i dati mancanti sono: `(1900, 1, 1, 0, 0, 0, 0, 1, -1)`.

Il supporto per la direttiva `%Z` si basa sui valori contenuti in `tzname`, e sul fatto che il valore di `daylight` risulti vero o meno. Per questo motivo, risulta dipendente dalla piattaforma, ad eccezione del riconoscimento di UTC e GMT che sono sempre noti (e vengono considerati fusi orari senza ora legale).

struct_time

Il tipo della sequenza dei valori del tempo restituita da `gmtime()`, `localtime()` e `strptime()`. Nuovo nella versione 2.2.

time()

Restituisce il tempo come un numero in virgola mobile espresso in secondi da epoch, in UTC. Notate che anche se il tempo viene sempre restituito come numero in virgola mobile, non tutti i sistemi forniscono il tempo con una precisione superiore ad un secondo. Dato che questa funzione normalmente restituisce valori non decrescenti, può restituire un valore inferiore rispetto a quello di una precedente chiamata, se l'orologio di sistema è stato reimpostato tra le due chiamate.

timezone

La distanza del fuso orario locale (non DST), in secondi a ovest di UTC (valore negativo nella maggior parte dell'Europa, positivo in USA, zero nel Regno Unito).

tzname

Una tupla di due stringhe: la prima è il nome del fuso orario locale non DST, la seconda è il nome del fuso orario DST. Se nessun fuso DST viene definito, la seconda stringa non dovrebbe venire usata.

tzset()

Reimposta le regole di conversione del tempo dalla libreria di sistema. La variabile d'ambiente TZ specifica come questo viene fatto. Nuovo nella versione 2.3.

Disponibilità: UNIX.

Note: Anche se in molti casi, modificare la variabile d'ambiente TZ può influire sull'output di funzioni come `localtime` senza chiamare `tzset`, non si dovrebbe fare affidamento su questo comportamento.

La variabile d'ambiente TZ non dovrebbe contenere spazi vuoti.

Il formato standard per la variabile d'ambiente TZ è (gli spazi vengono aggiunti per chiarezza):

std offset [dst [offset[,start[/time], end[/time]]]]

Dove:

std and dstTre o più alfanumerici indicanti l'abbreviazione del fuso orario. Queste verranno espanse in time.tzname

offsetL'offset ha la forma ± hh[:mm[:ss]]. Indica il valore da aggiungere al tempo locale per ottenere UTC. Se preceduto da un '-', il fuso orario è ad est del meridiano principale; altrimenti è a ovest. Se nessun offset segue il DST, l'ora estiva viene assunta come un'ora avanti rispetto all'ora standard.

start[/time,end[/time]] Indica quando cambiare da e verso DST. Il formato delle date start e end passati come argomento, è uno dei seguenti:

J*n*Il giorno *n* secondo il calendario Giuliano (1 <= *n* <= 365). I giorni bisestili non vengono conteggiati, perciò in tutti gli anni il 28 febbraio è il giorno 59 ed il primo marzo è il giorno 60.

*n*Il giorno secondo il calendario Giuliano, partendo da 0 (0 <= *n* <= 365). I giorni bisestili vengono conteggiati ed è possibile fare riferimento al 29 febbraio.

M*m.n.d*Il giorno *d*-esimo (0 <= *d* <= 6) o la settimana *n* del mese *m* dell'anno (1 <= *n* <= 5, 1 <= *m* <= 12, dove la settimana 5 indica l'ultimo giorno *d* del mese *m* che può presentarsi sia nella quarta

che nella quinta settimana). La settimana 1 è la prima settimana in cui cade il giorno *d*. Il giorno zero è Domenica.

time ha lo stesso formato di offset ad eccezione del fatto che il segno ('-' o '+') non è ammesso. Il valore predefinito, se non viene passata nessuna ora, è 02:00:00.

```
>>> os.environ['TZ'] = 'EST+05EDT,M4.1.0,M10.5.0'
>>> time.tzset()
>>> time.strftime('%X %x %Z')
'02:07:36 05/08/03 EDT'
>>> os.environ['TZ'] = 'AEST-10AEDT-11,M10.5.0,M3.5.0'
>>> time.tzset()
>>> time.strftime('%X %x %Z')
'16:08:12 05/08/03 AEST'
```

Su molti sistemi Unix (inclusi *BSD, Linux, Solaris e Darwin), è più conveniente usare il database di sistema zoneinfo (*tzfile*(5)) per specificare tutte le regole di fuso orario. Per fare questo, impostate la variabile d'ambiente TZ al percorso del file riferito al fuso orario richiesto, relativamente alla radice del database di sistema di 'zoneinfo', generalmente posto in '/usr/share/zoneinfo'. Per esempio, `'US/Eastern'`, `'Australia/Melbourne'`, `'Egypt'` o `'Europe/Amsterdam'`.

```
>>> os.environ['TZ'] = 'US/Eastern'
>>> time.tzset()
>>> time.tzname
('EST', 'EDT')
>>> os.environ['TZ'] = 'Egypt'
>>> time.tzset()
>>> time.tzname
('EET', 'EEST')
```

Vedete anche:

Modulo `datetime` (sezione 6.9):
Un'interfaccia più orientata agli oggetti per data e tempo.

Modulo `locale` (sezione 6.26):
Servizi di internazionalizzazione. Le impostazioni della localizzazione possono influenzare il valore restituito di alcune delle funzioni del modulo `time`.

Modulo `calendar` (sezione 5.19):
Generiche funzioni collegate al calendario. `timegm()` è l'inverso di `gmtime()` di questo modulo.

6.11 `sched` — Schedulatore degli eventi

Il modulo `sched` definisce una classe che implementa uno schedulatore di eventi per uso generale:

class `scheduler`(*timefunc*, *delayfunc*)
La classe `scheduler` definisce un'interfaccia generica per la schedulazione degli eventi. Sono necessarie due funzioni per accedere al "mondo esterno" — *timefunc* dovrebbe essere chiamabile senza argomenti, e restituire un numero (il "tempo", espresso in una qualsiasi unità di misura). La funzione *delayfunc* dovrebbe essere chiamabile con un argomento compatibile con il risultato della funzione *timefunc*, ed introdurre un ritardo pari alle unità specificate. *delayfunc* deve essere chiamata anche con argomento `0` dopo l'esecuzione di ciascun evento, per consentire agli altri thread di venire eseguiti in applicazioni multi threaded.

Esempio:

```
>>> import sched, time
>>> s=sched.scheduler(time.time, time.sleep)
>>> def print_time(): print "From print_time", time.time()
...
>>> def print_some_times():
...     print time.time()
...     s.enter(5, 1, print_time, ())
...     s.enter(10, 1, print_time, ())
...     s.run()
...     print time.time()
...
>>> print_some_times()
930343690.257
From print_time 930343695.274
From print_time 930343700.273
930343700.276
```

6.11.1 Oggetti dello scheduler

Le istanze di `scheduler` hanno i seguenti metodi:

`enterabs`(*time, priority, action, argument*)
: Aggiunge allo scheduler un nuovo evento. L'argomento *time* dovrebbe essere un tipo numerico compatibile con il valore restituito dalla funzione *timefunc*, passata al costruttore. Gli eventi schedulati per lo stesso valore di *time* verranno eseguiti secondo l'ordine di *priority* (NdT: priorità).

  Eseguire l'evento significa eseguire *action*`(*`*argument*`)`. *argument* deve essere una sequenza contenente i paramteri per l'azione *action*.

  Il valore restituito è un puntatore all'evento schedulato, che può venire usato in futuro per una sua cancellazione (vedete `cancel()`).

`enter`(*delay, priority, action, argument*)
: Schedula un evento per una quantità pari ad un valore di *delay* unità di tempo. Tranne il tempo relativo, gli altri argomenti, l'effetto ed il valore restituito sono uguali a quelli di `enterabs()`.

`cancel`(*event*)
: Rimuove *event* dalla coda degli eventi. Se l'evento non è presente nella coda, questo metodo solleverà un'eccezione `RuntimeError`.

`empty`()
: Restituisce vero se la coda degli eventi è vuota.

`run`()
: Esegue tutti gli eventi dello schedulatore. Questa funzione (attraverso l'uso della funzione `delayfunc` passata al costruttore) resterà in attesa dell'evento successivo, quindi lo eseguirà, e così via finché non esisteranno più eventi schedulati.

  Sia *action* che *delayfunc* possono sollevare un'eccezione. In entrambi i casi, lo schedulatore manterrà uno stato consistente e propagherà l'eccezione. Se un'eccezione viene sollevata da *action*, non si tenterà di eseguire l'evento alla successiva chiamata di `run()`.

  Se una sequenza di eventi richiede un tempo per l'esecuzione superiore a quello messo a disposizione prima dell'evento successivo, lo schedulatore rimarrà semplicemente attardato. Nessun evento verrà scartato; il codice chiamante è responsabile della cancellazione degli eventi non più pertinenti.

6.12 `mutex` — Supporto mutuamente esclusivo

Il modulo `mutex` definisce una classe che consente di implementare la mutua esclusione acquisendo e rilasciando i locks. Non richiede (o implica) threading o multi-tasking, anche se potrebbe essere utile per questi scopi.

Il modulo `mutex` definisce le seguenti classi:

class `mutex`()
Crea un nuovo mutex (non locked (NdT: non bloccato)).

Lo stato di un mutex è composto di due elementi — un bit "locked" ed una coda. Quando il mutex non è bloccato, la coda è vuota. Altrimenti, la coda contiene zero o più coppie (*funzione*, *argomento*) che rappresentano funzioni (o metodi) in attesa di acquisire il lock. Quando il mutex è non bloccato mentre la coda è non vuota, la prima voce nella coda viene rimossa e la corrispondente funzione (argomento) della coppia viene chiamata, implicando l'acquisizione del lock.

Naturalmente, non è implicato il multi-threading – da cui la simpatica interfaccia per il `lock()`, in cui una funzione viene chiamata dopo che il blocco è stato acquisito.

6.12.1 Oggetti Mutex

Gli oggetti `mutex` hanno i seguenti metodi:

`test`()
Controlla se il mutex è bloccato.

`testandset`()
Operazione "atomica" di tipo controlla-ed-imposta, prende il lock se non è impostato, e restituisce `True`, altrimenti `False`.

`lock`(*function, argument*)
Esegue *function*(*argument*), a meno che il mutex sia bloccato. Nel caso sia bloccato, inserisce la coppia funzione-argomento nella coda; vedete il metodo methodunlock per spiegazioni a riguardo dell'esecuzione di *funzione*(*argomento*) in questo caso.

`unlock`()
Sblocca il mutex se la coda è vuota, altrimenti esegue il primo elemento della coda.

6.13 `getpass` — Inserimento di password portabile

Il modulo `getpass` fornisce due funzioni:

`getpass`([*prompt*])
Fornisce all'utente il prompt per l'inserimento di una password, senza emissione di caratteri. L'utente ha a disposizione la stringa *prompt*, il cui valore predefinito è `'Password: '`. Disponibilità: Macintosh, UNIX, Windows.

`getuser`()
Restituisce il "nome di login" dell'utente. Disponibilità: UNIX, Windows.

Questa funzione controlla le variabili d'ambiente LOGNAME, USER, LNAME e USERNAME, nell'ordine elencato, e restituisce il valore della prima impostata ad una stringa non vuota. Se nessuna di queste variabili è impostata, viene restituito il nome di login dal database delle password, sui sistemi che supportano il modulo pwd, altrimenti viene sollevata un'eccezione.

6.14 `curses` — Gestione dei terminali per display a celle di caratteri

Modificato nella versione 1.6: Aggiunto il supporto per la libreria `ncurses` e convertito in un package.

Il modulo `curses` fornisce un'interfaccia alla libreria curses, lo standard di fatto per la gestione portabile ed avanzata dei terminali.

Anche se curses viene principalmente utilizzata in ambiente UNIX, sono disponibili anche delle versioni per DOS, OS/2 e altri sistemi. Questo modulo di estensione è progettato per corrispondere alle API ncurses, una libreria curses open source ospitata su Linux e sulle varianti BSD di UNIX.

Vedete anche:

Modulo `curses.ascii` (sezione 6.17):
Modulo di utilità per lavorare con caratteri ASCII, in modo indifferente alle impostazioni locali.

Modulo `curses.panel` (sezione 6.18):
Un'estensione panel stack che aggiunge profondità alle finestre curses.

Modulo `curses.textpad` (sezione 6.15):
Widget di testo editabile per curses che supporta i binding in stile **Emacs**.

Modulo `curses.wrapper` (sezione 6.16):
Comoda funzione per assicurare impostazione e reset appropriati all'entrata e all'uscita dalle applicazioni.

Curses Programming with Python
(http://www.python.org/doc/howto/curses/curses.html)
Tutorial sull'utilizzo di curses con Python, scritto da Andrew Kuchling ed Eric Raymond, è disponibile sul sito web di Python.

La directory 'Demo/curses/' nella distribuzione dei sorgenti Python contiene alcuni programmi di esempio sull'utilizzo dei binding curses forniti da questo modulo.

6.14.1 Funzioni

Il modulo `curses` definisce la seguente eccezione:
exception `error`
Eccezione sollevata quando una funzione di libreria curses restituisce un errore.
Note: Ogni volta che gli argomenti *x* o *y* di una funzione o metodo sono facoltativi, i loro valori predefiniti fanno riferimento alla posizione corrente del cursore. Ogni volta che *attr* è facoltativo, il suo valore predefinito è `A_NORMAL`.

Il modulo `curses` definisce le seguenti funzioni:

`baudrate``()`
Restituisce la velocità di output del terminale in bit al secondo. Sui software di emulazione di terminale avrà un alto valore fisso. Inclusa per ragioni storiche; nei primi tempi, veniva usata per scrivere cicli di output per ritardi di tempo, e occasionalmente per modificare le interfacce in fuzione della velocità di linea.

`beep``()`
Emette un breve suono di attenzione.

`can_change_color``()`
Restituisce vero o falso, a seconda del faccto che il programmatore possa o meno modificare i colori mostrati dal terminale.

`cbreak``()`
Entra in modalità cbreak. Nella modalità cbreak (a volte chiamata modalità "rare") il normale buffering di riga delle tty viene disattivato, ed i caratteri sono disponibili per essere letti uno per volta. Comunque, a differenza della modalità raw, i caratteri speciali (interruzione, uscita, sospensione, e controllo dil flusso) mantengono il loro effetto sul driver tty e sul programma chiamante. Chiamando prima `raw()` e quindi `cbreak()`, il terminale viene lasciato in modalità cbreak.

`color_content``(`*color_number*`)`
Restituisce l'intensità delle componenti di rosso, verde e blu (RGB) nel colore *color_number*, che deve essere compreso tra `0` e `COLORS`. Viene restituita una tupla di 3 elementi, contenente i valori di R, G, B per il colore dato, che sarà compreso tra `0` (nessun componente) e `1000` (massimo valore del singolo componente).

`color_pair``(`*color_number*`)`
Restituisce il valore dell'attributo utilizzato per mostrare il testo nel colore specificato. Questo valore d'attributo può venire combinato con `A_STANDOUT`, `A_REVERSE`, e con gli altri attributi `A_*`. `pair_number()` è la controparte di questa funzione.

`curs_set``(`*visibility*`)`
Imposta lo stato del cursore. *visibility* può venire impostato ai valori `0`, `1` o `2`, che indicano invisibile, normale, o molto visibile. Se il terminale supporta la visibilità richiesta, viene restituito lo stato precedente

del cursore; altrimenti, viene sollevata un'eccezione. Su molti terminali, la modalità "visibile" consiste in una linea di sottolineatura (NdT: underline), e la modalità "molto visibile" consiste in un cursore a blocco.

def_prog_mode()
Salva la modalità corrente del terminale come modalità "programma", la modalità impiegata quando il programma in esecuzione utilizza curses. (La sua controparte è la modalità "shell", impiegata quando il programma non utilizza curses.) Chiamate successive a `reset_prog_mode()` rispristineranno questa modalità.

def_shell_mode()
Salva la modalità corrente del terminale come modalità "shell", la modalità impiegata quando il programma non utilizza curses. (La sua controparte è la modalità "programma", impiegata quando il programma utilizza curses.) Chiamate successive a `reset_shell_mode()` rispristineranno questa modalità.

delay_output(*ms*)
Inserisce una pausa di *ms* millisecondi nell'output.

doupdate()
Aggiorna lo schermo fisico. La libreria curses mantiene due strutture dati, una rappresentante i contentuti dello schermo fisico corrente ed una lo schermo virtuale rappresentante il prossimo stato desiderato. La `doupdate()` aggiorna lo schermo fisico al corrispondente schermo virtuale.

Lo schermo virtuale può venire aggiornato da una chiamata a `noutrefresh()`, dopo che su una finestra sono state eseguite operazioni di scrittura come `addstr()`. La chiamata normale a `refresh()` è semplicemente `noutrefresh()` seguita da `doupdate()`; se dovete aggiornare finestre multiple, potete velocizzare le performance e forse ridurre l'instabilità dell'immagine dello schermo distribuendo chiamate a `noutrefresh()` su tutte le finestre, seguite da una singola `doupdate()`.

echo()
Entra nella modalità echo. Nella modalità echo, ogni carattere in input viene mostrato sullo schermo così come viene immesso.

endwin()
Deinizializza la libreria, e reimposta il terminale allo stato normale.

erasechar()
Restituisce il carattere di cancellazione corrente dell'utente. Nei sistemi operativi UNIX questa è una proprietà della tty che controlla il programma curses, e non viene impostato dalla libreria curses.

filter()
La procedura `filter()`, se usata, deve essere chiamata prima di `initscr()`. L'effetto è che, durante queste chiamate, LINES viene impostato a `1`; le proprietà lear, cup, cud, cud1, cuu1, cuu, vpa vengono disabilitate; e la stringa home viene impostata al valore di cr. L'effetto è quello di confinare il cursore alla riga corrente, ed effettuare gli aggiornamenti dello schermo. Tutto questo può venire usato per abilitare l'editing di riga di un carattere alla volta, senza toccare il resto dello schermo.

flash()
Fa lampeggiare lo schermo. Di fatto, lo modifica a reverse-video e lo riporta indietro in un breve intervallo. Alcune persone preferiscono questo genere di 'campanello visibile' al segnale di attenzione udibile prodotto da `beep()`.

flushinp()
Svuota tutti i buffer di input. Questo fa perdere ogni carattere digitato dall'utente che non è ancora stato elaborato dal programma.

getmouse()
Dopo che `getch()` ha restituito `KEY_MOUSE` per segnalare un evento del mouse, questo metodo dovrebbe venire chiamato per recuperare l'evento del mouse accodato, rappresentato come una tupla di 5 elementi (*id*, *x*, *y*, *z*, *bstate*). *id* è un valore ID usato per distinguere dispositivi multipli, e *x*, *y*, *z* sono le coordinate dell'evento (*z* al momento non viene usato). *bstate* è un valore intero i cui bit verranno impostati per indicare il tipo di evento, ed indicherà l'OR bit per bit di una o più di queste costanti, dove *n* è il numero del pulsante da 1 a 4: `BUTTON`*n*`_PRESSED`, `BUTTON`*n*`_RELEASED`, `BUTTON`*n*`_CLICKED`, `BUTTON`*n*`_DOUBLE_CLICKED`, `BUTTON`*n*`_TRIPLE_CLICKED`, `BUTTON_SHIFT`, `BUTTON_CTRL`, `BUTTON_ALT`.

getsyx()
Restitusce le coordinate correnti del cursore dello schermo virtuale in y e x. Se `leaveok` è vera al momento, allora viene restituito `-1,-1`.

getwin(*file*)
Legge i dati relativi alla finestra immagazzinati in file da una precedente chiamata a `putwin()`. La procedura quindi crea ed inizializza una nuova finestra usando quei dati, restituendo il nuovo oggetto window.

has_colors()
Restituisce vero se il terminale può mostrare i colori; altrimenti, restituisce falso.

has_ic()
Restituisce vero se il terminale ha le proprietà di inserire e cancellare i caratteri. Questa funzione viene inclusa solamente per ragioni storiche, visto che tutti i moderni software di emulazione di terminale possiedono queste proprietà.

has_il()
Restituisce vero se il terminale ha la proprietà di cancellare ed inserire righe, o può simularle usando le regioni scorrevoli. Questa funzione viene inclusa solamente per ragioni storiche, visto che tutti i moderni software di emulazione di terminale possiedono queste proprietà.

has_key(*ch*)
Prende un valore chiave *ch*, e restituisce vero se il tipo di terminale corrente riconosce una chiave con quel valore.

halfdelay(*tenths*)
Usato per la modalità half-delay, che è simile alla modalità cbreak per il fatto che i caratteri digitati dall'utente vengono immediatamente resi disponibili al programma. Comunque, dopo essere rimasto bloccato per *tenths* (NdT: decimi) di secondo, se non è stato digitato niente, viene sollevata un'eccezione. Il valore di *tenths* deve essere un numero nell'intervallo tra 1 e 255. Usate `nocbreak()` per abbandonare la modalità half-delay.

init_color(*color_number, r, g, b*)
Modifica la definizione di un colore, prendendo il numero del colore da modificare seguito da tre valori RGB (per le quantità dei componenti rosso, verde e blu). Il valore di *color_number* deve essere compreso tra `0` e `COLORS`. Ciascun valore di *r*, *g* e *b* deve essere compreso tra `0` e `1000`. Quando viene utilizzata la funzione `init_color()`, tutte le occorrenze di quel colore sullo schermo vengono immediatamente cambiate alla nuova definizione. Questa funzione non ha alcun effetto sulla maggior parte dei terminali; risulta attiva solo se `can_change_color()` restituisce `1`.

init_pair(*pair_number, fg, bg*)
Modifica la definizione di una coppia di colori. Prende tre argomenti: il numero della coppia di colori da modificare, il numero del colore in primo piano ed numero del colore di sfondo. Il valore di *pair_number* deve essere compreso tra `1` e `COLOR_PAIRS - 1` (la coppia dei colori `0` è contornata in bianco e nero e non può essere modificata). Il valore degli argomenti *fg* e *bg* deve essere compreso tra `0` e `COLORS`. Se la coppia dei colori è stata inizializzata precedentemente, lo schermo viene aggiornato e tutte le occorrenze di quella coppia di colori vengono modificate alla nuova definizione.

initscr()
Inizializza la libreria. Restituisce un `WindowObject` che rappresenta l'intero schermo. **Note:** Se si verifica un errore durante l'apertura del terminale, la libreria curses sottostante può causare l'uscita dall'interprete.

isendwin()
Restituisce vero se è stata chiamata `endwin()` (di fatto, la libreria curses è stata deinizializzata).

keyname(*k*)
Restituisce il nome del tasto numerato *k*. Il nome di un tasto che genera un carattere ASCII stampabile è il carattere del tasto. Il nome di una combinazione control-tasto è una stringa di due caratteri che consiste in un carattere di omissione seguito dal carattere ASCII stampabile corrispondente. Il nome di una combinazione alt-tasto (128-255) è una stringa che consiste nel prefisso 'M-' seguito dal nome del carattere ASCII corrispondente.

killchar()
Restituisce il carattere di kill di riga corrente dell'utente. Nei sistemi operativi UNIX questa è una proprietà della tty che controlla il programma curses, e non viene impostato della libreria curses.

longname()
Restituisce una stringa contenente il campo terminfo che descrive il terminale corrente. La lunghezza massima di una descrizione prolissa è di 128 caratteri. Viene definita solo dopo la chiamata a `initscr()`.

meta(*yes*)
Se *yes* viene impostato ad 1, permette l'input di caratteri a 8-bit. Se *yes* viene impostato a 0, ammette solamente caratteri a 7-bit.

mouseinterval(*interval*)
Imposta il tempo massimo in millisecondi che possono passare fra gli eventi di pressione e rilascio, in modo che vengano interpretati come click, e restituisce il valore dell'intervallo precedente. Il valore predefinito è 200 msec, o un quinto di secondo.

mousemask(*mousemask*)
Imposta la notifica degli eventi del mouse, e restituisce una tupla (*availmask*, *oldmask*). *availmask* indica quali tra gli eventi del mouse specificati possono venire riportati; in caso di completo fallimento restituisce 0. *oldmask* è il valore precedente della maschera degli eventi del mouse della finesta specificata. Se questa funzione non viene mai chiamata, nessun evento del mouse viene mai notificato.

napms(*ms*)
Rimane in attesa per *ms* millisecondi.

newpad(*nlines, ncols*)
Crea e restituisce un puntatore ad una nuova struttura dati pad con il dato numero di righe e colonne. Viene restituito un pad come oggetto finestra.

Un pad è simile ad una finestra, con l'eccezione che non viene ristretto dalla dimensione dello schermo e non viene necessariamente associato ad una particolare parte dello schermo. I pad possono venire usati quando c'è bisogno di una finestra grande, e solo una parte della finestra deve trovarsi sullo schermo in un dato momento. L'aggiornamento automatico dei pad (come dallo scorrimento o dall'echo dell'input) non viene messo in atto. I metodi `refresh()` e `noutrefresh()` di un pad richiedono 6 argomenti per specificare la parte del pad da mostrare, e la locazione sullo schermo da usare per il display. Gli argomenti sono pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol; gli argomenti p si riferiscono all'angolo in alto a sinistra della regione del pad che deve essere mostrata, e gli argomenti s definiscono il ritaglio sullo schermo nel quale deve essere mostrata la regione del pad.

newwin([*nlines, ncols,*] *begin_y, begin_x*)
Restituisce una nuova finestra, il cui angolo in alto a sinistra si trova a (*begin_y*, *begin_x*), e la cui altezza/larghezza è *nlines*/*ncols*.

In modo predefinito, la finestra si estenderà dalla posizione specificata fino all'angolo in basso a destra dello schermo.

nl()
Entra nella modalità newline. Questa modalità traduce il tasto invio in un fine riga durante l'input, e traduce il carattere di fine riga in caratteri di invio e line-feed durante l'output. La modalità newline è inizialmente attiva.

nocbreak()
Abbandona la modalità cbreak. Ritorna alla modalità normale "cooked" con il buffering di riga.

noecho()
Abbandona la modalità echo. L'echo dei caratteri in input viene disattivato.

nonl()
Abbandona la modalità newline. Disabilita la traduzione di invio in fine riga durante l'input, e disabilita la traduzione a basso livello del fine riga in line-feed/invio durante l'output (ma questa non cambia il comportamento di `addch('\n')`, che simula sempre l'equivalente dei caratteri di invio e line-feed sullo schermo virtuale). Con la traduzione disattivata, curses può saltuariamente accelerare un po' lo spostamento verticale; inoltre, sarà in grado di rilevare il tasto di invio in input.

noqiflush()

Quando viene utilizzata la procedura noqiflush, non sarà effettuato il normale svuotamento delle code di input e output associate ai caratteri NTR, QUIT e SUSP. Potreste voler chiamare `noqiflush()` in un gestore dei segnali dopo la sua uscita, se volete che l'output continui come se l'interruzione non fosse avvenuta.

`noraw``()`
Abbandona la modalità raw. Ritorna alla modalità normale "cooked" con il buffering di riga.

`pair_content``(`*pair_number*`)`
Restituisce una tupla `(`*fg*`, `*bg*`)` contenente i colori per la coppia di colore richesta. Il valore di *pair_number* deve trovarsi fra `0` e `COLOR_PAIRS - 1`.

`pair_number``(`*attr*`)`
Restituisce il numero della coppia di colore impostata dal valore attributo *attr*. `color_pair()` è la controparte di questa funzione.

`putp``(`*string*`)`
Equivalente a `tputs(str, 1, putchar)`; emette il valore di una specifica proprietà terminfo per il terminale corrente. Notate che l'ouput di putp viene sempre diretto allo standard output.

`qiflush``(` [*flag*] `)`
Se *flag* è `False`, l'effetto è lo stesso di chiamare `noqiflush()`. Se *flag* è `True`, o nessun argomento viene passato, le code verranno svuotate alla lettura di questi caratteri di controllo.

`raw``()`
Entra nella modalità raw. Nella modalità raw, il normale buffering di riga e l'elaborazione dei caratteri di interruzione, terminazione, sospensione, e gestione del flusso, vengono disabilitati; i caratteri vengono presentati alle funzioni di input di curses uno alla volta.

`reset_prog_mode``()`
Ripristina la modalità "program" del terminale, come salvata precedentemente dalla funzione `def_prog_mode()`.

`reset_shell_mode``()`
Ripristina la modalità "shell" del terminale, come salvata precedentemente dalla funzione `def_shell_mode()`.

`setsyx``(`*y, x*`)`
Imposta il cursone dello schermo virtule a *y*, *x*. Se *y* ed *x* valgono entrambi `-1`, viene impostata la `leaveok`.

`setupterm``(`[*termstr, fd*]`)`
Inizializza il terminale. *termstr* è una stringa che fornisce il nome del terminale; se omesso, verrà utilizzato il valore della variabile d'ambiente TERM. *fd* è il descrittore di file al quale verrà inviata ogni sequenza di inizializzazione; se non fornito, verrà usato il descritttore di file di `sys.stdout`.

`start_color``()`
Deve essere chiamata se il programmatore vuole usare i colori, e prima che venga chiamata ogni altra procedura di manipolazione dei colori. È buona norma chiamare questa procedura subito dopo `initscr()`.

`start_color()` inizializza otto colori di base (nero, rosso, verde, giallo, blu, magenta, ciano e bianco), e due variabili globali nel modulo `curses`, `COLORS` e `COLOR_PAIRS`, contenenti rispettivamente il massimo numero di colori e coppie colore che il terminale è in grado di supportare. Inoltre ripristina i colori sul terminale ai valori che possedevano al momento della sua accensione.

`termattrs``()`
Restituisce un OR logico di tutti gli attributi video supportati dal terminale. Questa informazione è utile quando un programma curses necessita di un controllo completo sull'aspetto dello schermo.

`termname``()`
Restituisce il valore della variabile d'ambiente TERM, troncata a 14 caratteri.

`tigetflag``(`*capname*`)`
Restituisce il valore della proprietà booleana corrispondente al nome della proprietà terminfo *capname*. Il valore `-1` viene restituito nel caso in cui *capname* non sia una proprietà booleana, o `0` se è cancellata o assente dalla descrizione del terminale.

tigetnum(*capname*)
Restituisce il valore della proprietà numerica corrispondente al nome della proprietà terminfo *capname*. Il valore `-2` viene restituito nel caso in cui *capname* non sia una proprietà numerica, o `-1` se è cancellata o assente dalla descrizione del terminale.

tigetstr(*capname*)
Restituisce il valore della proprietà stringa corrispondente al nome della proprietà terminfo *capname*. Viene restituito `None` nel caso in cui *capname* non sia una capability stringa, o se è cancellata o assente dalla descrizione del terminale.

tparm(*str*[,...])
Istanzia la stringa *str* con i parametri specificati, dove *str* dovrebbe essere una stringa parametrizzata, ottenuta dal database terminfo. Per esempio `tparm(tigetstr(cup), 5, 3)` potrebbe risultare come `'\033[6;4H'`, il risultato esatto dipende dal tipo del terminale.

typeahead(*fd*)
Specifica che il descrittore di file *fd* deve venire usato per il controllo typeahead. Se *fd* ha valore `-1`, allora non viene effettuato alcun controllo typeahead.

La libreria curses effettua "l'ottimizzazione line-breakout" cercando periodicamente il typeahead mentre viene aggiornato lo schermo. Se viene trovato dell'input, ed esso proviene da una tty, l'aggiornamento corrente viene posticipato fino a che non venga chiamata di nuovo una refresh o una doupdate, permettendo una risposta più rapida a comandi digitati in anticipo. Questa funzione permette di specificare un differente descrittore di file per il controllo typeahead.

unctrl(*ch*)
Restituisce una stringa che è la rappresentazione stampabile del carattere *ch*. I caratteri di controllo vengono mostrati come un carattere di omissione seguito dal carattere, per esempio come accde con `^C`. I caratteri stampabili vengono lasciati come tali.

ungetch(*ch*)
Accoda *ch* così da restituirlo alla prossima `getch()`. **Note:** Solo un *ch* può venire accodato prima che venga chiamata `getch()`.

ungetmouse(*id, x, y, z, bstate*)
Accoda un evento `KEY_MOUSE` sulla coda dell'input, associandogli il dato di stato fornito.

use_env(*flag*)
Se usata, questa funzione dovrebbe essere chiamata prima di `initscr()` o `newterm()`. Quando *flag* è falso, verranno utilizzati i valori di righe e colonne specificati nel database terminfo, anche se sono impostate le variabili di ambiente LINES e COLUMNS (usate in modo predefinito), o se curses viene eseguito in una finestra (nel qual caso il comportamento predefinito utilizzerebbe la dimensione della finestra se LINES e COLUMNS non fossere impostate).

use_default_colors()
Permette l'utilizzo dei colori predefiniti sui terminali che supportano questa possibilità. Usate questo metodo per supportare la trasperenza nella vostra applicazione. Il colore predefinito viene assegnato al colore numero -1. Dopo aver chiamato questa funzione, `init_pair(x, curses.COLOR_RED, -1)` inizializza, ad esempio, la coppia colore *x* ad un colore di primo piano rosso sullo sfondo predefinito.

6.14.2 Oggetti finestra

Gli oggetti finestra, come restituiti da `initscr()` e `newwin()` visti sopra, possiedono i seguenti metodi:

addch([*y, x,*] *ch*[, *attr*])
Note: Con *carattere* si intende un carattere C (un codice ASCII), invece di un carattere Python (una stringa di lunghezza 1). (Questa nota è valida ogni volta che la documentazione menziona un carattere.) La funzione built-in `ord()` è utile per convertire stringhe in codici.

Scrive il carattere *ch* alla posizione `(`*y*`, `*x*`)` con attributi *attr*, sovrascrivendo ogni carattere precedentemente scritto in quella locazione. In modo predefinito, la posizione e gli attributi del carattere sono le impostazioni correnti per l'oggetto finestra.

addnstr([*y, x,*] *str, n*[, *attr*])

Scrive al più *n* caratteri della stringa *str* alla posizione `(y, x)` con attributi *attr*, sovrascrivendo qualsiasi cosa si trovi in precedenza sul display.

`addstr``(`[*y, x,*] *str*[*, attr*]`)`
Scrive la stringa *str* alla posizione `(y, x)` con attributi *attr*, sovrascrivendo qualsiasi cosa si trovi in precedenza sul display.

`attroff``(`*attr*`)`
Rimuove l'attributo *attr* dall'insieme di "background" applicato a tutte le scritture sulla finestra corrente.

`attron``(`*attr*`)`
Aggiunge l'attributo *attr* all'insieme di "background" applicato a tutte le scritture sulla finestra corrente.

`attrset``(`*attr*`)`
Imposta ad *attr* l'insieme degli attributi di "background". Questo insieme ha inizialmente il valore impostato a `0` (nessun attributo).

`bkgd``(`*ch*[*, attr*]`)`
Imposta la proprieta di sfondo della finestra al carattere *ch*, con attributi *attr*. La modifica viene quindi applicata ad ogni locazione di carattere in quella finestra:

- L'attributo di ogni carattere nella finestra viene modificato al nuovo attributo dello sfondo.
- Ovunque appaia il precendente carattere di sfondo, questo viene modificato nel nuovo carattere dello sfondo.

`bkgdset``(`*ch*[*, attr*]`)`
Imposta lo sfondo della finestra. Lo sfondo di una finestra consiste di un carattere e di ogni combinazione di attributi. La parte degli attributi dello sfondo viene combinata (tramite OR) con tutti i caratteri non vuoti che vengono scritti sulla finestra. Sia i caratteri che le parti di attributi dello sfondo vengono combinati con i caratteri vuoti. Lo sfondo diventa una proprietà del carattere e si sposta con il carattere in ogni operazione di inserimento/cancellazione di riga/carattere.

`border``(`[*ls*[*, rs*[*, ts*[*, bs*[*, tl*[*, tr*[*, bl*[*, br*]]]]]]]]`)`
Disegna un bordo attorno ai lati della finestra. Ogni parametro specifica il carattere da utilizzare per una specifica parte del bordo; vedete la tabella sotto per ulteriori dettagli. I caratteri possono venire specificati come interi o come stringhe di un carattere.

Note: In caso di valore `0` per ogni parametro, verrà utilizzato il carattere predefinito per quel parametro. *Non* possono venire utilizzati parametri a parola chiave. I predefiniti vengono elencati in questa tabella:

Parametro	Descrizione	Valore predefinito
ls	Lato sinistro	`ACS_VLINE`
rs	Lato destro	`ACS_VLINE`
ts	Vertice	`ACS_HLINE`
bs	Base	`ACS_HLINE`
tl	Angolo in alto a sinistra	`ACS_ULCORNER`
tr	Angolo in alto a destra	`ACS_URCORNER`
bl	Angolo in basso a sinistra	`ACS_BLCORNER`
br	Angolo in basso a destra	`ACS_BRCORNER`

`box``(`[*vertch, horch*]`)`
Simile a `border()`, ma sia *ls* che *rs* sono *vertch*, e sia *ts* che *bs* sono *horch*. I caratteri di angolo predefiniti vengono sempre usati da questa funzione.

`clear``()`
Come `erase()`, ma causa anche il ridisegno dell'intera finestra alla prossima chiamata a `refresh()`.

`clearok``(`*yes*`)`
Se *yes* vale `1`, la prossima chiamata a `refresh()` pulirà completamente la finestra.

`clrtobot``()`
Cancella dal cursore fino alla fine della finestra: tutte le righe sotto il cursore vengono cancellate, e quindi viene eseguito l'equivalente di `clrtoeol()`.

`clrtoeol``()`
Cancella dal cursore alla fine della riga.

cursyncup()
Aggiorna la posizione corrente del cursore di tutti gli antenati della finestra, in modo da riflettere la posizione corrente del cursore della finestra.

delch([*x*, *y*])
Cancella ogni carattere alla posizione (*y*, *x*).

deleteln()
Cancella la riga su cui si trova il cursore. Tutte le righe seguenti vengono spostate in alto di una riga.

derwin([*nlines*, *ncols*,] *begin_y*, *begin_x*)
Un'abbreviazione di "finestra derivata", `derwin()` ha lo stesso effetto della chiamata a `subwin()`, eccetto che *begin_y* e *begin_x* sono relativi all'origine della finestra, piuttosto che all'intero schermo. Restituisce un oggetto finestra per la finestra derivata.

echochar(*ch*[, *attr*])
Aggiunge un carattere *ch* con attributo *attr*, e chiama immediatamente `refresh()` sulla finestra.

enclose(*y*, *x*)
Verifica che la coppia di coordinate di cella di carattere passata relativa allo schermo sia contenuta nella finestra data, restituendo vero o falso. È utile per determinare quale sotto insieme delle finestre dello schermo racchiude la locazione di un evento del mouse.

erase()
Pulisce la finestra.

getbegyx()
Restituisce una tupla (*y*, *x*) di coordinate dell'angolo in alto a sinistra.

getch([*x*, *y*])
Prende un carattere. Notate che *non* è necessario che l'intero restituito si trovi nell'intervallo ASCII: tasti funzione, tasti del tastierino numerico e altri restituiscono numeri più alti di 256. Nella modalità no-delay, se non c'è input viene restituito `-1`.

getkey([*x*, *y*])
Prende un carattere, restituiendo una stringa invece di un intero, come fa `getch()`. Tasti funzione, tasti del tastierino numerico ed altri restituiscono una stringa a byte multipli, contenente il nome del tasto. Nella modalità no-delay, viene sollevata un'eccezione in caso di mancanza di input.

getmaxyx()
Restituisce una tupla (*y*, *x*) contenente l'altezza e la larghezza della finestra.

getparyx()
Restituisce le coordinate iniziali di questa finestra relative alla sua finestra madre in due variabili intere y e x. Restituisce `-1,-1` se questa finestra non ha madre.

getstr([*x*, *y*])
Legge una stringa dall'utente, con proprietà di editing di riga elementare.

getyx()
Restituisce una tupla (*y*, *x*) della posizione corrente del cursore, relativa all'angolo in alto a sinistra della finestra.

hline([*y*, *x*,] *ch*, *n*)
Mostra una linea orizzontale che inizia a (*y*, *x*) e di lunghezza *n*, composta da caratteri *ch*.

idcok(*flag*)
Se *flag* è falso, non viene più utilizzata da `curses` la funzionalità hardware di inserimento/cancellazione dei caratteri del terminale; se *flag* è vero, l'utilizzo dell'inserimento e cancellazione dei caratteri viene abilitato. Al momento dell'inizializzazione di `curses`, viene abilitato in modalità predefinita l'utilizzo dei caratteri inserisci/cancella.

idlok(*yes*)
Se viene chiamato con *yes* uguale a `1`, `curses` cercherà di sfruttare le semplificazioni hardware di elaborazione di riga. Altrimenti, la possibilità di inserimento/cancellazione delle righe viene disabilitata.

immedok(*flag*)

Se *flag* è vero, ogni modifica nell'immagine della finestra provocherà l'immediato aggiornamento della finestra stessa; non dovrete più chiamare `refresh()` da voi stessi. In ogni caso, questo potrebbe degradare le prestazioni in modo considerevole, a causa delle ripetute chiamate a `wrefresh`. Questa opzione viene disabilitata in modo predefinito.

inch([*x, y*])
Restituisce il carattere nella finestra alla posizione data. Gli 8 bit di fondo sono il carattere vero e proprio, gli 8 bit di testa sono gli attributi.

insch([*y, x,*] *ch*[*, attr*])
Disegna il carattere *ch* alla posizione (*y, x*) con attributi *attr*, spostando la riga dalla posizione *x* a destra di un carattere.

insdelln(*nlines*)
Inserisce *nlines* righe nella finestra specificata al di sopra della riga corrente. Le *nlines* righe di fondo vengono perdute. In caso di *nlines* negativo, vengono cancellate *nlines* righe a partire da quella sotto il cursore, e vengono spostate in alto le righe rimanenti. Le *nlines* righe di fondo vengono rimosse. La posizione corrente del cursore rimane la stessa.

insertln()
Inserisce una riga vuota sotto il cursore. Tutte le righe seguenti vengono spostate in basso di una riga.

insnstr([*y, x,*] *str, n* [*, attr*])
Inserisce una stringa di caratteri (tanti caratteri quanti ne possano stare sulla riga) prima del carattere sotto il cursore, fino al massimo di *n* caratteri. Se *n* è zero o negativo, viene inserita l'intera stringa. Tutti i caratteri alla destra del cursore vengono spostati a destra, causando la perdita dei caratteri più a destra nella riga. La posizione del cursore non viene modificata (dopo essersi spostata in *y, x*, se specificato).

instr([*y, x*] [*, n*])
Restituisce una stringa di caratteri, estratta dalla finestra a partire dalla posizione corrente del cursore, o da *y, x* se specificato. Gli attributi vengono rimossi dai caratteri. Se *n* viene specificato, `instr()` restituisce una stringa lunga al massimo *n* caratteri (escluso il NUL finale).

is_linetouched(*line*)
Restituisce vero se la riga specificata è stata modificata dall'ultima chiamata a `refresh()`; altrimenti restituisce falso. Solleva l'eccezione `curses.error` se la riga *line* risulta non valida per la finestra data.

is_wintouched()
Restituisce vero se la finestra specificata è stata modificata dall'ultima chiamata a `refresh()`; altrimenti restituisce false.

keypad(*yes*)
Se *yes* viene impostato a `1`, le sequenze di escape generate da alcuni tasti (tastierino numerico, tasti funzione) verranno interpretate da `curses`. Se *yes* viene impostato a `0`, le sequenze di escape verranno lasciate invariate nel flusso di input.

leaveok(*yes*)
Se *yes* viene impostato a `1`, ad ogni aggiornamento il cursore viene lasciato dove si trova, invece di rimanere alla "cursor position". Questo riduce i movimenti del cursore dove possibile. Se possibile il cursore verrà reso invisibile.

Se *yes* viene impostato a `0`, il cursore si troverà sempre alla "cursor position" dopo un aggiornamento.

move(*new_y, new_x*)
Sposta il cursore in (*new_y, new_x*).

mvderwin(*y, x*)
Sposta la finestra dentro la sua finestra madre. I parametri relativi allo schermo della finestra restano invariati. Questa procedura viene utilizzata per mostrare parti differenti della finestra madre alla stessa posizione fisica sullo schermo.

mvwin(*new_y, new_x*)
Sposta la finestra in modo che il suo angolo in alto a sinistra si trovi in (*new_y, new_x*).

nodelay(*yes*)
Se *yes* viene impostato ad `1`, `getch()` sarà non bloccante.

notimeout(*yes*)
Se *yes* viene impostato ad `1`, le sequenze di escape non avranno un time out.

Se *yes* viene impostato a `0`, dopo qualche millisecondo, una sequenza di escape non verrà interpretata, e resterà invariata nel flusso di input.

noutrefresh()
Segna per un refresh ma attende. Questa funzione aggiorna la struttura dei dati che rappresentaino lo stato desiderato della finestra, ma non impone un aggiornamento dello schermo fisico. Per ottenerlo, chiamate `doupdate()`.

overlay(*destwin*[, *sminrow, smincol, dminrow, dmincol, dmaxrow, dmaxcol*])
Sovrappone la finestra in cima a *destwin*. Le finestre non devono necessariamente avere la stessa dimensione, viene copiata solo la regione che si sovrappone. Questa copia non è distruttiva, ciò significa che il carattere dello sfondo corrente non sovrascrive i vecchi contenuti di *destwin*.

Per ottenere un controllo fine sulla granatura della regione copiata, si può utilizzare la seconda forma di `overlay()`. *sminrow* e *smincol* sono le coordinate in alto a sinistra della finestra sorgente, e le altre variabili segnano un rettangolo nella finestra di destinazione.

overwrite(*destwin*[, *sminrow, smincol, dminrow, dmincol, dmaxrow, dmaxcol*])
Sovrascrive la finestra in cima a *destwin*. Le finestre non devono necessariamente avere la stessa dimensione, viene copiata solo la regione che si sovrappone. Questa copia è distruttiva, ciò significa che il carattere di sfondo corrente sovrascrive i vecchi contenuti di *destwin*.

Per ottenere un ontrollo fine sulla granatura della regione copiata, si può utilizzare la seconda forma di `overwrite()`. *sminrow* e *smincol* sono le coordinate in alto a sinistra della finestra sorgente, e le altre variabili segnano un rettangolo nella finestra di destinazione.

putwin(*file*)
Scrive nell'oggetto file passato tutti i dati associati alla finestra. Queste informazioni possono venire recuperate in seguito, utilizzando la funzione `getwin()`.

redrawln(*beg, num*)
Indica che le *num* righe dello schermo a partire dalla riga *beg*, sono corrotte e dovrebbero venire completamente ridisegnate nella prossima chiamata a `refresh()`.

redrawwin()
Corregge l'intera finestra, ridisegnandola completamente nella prossima chiamata a `refresh()`.

refresh([*pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol*])
Aggiorna il display immediatamente (sincronizza lo schermo reale con i precendenti metodi di disegno/cancellazione).

I 6 argomenti facoltativi possono venire specificati solo quando la finestra è un pad creato con `newpad()`. Gli argomenti aggiuntivi sono necessari in quanto indicano quale parte del pad e dello schermo vengono impiegate. *pminrow* e *pmincol* specificano l'angolo in alto a sinistra del rettangolo che deve essere mostrato sul pad. *sminrow*, *smincol*, *smaxrow* e *smaxcol*, specificano i bordi del rettangolo che deve essere mostrato sullo schermo. L'angolo in basso a destra del rettangolo da mostrare sul pad viene calcolato dalle coordinate dello schermo, visto che i rettangoli devono essere della stessa dimensione. Entrambi i rettangoli devono essere interamente contenuti nelle rispettive strutture. Valori negativi di *pminrow*, *pmincol*, *sminrow* o *smincol*, vengono trattati come se fossero zero.

scroll([*lines = 1*])
Fa scorrere lo schermo o le regioni scorrevoli verso l'alto di *lines* linee.

scrollok(*flag*)
Controlla ciò che accade quando il cursore di una finestra viene mosso fuori dal bordo della finestra o della regione di scorrimento, come risultato ad esempio di un'azione di fine riga sulla linea di fondo, o digitando l'ultimo carattere dell'ultima linea. Se *flag* è falso, il cursore viene lasciato sulla riga di fondo. Se *flag* è vero, la finestra viene fatta scorrere in alto di una riga. Notate che per ottenere l'effetto di scorrimento fisico sul terminale, è necessario chiamare anche `idlok()`.

setscrreg(*top, bottom*)
Imposta le regione di scorrimento dalla riga *top* alla riga *bottom*. Tutte le azioni di scorrimento avranno luogo in questa regione.

standend()
Disabilita l'attributo *standout*. Su alcuni terminali può avere l'effetto collaterale di disabilitare tutti gli attributi.

standout()
Abilita l'attributo *A_STANDOUT*.

subpad([*nlines, ncols,*] *begin_y, begin_x*)
Restituisce una sotto finestra, il cui angolo in alto a sinistra si trova in (*begin_y, begin_x*), e la cui larghezza/altezza è *ncols*/*nlines*.

subwin([*nlines, ncols,*] *begin_y, begin_x*)
Restituisce una sotto finestra, il cui angolo in alto a sinistra si trova in (*begin_y, begin_x*), e la cui larghezza/altezza è *ncols*/*nlines*.

In modo predefinito, la sotto finestra si estenderà a partire dalla posizione specificata verso l'angolo in basso a destra della finestra.

syncdown()
Corregge ogni locazione nella finestra che è stata corretta in una qualsiasi delle sue finestre antenate. Questa procedura viene chiamata da `refresh()`, quindi non dovrebbe essere mai necessario chiamarla manualmente.

syncok(*flag*)
Se viene chiamata con *flag* impostata a vero, allora `syncup()` viene chiamata automaticamente ogni volta che avviene una modifica sulla finestra.

syncup()
Corregge tutte le locazioni negli antenati nella finestra che sono state modificate nella finestra stessa.

timeout(*delay*)
Imposta il comportamento bloccante o non bloccante alla lettura dell'input per la finestra. Se *delay* è negativo, viene usata la lettura bloccante (che attende indefinitivamente l'input). Se *delay* viene impostato a zero, allora viene usata la lettura non bloccante, e `getch()` restituirà `-1` in caso di assenza di input in attesa. Se *delay* è positivo, allora `getch()` attenderà *delay* millisecondi, e restituirà `-1` se non sopraggiungerà alcun input al termine di questo periodo.

touchline(*start, count*)
Simula che *count* righe siano state modificate, a partire dalla riga *start*.

touchwin()
Simula che l'intera finestra sia stata modificata, per scopi di ottimizzazione del disegno.

untouchwin()
Segna tutte le righe nella finestra come se non fossero state modificate dall'ultima chiamata a `refresh()`.

vline([*y, x,*] *ch, n*)
Mostra una linea verticale che inizia a (*y, x*) e di lunghezza *n*, composta da caratteri *ch*.

6.14.3 Costanti

Il modulo `curses` definisce i seguenti membri dato:

ERR
Alcune procedure curses che restituiscono un intero, come `getch()`, restituiscono `ERR` in caso di insuccesso.

OK
Alcune procedure curses che restituiscono un intero, come `napms()`, restituiscono `OK` in caso di successo.

version
Una stringa che rappresenta la versione corrente del modulo. Anche disponibile come `__version__`.

Sono disponibili alcune costanti per specificare gli attributi delle celle di caratteri:

Attributo	Significato
A_ALTCHARSET	Modalità di impostazione alternata dei caratteri.
A_BLINK	Modalità lampeggiante.
A_BOLD	Modalità grassetto.
A_DIM	Modalità offuscata.
A_NORMAL	Attributo normale.
A_STANDOUT	Modalità standout.
A_UNDERLINE	Modalità sottolineata.

I tasti vengono indicati tramite constanti intere, il cui nome inizia per 'KEY_'. I tasti disponibili dipendono dal sistema.

Costante	Chiave
KEY_MIN	Valore minimo di tasto
KEY_BREAK	Tasto break (inaffidabile)
KEY_DOWN	Freccia giù
KEY_UP	Freccia sù
KEY_LEFT	Freccia sinistra
KEY_RIGHT	Freccia destra
KEY_HOME	Tasto Home (upward+freccia sù)
KEY_BACKSPACE	Backspace (inaffidabile)
KEY_F0	Tasti funzione. Vengono supportati fino a 64 tasti funzione.
KEY_F*n*	Valore del tasto funzione *n*
KEY_DL	Cancella riga
KEY_IL	Inserisce riga
KEY_DC	Cancella carattere
KEY_IC	Inserisce carattere o entra nella modalità inserimento
KEY_EIC	Esce dalla modalità di inserimento carattere
KEY_CLEAR	Pulisce lo schermo
KEY_EOS	Cancella fino alla fine dello schermo
KEY_EOL	Cancella fino alla fine della riga
KEY_SF	Scorre di 1 riga in avanti
KEY_SR	Scorre di 1 riga indietro (inverso)
KEY_NPAGE	Pagina seguente
KEY_PPAGE	Pagina precedente
KEY_STAB	Imposta il Tab
KEY_CTAB	Cancella il Tab
KEY_CATAB	Cancella tutti i Tab
KEY_ENTER	Invio (inaffidabile)
KEY_SRESET	Soft (parziale) reset (inaffidabile)
KEY_RESET	Reset o hard reset (inaffidabile)
KEY_PRINT	Stampa
KEY_LL	Home giù o in fondo (lower left)
KEY_A1	In alto a sinistra del tastierino numerico
KEY_A3	In alto a destra del tastierino numerico
KEY_B2	Centro del tastierino numerico
KEY_C1	In basso a sinistra del tastierino numerico
KEY_C3	In basso a destra del tastierino numerico
KEY_BTAB	Tab indietro
KEY_BEG	Beg (inizio)
KEY_CANCEL	Cancella
KEY_CLOSE	Chiudi
KEY_COMMAND	Cmd (comando)
KEY_COPY	Copia
KEY_CREATE	Crea
KEY_END	Fine
KEY_EXIT	Esci
KEY_FIND	Trova

Costante	Chiave
KEY_HELP	Aiuto
KEY_MARK	Segna
KEY_MESSAGE	Messaggio
KEY_MOVE	Muovi
KEY_NEXT	Successivo
KEY_OPEN	Apri
KEY_OPTIONS	Opzioni
KEY_PREVIOUS	Prev (Precedente)
KEY_REDO	Ripeti
KEY_REFERENCE	Ref (riferimento)
KEY_REFRESH	Rinfresca
KEY_REPLACE	Sostituisci
KEY_RESTART	Riavvia
KEY_RESUME	Riprendi
KEY_SAVE	Salva
KEY_SBEG	Beg + Shift (inizio)
KEY_SCANCEL	Cancella + Shift
KEY_SCOMMAND	Comando + Shift
KEY_SCOPY	Copia + Shift
KEY_SCREATE	Crea + Shift
KEY_SDC	Cancella carattere + Shift
KEY_SDL	Cancella riga + Shift
KEY_SELECT	Scegli
KEY_SEND	Fine + Shift
KEY_SEOL	Cancella riga + Shift
KEY_SEXIT	Esci + Shift
KEY_SFIND	Trova + Shift
KEY_SHELP	Aiuto + Shift
KEY_SHOME	Home + Shift
KEY_SIC	Input + Shift
KEY_SLEFT	Freccia sinistra + Shift
KEY_SMESSAGE	Messaggio + Shift
KEY_SMOVE	Muovi + Shift
KEY_SNEXT	Successivo + Shift
KEY_SOPTIONS	Opzioni + Shift
KEY_SPREVIOUS	Precedente + Shift
KEY_SPRINT	Stampa + Shift
KEY_SREDO	Ripeti + Shift
KEY_SREPLACE	Sostituisci + Shift
KEY_SRIGHT	Freccia destra + Shift
KEY_SRSUME	Riprendi + Shift
KEY_SSAVE	Salva + Shift
KEY_SSUSPEND	Sospendi + Shift
KEY_SUNDO	Undo + Shift
KEY_SUSPEND	Sospendi
KEY_UNDO	Undo (ripristina lo stato precedente)
KEY_MOUSE	Si è verificato un evento del mouse
KEY_RESIZE	Evento di ridimensionamento del terminale
KEY_MAX	Massimo valore di chiave

Sui VT100 e loro emulazioni software, come gli emulatori di terminale sotto X, esistono normalmente almeno quattro tasti funzione disponibili, (KEY_F1, KEY_F2, KEY_F3, KEY_F4) ed i tasti freccia mappati come KEY_UP, KEY_DOWN, KEY_LEFT e KEY_RIGHT nell'ovvio ordine. Se la vostra macchina possiede una tastiera da PC, è ragionevole aspettarsi la presenza dei tasti freccia e di dodici tasti funzione (le tastiere da PC più vecchie potrebbero avere solo dieci tasti funzione); inoltre, le seguenti mappature del tastierino numerico sono standard:

Tasto	Costante
Insert	KEY_IC
Delete	KEY_DC
Home	KEY_HOME
End	KEY_END
Page Up	KEY_NPAGE
Page Down	KEY_PPAGE

La seguente tabella elenca i caratteri dall'insieme dei caratteri alternativi. Vengono ereditati dal terminale VT100 e generalmente sarà disponibile sulle emulazioni software, come i terminali sotto X. Quando non è disponibile la grafica, curses torna ad una cruda approssimazione ASCII stampabile. **Note:** I caratteri seguenti sono disponibili solo dopo aver chiamato `initscr()`.

Codice ACS	Significato
ACS_BBSS	nome alternativo per l'angolo in alto a destra
ACS_BLOCK	blocco solido quadrato
ACS_BOARD	tavola di quadrati
ACS_BSBS	nome alternativo per riga orizzontale
ACS_BSSB	nome alternativo per l'angolo in alto a sinistra
ACS_BSSS	nome alternativo per la top tee
ACS_BTEE	tee di fondo
ACS_BULLET	proiettile
ACS_CKBOARD	scacchiera (punteggiata)
ACS_DARROW	freccia che punta verso il basso
ACS_DEGREE	simbolo dei gradi
ACS_DIAMOND	diamante
ACS_GEQUAL	maggiore o uguale a
ACS_HLINE	riga orizzontale
ACS_LANTERN	simbolo della lanterna
ACS_LARROW	freccia sinistra
ACS_LEQUAL	minore o uguale a
ACS_LLCORNER	angolo in basso a sinistra
ACS_LRCORNER	angolo in basso a destra
ACS_LTEE	tee sinistra
ACS_NEQUAL	segno di non uguale
ACS_PI	pi greco
ACS_PLMINUS	segno più/meno
ACS_PLUS	segno più (grande)
ACS_RARROW	freccia destra
ACS_RTEE	tee destra
ACS_S1	scansiona riga 1
ACS_S3	scansiona riga 3
ACS_S7	scansiona riga 7
ACS_S9	scansiona riga 9
ACS_SBBS	nome alternativo per l'angolo in basso a destra
ACS_SBSB	nome alternativo per la linea verticale
ACS_SBSS	nome alternativo per la tee destra
ACS_SSBB	nome alternativo per l'angolo in basso a sinistra
ACS_SSBS	nome alternativo per la tee di fondo
ACS_SSSB	nome alternativo per la tee sinistra
ACS_SSSS	nome alternativo per incrocio o più (grande)
ACS_STERLING	sterlina
ACS_TTEE	tee in alto
ACS_UARROW	freccia sù
ACS_ULCORNER	angolo in alto a sinistra
ACS_URCORNER	angolo in alto a destra
ACS_VLINE	riga verticale

La seguente tabella elenca i colori predefiniti:

Costante	Colore
COLOR_BLACK	Nero
COLOR_BLUE	Blu
COLOR_CYAN	Ciano (blu verdino - acquamarina)
COLOR_GREEN	Verde
COLOR_MAGENTA	Magenta (rosso porpora)
COLOR_RED	Rosso
COLOR_WHITE	Bianco
COLOR_YELLOW	Giallo

6.15 curses.textpad — Widget per l'input di testo nei programmi curses

Nuovo nella versione 1.6.

Il modulo curses.textpad fornisce una classe Textbox che gestisce l'inserimento elementare di testo all'interno di una finestra curses, supportando un insieme di keybinding somiglianti a quelli di Emacs (e quindi, anche di Netscape Navigator, BBedit 6.x, FrameMaker, e molti altri programmi). Il modulo fornisce anche una funzione di disegno dei rettangoli, utile per incorniciare riquadri di testo o per altri scopi.

Il modulo curses.textpad definisce la seguente funzione:

rectangle(*win, uly, ulx, lry, lrx*)
Disegna un rettangolo. Il primo argomento deve essere un'oggetto finestra; gli argomenti rimanenti sono coordinate relative a quella finestra. Il secondo ed il terzo argomento sono le coordinate y ed x dell'angolo in alto a sinistra del rettangolo da disegnare; il quarto ed il quinto argomento sono le coordinate y e x dell'angolo in basso a destra. Il rettangolo verrà disegnato utilizzando le forme dei caratteri VT100/IBM PC sui terminali che lo rendono possibile (inclusa la xterm e la gran parte dei software di emulazione di terminale). Altrimenti verrà disegnato in ASCII, utilizzando tratteggi, barre verticali e segni.

6.15.1 Oggetti Textbox

Potete instanziare un oggetto Textbox nel modo seguente:

class Textbox(*win*)
Restituisce un oggetto widget textbox (NdT: casella di testo). L'argomento *win* dovrebbe essere un oggetto finestra curses WindowObject in cui deve essere contenuto il textbox. Il cursore di modifica del textbox inizialmente si trova nell'angolo in alto a sinistra della finestra contenitore, con coordinate (0, 0). L'opzione stripspaces dell'istanza è inizialmente attiva.

Gli oggetti Textbox hanno i seguenti metodi:

edit([*validator*])
Questo è il punto di ingresso che userete normalmente. Accetta digitazioni finchè non viene immessa una digitazione di terminazione. Se viene passato *validator*, dovrà essere una funzione. Verrà chiamata per ogni digitazione immessa, con quest'ultima come parametro; come risultato viene dato l'invio di un comando. Questo metodo restituisce i contenuti della finestra come una stringa; la presenza di caratteri vuoti dipende dal membro stripspaces.

do_command(*ch*)
Elabora una singola digitazione. Ecco le digitazione supportate:

Digitazione	Azione
`Control-A`	Vai al lato sinistro della finestra.
`Control-B`	Cursore a sinistra, riavvolgendo la prossima riga se necessario.
`Control-D`	Cancella il carattere sotto il cursore.
`Control-E`	Vai al lato destro (stripspaces off) o alla fine della riga (stripspaces on).
`Control-F`	Cursore a destra, riavvolgendo la prossima linea dove necessario.
`Control-G`	Termina, restituendo il contenuto della finestra.
`Control-H`	Cancella il carattere precedente.
`Control-J`	Termina nel caso la finestra possieda una sola riga, altrimenti inserisce un fine riga.
`Control-K`	Se la riga è vuota, cancellala, altrimenti pulisci tutto fino alla fine della riga.
`Control-L`	Aggiorna lo schermo.
`Control-N`	Cursore giù; spostati in basso di una riga.
`Control-O`	Inserisci una riga vuota sotto la posizione del cursore.
`Control-P`	Cursore sù; muoviti in alto di una riga.

Le operazioni di movimento non hanno effetto se il cursore si trova su un bordo, dove i movimenti non sono possibili. I seguenti sinonimi vengono supportati dove possibile:

Costante	Digitazione
`KEY_LEFT`	`Control-B`
`KEY_RIGHT`	`Control-F`
`KEY_UP`	`Control-P`
`KEY_DOWN`	`Control-N`
`KEY_BACKSPACE`	`Control-h`

Tutte le altre digitazioni vengono trattate come comandi d'inserimento del carattere fornito e poi si spostano a destra (con l'intera riga).

`gather``()`
Questo metodo restituisce i contenuti della finestra come una stringa; la presenza di caratteri vuoti dipende dal membro `stripspaces`.

`stripspaces`
Questo membro dato è un'opzione che controlla l'interpretazione dei caratteri vuoti nella finestra. Quando è attiva, i caratteri finali vuoti di ogni riga vengono ignorati; ogni movimento del cursore che portrebbe lo stesso cursore a posizionarsi su uno di questi caratteri vuoti, lo porta invece alla fine della riga, ed essi non vengono riportati quando il contenuto della finestra viene raccolto.

6.16 `curses.wrapper` — Gestore del terminale per i programmi curses

Nuovo nella versione 1.6.

Questo modulo fornisce una funzione, `wrapper()`, che lancia un'altra funzione che dovrebbe essere il resto della vostra applicazione che usa le curses. Se l'applicazione solleva un'eccezione, `wrapper()` ripristinerà il terminale in uno stato ottimale prima di passarlo oltre lo stack e generare una traceback.

`wrapper`(*func*, ...)
Funzione wrapper che inizializza curses e chiama un'altra funzione, *func*, ripristinando il comportamento normale di tastiera/schermo in caso di errore. All'oggetto chiamabile *func* viene quindi passata la finestra principale 'stdscr' come primo argomento, seguito da ogni altro argomento passato a `wrapper()`.

Prima di chiamare la funzione di hook (NdT: gancio), `wrapper()` attiva la modalità cbreak, disattiva l'echo, abilita il tastierino numerico del terminale, ed inizializza i colori nel caso in cui il terminale li supporti. Al momento dell'uscita (sia normale che a causa di un'eccezione) rispristina la modalità cooked, attiva l'echo, e disabilita il tastierino numerico.

6.17 `curses.ascii` — Utilità per i caratteri ASCII

Nuovo nella versione 1.6.

Il modulo `curses.ascii` fornisce dei nomi costanti per i caratteri ASCII, e delle funzioni per testarne l'appartenenza a varie classi di caratteri ASCII. Le costanti fornite sono nomi di caratteri di controllo come segue:

Nome	Significato
NUL	
SOH	Inizio dell'intestazione, segnale di interrupt della console
STX	Inizio del testo
ETX	Fine del testo
EOT	Fine della trasmissione
ENQ	Interrogazione, esegue un controllo di flusso con ACK
ACK	Dichiarazione
BEL	Campanella
BS	Backspace
TAB	Tab
HT	Sinonimo per TAB: "Tab orizzontale"
LF	Avanzamento di riga
NL	Sinonimo per LF: "Nuova riga"
VT	Tab verticale
FF	Avanzamento di pagina
CR	Ritorno del carrello (NdT: ritorno a capo)
SO	Shift-out, inizia ad alternare l'insieme dei caratteri
SI	Shift-in, ripristina l'insieme dei caratteri predefiniti
DLE	Escape Data-link
DC1	XON, per il controllo del flusso
DC2	Dispositivo di controllo 2, modo bloccante per il controllo di flusso
DC3	XOFF, per il controllo di flusso
DC4	Dispositivo di controllo 4
NAK	Negazione
SYN	Sincronia dell'idle
ETB	Fine del blocco di trasmissione
CAN	Cancella
EM	Fine del mezzo di trasmissione
SUB	Sostituisci
ESC	Esci
FS	Separatore di file
GS	Separatore di gruppo
RS	Separatore di record, termina la modalità bloccante
US	Unisci i separatori
SP	Spazio
DEL	Cancella

Notate che nell'uso moderno molti di questi termini possiedono un significato pratico ridotto. I significati mnemonici derivano dalle convenzioni delle telescriventi che hanno preceduto i calcolatori digitali.

Il modulo fornisce le seguenti funzioni, sulla base di quelle presenti nella libreria standard C.

`isalnum`(*c*)
Verifica che *c* sia un carattere ASCII alfanumerico; è equivalente a '`isalpha(`*c*`) o isdigit(`*c*`)`'.

`isalpha`(*c*)
Verifica che *c* sia un carattere ASCII alfabetico; è equivalente a '`isupper(`*c*`) o islower(`*c*`)`'.

`isascii`(*c*)
Verifica che il valore di un carattere *c* sia contenuto nell'insieme a 7-bit ASCII.

`isblank`(*c*)
Verifica che *c* sia uno spazio ASCII.

`iscntrl`(*c*)
Verifica che *c* sia un carattere di controllo ASCII (nell'intervallo 0x00 - 0x1f).

isdigit(*c*)
Verifica che *c* sia una cifra decimale ASCII, da '`0`' a '`9`'. È equivalente a '*c* `in string.digits`'.

isgraph(*c*)
Verifica che *c* sia un carattere ASCII stampabile, ad eccezione dello spazio.

islower(*c*)
Verifica che *c* sia un carattere ASCII minuscolo.

isprint(*c*)
Verifica che *c* sia un carattere ASCII stampabile, incluso lo spazio.

ispunct(*c*)
Verifica che *c* sia un carattere ASCII stampabile, diverso dallo spazio o da un carattere alfanumerico.

isspace(*c*)
Verifica che *c* sia uno spazio vuoto ASCII; cioè uno tra: spazio, line-feed, ritorno del carrello, avanzamento di pagina, tab orizzontale, tab verticale.

isupper(*c*)
Verifica che *c* sia una lettera ASCII maiuscola.

isxdigit(*c*)
Verifica che *c* sia una cifra ASCII esadecimale. È equivalente a '*c* `in string.hexdigits`'.

isctrl(*c*)
Verifica che *c* sia un carattere di controllo ASCII (valori ordinali da `0` a `31`).

ismeta(*c*)
Verifica che *c* non sia un carattere ASCII (valori ordinali oltre 0x80).

Queste funzioni accettano sia interi che stringhe; quando l'argomento è una stringa, viene prima convertito usando la funzione `ord()`.

Notate che tutte queste funzioni verificano i valori bit ordinali derivati dal primo carattere della stringa che viene passata; non sanno nulla riguardo la codifica dei caratteri della macchina ospite. Per le funzioni che riconoscono la codifica dei caratteri (e gestiscono in modo appropriato l'internazionalizzazione) vedete il modulo `string`.

Le seguenti due funzioni accettano sia una stringa di un unico carattere, che un valore di bit intero; restituiscono un valore dello stesso tipo.

ascii(*c*)
Restituisce il valore ASCII corrispondente ai primi 7 bit di *c*.

ctrl(*c*)
Restituisce il carattere di controllo corrispondente al carattere dato (sul valore di bit del carattere viene eseguito un and bit per bit con 0x1f).

alt(*c*)
Restituisce il carattere a 8 bit corrispondente al carattere ASCII dato (sul valore di bit del carattere viene eseguito un or bit per bit con 0x80).

La seguente funzione accetta sia una stringa di un unico carattere, che un valore intero; restituisce una stringa.

unctrl(*c*)
Restituisce una rappresentazione sotto forma di stringa del carattere ASCII di *c*. Se *c* è stampabile, la stringa sarà il carattere stesso. Se è un carattere di controllo (0x00-0x1f), la stringa sarà composta da un carattere di omissione ('`^`') seguito dalla corrispondente lettere maiuscola. Se il carattere è un ASCII delete (0x7f), la stringa sarà `'^?'`. Se il carattere ha il suo meta bit (0x80) impostato, quest'ultimo verrà rimosso, le precedenti regole verranno applicate, e infine verrà anteposto il carattere '`!`' al risultato.

controlnames
Un array stringa di 33 elementi che contiene i mnemonic ASCII per i 32 caratteri di controllo ASCII da 0 (NUL) a 0x1f (US), in ordine, più il mnemonico '`SP`' per il carattere spazio.

6.18 `curses.panel` — Un'estensione panel stack per curses

I pannelli sono finestre con la caratteristica della profondità, così da poter essere sovrapposti uno sull'altro, e solo la porzione visibile di ciascuna finestra verrà mostrata. I pannelli possono venire aggiunti, mossi in alto o in basso nella pila, e rimossi.

6.18.1 Funzioni

Il modulo `curses.panel` definisce le seguenti funzioni:

`bottom_panel``()`
Restituisce il pannello in fondo al panel stack.

`new_panel``(`*win*`)`
Restituisce un oggetto pannello, associandolo alla data finestra *win*.

`top_panel``()`
Restituisce il pannello in cima al panel stack.

`update_panels``()`
Aggiorna lo schermo virtuale dopo le modifiche al panel stack. Questa funzione non chiama `curses.doupdate()`, perciò dovrete farlo voi stessi.

6.18.2 Oggetti Panel

Gli oggetti panel, come restituiti dalla funzione `new_panel()` vista sopra, sono delle finestre con un ordine di sovrapposizione. Esiste sempre una finestra associata ad un pannello che ne determina il contenuto, mentre i metodi del pannello sono responsabili della profondità della finestra nel panel stack.

Gli oggetti panel possiedono i seguenti metodi:

`above``()`
Restituisce il pannello posto sopra il pannello attuale.

`below``()`
Restituisce il pannello posto sotto il pannello attuale.

`bottom``()`
Spinge il pannello in fondo allo stack.

`hidden``()`
Restituisce vero se il pannello è nascosto (non visibile), falso altrimenti.

`hide``()`
Nasconde il pannello. Ciò non comporta l'eliminazione dell'oggetto, semplicemente rende la finestra invisibile sullo schermo.

`move``(`*y*`, `*x*`)`
Sposta il pannello alle coordinate dello schermo `(`*y*`, `*x*`)`.

`replace``(`*win*`)`
Cambia la finestra associata al pannello con la finestra *win*.

`set_userptr``(`*obj*`)`
Imposta l'indicatore utente del pannello a *obj*. Questo viene utilizzato per associare una porzione di dati arbitraria al pannello, e può essere un qualsiasi oggetto Python.

`show``()`
Mostra il pannello (che può essere stato nascosto).

`top``()`
Spinge il pannello il cima allo stack.

userptr()
Restituisce l'indicatore utente del pannello. Può essere un qualsiasi oggetto Python.

window()
Restituisce l'oggetto finestra associato al pannello.

6.19 getopt — Parser per le opzioni da riga di comando

Questo modulo consente agli script ad analizzare gli argomenti da riga di comando in sys.argv. Supporta le stesse convenzioni della funzione UNIX getopt() (inclusi i significati speciali degli argomenti nella forma '-' e '--'). Le opzioni lunghe, analogamente a quelle supportate dai software GNU, possono venire usate anche tramite un terzo argomento facoltativo. Questo modulo fornisce una singola funzione ed una eccezione:

getopt(*args*, *options*[, *long_options*])
Analizza le opzioni da riga di comando e la lista dei parametri. *args* è l'argomento della lista da analizzare, senza il riferimento iniziale al programma in esecuzione. Tipicamente, questo significa 'sys.argv[1:]'. *options* è la stringa delle lettere-opzioni che lo script deve riconoscere, con le opzioni che richiedono un argomento, seguite da un segno di due punti (':'; per esempio, lo stesso formato che usa getopt() in UNIX).

Note: Diversamente dal getopt() GNU, dopo un argomento non-opzione, anche i successivi argomenti vengono considerati come non-opzioni. Questo comportamento è simile al modo in cui operano i sistemi UNIX non-GNU.

long_options, se specificato, deve essere una lista di stringhe con i nomi delle opzioni lunghe che dovrebbero venire supportate. I caratteri '--' iniziali non dovrebbero venire inclusi nel nome dell'opzione. Le opzioni lunghe che richiedono un argomento dovrebbero essere seguite dal segno di uguale ('='). Per accettare solo opzioni lunghe, *options* dovrebbe essere una stringa vuota. Le opzioni lunghe da riga di comando possono venire riconosciute come tali, purché contengano un prefisso del nome dell'opzione che corrisponda esattamente con una delle opzioni accettate. Per esempio, se *long_options* è ['foo', 'frob'], l'opzione **--fo** corrisponderà con **--foo**, ma **--f** non verrà considerata una corrispondenza univoca, così da sollevare un'eccezione GetoptError.

Il valore restituito consiste di due elementi: il primo è una lista di coppie (*option*, *value*); il secondo è la lista degli argomenti del programma rimasti a sinistra, dopo che la lista delle opzioni è stata tolta (questa è la suddivisione in fette di *args* (NdT: per mezzo di *split()*)). Ogni coppia opzione-valore restituita ha l'opzione come primo elemento, avente come prefisso un trattino per le opzioni brevi (per es. '-x') o due trattini per le opzioni lunghe (per es. '--long-option'), e l'argomento dell'opzione come suo secondo argomento, o una stringa vuota se l'opzione non ha argomenti. Le opzioni compaiono nella lista nello stesso ordine nel quale sono state individuate, permettendo così occorrenze nultiple. Le opzioni lunghe e brevi si possono mescolare.

gnu_getopt(*args*, *options*[, *long_options*])
Questa funzione opera come getopt(), con la differenza che lo stile GNU di modalità di scansione viene usato in modo predefinito. Questo significa che gli argomenti di opzione e di non-opzione possono venire mescolati tra loro. La funzione getopt() ferma l'elaborazione delle opzioni non appena viene incontrato un argomento di non-opzione.

Se il primo carattere dell'opzione stringa è '+', o se la variabile d'ambiente POSIXLY_CORRECT è impostata, allora l'elaborazione delle opzioni si ferma non appena viene incontrato un argomento di non-opzione.

exception GetoptError
Viene sollevata quando viene trovata un'opzione sconosciuta nella lista degli argomenti, oppure quando non viene fornito nessun argomento ad una opzione che ne richieda. L'argomento dell'eccezione è una stringa indicante la causa dell'errore. Per le opzioni lunghe, un argomento fornito ad un'opzione che non ne richieda, provocherà il sollevamento dell'eccezione. Gli attributi msg e opt danno il messaggio d'errore e le opzioni connesse; se non esiste l'opzione specifica alla quale l'eccezione si riferisce, allora opt è una stringa vuota.

Modificato nella versione 1.6: Introdotto GetoptError come sinonimo di error.

exception `error`
Sinonimo per `GetoptError`; per retrocompatibilità.

Un esempio usando solamente lo stile delle opzioni UNIX:

```
>>> import getopt
>>> args = '-a -b -cfoo -d bar a1 a2'.split()
>>> args
['-a', '-b', '-cfoo', '-d', 'bar', 'a1', 'a2']
>>> optlist, args = getopt.getopt(args, 'abc:d:')
>>> optlist
[('-a', ''), ('-b', ''), ('-c', 'foo'), ('-d', 'bar')]
>>> args
['a1', 'a2']
```

L'uso dei nomi lunghi delle opzioni è altrettanto facile:

```
>>> s = '--condition=foo --testing --output-file abc.def -x a1 a2'
>>> args = s.split()
>>> args
['--condition=foo', '--testing', '--output-file', 'abc.def', '-x', 'a1', 'a2']
>>> optlist, args = getopt.getopt(args, 'x', [
...     'condition=', 'output-file=', 'testing'])
>>> optlist
[('--condition', 'foo'), ('--testing', ''), ('--output-file', 'abc.def'), ('-x',
 '')]
>>> args
['a1', 'a2']
```

In uno script, l'uso tipico è qualcosa di simile a questo:

```
import getopt, sys

def main():
    try:
        opts, args = getopt.getopt(sys.argv[1:], "ho:v", ["help", "output="])
    except getopt.GetoptError:
        # stampa l'informazione di aiuto ed esce:
        usage()
        sys.exit(2)
    output = None
    verbose = False
    for o, a in opts:
        if o == "-v":
            verbose = True
        if o in ("-h", "--help"):
            usage()
            sys.exit()
        if o in ("-o", "--output"):
            output = a
    # ...

if __name__ == "__main__":
    main()
```

Vedete anche:

Modulo `optparse` (sezione 6.20):
Una riga di comando per l'analisi delle opzioni maggiormente orientata agli oggetti.

6.20 optparse — Un potente analizzatore per le opzioni da riga di comando.

Nuovo nella versione 2.3.

Il modulo optparse è una libreria potente, flessibile ed estensibile, da riga di comando, semplice da usare, per analizzare le opzioni. Con l'utilizzo di optparse, sarete in grado di gestire con poca fatica e dalla linea di comando, delle opzioni sofisticate ai vostri script.

Ecco un esempio di utlizzo di optparse, per aggiungere alcune opzioni da linea di comando ad un semplice script:

```
from optparse import OptionParser

parser = OptionParser()
parser.add_option("-f", "--file", dest="filename",
                  help="scrivi un rapporto su FILE", metavar="FILE")
parser.add_option("-q", "--quiet",
                  action="store_false", dest="verbose", default=True,
                  help="non stampare i messaggi di stato sullo stdout")

options, args = parser.parse_args()
```

Con queste poche righe di codice, gli utilizzatori del vostro script possono ora fare la "solita cosa" sulla linea di comando:

```
$ <yourscript> -f outfile --quiet
$ <yourscript> -qfoutfile
$ <yourscript> --file=outfile -q
$ <yourscript> --quiet --file outfile
```

(Tutti questi risultati in options.filename == outfile e options.verbose == False, esattamente come vi aspettereste.)

In modo ancor più elegante, l'utente potrebbe eseguire uno tra

```
$ <yourscript> -h
$ <yourscript> --help
```

e optparse stamperà un breve sommario delle opzioni del vostro script:

```
uso: <vostroscript> [opzioni]

opzioni:
  -h, --help           mostra questo messaggio ed esci
  -fFILE, --file=FILE  scrivi un rapporto su FILE
  -q, --quiet          non stampare i messaggi di stato sullo stdout
```

Questo è solo un assaggio della flessibilità che il modulo optparse può darvi per interpretare la linea di comando.

6.20.1 Filosofia

La finalità di optparse è quella di rendere possbile la presenza di una interfaccia ai programmi UNIX da linea di comando, che sia aderente agli standard, e semplice ed intuitiva da usare. La filosofia del modulo optparse è pesantemente influenzata dagli strumenti UNIX e GNU, e la presenza di questa sezione viene intesa per descrivere questa filosofia.

Terminologia

Innanzitutto dobbiamo stabilire un po' di terminologia.

argument
: una porzione di testo che l'utente digita sulla riga di comando, e che la shell passa a `execl()` o `execv()`. In Python, gli argomenti sono elementi di `sys.argv[1:]`. (`sys.argv[0]` è il nome che sta per essere eseguito; nell'ambito dell'analisi degli argomenti ciò non è molto importante. Anche le shell UNIX utilizzano il termine "word" (NdT: parola).

  Sarebbe auspicabile occasionalmente, utilizzare una lista argomenti differente da `sys.argv[1:]`, in modo che possiate leggere "argomento" come "un elemento di `sys.argv[1:]` oppure di qualche altra lista fornita in sostituzione di `sys.argv[1:]`".

option
: un argomento utilizzato per fornire informazioni supplementari al fine di guidare o personalizzare l'esecuzione di un programma. Esistono svariate differenze di sintassi per le opzioni; la sintassi tradizionale di UNIX è un trattino (-) seguito da una lettera singola, per esempio **-x** o **-F**. Inoltre la sintassi tradizionale di UNIX consente che le opzioni multiple vengano riunite in un singolo argomento, per esempio **-x -F** è equivalente a **-xF**. Il progetto GNU ha introdotto il doppio trattino (--) seguito da una serie di parole separate da un trattino, per esempio **--file** o **--dry-run**. Queste sono le sole due opzioni sintattiche concesse da `optparse`.

  Alcune altre opzioni di sintassi utilizzate dalla comunità informatica:

  - un trattino seguito da alcune lettere, per esempio **-pf** (questo *non* corrisponde ad opzioni multiple riunite in un singolo argomento.)
  - un trattino seguito da un'intera parola, per esempio **-file** (questo tecnicamente è equivalente alla precedente sintassi, ma solitamente non si trovano insieme nello stesso programma).
  - un segno "più" seguito da una lettera singola o da alcune lettere, oppure da una parola, per esempio **+f**, **+rgb**.
  - uno slash seguito da una lettera o da alcune lettere, oppure da una parola, per esempio **/f**, **/file**.

  `optparse` non supporta queste opzioni sintattiche, né lo farà in futuro. (Se volete veramente utilizzare una di queste opzioni di sintassi, dovrete creare una classe derivata di `OptionParser` e ridefinire tutti i singoli pezzi. Ma per favore non lo fate! `optparse` opera deliberatamente nel modo tradizionale proprio di UNIX/GNU; le prime tre opzioni non sono standard su nessun sistema, e l'ultima ha senso esclusivamente se state programmando per piattaforme MS-DOS/Windows e/o VMS.)

option argument
: un argomento che segue un'opzione, è strettamente collegato ad essa, e viene rimosso dalla lista degli argomenti contemporaneamente all'opzione stessa. Spesso, gli argomenti di un'opzione possono anche far parte dello stesso argomento costituente l'opzione, ad esempio:

```
["-f", "foo"]
```

  sarebbe equivalente a:

```
["-ffoo"]
```

  (`optparse` supporta questa sintassi.)

  Alcune opzioni non prendono mai un argomento. Altre invece richiedono sempre un argomento. Molti utenti desiderano avere la possibilità di passare "argomenti facoltativi ad un'opzione", nel senso che alcune opzioni prenderanno un argomento nel caso lo rilevino sulla riga di comando, altrimenti non ne prenderanno alcuno. Questo comportamento è in qualche modo controverso, perché rende ambigua l'analisi: se **-a** e **-b** sono entrambe opzioni, e **-a** prende un argomento facoltativo, come deve venire interpretato **-ab**? Il modulo `optparse` non supporta l'uso degli argomenti facoltativi di un'opzione.

argomento posizionale
: qualche volta presente come residuo della lista degli argomenti, dopo che le opzioni sono state analizzate; ovvero, dopo che le opzioni ed i loro argomenti siano stati analizzati e rimossi dalla lista degli argomenti.

opzione richiesta
: un'opzione che deve essere fornita sulla riga di comando. La frase "opzione richiesta" è un ossimoro; la presenza di "opzioni richieste" in un programma è di solito indice di una qualche carenza nel progetto dell'interfaccia utente. `optparse` non vi impedisce di implementare questo tipo di opzioni, tuttavia non vi fornisce neppure molto aiuto per farlo. Vedete "Esempi estesi" (sezione 6.20.5) per due modi di implementare le opzioni di questo tipo con `optparse`.

Per esempio, considerate questa ipotetica riga di comando:

```
prog -v --report /tmp/report.txt foo bar
```

-v e **--report** sono entrambe opzioni. Supponendo che l'opzione **--report** richieda un argomento, `/tmp/report.txt` è un argomento dell'opzione. `foo` e `bar` sono invece argomenti posizionali.

A cosa servono le opzioni?

Le opzioni vengono usate per fornire informazioni aggiuntive al programma, al fine di regolarne o personalizzarne l'esecuzione. Nel caso non fosse ancora chiaro, le opzioni dovrebbero essere *facoltative*. Un programma dovrebbe essere in grado comunque di funzionare correttamente senza alcuna opzione. (Prendete un programma qualsiasi tra gli strumenti forniti da UNIX o da GNU. Può girare senza alcuna opzione e comportarsi ancora sensatamente? Le sole eccezioni che mi vengono in mente sono **find**, **tar** e **dd**—che si comportano tutti e tre in maniera stravagante e variabile, e che sono stati giustamente criticati per la loro sintassi non standard e la loro interfaccia piuttosto confusa.)

Molte persone desiderano che i loro programmi abbiano delle "opzioni richieste". Pensateci. Se è obbligatoria, allora un'opzione *non è facoltativa*! Se vi è una qualche informazione che il vostro programma richiede assolutamente per poter funzionare con successo, per quello ci sono gli argomenti posizionali. (Tuttavia, se insistete ad aggiungere delle "opzioni richieste" ai vostri programmi, date un'occhiata alla sezione "Esempi estesi" (sezione 6.20.5) per due modi di implementarle con `optparse`.)

Considerate il semplice comando **cp**, per la copia dei file. Non ha molto senso provare a copiare dei file senza fornire un file di destinazione e almeno uno da copiare. Quindi, **cp** fallisce se lo eseguite senza argomenti. Tuttavia ha una sintassi utile e flessibile, che non dipende assolutamente dalle opzioni:

```
$ cp SOURCE DEST
$ cp SOURCE ... DEST-DIR
```

Potete fare moltissimo con solo questo comando. Molte implementazioni di **cp** forniscono una serie di opzioni per modificare leggermente il modo in cui i file debbano essere copiati: è possibile conservare la modalità (di lettura/scrittura/accesso) del file e l'ora di modifica, evitare di seguire i link simbolici, chiedere conferma prima di sovrascrivere file esistenti, etc. Ma nessuna di queste modifica lo scopo principale di **cp**, che è quello di copiare un file in un altro, o N file in un'altra directory.

A cosa servono gli argomenti posizionali?

Nel caso non fosse chiaro dall'esempio precendente: gli argomenti posizionali forniscono le informazioni che il vostro programma richiede assolutamente, concretamente, per poter funzionare.

Una buona interfaccia utente dovrebbe avere il minor numero possibile di argomenti assoluti. Se il vostro programma richiede 17 distinti pezzi d'informazione per poter funzionare correttamente, non importa poi molto *come* si ottengano queste informazioni dall'utente—la maggior parte delle persone rinuncerà e lascerà perdere prima di poter eseguire il programma con successo. Questo vale per qualsiasi tipo di interfaccia utente, sia la riga di coman-

do, un file di configurazione, una GUI, o qualunque altra cosa: se fate troppe richieste ai vostri utenti, la maggior parte di loro semplicemente lascerà perdere.

Per farla breve, provate a ridurre al minimo la quantità di informazioni che l'utente è assolutamente obbligato a fornire—usate dei ragionevoli valori predefiniti ogni volta che è possibile. Naturalmente, voi vorreste anche rendere i vostri programmi ragionevolmente flessibili. Le opzioni esistono per questo. Inoltre, a prescindere che esse siano valori di un file di configurazione, caselle di controllo nella finestra delle "Preferenze" di una GUI, o opzioni da specificare da riga di comando—quante più opzioni implementate, tanto più flessibile sarà il vostro programma, e tanto più complicata ne risulterà l'implementazione. È abbastanza probabile che gli utenti (e voi stessi!) vengano sopraffatti dalla troppa flessibilità, perciò siate prudenti a questo proposito.

6.20.2 Utilizzo di base

Dato che `optparse` è abbastanza flessibile e potente, non dovete fare i salti mortali o leggere risme di documentazione per poterlo utilizzare nei casi più semplici. Questo documento cerca di mostrare qualche semplice modello di utilizzo che vi permetterà di usare `optparse` nei vostri script.

Per analizzare il contenuto di una riga di comando con `optparse`, dovete creare una istanza di `OptionParser` e popolarla. Ovviamente dovrete importare le classi `OptionParser` in ogni script che utilizza `optparse`:

```
from optparse import OptionParser
```

All'inizio del corpo principale del programma, create un parser:

```
parser = OptionParser()
```

Quindi potete iniziare a popolare il parser con le opzioni. In realtà, ogni opzione è un insieme di stringhe di opzione sinonime; più comunemente, avrete una stringa di opzione in forma breve ed una stringa di opzione in forma lunga — ad esempio **-f** e **--file**:

```
parser.add_option("-f", "--file", ...)
```

La cosa interessante, naturalmente, è ciò che viene dopo le stringhe di opzione. Per il momento, noi tratteremo solo quattro fra le cose che potete prevedere al riguardo: *action*, *type*, *dest* e *help*.

L'azione di *memorizzazione*

L'azione istruisce `optparse` su cosa fare quando riceve una stringa di opzione per questa opzione, da riga di comando. Per esempio, l'azione di *memorizzazione* significa: prendi il prossimo argomento (o ciò che rimane dell'argomento corrente), assicurati che sia del tipo giusto, e memorizzalo nella destinazione prescelta.

Per esempio, inserite nel "..." dell'ultima opzione:

```
parser.add_option("-f", "--file",
                  action="store", type="string", dest="filename")
```

Adesso si crei una finta riga di comando e si chieda a `optparse` di analizzarla:

```
args = ["-f", "foo.txt"]
options, args = parser.parse_args(args)
```

(Notate che se non viene passata una lista di argomenti a `parse_args()`, automaticamente la funzione utilizzerà `sys.argv[1:]`.)

Quando `optparse` trova **-f**, elabora l'argomento seguente—`foo.txt`—e lo memorizza nell'attributo `filename` di uno speciale oggetto. Questo oggetto è il primo valore restituito da `parse_args()`, così:

```
print options.filename
```

stamperà `foo.txt`.

Altri tipi di opzioni forniti da `optparse` sono `int` e `float`. Ecco un'opzione che si aspetta un argomento di tipo intero:

```
parser.add_option("-n", type="int", dest="num")
```

Questo esempio non fornisce un tipo di opzione lunga, che è perfettamente accettabile. Così come non specifica l'azione—che viene impostata a "store" (NdT: memorizza) come valore predefinito.

Adesso facciamo l'analisi di un'altra riga di comando finta. Questa volta comprimeremo l'argomento dell'opzione e l'opzione stessa, per cui **-n42** (un argomento) è equivalente a **-n 42** (due argomenti).

```
options, args = parser.parse_args(["-n42"])
print options.num
```

Verrà stampato `42`.

La prova con un argomento di tipo "float" viene lasciata per esercizio al lettore.

Se non specificate un tipo, `optparse` presuppone il tipo "string". In combinazione col fatto che l'azione predefinita è "store", ciò significa che il nostro primo esempio può essere scritto in forma molto più breve:

```
parser.add_option("-f", "--file", dest="filename")
```

Se non prevedete una destinazione, `optparse` calcola un ragionevole valore predefinito dalla stringa delle opzioni: se la prima stringa dell'opzione lunga è **--foo-bar**, allora la destinazione predefinita sarà `foo_bar`. Se non ci sono stringhe di opzioni lunghe, `optparse` cerca nella prima opzione corta: la destinazione predefinita per **-f** è `f`.

Aggiungere ulteriori tipi è abbastanza semplice; vedete la sezione 6.20.5, "Aggiungere nuovi tipi".

Ulteriori azioni *store_**

Si evidenzia che il comportamento per i flag delle opzioni sono a comune—impostano una variabile a valore vero o falso quando viene rilevata una particolare opzione. `optparse` le supporta con due azioni distinte, "store_true" e "store_false". Per esempio, potreste avere un flag *verbose* che viene attivato con **-v** e disattivato con **-q**:

```
parser.add_option("-v", action="store_true", dest="verbose")
parser.add_option("-q", action="store_false", dest="verbose")
```

In questo caso abbiamo due opzioni differenti con la stessa destinazione, cosa che risulta perfettamente nella norma. (Questo significa che dovrete solo stare attenti al momento di impostare i valori predefiniti–vedete di seguito).

Quando `optparse` rileva **-v** sulla riga di comando, imposta `options.verbose` a `True`; quando rileva **-q**, imposta `options.verbose` a `False`.

Impostare i valori predefiniti

Tutti i precedenti esempi necessitano dell'impostazione di alcune variabili (la "destinazione") quando vengono rilevate certe opzioni da riga di comando. Cosa succede se queste opzioni non venissero mai rilevate? Se non vengono impostate delle opzioni predefinite, vengono tutte impostate a `None`. Questo comportamento qualche volta può essere accettabile (e questo è il motivo per cui è il predefinito), ma qualche volta potreste volere un un maggiore controllo. Per soddisfare questa necessità, `optparse` vi permette di impostare un valore predefinito per ogni destinazione, che viene assegnato prima che la riga di comando venga analizzata.

Innanzitutto considerate l'esempio prolisso/quieto. Se vogliamo che `optparse` imposti `verbose` a `True` senza che **-q** venga rilevato, allora possiamo fare così:

```
parser.add_option("-v", action="store_true", dest="verbose", default=True)
parser.add_option("-q", action="store_false", dest="verbose")
```

Stranamente, questo è esattamente equivalente:

```
parser.add_option("-v", action="store_true", dest="verbose")
parser.add_option("-q", action="store_false", dest="verbose", default=True)
```

Questi sono equivalenti in quanto avete fornito un valore predefinito per la *destinazione* delle opzioni, e queste due opzioni hanno la stessa destinazione (la variabile `verbose`)

Considerate questo:

```
parser.add_option("-v", action="store_true", dest="verbose", default=False)
parser.add_option("-q", action="store_false", dest="verbose", default=True)
```

Di nuovo, il valore predefinito per `verbose` sarà `True`: l'ultimo valore predefinito fornito per ogni destinazione è l'unico che ha importanza.

Generazione dell'help

L'ultima funzionalità che utilizzerete in ogni script sarà quella che permette a `optparse` di generare messaggi di *help*. Tutto quello che dovrete fare sarà fornire un argomento *help* quando aggiungerete un'opzione. Creiamo un nuovo analizzatore, e popoliamolo includendovi delle opzioni (documentate) semplificative:

```
usage = "utilizzo: %prog [opzioni] arg1 arg2"
parser = OptionParser(usage=usage)
parser.add_option("-v", "--verbose",
                  action="store_true", dest="verbose", default=True,
                  help="fa un sacco di rumore [predefinito]")
parser.add_option("-q", "--quiet",
                  action="store_false", dest="verbose",
                  help="è mooolto silenzioso (sono a caccia di conigli)")
parser.add_option("-f", "--file", dest="filename",
                  metavar="FILE", help="scrive l'output su FILE"),
parser.add_option("-m", "--mode",
                  default="intermediate",
                  help="modalità interattiva: specificare tra 'principiante', "
                       "'intermedio' [predefinito], 'esperto'")
```

Se `optparse` incontra **-h** o **--help** sulla riga di comando, oppure se invocate `parser.print_help()`, stamperà sullo stdout quanto segue:

```
utilizzo: <vostroscript> [opzioni] arg1 arg2

opzioni:
  -h, --help           mostra questo messaggio ed esce
  -v, --verbose        fa un sacco di rumore [predefinito]
  -q, --quiet          è mooolto silenzioso (sono a caccia di conigli)
  -fFILE, --file=FILE  scrive l'output su FILE
  -mMODE, --mode=MODE  modalità interattiva: specificare tra 'principiante',
                       'intermedio' [predefinito], 'esperto'
```

Ci sono molti argomenti per aiutare `optparse` a genereare il miglior messaggio d'aiuto possibile:

- lo script definisce il messaggio del proprio utilizzo:

  ```
  usage = "utilizzo: %prog [opzioni] arg1 arg2"
  ```

  `optparse` espande '`%prog`' nella stringa di utilizzo fino al nome dello script corrente, cioè `os.path.basename(sys.argv[0])`. La stringa espansa viene quindi stampata prima della descrizione dettagliata delle opzioni di aiuto.

  Se non fornite una stringa di utilizzo, `optparse` ne userà una predefinita, minimale ma comprensibile: `usage: %prog [options]`, che è perfetta se il vostro script non richiede nessun argomento posizionale.

- per ogni opzione viene definita una stringa di aiuto, e non preoccupatevi della formattazione delle righe di output—`optparse` se ne occupa autonomamente, rendendo gradevole alla vista l'output di aiuto.

- le opzioni che prendono un valore, lo indicano nel loro messaggio di aiuto generato automaticamente, per esempio per l'opzione "mode":

  ```
  -mMODE, --mode=MODE
  ```

  In questo caso "MODE" viene chiamata meta variabile: cioè resta in attesa dell'argomento fornito dall'utente a **-m**/**--mode**. `optparse`, come comportamento predefinito, converte il nome della variabile di destinazione in lettere maiuscole, e lo utilizza per indicare la meta variabile. Qualche volta, questo non è ciò che desiderate—per esempio, l'opzione che specifica il nome di un file, *filename*, crea esplicitamente `metavar=FILE`, con la seguente descrizione dell'opzione generata automaticamente:

  ```
  -fFILE, --file=FILE
  ```

  Questo, tuttavia, è importante anche per ragioni che vanno oltre il mero risparmio di spazio: il testo di aiuto scritto manualmente usa la meta variabile "FILE" per suggerire all'utente che esiste una correlazione tra la sintassi formale "-fFILE" e la descrizione semantica informale "scrive l'output su FILE". Questa è una maniera semplice ma efficace per rendere il vostro testo di aiuto assai più chiaro e fruibile per gli utilizzatori finali.

Quando si ha a che fare con molte opzioni, conviene raggrupparle per avere un migliore output di aiuto. Una classe `OptionParser` può contenere diversi gruppi di opzioni, ognuno dei quali può contenere diverse opzioni.

Continuando col parser definito sopra, è semplice aggiungere un `OptionGroup`:

```
group = OptionGroup(parser, "Opzioni Pericolose", "Attenzione:
utilizza queste opzioni a tuo rischio e pericolo.  "
"Alcune di queste sono note per creare dei danni.")
group.add_option("-g", action="store_true", help="Opzione di un gruppo.")
parser.add_option_group(group)
```

Il risultato dovrebbe essere l'output di aiuto seguente:

```
utilizzo: <vostroscript> [opzioni] arg1 arg2

opzioni:
  -h, --help           mostra questo messaggio ed esce
  -v, --verbose        fa un sacco di rumore [predefinito]
  -q, --quiet          è mooolto silenzioso (sono a caccia di conigli)
  -fFILE, --file=FILE  scrive l'output su FILE
  -mMODE, --mode=MODE  modalità interattiva: specificare tra 'principiante',
                       'intermedio' [predefinito], 'esperto'

  Opzioni pericolose:
    Attenzione: utilizza quest opzioni a tuo rischio e
    pericolo. Alcune di queste sono note per creare dei danni.
    -g                 Opzione di un gruppo.
```

Stampa un numero di versione

In modo analogo alle brevi stringhe sull'utilizzo, `optparse` può anche stampare una stringa che indichi la versione del vostro programma. Dovrete fornire voi la stringa, così come l'argomento *version* a `OptionParser`:

```
parser = OptionParser(usage="%prog [-f] [-q]", version="%prog 1.0")
```

version può contenere qualsiasi cosa vogliate; `%prog` viene espanso in *version* allo stesso modo in cui si verifica con *usage*. Quando fornite l'argomento *version*, `optparse` automaticamente aggiunge un'opzione **--version** al vostro parser. Se incontra questa opzione sulla riga di comando, espande la vostra stringa *version* (sostituendo `%prog`), la stampa sullo stdout, e termina il programma.

Per esempio, se il vostro script si chiama /usr/bin/foo, un utente potrebbe fare:

```
$ /usr/bin/foo --version
foo 1.0
```

Gestione degli errori

La sola cosa che dovete conoscere per un utilizzo basilare è il comportamento che `optparse` tiene quando incontra un errore sulla riga di comando—per esempio **-n 4x** dove **-n** è un valore di tipo intero. In questo caso, `optparse` stampa la vostra stringa di utilizzo sullo stderr, seguito da un messaggio di errore utile e comprensibile. Quindi termina (chiamando la funzione `sys.exit()`) con uno stato di uscita diverso da zero.

Se questo non fosse il vostro comportamento desiderato, create una classe derivata da `OptionParser`, e sovrascrivete il metodo `error()`. Vedete la sezione 6.20.5, "Estendere `optparse`."

Assemblare il tutto

Ecco come appare solitamente uno script basato su `optparse`:

```
from optparse import OptionParser
[...]
def main():
    usage = "uso: \%prog [-f] [-v] [-q] primoarg secondoarg"
    parser = OptionParser(usage)
    parser.add_option("-f", "--file", type="string", dest="filename",
                      help="legge i dati da FILENAME")
    parser.add_option("-v", "--verbose",
                      action="store_true", dest="verbose")
    parser.add_option("-q", "--quiet",
                      action="store_false", dest="verbose")

    options, args = parser.parse_args()
    if len(args) != 1:
        parser.error("numero di argomenti errato")

    if options.verbose:
        print "sto leggendo \%s..." \% options.filename
    [... corpo del programma ...]

if __name__ == "__main__":
    main()
```

6.20.3 Uso avanzato

Creare e popolare il parser

Ci sono modi diversi per popolare il parser con delle opzioni. Uno di questi consiste nel passare una lista di `Options` al costruttore della classe `OptionParser`

```
from optparse import OptionParser, make_option
[...]
parser = OptionParser(option_list=[
    make_option("-f", "--filename",
                action="store", type="string", dest="filename"),
    make_option("-q", "--quiet",
                action="store_false", dest="verbose")])
```

(`make_option()` è una funzione factory per la generazione di oggetti `Option`.)

Per avere un elenco lungo di opzioni, può essere più conveniente/leggibile creare le liste separatamente:

```
option_list = [make_option("-f", "--filename",
                           action="store", type="string", dest="filename"),
               [... opzioni ulteriori ...]
               make_option("-q", "--quiet",
                           action="store_false", dest="verbose")]
parser = OptionParser(option_list=option_list)
```

Oppure potete usare il metodo `add_option()` di `OptionParser` per aggiungere una opzione alla volta:

```
parser = OptionParser()
parser.add_option("-f", "--filename",
                  action="store", type="string", dest="filename")
parser.add_option("-q", "--quiet",
                  action="store_false", dest="verbose")
```

Questo metodo consente di tenere più facilmente traccia delle eccezioni sollevate dal costruttore `Option`, eccezioni che sono comuni a causa delle complicate interdipendenze tra i vari argomenti. (Se sbagliate, `optparse` solleva un'eccezione `OptionError`.)

`add_option()` può venire invocato in due diversi modi:

- passandogli una istanza di `Option` (come restituita da `make_option()`)
- passandogli una qualsiasi combinazione di argomenti posizionali e parole chiave che siano accettabili da `make_option()` (per esempio al costruttore `Option`), in modo che generi l'istanza `Option` al vostro posto (come mostrato sopra).

Definire le opzioni

Ogni instanza di `Option` rappresenta un insieme di sinonimi per le opzioni da riga di comando, cioè opzioni che abbiano lo stesso significato e gli stessi effetti, ma differente ortografia. Potete specificare qualsiasi numero di opzioni in forma di stringa breve o estesa, ma dovete specificare almeno una stringa di opzione.

Per definire una opzione che abbia solamente una stringa di opzione in forma breve:

```
make_option("-f", ...)
```

E per definire una opzione che abbia solamente una stringa di opzione in forma estesa:

```
make_option("--foo", ...)
```

I puntini di sospensione "..." indicano un insieme di parole chiave che definiscono gli attributi dell'oggetto `Option`. Le regole che determinano quali parole chiave dovete passare per una data `Option` sono abbastanza complicate, ma *qualche* parola chiave dovrete passarla in ogni caso. Se qualcosa va storto, `optparse` solleva un'eccezione `OptionError`, mostrandovi l'errore commesso.

L'attributo più importante di un'opzione è la sua azione, ad esempio che cosa fare quando viene incontrata questa opzione sulla riga di comando. Le azioni possibili sono:

Azione	**Significato**
store	memorizza l'argomento dell'opzione (predefinita)
store_const	memorizza il valore in una costante
store_true	memorizza un valore vero
store_false	memorizza un valore falso
append	accoda l'argomento dell'opzione ad una lista
count	incrementa un contatore di uno
callback	chiama una funzione specificata
help	stampa un messaggio che spiega l'utilizzo, e che include tutte le opzioni e la rispettiva documentazione

(Se non specificate un'azione, quella predefinita è "store". Per questa azione, potete anche fornire le parole chiave *type* e *dest*; vedete più avanti.)

Come potete vedere la maggior parte delle azioni comporta la memorizzazione o l'aggiornamento di un valore da qualche parte. `optparse` crea sempre un oggetto particolare (un'istanza della classe `Values`) prorio per questo

scopo. Gli argomenti di `Option` (e alcuni altri valori) vengono memorizzati come attributi di questo oggetto, a seconda dell'argomento *dest* passato a `make_option()`/`add_option()`.

Per esempio, quando chiamate:

```
parser.parse_args()
```

una delle prime cose che fa `optparse` consiste nel creare un oggetto `values`:

```
values = Values()
```

Se una delle opzioni di questo parser viene definita con:

```
make_option("-f", "--file", action="store", type="string", dest="filename")
```

e la riga di comando da analizzare include una qualsiasi tra le seguenti:

```
-ffoo
-f foo
--file=foo
--file foo
```

allora `optparse`, vedendo l'opzione **-f** o **--file**, farà l'equivalente di questo:

```
values.filename = "foo"
```

Chiaramente, gli argomenti *type* e *dest* sono importanti quasi quanto l'azione *action*. Tuttavia, *action* è il solo attributo significativo per *tutte* le opzioni, per questo è il più importante.

Azioni delle opzioni

Le varie azioni delle opzioni hanno tutte requisiti ed effetti leggermente differenti. Ad eccezione dell'azione "help", dovete fornire almeno un'altro argomento chiave quando create la `Option`; i requisiti esatti per ogni azione vengono elencati qui:

store

[requisiti pertinenti: *type*, *dest*, *nargs*, *choices*]

L'opzione deve essere seguita da un argomento, che viene convertito in un valore secondo quanto specificato in *type*, e memorizzato in *dest*. Se `nargs > 1`, possono venire elaborati più argomenti dalla riga di comando; tutti gli argomenti verranno convertiti secondo quanto specificato in *type*, e memorizzati sotto forma di tupla in *dest*. Vedete la sezionne 6.20.3, "Tipi delle opzioni", più sotto.

Se viene fornito *choices* (una sequenza di stringhe), il tipo predefinito è "choice".

Se *type* non viene specificato, il tipo predefinito è "string".

Se *dest* non viene specificato, `optparse` deriva il nome della variabile di destinazione dalla prima opzione in forma lunga (per esempio **--foo-bar** diventa `foo_bar`). Se non ci sono opzioni in forma lunga, `optparse` deriva il nome della destinazione dalla prima opzione in forma breve (per esempio, **-f** diventa `f`).

Esempio:

```
make_option("-f")
make_option("-p", type="float", nargs=3, dest="point")
```

Data la seguente riga di comando:

```
-f foo.txt -p 1 -3.5 4 -fbar.txt
```

`optparse` imposterà:

```
values.f = "bar.txt"
values.point = (1.0, -3.5, 4.0)
```

(In effetti, `values.f` viene impostato due volte, ma solo la seconda volta è visibile nella parte finale.)

store_const

[richiesti: *const*, *dest*]

Il valore *const* fornito al costruttore di `Option` viene memorizzato in *dest*.

Esempio:

```
make_option("-q", "--quiet",
       action="store_const", const=0, dest="verbose"),
make_option("-v", "--verbose",
       action="store_const", const=1, dest="verbose"),
make_option("--noisy",
       action="store_const", const=2, dest="verbose"),
```

Se viene visto **--noisy**, `optparse` imposterà:

```
values.verbose = 2
```

store_true

[richiesto: *dest*]

Uno caso speciale di “store_const”, che memorizza `True` in *dest*.

store_false

[richiesto: *dest*]

Come “store_true”, ma memorizza `False`.

Esempio:

```
make_option(None, "--clobber", action="store_true", dest="clobber")
make_option(None, "--no-clobber", action="store_false", dest="clobber")
```

append

[requisiti pertinenti: *type*, *dest*, *nargs*, *choices*]

L’opzione deve essere seguita da un argomento, che viene accodato alla lista in *dest*. Se non viene fornito alcun valore predefinito per *dest* (cioè il valore predefinito è `None`), non appena `optparse` incontra questa opzione sulla riga di comando viene automaticamente creata una lista vuota. Se `nargs > 1`, vengono elaborati più argomenti, e una tupla di lunghezza pari a *nargs* viene aggiunta a *dest*.

I valori predefiniti per *type* e *dest* sono uguali a quelli dell’azione “store”.

Esempio:

```
make_option("-t", "--tracks", action="append", type="int")
```

Se l’argomento **-t3** viene individuato sulla riga di comando, `optparse` esegue l’equivalente di:

```
values.tracks = []
values.tracks.append(int("3"))
```

Se, un po' più avanti, venisse individuato **--tracks=4**, verrebbe eseguito:

```
values.tracks.append(int("4"))
```

Vedete "Gestione degli Errori" (sezione 6.20.2) per informazioni su modo in cui `optparse` si comporta con qualcosa come **--tracks=x**.

count

[richiesto: *dest*]

Incrementa il valore intero memorizzato in *dest*. *dest* viene impostato a zero prima di venire incrementato la prima volta (a meno che non abbiate fornito un valore predefinito).

Esempio:

```
make_option("-v", action="count", dest="verbosity")
```

La prima molta che **-v** viene individuato sulla riga di comando, `optparse` esegue l'equivalente di:

```
values.verbosity = 0
values.verbosity += 1
```

Ogni occorrenza successiva di **-v** risulta tale:

```
values.verbosity += 1
```

callback

[richiesto: *callback*; requisiti pertinenti: *type*, *nargs*, *callback_args*, *callback_kwargs*]

Chiama la funzione specificata da *callback*. La dichiarazione della funzione dovrebbe essere:

```
func(option : Option,
     opt : string,
     value : any,
     parser : OptionParser,
     *args, **kwargs)
```

Le opzioni di callback vengono trattate dettagliatamente nella sezione 6.20.4, "Opzioni di callback".

help

[richiesto: nessuno]

Stampa un completo messagio di aiuto per tutte le opzioni nell'istanza corrente del parser delle opzioni. Il messaggio di aiuto viene costruito dalle stringhe *usage* passate al costruttore di `OptionParser`, e dalla stringa *help* passata singolarmente a ciascuna opzione.

Se non viene fornita nessuna stringa *help* per un'opzione, questa viene comunque mostrata nel messaggio di aiuto. Per omettere completamente una opzione, usate il valore speciale `optparse.SUPPRESS_HELP`.

Esempio:

```
from optparse import Option, OptionParser, SUPPRESS_HELP

usage = "usage: %prog [options]"
parser = OptionParser(usage, option_list=[
  make_option("-h", "--help", action="help"),
  make_option("-v", action="store_true", dest="verbose",
              help="Sii moderatamente verboso")
  make_option("--file", dest="filename",
              help="File di input da cui leggere i dati"),
  make_option("--secret", help=SUPPRESS_HELP)
])
```

Se `optparse` incontra **-h** oppure **--help** sulla riga di comando, verrà stampato sullo stdout qualcosa di simile al seguente messaggio di aiuto:

```
usage: <yourscript> [options]

options:
  -h, --help        Visualizza questo messaggio ed esci
  -v                Sii moderatamente verboso
  --file=FILENAME   File di input da cui leggere i dati
```

Dopo aver stampato il messaggio di aiuto, `optparse` termina il vostro processo con il codice di uscita `sys.exit(0)`.

version
[richiesto: nessuno]

Stampa il numero di versione fornito ad `OptionParser` sullo stdout, ed esce. Il numero di versione viene effettivamente formattato e stampato dal metodo `print_version()` di `OptionParser`. Generalmente è rilevante solo se l'argomento *version* viene fornito al costruttore di `OptionParser`.

Tipi delle opzioni

`optparse` supporta nativamente sei tipi di dato: *string*, *int*, *long*, *choice*, *float* e *complex*. (Di questi, stringa, numero intero, numero in virgola mobile, e scelta/selezione, sono quelli usati più fequentemente—intero long e numero complesso vengono supportati principalmente per completezza.) È facile aggiungere nuovi tipi di dato tramite la creazione di classi derivate di `Option`; vedete la sezione 6.20.5, "Estendere `optparse`".

Gli argomenti destinati alle opzioni di tipo stringa non vengono controllati o convertiti in alcun modo: il testo sulla riga di comando viene memorizzato nella destinazione (o passato alla funzione di callback) così come si trova.

Gli argomenti di tipo intero vengono passati a `int()` per essere convertiti in interi Python. Se `int()` fallisce, allora anche `optparse` lo farà, anche se con un messaggio di errore più utile e comprensibile. Internamente, `optparse` solleva l'eccezione `OptionValueError` in `optparse.check_builtin()`; ad un livello più alto (in `OptionParser`), `optparse` cattura questa eccezione, e termina il programma con un messaggio di errore comprensibile.

Allo stesso modo, gli argomenti di tipo in virgola mobile vengono passati a `float()` per la conversione, quelli di tipo long a `long()`, ed i complessi a `complex()`. A parte questo, vengono trattati alla stessa stregua degli argomenti di tipo intero.

Le opzioni di tipo "choice" (NdT: scelta/selezione) sono dei tipi derivati dalle opzioni "string". Una lista o tupla principale di scelte/selezioni (stringhe) deve venire passata al costruttore dell'opzione (`make_option()` oppure a `OptionParser.add_option()`) come chiave *choices*. Gli argomenti di un'opzione "choice" vengono confrontati con i valori della lista pincipale in `optparse.check_choice()`, e l'eccezione `OptionValueError` viene sollevata quando viene fornita una stringa sconosciuta.

Interrogare e manipolare il parser delle opzioni

A volte, può essere utile ficcare il naso nel parser delle opzioni e vedere che cosa c'è dentro. `OptionParser` fornisce un paio di metodi per aiutarvi in tal senso:

`has_option`(*opt_str*)
: Data come argomento una stringa di opzione come **-q** o **--verbose**, restituisce vero se `OptionParser` ha un'opzione con quella stringa d'opzione.

`get_option`(*opt_str*)
: Restituisce l'istanza di `Option` che implementa la stringa d'opzione da voi fornita, o `None` se nessuna opzione la implementa.

`remove_option`(*opt_str*)
: Se l'oggetto `OptionParser` ha un'opzione corrispondente a *opt_str*, l'opzione viene rimossa. Se quell'opzione forniva qualche altra stringa di opzioni, tutte quelle stringhe verranno ritenute non valide.

  Se *opt_str* non è presente in alcuna opzione appartenente a questo `OptionParser`, verrà sollevata l'eccezione `ValueError`.

Conflitti tra opzioni

Se non state attenti, è facile definire delle opzioni in mutuo conflitto:

```
parser.add_option("-n", "--dry-run", ...)
...
parser.add_option("-n", "--noisy", ...)
```

(Questo è più facile che si verifichi se avete definita una vostra propria classe derivata di `OptionParser` con qualche opzione standard.)

Basandosi sull'assunto che ciò è usualmente un errore, `optparse` solleva in modo predefinito un'eccezione (`OptionConflictError`) quando questo si verifica. Dato che questo è un errore di programmazione facilmente correggibile, non dovreste tentare di catturare l'eccezione — meglio correggere l'errore e tirare a campare.

A volte potreste volere che opzioni più recenti rimpiazzino deliberatamente le stringhe usate dalle opzioni più vecchie. Potete fare mediante la chiamata:

```
parser.set_conflict_handler("resolve")
```

che istruisce `optparse` a risolvere i conflitti tra opzioni in modo intelligente.

Ecco come funziona: ogni volta che aggiungete un'opzione, `optparse` verifica gli eventuali conflitti tra le opzioni già definite. Se ne trova qualcuno, chiama il meccanismo di gestione dei conflitti che viene specificato al costruttore di `OptionParser`:

```
parser = OptionParser(..., conflict_handler="resolve")
```

oppure tramite il metodo `set_conflict_handler()`.

Il meccanismo predefinto di gestione dei conflitti è `error`.

Ecco un esempio: prima definite un insieme di `OptionParser` per risolvere intelligentemente i conflitti:

```
parser = OptionParser(conflict_handler="resolve")
```

Adesso aggiungete le opzioni:

```
parser.add_option("-n", "--dry-run", ..., help="opzione dry-run originale")
...
parser.add_option("-n", "--noisy", ..., help="sii loquace")
```

A questo punto `optparse` individua che l'opzione precedentemente definita sta già usando la stringa di opzione **-n**. Dal momento che `conflict_handler == resolve`, la situazione viene risolta con la rimozione dell'opzione **-n** dalla precedente lista delle stringhe di opzioni. Ora, **--dry-run** è il solo modo per l'utente di attivare quell'opzione. Se l'utente chiederà aiuto, il relativo messaggio rifletterà questa situazione, cioè:

```
options:
  --dry-run     opzione dry-run originale
  ...
  -n, --noisy   sii loquace
```

Notate il fatto che è possibile ridurre via via le stringhe di opzione da una opzione già esistente, fino a che non ne sia rimasta nessuna, così che l'utente non abbia più modo di invocare l'opzione da riga di comando. In questo caso, `optparse` rimuove l'opzione completamente, e questa non verrà mostrata né nel messaggio di aiuto né altrove. Cioè, se continuiamo con il nostro `OptionParser`:

```
parser.add_option("--dry-run", ..., help="nuova opzione dry-run")
```

A questo punto, la prima opzione **-n**/**--dry-run** non è più accessibile, e quindi `optparse` la rimuove. Se l'utente chiede aiuto, otterrà qualcosa di simile:

```
options:
  ...
  -n, --noisy   sii loquace
  --dry-run     nuova opzione dry-run
```

6.20.4 Opzioni di callback

Se le azioni e i tipi predefiniti da `optparse` non si adattano alle esigenze del programmatore, e non vale la pena estendere `optparse` per definire nuove azioni o nuovi tipi, allora potreste aver bisogno di definire una opzione di callback. La definizione delle opzioni di callback è abbastanza semplice; il trucco consiste nel costruire una buona funzione di callback (la funzione che viene chiamata quando `optparse` incontra l'opzione sulla riga di comando).

Definire una opzione di callback

Come sempre, potete definire una opzione di callback sia istanziando direttamente la classe `Option`, sia usando il metodo `add_option()` dell'oggetto `OptionParser`. Il solo attributo dell'opzione che dovete specificare è *callback*, la funzione da chiamare:

```
parser.add_option("-c", callback=my_callback)
```

Notate che voi fornite un oggetto funzione —così dovrete aver già definito una funzione `my_callback()` nel momento in cui definite l'opzione di callback. In questo semplice caso, `optparse` non sa nulla sugli argomenti che l'opzione **-c** si aspetta di prendere – la mera presenza di **-c** sulla riga di comando è tutto ciò che ha bisogno di sapere. In alcune circostanze, tuttavia, potreste volere che la vostra funzione di callback elabori un numero

arbitrario di argomenti da riga di comando. Questo è il caso in cui bisogna applicare qualche trucco nella scrittura delle funzioni di callback; lo vedremo più avanti nel documento.

Ci sono alcuni altri attributi dell'opzione che potreste specificare quando definite un attributo opzione:

type
ha il suo significato usuale: così come nelle azioni "store" o "append", `optparse` viene istruito per elaborare un argomento che sia convertibile nel tipo specificato. Tuttavia, invece di memorizzare il valore (o i valori) da qualche parte, `optparse` lo converte nel tipo (*type*) specificato e lo passa alla vostra funzione di callback.

nargs
anche questo ha il suo significato abituale: se viene specificato e '`nargs > 1`', `optparse` elaborerà *nargs* argomenti, ognuno dei quali deve essere convertibile nel tipo (*type*) specificato. Quindi passa una tupla dei valori convertiti alla vostra funzione di callback.

callback_args
una tupla di argomenti posizionali extra da passare alla funzione di callback.

callback_kwargs
un dizionario extra di argomenti chiave da passare alla funzione di callback.

Come vengono chiamate le funzioni di callback

Tutte le funzioni di callback vengono chiamate nel modo seguente:

```
func(option, opt, value, parser, *args, **kwargs)
```

dove

option
è l'istanza `Option` che sta chiamando la funzione di callback.

opt
è la stringa di opzione trovata nella riga di comando che provoca la chiamata alla funzione di callback. (Se viene usata la forma abbreviata di un'opzione in forma estesa, *opt* conterrà quest'ultima, la stringa di opzione canonica—per esempio, se l'utente immette **--foo** sulla riga di comando come abbreviazione di **--foobar**, allora *opt* sarà **--foobar**.)

value
è l'argomento dell'opzione incontrata sulla riga di comando. `optparse` si aspetta solo un argomento se *type* viene impostato; il tipo di dato del valore *value* sarà del tipo sottinteso dallo stesso *type* dell'opzione (vedete 6.20.3, "Tipi delle opzioni"). Se il tipo dell'opzione *type* vale `None` (nessun argomento atteso), allora *value* varrà `None`. Se '`nargs > 1`', *value* sarà una tupla di valori del tipo appropriato.

parser
è l'istanza di `OptionParser` che chiama la funzione, utile principalmente perché attraverso essa potete accedere ad altri dati interessanti, come gli attributi dell'istanza:

parser.rargs
la lista corrente degli argomenti rimanenti, ovvero con *opt* rimosso (e *value*, se esistente) e con presenti i soli argomenti che li seguono. La lista `parser.rargs` può venire liberamente modificata, per esempio rimuovendo più argomenti.

parser.largs
l'insieme corrente di argomenti rimasti, cioè quegli argomenti che sono stati elaborati ma che non sono stati riconosciuti come opzioni (o come argomenti di opzione). `parser.largs` può venire liberamente modificato, ad esempio aggiungendovi ulteriori argomenti.

parser.values
: l'oggetto in cui i valori delle opzioni vengono memorizzati in modo predefinito. L'oggetto è utile perché permette alle funzioni di callback di utilizzare lo stesso meccanismo di `optparse` per la memorizzazione dei valori delle opzioni; non avete bisogno di pasticciare con variabili globali o valori restituiti. Potete anche accedere al(i) valore(i) di qualsiasi opzione incontrata sulla riga di comando.

args
: è una tupla di argomenti posizionali arbitrari fornita tramite l'attributo *callback*_args dell'opzione.

kwargs
: è un dizionario di chiavi arbitrarie fornito tramite *callback_kwargs*.

Dal momento che *args* e *kwargs* sono facoltativi (vengono passati solo definendo *callback_args* e/o *callback_kwargs* in fase di definizione della opzione di callback), la funzione di callback più semplice è:

```
def my_callback(option, opt, value, parser):
    pass
```

Gestione degli errori

Nel caso si presentassero dei problemi con l'opzione o i suoi argomenti, la funzione di callback dovrebbe generare l'eccezione `OptionValueError`. `optparse` cattura questa eccezione e termina il programma, stampando il messaggio di errore che avete impostato sullo stderr. Il messaggio dovrebbe essere chiaro, conciso, accurato e dovrebbe menzionare l'opzione che ha provocato l'eccezione. Altrimenti l'utente potrebbe perdere molto tempo per dover capire cosa possa aver sbagliato.

Esempi

Ecco un'esempio di opzione di callback che non accetta argomenti, e registra solamente il fatto che l'opzione è stata specificata:

```
def record_foo_seen(option, opt, value, parser):
    parser.saw_foo = 1

parser.add_option("--foo", action="callback", callback=record_foo_seen)
```

Naturalmente, potreste fare questo con l'azione "store_true". Ecco un esempio un po' più interessante: viene registrato il fatto che **-a** è stato specificato, ma se specificato dopo **-b** sulla riga di comando, viene considerato non valido.

```
def check_order(option, opt, value, parser):
    if parser.values.b:
        raise OptionValueError("can't use -a after -b")
    parser.values.a = 1
...
parser.add_option("-a", action="callback", callback=check_order)
parser.add_option("-b", action="store_true", dest="b")
```

Se volete riutilizzare questa funzione di callback per diverse opzioni simili (definire una stringa di opzione, ma invalidarla se **-b** è stata già invocata), è necessario lavorarci sopra: il messaggio di errore e la stringa di opzione su cui viene impostato hanno bisogno di venire generalizzati.

```
def check_order(option, opt, value, parser):
    if parser.values.b:
        raise OptionValueError("can't use %s after -b" % opt)
    setattr(parser.values, option.dest, 1)
...
parser.add_option("-a", action="callback", callback=check_order, dest='a')
parser.add_option("-b", action="store_true", dest="b")
parser.add_option("-c", action="callback", callback=check_order, dest='c')
```

Naturalmente, potete valutare qualsiasi condizione — non siete obbligati a controllare i valori delle opzioni già definite. Per esempio, se avete delle opzioni che non dovrebbero venire chiamate quando c'è luna piena, ciò che bisogna fare è questo:

```
def check_moon(option, opt, value, parser):
    if is_full_moon():
        raise OptionValueError("%s option invalid when moon full" % opt)
    setattr(parser.values, option.dest, 1)
...
parser.add_option("--foo",
                  action="callback", callback=check_moon, dest="foo")
```

La definizione di `is_full_moon()` viene lasciato come esercizio al lettore.

Argomenti costanti

Le cose diventano un po' più interessanti quando si definiscono opzioni di callback che accettano un numero di argomenti prefissato. La specificazione che una opzione di callback accetti argomenti è simile alla definizione di una opzione "store" o "append": se definite un *tipo*, allora l'opzione accetta un argomento che sia convertibile in quel tipo; se in seguito definirete *nargs*, allora l'opzione accetterà tutti quegli argomenti.

Ecco un esempio che emula l'azione standard "store":

```
def store_value(option, opt, value, parser):
    setattr(parser.values, option.dest, value)
...
parser.add_option("--foo",
                  action="callback", callback=store_value,
                  type="int", nargs=3, dest="foo")
```

Notate che `optparse` si occupa di elaborare tre argomenti e di convertirli in interi al posto vostro; tutto quello che voi dovrete fare sarà memorizzarli. (O qualcosa di simile: ovviamente non c'è bisogno di una funzione di callback per questo esempio. Usate la vostra immaginazione!)

Argomenti variabili

Le cose si complicheranno nel momento in cui vorrete un'opzione che accetti un numero variabile di argomenti. In questo caso dovrete scrivere una funzione di callback; `optparse` non possiede nessuna funzionalità built-in per poterlo fare Dovrete confrontarvi con la ben strutturata sintassi per poter effettuare l'analisi attraverso la riga di comando convenzionale UNIX. (In precedenza `optparse` si prendeva in carico questo compito, ma non funzionava bene. L'errore è stato corretto, al costo però di rendere più complesso questo tipo di callback.) In particolare, le funzioni di callback devono preoccuparsi degli argomenti nudi -- e -; la convenzione é:

- mero --, se non specificato come argomento a qualche opzione, comporta l'arresto dell'interpretazione della riga di comando ed il -- stesso viene perso.
- mero - analogamente, causa l'arresto dell'interpretazione della riga di comando, ma lo stesso - viene mantenuto.
- sia -- che - possono essere argomenti di un'opzione.

Se desiderate un'opzione che accetti un numero variabile di argomenti, bisogna che abbiate scaltrezza ed ingegno, per via di diverse difficoltà che potrebbero presentarvisi. L'esatta implementazione che verrà scelta dovrà essere basata su quello che desiderate e su quanto sarete disposti a fare per la vostra applicazione (questo è il motivo per cui `optparse` non supporta direttamente questo tipo di operazioni).

Comunque, ecco un tentativo di una opzione con argomenti variabili su di una funzione di callback:

```
def varargs(option, opt, value, parser):
    assert value is None
    done = 0
    value = []
    rargs = parser.rargs
    while rargs:
        arg = rargs[0]

        #  Stop se viene intercettato un argomento come "--foo", "-a",
#+ "-fx", "--file=f", etc.
#  Notate che che lo stop avviene anche su "-3" o "-3.0",
#+ così se la vostra opzione accetta valori numerici, dovrete
        #+ gestirli in questo modo.

        if ((arg[:2] == "--" and len(arg) > 2) or
            (arg[:1] == "-" and len(arg) > 1 and arg[1] != "-")):
            break
        else:
            value.append(arg)
            del rargs[0]

     setattr(parser.values, option.dest, value)

...
parser.add_option("-c", "--callback",
                  action="callback", callback=varargs)
```

La principale debolezza di questa particolare implementazione, è che i numeri negativi negli argomenti che seguono **-c** verranno interpretati come opzioni successive, piuttosto che come argomenti di **-c**. La risoluzione di questo problema viene lasciata per esercizio al lettore.

6.20.5 Estendere `optparse`

Poiché i due fattori principali di controllo nel modo in cui `optparse` interpreta le opzioni della riga di comando sono l'azione ed il tipo per ogni opzione, la via più facile per le estensioni è quella di aggiungere nuove azioni e nuovi tipi.

Inoltre, la sezione degli esempi include diverse dimostrazioni su come estendere `optparse` in diversi modi: per esempio una opzione del parser non sensibile alle differenze tra maiuscolo e minuscolo, o due tipi di opzioni di parser che implementino "opzioni obbligatorie".

Aggiunta di nuovi tipi

Per aggiungere nuovi tipi, dovrete definire una vostra classe derivata della classe `Option` di `optparse`. Questa classe possiede una coppia di attributi che definiscono i tipi di `optparse`: `TYPES` e `TYPE_CHECKER`.

`TYPES` è una tupla di nomi dei tipi; nella nuova classe derivata, definisce semplicemente una nuova tupla `TYPES` che si costruisce su quella standard.

`TYPE_CHECKER` è un dizionario che associa nomi di tipo alle funzione di controllo tipo. Una funzione di controllo tipo possiede la seguente definizione:

```
def check_foo(option : Option, opt : string, value : string)
              -> foo
```

La si può chiamare con qualsiasi nome, e fare in modo che restituisca ogni tipo voluto. Il valore restituito dalla funzione di controllo tipo può venire inserito nell'istanza `OptionValues` restituita da `OptionParser.parse_args()`, o venire passato alle funzioni di callback come parametro *valore*.

La vostra funzione di controllo tipo dovrebbe sollevare l'eccezione `OptionValueError` qualora incontrasse un qualsiasi problema. `OptionValueError` accetta un singolo argomento stringa, che viene passato senza trattamenti al metodo `error()` di `OptionParser`, che a sua volta prepone il nome del programma, e la stringa "error:" e stampa il tutto su stderr prima di terminare il processo.

Ecco un semplice esempio che dimostra come aggiungere un tipo di opzione "complex" per analizzare un numero complesso nello stile Python sulla riga di comando. (Questo è troppo semplice da potersi usare così com'è, perché `optparse` 1.3 aggiunge il supporto built-in per numeri complessi [semplicemente per completezza] ma non fateci caso.)

Per prima cosa, i necessari import:

```
from copy import copy
from optparse import Option, OptionValueError
```

Dovete definire prima il vostro controllo tipo, per potervi fare riferimento successivamente (nell'attributo di classe `TYPE_CHECKER` della propria classe derivata di `Option`):

```
def check_complex(option, opt, value):
    try:
        return complex(value)
    except ValueError:
        raise OptionValueError(
            "option %s: invalid complex value: %r" % (opt, value))
```

Finalmente, la classe derivata di `Option`:

```
class MyOption(Option):
    TYPES = Option.TYPES + ("complex",)
    TYPE_CHECKER = copy(Option.TYPE_CHECKER)
    TYPE_CHECKER["complex"] = check_complex
```

(Se non si fa il `copy()` di `Option.TYPE_CHECKER`, si potrebbe terminare la modificando l'attributo `TYPE_CHECKER` della classe `Option` di `optparse`. Questo è Python, niente ferma il programmatore tranne le buone maniere ed il buon senso.)

Questo è tutto! Adesso potete scrivere uno script che usi il nuovo tipo di opzione come un qualsiasi altro script basato su `optparse`, eccetto per il fatto che dovrete istruire il vostro `OptionParser` ad usare `MyOption` invece di `Option`:

```
parser = OptionParser(option_class=MyOption)
parser.add_option("-c", action="store", type="complex", dest="c")
```

In alternativa, potete costruire la vostra lista di opzioni e passarla a `OptionParser`: se non utilizzate `add_option()` nel modo descritto, non avrete bisogno di comunicare a `OptionParser` quale opzione di classe usare:

```
option_list = [MyOption("-c", action="store", type="complex", dest="c")]
parser = OptionParser(option_list=option_list)
```

Aggiungere nuove azioni

Aggiungere nuove azioni è un pò più complesso, perché dovete comprendere il fatto che `optparse` possiede una coppia di classificazioni per le azioni:

azioni "store"
: azioni che appaiono in `optparse` memorizzando un valore in un attributo dell'istanza OptionValues; queste opzioni richiedono un attributo *dest* da fornire nel costruttore Option

azioni "typed"
: azioni che prendono un valore da riga di comando, e si aspettano che sia di un certo tipo; o diversamente, una stringa che può venire convertita in un certo tipo. Queste opzioni richiedono un attributo di tipo (NdT: *type*) nel costruttore Option.

Alcune azioni predefinite "store" sono *store*, *store_const*, *append* e *count*. Le predefinite per l'azione "typed" sono *store*, *append* e *callback*.

Quando aggiungete un'azione, dovrete decidere se si tratta di una azione "store" o "typed", nessuna delle due, o entrambe. Tre attributi di `Option` (o di una sua classe derivata) controllano questo fatto:

`ACTIONS`
: Tutte le azioni devono venire elencate come stringhe in ACTIONS.

`STORE_ACTIONS`
: Le azioni "store" vengono aggiunte qui.

`TYPED_ACTIONS`
: Le azioni "typed" vengono aggiunte qui.

Per poter implementare la vostra nuova azione, dovrete sovrascrivere il metodo `take_action()` di `Option`, e implementare un caso che riconosca la vostra azione.

Per esempio, aggiungete un'azione "extend". Questa è simile all'azione standard "append", ma invece di prendere un singolo valore dalla riga di comando e aggiungerlo alla lista esistente, "extend" prenderà valori multipli in una stringa delimitata da virgole, e quindi estenderà con essi una lista esistente. Così, se **--names** è un'opzione "extend" di tipo stringa, la riga di comando:

```
--names=foo,bar --names blah --names ding,dong
```

potrebbe risultare in un lista:

```
["foo", "bar", "blah", "ding", "dong"]
```

Definiamo ancora una classe derivata di `Option`:

```
class MyOption(Option):

    ACTIONS = Option.ACTIONS + ("extend",)
    STORE_ACTIONS = Option.STORE_ACTIONS + ("extend",)
    TYPED_ACTIONS = Option.TYPED_ACTIONS + ("extend",)

    def take_action(self, action, dest, opt, value, values, parser):
        if action == "extend":
            lvalue = value.split(",")
            values.ensure_value(dest, []).extend(lvalue)
        else:
            Option.take_action(
                self, action, dest, opt, value, values, parser)
```

Note aggiuntive:

- "extend" si aspetta un valore nella riga di comando e memorizza quel valore da qualche parte, così agisce sia in `STORE_ACTIONS` che in `TYPED_ACTIONS`.
- `MyOption.take_action()` implementa solo questa nuova azione, e passa il controllo di nuovo a `Option.take_action()` per le azioni standard di `optparse`.
- *values* è una istanza della classe `Values`, che fornisce l'utile metodo `ensure_value()`. `ensure_value()` è essenzialmente `getattr()` con in più una valvola di sicurezza; viene chiamato come:

```
values.ensure_value(attr, value)
```

Se l'attributo `attr` di *values* non esiste o vale `None`, per prima cosa `ensure_value()` lo imposta a *value* e quindi restituisce *value*. Questo è molto utile per azioni come "extend", "append" e "count", ciascuna delle quali accumula dati in una variabile, e si aspetta che la variabile sia di un certo tipo (una lista per le prime due, un intero per l'ultima). L'uso di `ensure_value()` indica che che gli script contenenti la vostra azione non si debbano preoccupare di impostare un valore predefinito per le destinazioni della opzione in questione; possono lasciare semplicemente il valore predefinito a `None`, e `ensure_value()` si prenderà cura di restituire il valore giusto quando richiesto.

Altre motivazioni per estendere `optparse`

Aggiungere nuovi tipi e nuove azioni sono le principali, ovvie ragioni per cui possiate voler estendere `optparse`. Si può pensare ad almeno due motivi.

Il primo metodo, quello semplice: `OptionParser` cerca di essere d'aiuto chiamando `sys.exit()` quando necessario, per esempio quando si verifica un errore sulla riga di comando o quando l'utente richiama l'help. Nel primo caso, il metodo tradizionale di lasciare che lo script si blocchi con un traceback è inaccettabile; questo farà pensare agli utenti che lo script abbia un difetto quando invece causano un errore da riga di comando. Nel secondo caso, non c'è molto da fare dopo che è stato stampato il messaggio di aiuto.

Se questo comportamento non vi soddisfa, non dovrebbe essere troppo difficile "correggerlo". Dovrete:

1. creare una classe derivata di OptionParser e sovrascrivere il metodo `error()`.
2. creare una classe derivata di Option e sovrascrivere il metodo `take_action()` — dovrete fornire la vostra gestione dell'azione di "help", che non chiami sys.exit().

Il secondo metodo, molto più complesso, è quello di sovrascrivere la sintassi della riga di comando implementata da `optparse`. In questo caso, dovrete abbandonare a se stessa l'intera struttura delle azioni sulle opzioni e sui

tipi, riscrivendo il codice che processa `sys.argv`. Dovrete comunque derivare una classe da `OptionParser`; in funzione di quanto sarete radicali nella riscrittura, probabilmente dovrete sovrascrivere uno o tutti tra i metodi `parse_args()`, `_process_long_opt()` e `_process_short_opts()`.

Entrambi questi approcci vengono lasciati come un esercizio per il lettore. Non ho provato a implementarli personalmente, in quanto mi trovo già bene con il comportamento predefinito di `optparse` (naturalmente).

Buon Hacking, e non dimenticate: Usa il Sorgente, Luke.

Esempi

Ecco alcuni esempi su di estensione del modulo `optparse`.

Per prima cosa, cambiate l'analisi delle opzioni in modo che non siano sensibili a maiuscole e minuscole.

```
from optparse import Option, OptionParser, _match_abbrev

#  Questo parser di opzioni insensibile a maiuscole/minuscole richiede
#+ che sia disponibile un tipo di dizionario con la stessa caratteristica
#+ nei confronti di maiuscole/minuscole. Questo è uno per il Python 2.2.
#+ Notate che un dizionario reale insensibile a maiuscole/minuscole
#+ richiede anche che siano implementati __new__(), update(), e
#+ setdefault() -- ma non è questo l'oggetto di questo esercizio.

class caseless_dict (dict):
    def __setitem__ (self, key, value):
        dict.__setitem__(self, key.lower(), value)

    def __getitem__ (self, key):
        return dict.__getitem__(self, key.lower())

    def get (self, key, default=None):
        return dict.get(self, key.lower())

    def has_key (self, key):
        return dict.has_key(self, key.lower())


class CaselessOptionParser (OptionParser):

    def _create_option_list (self):
        self.option_list = []
        self._short_opt = caseless_dict()
        self._long_opt = caseless_dict()
        self._long_opts = []
        self.defaults = {}

    def _match_long_opt (self, opt):
        return _match_abbrev(opt.lower(), self._long_opt.keys())


if __name__ == "__main__":
    from optik.errors import OptionConflictError

    # test 1: nessuna opzione per partire con
    parser = CaselessOptionParser()
    try:
        parser.add_option("-H", dest="blah")
    except OptionConflictError:
        print "ok: ho OptionConflictError per -H"
    else:
        print "non è ok: nessuno conflitto tra  -h e -H"
```

```
parser.add_option("-f", "--file", dest="file")
#print repr(parser.get_option("-f"))
#print repr(parser.get_option("-F"))
#print repr(parser.get_option("--file"))
#print repr(parser.get_option("--fIlE"))
(options, args) = parser.parse_args(["--FiLe", "foo"])
assert options.file == "foo", options.file
print "ok: l'opzione lunga insensibile a maiuscole/minuscole lavora"

(options, args) = parser.parse_args(["-F", "bar"])
assert options.file == "bar", options.file
print "ok: l'opzione lunga insensibile a maiuscole/minuscole lavora"
```

E due modi di implementare le "opzioni richieste" con `optparse`.

Versione 1: Aggiungere un metodo a `OptionParser`, le cui applicazioni debbono chiamare dopo che gli argomenti siano stati analizzati:

_1.py

Versione 2: Estendere `Option`, ed aggiungere un attributo `required`; estendere `OptionParser` per assicurarsi che le opzioni `required` siano presenti dopo l'analisi.

_2.py

Vedete anche:

Modulo `getopt` (sezione 6.19):
Una più tradizionale riga di comando in stile UNIX per l'analisi delle opzioni.

6.21 `tempfile` — Generare file e directory temporanei

Questo modulo genera file e directory temporanei. Funziona su tutte le piattaforme supportate.

Nella versione 2.3 di Python, questo modulo è stato revisionato per accrescerne la sicurezza. Adesso fornisce tre nuove funzioni `NamedTemporaryFile()`, `mkstemp()` e `mkdtemp()` che dovrebbero eliminare tutti i casi in cui si debba usare la funzione insicura `mktemp()`. I nomi dei file temporanei creati da questo modulo non contengono più l'ID del processo; viene invece usata una stringa di sei caratteri casuali.

Inoltre, tutte le funzioni chiamabili dall'utente ora prendono argomenti aggiuntivi, consentendo il controllo diretto sulla posizione e sul nome dei file temporanei. Non è più necessario usare le variabili globali *tempdir* e *template*. Per mantenere la compatibilità all'indietro, l'ordine degli argomenti risulta piuttosto bizzarro; viene raccomandato, per chiarezza, di usare gli argomenti a parola chiave, keyword.

Il modulo definisce le seguenti funzioni chiamabili dall'utente:

`TemporaryFile`([*mode*=`'w+b'`[, *bufsize*=`-1`[, *suffix*[, *prefix*[, *dir*]]]]])
Restituisce un oggetto file (o simile a file) che può venire usato come area temporanea di memorizzazione. Il file viene creato usando `mkstemp`. Verrà distrutto non appena chiuso (includendo in questo una implicita chiusura quando l'oggetto viene raccolto dal garbage collector). Sotto UNIX, il parametro directory per il file viene rimosso non appena il file viene creato. Altre piattaforme non supportano questo comportamento; il vostro codice non dovrebbe fare affidamento su di un file temporaneo creato usando questa funzione, abbia o meno un nome visibile nel filesystem.

Il parametro *mode* viene impostato in modo predefinito a `'w+b'`, in modo tale che il file creato possa venire letto e scritto senza essere chiuso. Viene usata la modalità binaria, in modo tale che si comporti coerentemente su tutte le piattaforme, indipendentemente dal tipo dei dati immagazzinati. La variabile *bufsize* viene impostata al valore predefinito `-1`, a significare che viene usato il sistema operativo predefinito.

I parametri *dir*, *prefix* e *suffix* vengono passati a `mkstemp()`.

NamedTemporaryFile([*mode='w+b'*[, *bufsize=-1*[, *suffix*[, *prefix*[, *dir*]]]]])
Questa funzione opera esattamente come la funzione `TemporaryFile()`, eccetto che al file viene garantito di avere un nome visibile nel filesystem (su UNIX, il parametro directory non è scollegato). Questo nome può venire ritrovato dal `name` del membro del file oggetto. Che il nome possa venire usato per aprire il file una seconda volta, mentre il file temporaneo nominato resta ancora aperto, è un comportamento variabile a con la piattaforma (valido così su UNIX; viceversa non valido su Windows NT o successivi). Nuovo nella versione 2.3.

mkstemp([*suffix*[, *prefix*[, *dir*[, *text*]]]])
Crea un file temporaneo nel modo più sicuro possibile. Non ci saranno preferenze di "razza" nella creazione del file, assumendo che la piattaforma sottostante implementi in modo appropriato l'opzione `O_EXCL` per `os.open()`. Il file sarà leggibile e scrivibile solamente dall'utente che possiede l'ID associato alla creazione del file stesso. Se la piattaforma usa i bit dei permessi per indicare l'eseguibilità del file, il file non risulterà eseguibile da nessun altro utente. Il descrittore del file non viene ereditato dal processo derivato.

Diversamente da `TemporaryFile()`, l'utente di `mkstemp()` è responsabile della cancellazione del file temporaneo, quando realizzato con questa funzione.

Se il suffisso *suffix* viene specificato, il nome del file terminerà con quel suffisso, altrimenti non avrà suffisso. `mkstemp()` non inserisce un punto tra il nome del file ed il suffisso; se ne avete bisogno, mettetelo all'inizio del suffisso *suffix*.

Se il prefisso *prefix* viene specificato, il nome del file inizierà con quel prefisso; altrimenti verrà utilizzato un prefisso predefinito.

Se *dir* viene specificata, il file verrà creato in quella directory; altrimenti verrà usata una directory predefinita.

Se *text* viene specificato, questo indicherà se il file debba venire aperto in modalità binaria (predefinita) o testuale. Su qualche piattaforma, questo non fa differenza.

`mkstemp()` restituisce una tupla contenente un descrittore di file aperto (come dovrebbe venire restituito da `os.open()`) e il percorso assoluto di quel file, in questo ordine. Nuovo nella versione 2.3.

mkdtemp([*suffix*[, *prefix*[, *dir*]]])
Crea un directory temporanea nella maniera più sicura possibile. Non ci saranno preferenze di "razza" nella creazione della directory. La directory sarà leggibile, scrivibile, e attraversabile solamente dall'utente con che possiede l'ID associato alla creazione della directory stessa.

L'utente di `mkdtemp()` è responsabile della cancellazione della directory temporanea e del suo contenuto, quando realizzata con questa funzione.

Gli argomenti *prefix*, *suffix* e *dir* sono gli stessi della funzione `mkstemp()`.

`mkdtemp()` restituisce il percorso assoluto della nuova directory. Nuovo nella versione 2.3.

mktemp([*suffix*[, *prefix*[, *dir*]]])
Deprecato dalla versione 2.3. Usate invece `mkstemp()`.

Restituisce un percorso assoluto di un file che non esiste al momento della chiamata. Gli argomenti *prefix*, *suffix* e *dir* sono gli stessi della funzione `mkstemp()`.

Avvertenze: L'uso di questa funzione può introdurre un buco di sicurezza nel vostro programma. Nel tempo che voi impiegate per fare qualcosa con il nome del file restituito, qualcun altro potrebbe avervi già abilmente raggirato.

Il modulo usa due variabili globali che dicono ad esso come costruire un nome temporaneo. Queste vengono inizializzate alla prima chiamata di una delle funzioni sottostanti. Il chiamante le può modificare, ma questo è sconsigliato; usate invece gli appropriati argomenti della funzione.

tempdir
Quando impostata ad un valore diverso da `None`, questa variabile definisce il valore predefinito dell'argomento *dir* per tutte le funzioni definite in questo modulo.

Se `tempdir` non è impostato o è `None` per nessuna chiamata e a nessuna delle funzioni precedenti, Python cerca una lista standard di directory ed imposta *tempdir* alla prima directory nella quale l'utente che effettua la chiamata abbia il permesso di creare dei file. La lista è:

1. La directory indicata dalla variabile d'ambiente TMPDIR.

2.La directory indicata dalla variabile d'ambiente TEMP.

3.La directory indicata dalla variabile d'ambiente TMP.

4.Una posizione specifica, dipendente dalla piattaforma:

- Su Macintosh, la cartella 'Temporary Items'.
- Su RiscOS, la directory ndicata dalla variabile d'ambiente Wimp$ScrapDir.
- Su Windows, le directory 'C:\TEMP', 'C:\TMP', '\TEMP' e '\TMP', in questo ordine.
- Su tutte le altre piattaforme, le directory '/tmp', '/var/tmp', e '/usr/tmp', in questo ordine.

5.Come ultima risorsa, la directory di lavoro corrente.

gettempdir()
: Restituisce la directory correntemente selezionata per crearvi un file temporaneo. Se `tempdir` non è `None`, allora semplicemente restituisce il suo contenuto; altrimenti, la ricerca decritta più avanti viene svolta e restituito il risultato.

template
: **Deprecato dalla versione 2.0.** Usate invece `gettempprefix()`.

  Quando impostata ad un valore diverso da `None`, questa variabile definisce il prefisso del componente finale del nome del file restituito da `mktemp()`. Una stringa di sei lettere e numeri casuali viene aggiunta al prefisso, per rendere univoco il nome del file. Su Windows, il prefisso predefinito è '`~T`'; su tutti gli altri sistemi è 'tmp'.

  Le versioni più vecchie di questo modulo usualmente richiedono che questo `template` sia impostato a `None` dopo una chiamata `os.fork()`; questo non è più necessario dalla versione 1.5.2.

gettempprefix()
: Restituisce il prefisso del nome del file usato per creare file temporanei. Questo non contiene componenti della directory. Usare questa funzione è preferibile, piuttosto che leggere direttamente la variabile da *template*. Nuovo nella versione 1.5.2.

6.22 errno — Sistema standard dei simboli di errore

Questo modulo rende disponibile il sistema standard dei simboli di errore di sistema (`errno`). Il valore di ogni simbolo è il corrispondente valore intero. Il nome e la descrizione vengono presi in prestito dal file 'linux/include/errno.h', che dovrebbe contenerli tutti.

errorcode
: Il dizionario che fornisce una mappa dal valore di errno al nome della stringa del sistema sottostante. Per esempio, `errno.errorcode[errno.EPERM]` mappa a `'EPERM'`.

Per tradurre un codice numerico di errore in un messaggio di errore, usate la funzione `os.strerror()`.

Della seguente lista, i simboli che non vengono usati nella piattaforma corrente non vengono definiti dal modulo. La lista specifica dei simboli definiti è disponibile come `errno.errorcode.keys()`. I simboli disponibili possono contenere:

EPERM
: Operation not permitted

ENOENT
: No such file or directory

ESRCH
: No such process

EINTR
: Interrupted system call

EIO
: I/O error

ENXIO
: No such device or address

E2BIG
Arg list too long

ENOEXEC
Exec format error

EBADF
Bad file number

ECHILD
No child processes

EAGAIN
Try again

ENOMEM
Out of memory

EACCES
Permission denied

EFAULT
Bad address

ENOTBLK
Block device required

EBUSY
Device or resource busy

EEXIST
File exists

EXDEV
Cross-device link

ENODEV
No such device

ENOTDIR
Not a directory

EISDIR
Is a directory

EINVAL
Invalid argument

ENFILE
File table overflow

EMFILE
Too many open files

ENOTTY
Not a typewriter

ETXTBSY
Text file busy

EFBIG
File too large

ENOSPC
No space left on device

ESPIPE
Illegal seek

EROFS

Read-only file system

EMLINK
Too many links

EPIPE
Broken pipe

EDOM
Math argument out of domain of func

ERANGE
Math result not representable

EDEADLK
Resource deadlock would occur

ENAMETOOLONG
File name too long

ENOLCK
No record locks available

ENOSYS
Function not implemented

ENOTEMPTY
Directory not empty

ELOOP
Too many symbolic links encountered

EWOULDBLOCK
Operation would block

ENOMSG
No message of desired type

EIDRM
Identifier removed

ECHRNG
Channel number out of range

EL2NSYNC
Level 2 not synchronized

EL3HLT
Level 3 halted

EL3RST
Level 3 reset

ELNRNG
Link number out of range

EUNATCH
Protocol driver not attached

ENOCSI
No CSI structure available

EL2HLT
Level 2 halted

EBADE
Invalid exchange

EBADR
Invalid request descriptor

EXFULL
Exchange full

ENOANO
No anode

EBADRQC
Invalid request code

EBADSLT
Invalid slot

EDEADLOCK
File locking deadlock error

EBFONT
Bad font file format

ENOSTR
Device not a stream

ENODATA
No data available

ETIME
Timer expired

ENOSR
Out of streams resources

ENONET
Machine is not on the network

ENOPKG
Package not installed

EREMOTE
Object is remote

ENOLINK
Link has been severed

EADV
Advertise error

ESRMNT
Srmount error

ECOMM
Communication error on send

EPROTO
Protocol error

EMULTIHOP
Multihop attempted

EDOTDOT
RFS specific error

EBADMSG
Not a data message

EOVERFLOW
Value too large for defined data type

ENOTUNIQ
Name not unique on network

EBADFD

File descriptor in bad state

EREMCHG
Remote address changed

ELIBACC
Can not access a needed shared library

ELIBBAD
Accessing a corrupted shared library

ELIBSCN
.lib section in a.out corrupted

ELIBMAX
Attempting to link in too many shared libraries

ELIBEXEC
Cannot exec a shared library directly

EILSEQ
Illegal byte sequence

ERESTART
Interrupted system call should be restarted

ESTRPIPE
Streams pipe error

EUSERS
Too many users

ENOTSOCK
Socket operation on non-socket

EDESTADDRREQ
Destination address required

EMSGSIZE
Message too long

EPROTOTYPE
Protocol wrong type for socket

ENOPROTOOPT
Protocol not available

EPROTONOSUPPORT
Protocol not supported

ESOCKTNOSUPPORT
Socket type not supported

EOPNOTSUPP
Operation not supported on transport endpoint

EPFNOSUPPORT
Protocol family not supported

EAFNOSUPPORT
Address family not supported by protocol

EADDRINUSE
Address already in use

EADDRNOTAVAIL
Cannot assign requested address

ENETDOWN
Network is down

ENETUNREACH
Network is unreachable

ENETRESET
Network dropped connection because of reset

ECONNABORTED
Software caused connection abort

ECONNRESET
Connection reset by peer

ENOBUFS
No buffer space available

EISCONN
Transport endpoint is already connected

ENOTCONN
Transport endpoint is not connected

ESHUTDOWN
Cannot send after transport endpoint shutdown

ETOOMANYREFS
Too many references: cannot splice

ETIMEDOUT
Connection timed out

ECONNREFUSED
Connection refused

EHOSTDOWN
Host is down

EHOSTUNREACH
No route to host

EALREADY
Operation already in progress

EINPROGRESS
Operation now in progress

ESTALE
Stale NFS file handle

EUCLEAN
Structure needs cleaning

ENOTNAM
Not a XENIX named type file

ENAVAIL
No XENIX semaphores available

EISNAM
Is a named type file

EREMOTEIO
Remote I/O error

EDQUOT
Quota exceeded

6.23 `glob` — Modello di espansione del percorso in stile UNIX

Il modulo `glob` individua tutti i percorsi corrispondenti ad uno specificato modello, secondo le regole della shell UNIX. Non viene effettuata l'espansione della tilde, ma `*`, `?` e gli intervalli di caratteri espressi con `[]` corrisponderanno correttamente. Questo comportamento viene realizzato utilizzando di concerto le funzioni `os.listdir()` e `fnmatch.fnmatch()`, e senza invocare una subshell. (Usate `os.path.expanduser()` e `os.path.expandvars()` per l'espansione della variabile tilde e delle altre variabili di shell).

`glob`(*pathname*)
: Restituisce una lista possibile-vuota dei nomi di percorso che corrispondono con *pathname*, che deve essere una stringa contenente uno specifico percorso all'interno del filesystem. *pathname* può essere sia assoluta (come '/usr/src/Python-1.5/Makefile') o relativa (come '../../Tools/*/*.gif'), e che può contenere i caratteri jolly previsti dalla shell.

Per esempio, considerate una directory contenente solo i seguenti file: '1.gif', '2.txt' e 'card.gif'. La funzione `glob()` produrrà i seguenti risultati. Notate come tutti i componenti iniziali del percorso vengano preservati.

```
>>> import glob
>>> glob.glob('./[0-9].*')
['./1.gif', './2.txt']
>>> glob.glob('*.gif')
['1.gif', 'card.gif']
>>> glob.glob('?.gif')
['1.gif']
```

Vedete anche:

Modulo `fnmatch` (sezione 6.24):
: Espansione dei nomi dei file (non dei percorsi) in stile shell

6.24 `fnmatch` — Modello di corrispondenza dei nomi di file in stile UNIX

Questo modulo fornisce il supporto per i caratteri jolly nello stile delle shell UNIX, che *non* sono gli stessi delle espressioni regolari (documentate nel modulo `re`). I caratteri speciali usati dai caratteri jolly sono:

Modello	Significato
*	corrisponde con tutti i caratteri
?	corrisponde con un qualsiasi carattere singolo
[*seq*]	corrisponde con ogni carattere di *seq*
[!*seq*]	corrisponde con ogni carattere non contenuto in *seq*

Notate che il separatore del nome del file (`'/'` su UNIX) *non* è speciale per questo modulo. Vedete il modulo `glob` per l'espansione del percorso (`glob` utilizza `fnmatch()` per ricercare le corrispondenze nei segmenti di percorso). In modo simile, i nomi dei file che iniziano con un punto non sono speciali per questo modulo, e vengono corrisposti dai modelli `*` e `?`.

`fnmatch`(*filename, pattern*)
: Esamina se la stringa diel nome del file *filename* corrisponde con la stringa del modello *pattern*, restituendo vero o falso. Se il sistema operativo non è sensibile alle differenze tra maiuscole e minuscole, allora entrambi i parametri verranno normalizzati tutti in maiuscolo o in minuscolo prima di effettuare il confronto. Se avete bisogno di un confronto sensibile alle differenze tra maiuscole e minuscole indipendente dallo standard del vostro sistema operativo, usate invece `fnmatchcase()`.

`fnmatchcase`(*filename, pattern*)
: Verifica se il nome del file *filename* corrisponde con il modello *pattern*, restituendo vero o falso; il confronto è sensibile alla differenza tra maiuscole e minuscole.

`filter`(*names, pattern*)
: Restituisce il sotto insieme della lista dei nomi, *names*, che corripondono con il modello *pattern*. È la stessa

cosa di `[n for n in names if fnmatch(n, pattern)]`, ma viene implementato in modo più efficiente. Nuovo nella versione 2.2.

Vedete anche:

Modulo `glob` (sezione 6.23):
Espansione dei percorsi nello stile della shell UNIX.

6.25 `shutil` — Operazioni di alto livello sui files

Il modulo `shutil` offre numerose operazioni di alto livello sui file e sugli insiemi di file. In particolare, vengono fornite funzioni che supportano la copia e la rimozione dei file.

Avvertenza: Su MacOS, la risorsa fork ed altri metadati non vengono usati. Per la copia dei file, questo significa che le risorse verranno perse e il tipo del file ed i codici di creazione non saranno corretti.

`copyfile`(*src*, *dst*)
Copia il contenuto del file chiamato *src* nel file chiamato *dst*. La posizione della destinazione deve avere il permesso di scrittura; altrimenti verrà sollevata un'eccezione `IOError`. Se *dst* già esiste, verrà sostituito. I file speciali come i dispositivi a caratteri o a blocchi e le pipe non possono venire copiati con questa funzione. *src* e *dst* sono i nomi dei percorsi forniti come stringhe.

`copyfileobj`(*fsrc*, *fdst*[, *length*])
Copia il contenuto dell'oggetto simile a file *fdst* nell'oggetto simile a file *fdst*. L'intero *length*, se fornito, rappresenta la dimensione del buffer. In particolare, un valore negativo di *length* significa copiare i dati senza eseguire un ciclo sopra la sorgente dei dati spezzettati; in modo predefinito i dati vengono letti in singole porzioni, per evitare consumi di memoria incontrollati.

`copymode`(*src*, *dst*)
Copia i bit dei permessi da *src* a *dst*. Il contenuto del file, il suo proprietario ed il relativo gruppo di appartenenza rimangono inalterati. *src* e *dst* sono i nomi dei percorsi forniti come stringhe.

`copystat`(*src*, *dst*)
Copia i bit dei permessi, il tempo rappresentante l'ultimo accesso e il tempo dell'ultima modifica da *src* a *dst*. Il contenuto del file, il suo proprietario ed il relativo gruppo di appartenenza rimangono inalterati. *src* e *dst* sono i nomi dei percorsi forniti come stringhe.

`copy`(*src*, *dst*)
Copia il file *src* nel file o directory *dst*. Se *dst* è una directory, un file con lo stesso prefisso del percorso di *src* viene creato (o sovrascritto) nella directory specificata. I bit dei permessi vengono copiati. *src* e *dst* sono i nomi dei percorsi forniti come stringhe.

`copy2`(*src*, *dst*)
Simile a `copy()`, ma vengono preservate le date di ultimo accesso e di ultima modifica. Questo è simile al comando UNIX **cp -p**.

`copytree`(*src*, *dst*[, *symlinks*])
Copia ricorsivamente un'intero albero di directory avente radice in *src*. La directory di destinazione, chiamata da *dst*, non deve esistere; verrà creata al momento. I singoli file vengono copiati usando `copy2()`. Se *symlinks* è impostato a `True`, i collegamenti simbolici nell'albero sorgente verranno rappresentati come collegamenti nel nuovo albero; se impostato a `False` o omesso, i contenuti dei file collegati da link simbolici verrnno copiati nel nuovo albero. Se si presentassero dei casi particolari, verrà sollevata un'eccezione `Error` con un elenco delle motivazioni che l'hanno provocata.

In questi casi il codice sorgente dovrebbe essere considerato un esempio piuttosto che uno strumento. Modificato nella versione 2.3: Viene sollevata un'eccezione `Error` per ogni eccezione intercettata durante la copia, invece she stampare solo un messaggio.

`rmtree`(*path*[, *ignore_errors*[, *onerror*]])
Cancella un intero albero di directory. Se *ignore_errors* è impostato a `True`, gli errori risultanti da una fallita rimozione verranno ignorati; se impostato a `False` o omesso, tali errori verranno trattati da un gestore specificato da *onerror* o, se questo viene omesso, solleveranno un'eccezione.

Se viene fornito *onerror*, deve essere un oggetto chiamabile che accetti tre parametri; *function*, *path* ed

excinfo. Il primo parametro, *function*, è la funzione che solleva l'eccezione; sarà una tra `os.remove()` o `os.rmdir()`. Il secondo parametro, *path*, sarà il nome del percorso passato alla funzione *function*. Il terzo parametro, *excinfo*, sarà l'informazione sull'eccezione restituita da `sys.exc_info()`. Le eccezioni sollevate da *onerror* non verranno intercettate.

`move`(*src*, *dst*)
Sposta ricorsivamente un file o una directory in un altra posizione.

Se la destinazione è nel nostro filesystem corrente, verrà usato semplicemente rename. Altrimenti, copierà la sorgente nella destinazione, rimuovendo successivamente la sorgente.

Nuovo nella versione 2.3.

exception `Error`
Questa eccezione raccoglie le eccezioni che vengono sollevate durante una operazione su file multipli. Per `copytree`, l'argomento dell'eccezione è una tuple a tre elementi (*srcname*, *dstname*, *exception*).

Nuovo nella versione 2.3.

6.25.1 Esempio

Questo esempio è l'implementazione della funzione `copytree()`, descritta precedentemente, con la docstring omessa. Questo esempio mostra molte delle altre funzioni fornite da questo modulo.

```
def copytree(src, dst, symlinks=0):
    names = os.listdir(src)
    os.mkdir(dst)
    for name in names:
        srcname = os.path.join(src, name)
        dstname = os.path.join(dst, name)
        try:
            if symlinks and os.path.islink(srcname):
                linkto = os.readlink(srcname)
                os.symlink(linkto, dstname)
            elif os.path.isdir(srcname):
                copytree(srcname, dstname, symlinks)
            else:
                copy2(srcname, dstname)
        except (IOError, os.error), why:
            print "Non posso copiare %s in %s: %s" % (‘srcname‘, ‘dstname‘, str(motivo))
```

6.26 `locale` — Servizi per l'internazionalizzazione

Il modulo `locale` accede al database delle localizzazioni POSIX, ed alle sue funzionalità. Il meccanismo della localizzazione POSIX consente ai programmatori di occuparsi solamente dei messaggi inviati da un'applicazione, senza dover conoscere le specifiche dei Paesi in cui il software verrà eseguito.

Il modulo `locale` viene implementato sul modulo `_locale`, che usa a sua volta una implementzione della localizzazione ANSI C, se disponibile.

Il modulo `locale` definisce le seguenti funzioni ed eccezioni:

exception `Error`
Eccezione sollevata quando `setlocale()` fallisce.

`setlocale`(*category*[, *locale*])
Se il parametro *locale* viene specificato, potrebbe essere una stringa, una tupla nella forma `(`*language code*`,` *encoding*`)`, o `None`. Se è una tupla, viene convertita in una stringa usando il motore di aliasing locale. Se *locale* viene fornito e non è `None`, `setlocale()` modifica l'impostazione locale per la categoria *category*. Le categorie disponibili vengono elencate nella descrizione qui sotto. Il valore è il nome di una localizzazione. Una stringa vuota specifica le impostazioni predefinite dell'utente. Se la modifica della

localizzazione fallisce, viene sollevata l'eccezione `Error`. Se ha successo, viene restituita l'impostazione della nuova localizzazione.

Se *locale* viene omesso o è `None`, viene restituito il valore corrente per *category*.

`setlocale()` non garantisce la sicurezza del thread sulla maggior parte dei sistemi. Le applicazioni si avviano tipicamente con una chiamata di

```
import locale
locale.setlocale(locale.LC_ALL, '')
```

Questa imposta la localizzazione per tutte le categorie dei parametri utente (solitamente specificati nella variabile d'ambiente LANG). Se da questo punto in poi la localizzazione non viene modificata, usare il multithreading non dovrebbe causare problemi.

Modificato nella versione 2.0: Aggiunto il supporto per valori tupla del parametro *locale*.

`localeconv``()`

Restituisce il database delle convenzioni locali in forma di dizionario. Questo dizionario ha le seguenti stringhe come chiavi:

Chiave	Categoria	Significato
LC_NUMERIC	`'decimal_point'`	Carattere del punto decimale.
	`'grouping'`	Sequenza di numeri che specifica in quali posizioni relative il `'thousands_sep'` è atteso. Se la sequenza viene terminata con `CHAR_MAX`, non verrà effettuato nessun altro ragguppamento. Se la sequenza termina con `0`, la misura dell'ultimo gruppo viene usata ripetutamente.
	`'thousands_sep'`	Carattere usato tra gruppi.
LC_MONETARY	`'int_curr_symbol'`	Simbolo di valuta internazionale.
	`'currency_symbol'`	Simbolo di valuta locale.
	`'mon_decimal_point'`	Punto decimale usato per valori monetari.
	`'mon_thousands_sep'`	Separatore di gruppo usato per valori monetari.
	`'mon_grouping'`	Equivalente a `'grouping'`, usato per valori monetari.
	`'positive_sign'`	Simbolo usato per annotare un valore monetario positivo.
	`'negative_sign'`	Simbolo usato per annotare un valore monetario negativo.
	`'frac_digits'`	Numero di cifre frazionarie usate nella formattazione locale dei valori monetari.
	`'int_frac_digits'`	Numero di cifre frazionarie usate nella formattazione internazionale dei valori monetari.

I possibili valori per `'p_sign_posn'` e `'n_sign_posn'` vengono forniti qui sotto.

Valore	Spiegazione
0	La valuta ed il valore vengono racchiusi tra parentesi.
1	Il segno dovrebbe precedere il valore ed il simbolo di valuta.
2	Il segno dovrebbe seguire il valore ed il simbolo di valuta.
3	Il segno dovrebbe precedere immediatamente il valore.
4	Il segno dovrebbe seguire immediatamente seguire il valore.
`LC_MAX`	Non è specificato niente in questa localizzazione.

`nl_langinfo``(`*option*`)`

Restituisce alcune informazioni sulle specifiche della localizzazione, sotto forma di stringa. Questa funzione non è disponibile in tutti i sistemi, e l'insieme delle possibili opzioni potrebbe variare attraverso le piattaforme. I possibili valori degli argomenti sono numeri, per i quali le costanti simboliche sono disponibili nel modulo locale.

`getdefaultlocale``([`*envvars*`])`

Tenta di determinare le impostazioni predefinite della localizzazione, e le restituisce come una tupla nella forma (*language code*, *encoding*).

In accordo con POSIX, un programma che non ha effettuato una chiamata a `setlocale(LC_ALL, '')` viene eseguito usando la localizzazione portabile `'C'`. La chiamata a `setlocale(LC_ALL, '')` fa invece in modo che il programma usi la localizzazione predefinita, impostata dalla variabile LANG. Poiché non vogliamo interferire con l'impostazione della localizzazione corrente, emuliamo così il comportamento nel modo descritto sopra.

Per mantenere la compatibilità verso altre piattaforme, non viene verificata solo la variabile LANG, ma anche tutta una lista di variabili definite come parametri di ambiente. La prima trovata come definita verrà usata. I valori predefiniti di *envvars* vengono impostati dal percorso di ricerca usato da GNU Gettext; questo deve sempre contenere il nome della variabile '`LANG`'. Il percorso di ricerca di GNU Gettext contiene le varibili di ambiente `'LANGUAGE'`, `'LC_ALL'`, `'LC_CTYPE'` e `'LANG'`, in questo ordine.

Fatta eccezione per il linguaggio `'C'`, l'identificazione del linguaggio viene effettuata secondo il canone stabilito dalla RFC 1766. *language code* e *encoding* possono essere impostate a `None` se il loro valore non può essere determinato. Nuovo nella versione 2.0.

`getlocale`([*category*])
Restituisce l'impostazione corrente per la data categoria di localizzazione, fornita come una sequenza contenente *language code*, *encoding*. *category* può assumere uno dei valori `LC_*`, con l'eccezione di `LC_ALL`. Il suo valore predefinito è `LC_CTYPE`.

Fatta eccezione per il linguaggio `'C'`, l'identificazione del linguaggio viene effettuata secondo il canone stabilito dalla RFC 1766. *language code* e *encoding* possono essere impostate a `None` se il loro valore non può essere determinato. Nuovo nella versione 2.0.

`getpreferredencoding`([*do_setlocale*])
Restituisce la codifica utilizzata per i dati in formato testo, in accordo con le preferenze dell'utente. Le preferenze dell'utente vengono espresse differentemente su differenti sistemi operativi, e potrebbero non essere disponibili in modo programmato su alcuni sistemi, così questa funzione restituisce solo una congettura.

Su alcuni sistemi è necessaria per invocare `setlocale` per ottenere le preferenze dell'utente, quindi questa funzione non è thread-safe. Se l'invocazione di `setlocale` non è necessaria o desiderata, *do_setlocale* dovrebbe venire impostata a `False`.

Nuovo nella versione 2.3.

`normalize`(*localename*)
Restituisce un codice normalizzato e localizzato secondo il nome definibile tramite la localizzazione fornita. Il codice locale restituito viene formattato per poter essere utilizzato con `setlocale()`. Se la normalizzazione fallisce, viene restituito inalterato il nome originale.

Se la codifica fornita non è conosciuta, la funzione imposta il valore a quello della codifica predefinita per la localizzazione, esattamente come per `setlocale()`. Nuovo nella versione 2.0.

`resetlocale`([*category*])
Imposta la localizzazione di *category* al valore predefinito.

L'impostazione predefinita è determinata dalla chiamata a `getdefaultlocale()`. Il valore predefinito per *category* è `LC_ALL`. Nuovo nella versione 2.0.

`strcoll`(*string1*, *string2*)
Confronta due stringhe secondo l'impostazione corrente di `LC_COLLATE`. Come ogni altra funzione di confronto, restituisce un valore negativo, positivo, o `0`, a seconda che *string1* sia collazionata prima o dopo *string2*, o che sia uguale ad essa.

`strxfrm`(*string*)
Trasforma una stringa in un'altra stringa che puossa veniore utilizzata dalla funzione built-in `cmp()`, e restituisce sempre dei risultati informativi sulla localizzazione. Questa funzione può venire usata quando la stessa stringa viene confrontata ripetutamente, ad esempio quando viene collazionata una sequenza di stringhe.

`format`(*format*, *val*[, *grouping*])
Formatta un numero *val* secondo l'impostazione corrente di `LC_NUMERIC`. Il formato segue le convenzioni dell'operatore `%`. Per valori in virgola mobile il punto decimale, se necessario, viene modificato. Se *grouping* ha valore vero, prende anche in considerazione il grouping nel risultato.

str(*float*)
Formatta un numero in virgola mobile usando lo stesso formato della funzione built-in `str(`*float*`)`, ma prende in considerazione il punto decimale nel risultato.

atof(*string*)
Converte una stringa in un numero in virgola mobile, seguendo le impostazioni di `LC_NUMERIC`.

atoi(*string*)
Converte una stringa in un intero, seguendo le convenzioni di `LC_NUMERIC`.

LC_CTYPE
Categoria di localizzazione per le funzioni di tipo carattere. A seconda delle impostazioni di questa categoria, le funzioni relazionate al modulo `string`, cambiano il loro comportamento.

LC_COLLATE
Categoria di localizzazione utilizzata per l'ordinamento delle stringhe. Afferisce le funzioni `strcoll()` e `strxfrm()` del modulo `locale`.

LC_TIME
Categoria di localizzazione per la formattazione del tempo. La funzione `time.strftime()` segue queste convenzioni.

LC_MONETARY
Categoria di localizzazione per la formattazione dei valori monetari. Le opzioni vengono rese disponibili dalla funzione `localeconv()`.

LC_MESSAGES
Categoria di localizzazione per la visualizzazione dei messaggi. Python attualmente non supporta messaggi specifici sulle informazioni di localizzazione. I messaggi mostrati dal sistema operativo, come quelli restituiti da `os.strerror()` possono venire influenzati da questa categoria.

LC_NUMERIC
Categoria di localizzazione per la formattazione dei numeri. Le funzioni `format()`, `atoi()`, `atof()` e `str()` del modulo `locale` possono venire influenzate da questa categoria. Tutte le altre operazioni di formattazione numeriche non ne sono interessate.

LC_ALL
Combinazione di tutte le impostazioni di localizzazione. Se questa opzione viene usata quando la localizzazione viene modificata, viene tentata la modifica della localizzazione per tutte le categorie. Se ciò fallisce per ogni categoria, allora non ne viene modificata nessuna. Quando la localizzazione viene richiamata usando questa opzione, viene restituita una stringa indicante le impostazioni di tutte le categorie. Questa stringa può venire usata successivamente per ripristinare le impostazioni.

CHAR_MAX
Questa è una costante simbolica usata per differenti valori restituiti da `localeconv()`.

La funzione `nl_langinfo` accetta una delle seguenti chiavi. Molte di queste descrizioni sono tratte dalle corrispondenti descrizioni della libreria GNU C.

CODESET
Restituisce una stringa con il nome del carattere codificato nella localizzazione selezionata.

D_T_FMT
Restituisce una stringa che può venire usata come stringa di formattazione per strftime(3), per rappresentare l'ora e la data nel modo indicato dalla specifica localizzazione.

D_FMT
Restituisce una stringa che può venire usata come stringa di formattazione per strftime(3), per rappresentare la data nel modo indicato dalla specifica localizzazione.

T_FMT
Restituisce una stringa che può venire usata come stringa di formattazione per strftime(3), per rappresentare l'ora nel modo indicato dalla specifica localizzazione.

T_FMT_AMPM
Il valore restituito può venire usato come stringa di formattazione per 'strftime', per rappresentare l'ora nel

formato am/pm.

DAY_1 ... DAY_7
Restituisce il nome del giorno n-esimo della settimana. **Avvertenze:** Questo segue la convenzione americana di `DAY_1` che inizia con la Domenica, non la convenzione internazionale (ISO 8601) in cui il primo giorno della settimana è il Lunedì.

ABDAY_1 ... ABDAY_7
Restituisce il nome abbreviato del giorno n-esimo della settimana.

MON_1 ... MON_12
Restituisce il nome del mese n-esimo.

ABMON_1 ... ABMON_12
Restituisce il nome abbreviato del mese n-esimo.

RADIXCHAR
Restituisce un carattere radice (punto decimale, virgola decimale, ecc).

THOUSEP
Restituisce il crattere separatore per le migliaia (gruppi di tre cifre).

YESEXPR
Restituisce una espressione regolare che può venire usata con la funzione regex per individuare un responso positivo ad una domanda di tipo sì/no. **Avvertenze:** L'espressione è nella sintassi adatta alla funzione `regex()` della libreria C, che può differire dalla sintassi usata in `re`.

NOEXPR
Restituisce una espressione regolare che può venire usata con la funzione regex(3) per individuare un responso negativo ad una domanda di tipo sì/no.

CRNCYSTR
Restituisce il simbolo di valuta corrente, prceduto da - se il simbolo dovrebbe apparire prima del valore, + se il simbolo dovrebbe apparire dopo il valore, o . se il simbolo dovrebbe sostituire il carattere radice.

ERA
Il valore restituito rappresenta l'era usata nella localizzazione corrente.

Molte localizzazioni non definiscono questo valore. Un esempio di localizzazione che definisce questo valore è la Giapponese. In Giappone, la rappresentazione tradizionale delle date include il nome dell'era corrispondente al regno di un determinato imperatore.

Normalmente non dovrebbe essere necessario usare direttamente questo valore. Specificando il modificatore `E` nelle stringhe di formattazione, la funzione `strftime` userà questa informazione. Il formato della stringa restituita non viene specificato, e quindi non dovreste darne per scontata la conoscenza su sistemi differenti.

ERA_YEAR
Il valore restituisce l'anno nell'era corrente della localizzazione.

ERA_D_T_FMT
Restituisce un valore che può venire usato come stringa di formattazione per la funzione `strftime`, per rappresentare nella localizzazione indicata, la data ed il tempo nel modo basato sull'era.

ERA_D_FMT
Questo restituisce un valore che può venire usato come stringa di formattazione per la funzione `strftime`, per rappresentare nella localizzazione indicata, il tempo nel basato sull'era.

ALT_DIGITS
Il valore restituito è una rappresentazione dei numeri fino a 100, usati per rappresentare i valori da 0 a 99.

Esempio:

```
>>> import locale
>>> loc = locale.setlocale(locale.LC_ALL)    #  prende la localizzazione corrente
>>> locale.setlocale(locale.LC_ALL, 'de_DE') #+ usa la localizzazione tedesca;
                                             #+ il nome potrebbe
                                             #+ variare con la piattaforma
>>> locale.strcoll('f\xe4n', 'foo')          #  confronta una stringa
                                             #+ contenente una umlaut
>>> locale.setlocale(locale.LC_ALL, '')      #  usa il localizzazione
                                             #+ definita dall'utente
>>> locale.setlocale(locale.LC_ALL, 'C')     #  usa la localizzazione predefinita (C)
>>> locale.setlocale(locale.LC_ALL, loc)     # ripristina la localizzazione memorizzata
```

6.26.1 Ambiente, dettagli, suggerimenti, trucchi ed avvertimenti

Lo standard C definisce la localizzazione come una proprietà del programma che può essere relativamente difficile da cambiare. Oltre a questo, alcune implementazioni vengono rovinate in modo tale che freqenti cambiamenti della localizzazione possono causare dei core dump. Tutto questo rende la localizzazione qualcosa di particolarmente complicato per poter essere usato correttamente.

Inizialmente , quando un programma viene avviato, la localizzazione corrisponde alla localizzazione '`C`', indipendentemente da quella definita dall'utente. Il programma deve esplicitamente dire che vuole utilizzare le localizzazioni impostate dall'utente, attraverso una chiamata a `setlocale(LC_ALL, "")`.

È generalmente una cattiva idea chiamare `setlocale()` in qualche routine di libreria, poiché l'effetto collaterale sarebbe quello di influenzare l'intero programma. Salvare e ripristinare la localizzazione è un male quasi identico, in quanto è dispendioso ed influenza altri thread che potrebbero essere mandati in esecuzione prima che le impostazioni vengano ripristinate.

Se, usando un modulo di codifica per uso generale, avete bisogno di una versione di localizzazione indipendente da una operazione che è stata interessata dalla stessa localizzazione (come `string.lower()`, o certi formati usati con `time.strftime()`), dovreste avere trovato un modo per fare questo senza usare le routine delle libreria standard. Sarebbe ancora meglio se vi convinceste che usare le localizzazioni impostate è giusto. Solo come ultima risorsa dovreste documentare che il vostro modulo non è compatibile con le impostazioni di localizzazione non '`C`'.

La funzioni di conversione delle maiuscole nel modulo `string` vengono influenzate dalle impostazioni della localizzazione. Quando una chiamata alla funzione `setlocale()` cambia le impostazioni di `LC_CTYPE`, le variabili `string.lowercase`, `string.uppercase` e `string.letters` vengono ricalcolate. Notate che questo codice che usa queste variabili attraverso '`from` ... `import` ...', per esempio `from string import letters`, non viene influenzato dalle chiamate successive a `setlocale()`.

L'unico modo per utilizzare operazioni numeriche secondo la localizzazione è quello di utilizzare le funzioni speciali definite da questo modulo: `atof()`, `atoi()`, `format()`, `str()`.

6.26.2 Per coloro che scrivono estensioni e programmi che includono Python

I moduli di estensione non dovrebbero mai chiamare `setlocale()`, se non per individuare la localizzazione corrente. Ma poiché il valore restituito può venire usato in modo portabile solo per ripristinarlo, questo non è molto utile (eccetto forse che per capire se la localizzazione è '`C`' oppure no).

Quando Python è incluso in una applicazione, se l'applicazione imposta la localizzazione a qualcosa di specifico prima di inizializzare Python, questo generalmente va bene, e Python userà qualsiasi localizzazione impostata, *ad eccezione* di `LC_NUMERIC` che dovrebbe essere sempre impostata a '`C`'.

La funzione `setlocale()` nel modulo `locale` dà al programmatore Python l'impressione che possa manipolare le impostazioni locali di `LC_NUMERIC`, ma questo non è il caso a livello di C: il codice C troverà sempre che l'impostazione locale di `LC_NUMERIC` è posta a '`C`'. Questo perché troppe cose verrebbero rovinate qualora il carattere di punto decimale venisse impostato su qualcosa di diverso che non sia il punto (per esempio il parser di Python si interromperebbe). **Avvertenze:** I thread che vengono eseguiti senza tenere conto del lock globa-

le dell'interprete Python, possono occasionalmente trovare delle differenze nelle impostazioni numeriche della localizzazione; questo perché l'unico modo portabile per implementare tale caratteristica è quello di impostare la localizzazione numerica sulle impostazioni richieste dall'utente, estraendo le caratteristiche rilevanti, e quindi ripristinando la localizzazione numerica a '`C`'.

Quando il codice Python usa il modulo `locale` per cambiare la localizzazione, vengono influenzate anche le applicazioni incorporate. Se l'applicazione incorporata non vuole che questo accada, dovrebbe rimuovere il modulo di estensione `_locale` (che in pratica fa tutto il lavoro) dalla tavola dei moduli built-in nel file 'config.c', e accertandosi che il modulo `_locale` non sia accessibile dalle librerie condivise.

6.26.3 Accesso al cataloghi dei messaggi

Il modulo locale presenta l'interfaccia Gettext della libreria del C su quei sistemi che forniscono questa interfaccia. Essa si compone delle funzioni `gettext()`, `dgettext()`, `dcgettext()`, `textdomain()` e `bindtextdomain()`. Queste sono simili alle stesse funzioni nel modulo `gettext`, ma usano i binari della libreria C per i cataloghi dei messaggi, e gli algoritmi di ricerca della libreria C per la ricerca nei messaggi dei cataloghi.

Le applicazioni Python non dovrebbero normalmente avere bisogno di invocare queste funzioni, e dovrebbero usare al loro posto `gettext`. Una eccezione conosciuta a questa regola è costituita da quelle applicazioni i cui collegamenti usano le librerie addizionali C, che internamente invocano `gettext()` o `cdgettext()`. Per queste applicazioni, potrebbe essere necessario collegarle al dominio del testo, così che le librerie possano propriamente individuare i propri cataloghi dei messaggi.

6.27 `gettext` — Servizio di internazionalizzazione multilingua

Il modulo `gettext` fornisce i servizi di internazionalizzazione (I18N) e localizzazione (L10N) per i vostri moduli ed applicazioni Python. Supporta sia i messaggi GNU `gettext` del catalogo API e, a più alto livello, una classe basata su API che può essere maggiormente appropriata per i file Python. L'interfaccia descritta di seguito vi permetterà di scrivere i messaggi dei vostri moduli e delle vostre applicazioni in un linguaggio naturale, e fornisce un catalogo di messaggi tradotti per poter funzionare sotto differenti linguaggi naturali.

Vengono forniti anche alcuni suggerimenti sulla localizzazione dei vostri moduli o programmi Python.

6.27.1 API GNU **gettext**

Il modulo `gettext` definisce la seguente API, che è molto simile alla API GNU **gettext**. Se usate questa API i suoi effetti interesseranno la traduzione della vostra applicazione globalmente. Spesso è ciò che desiderate se la vostra applicazione è monolingua, con la scelta della localizzazione dipendente da quella del vostro utente. Se state localizzando un modulo Python, o se la vostra applicazione ha bisogno di cambiare linguaggio al volo, probabilmente vorrete usare, al suo posto, la classe basata su API.

`bindtextdomain`(*domain*[, *localedir*])

Collega il dominio *domain* alla directory locale *localedir*. Più concretamente, `gettext` cercherà i file binari '.mo' per il dominio fornito, usando il percorso (su UNIX): '*localedir*/*language*/LC_MESSAGES/*domain*.mo', dove *languages* viene cercato rispettivamente nelle variabili d'ambiente LANGUAGE, LC_ALL, LC_MESSAGES e LANG.

Se *localedir* viene omesso o è `None`, allora viene restituito il corrente collegamento per il dominio *domain*.[2]

`textdomain`([*domain*])

Cambia o interroga il corrente dominio globale. Se *domain* è `None`, viene restituito il corrente dominio globale, altrimenti questo viene impostato e restituito a *domain*.

[2] La directory predefinita contenente la localizzazione è dipendente dal sistema; per esempio,su RedHat Linux è '/usr/share/locale', ma su Solaris è '/usr/lib/locale'. Il modulo `gettext` non cerca di supportare queste impostazioni dipendenti dal sistema, ma mantiene la propria impostazione predefinita in '`sys.prefix`/share/locale'. Per questo motivo, è sempre meglio chiamare `bindtextdomain()` con il suo percorso assoluto, all'inizio della vostra applicazione.

gettext(*message*)
Restituisce la traduzione localizzata del messaggio *message*, basata sui correnti dominio globale, linguaggio e directory di localizzazione. Questa funzione viene di solito usata come _ nello spazio dei nomi locale (vedete gli esempi a seguito).

dgettext(*domain, message*)
Come `gettext()`, ma cerca il messaggio nel dominio *domain* specificato.

ngettext(*singular, plural, n*)
Come `gettext()`, ma considera le forme plurali. Se viene trovata una traduzione, applica la formula plurale a *n*, e restituisce il messaggio risultante (alcuni linguaggi hanno più di due forme plurali). Se non viene rinvenuta alcuna traduzione, restituisce *singular* (NdT: singolare) se *n* è 1; altrimenti restituisce *plural* (NdT: plurale).

La formula Plurale viene presa dall'header del catalogo. È un'espressione in C o in Python che ha una variabile libera n; l'espressione valuta l'indice del plurale nel catalogo. Vedete la documentazione di GNU gettext per la sintassi esatta da usare nei file .po, e le formule per i vari linguaggi.

Nuovo nella versione 2.3.

dngettext(*domain, singular, plural, n*)
Come `ngettext()`, ma cerca il messaggio nel dominio *domain* specificato.

Nuovo nella versione 2.3.

Notate che GNU **gettext** definisce anche un metodo `dcgettext()`, ma questo è stato definito inutile e quindi non più implementato.

Ecco un esempio di uso tipico di questa API:

```
import gettext
gettext.bindtextdomain('myapplication', '/path/to/my/language/directory')
gettext.textdomain('myapplication')
_ = gettext.gettext
# ...
print _('This is a translatable string.')
```

6.27.2 Classe basata su API

La classe basata su API del modulo `gettext` vi dà maggiore flessibiltà e risulta moplto più utile dell'API di GNU **gettext** GNU. Questo è il modo raccomandato per la localizzazione delle vostre applicazioni e moduli Python. `gettext` definisce una classe "translations" che implementa l'analisi del formato dei file GNU '.mo', e ha dei metodi per restituire sia stringhe standard a 8 bit che stringhe Unicode. Le istanze di traduzione possono anche installare se stesse nello spazio dei nomi built-in come la funzione `_()`.

find(*domain*[, *localedir*[, *languages*[, *all*]]])
Questa funzione implementa l'algoritmo standard di ricerca dei file '.mo'. Prende il dominio *domain* identico a quello preso da `textdomain()`. Il valore opzionale di *localedir* è identico a quello della funzione `bindtextdomain()`. L'opzione *languages* è una lista di stringhe, in cui ogni stringa è un codice di linguaggio.

Se non viene passato *localedir*, verrà usata la directory predefinita del locale di sistema.[3] Se non viene passato *languages*, verranno ricercate le seguenti variabili d'ambiente: LANGUAGE, LC_ALL, LC_MESSAGES e LANG. La prima che restituisce un valore non vuoto, viene utilizzata per la variabile *languages*. Le variabili d'ambiente dovrebbero contenere un elenco di linguaggi separati da due punti, che dovrebbero essere separati da quel carattere proprio per produrre la lista di stringhe del codice del linguaggio corrispondente.

Quindi `find()` espande e normalizza i linguaggi, e itera su di essi, cercando un file esistente costituito da questi componenti:

'*localedir*/*language*/LC_MESSAGES/*domain*.mo'

[3] Vedete le note a piè di pagina per `bindtextdomain()`.

Il primo nome di file esistente viene restituito da `find()`. Se non viene rinvenuto alcun file, viene restituito `None`. Se viene passato *all*, viene restituito un elenco di tutti i nomi dei file, nell'ordine del quale compare la lista dei linguaggi o delle variabili d'ambiente.

`translation`(*domain*[, *localedir*[, *languages*[, *class_*,[*fallback*]]]])
Restituisce un'istanza di `Translations` basata su *domain*, *localedir* e *languages*, che vengono prima passati a `find()` per acquisire la lista dei percorsi dei file '.mo' associati. Istanze con nomi di file '.mo' identici vengono sovrascritte. L'attuale classe istanziata è una classe *class_* se fornita, altrimenti è del tipo `GNUTranslations`. La classe del costruttore deve prendere un singolo file oggetto come argomento.

Se vengono rinvenuti file multipli, gli ultimi file vengono usati per restituire i primi. Per permettere l'impostazione della restituzione, viene usata `copy.copy`, per clonare ogni oggetto traduzione dalla cache; l'attuale istanza di dati viene ancora condivisa con la cache.

Se non viene rinvenuto alcun file '.mo', questa funzione solleva un'eccezione `IOError` se *fallback* è falso (predefinito), e restituisce una istanza `NullTranslations` se *fallback* è vero.

`install`(*domain*[, *localedir*[, *unicode*]])
Installa la funzione _ nello spazio dei nomi built-in di Python, basata su *domain* e *localedir*, che sono stati passati per la funzione `translation()`. L'opzione *unicode* viene passata al risultante oggetto traduzione dal metodo `install`.

Come vedrete sotto, di solito vengono marcate le stringhe delle vostre applicazioni che sono candidate alla traduzione, inserendole all'interno di una chiamata alla funzione `_()`, come questa:

```
print _('Questa stringa deve essere tradotta.')
```

Per comodità, vorrete che la funzione `_()` venga installata nello spazio dei nomi built-in di Python, così da renderla più facilmente accessibile da tutti i moduli delle vostre applicazioni.

La classe `NullTranslations`

Le classi Translation sono quelle che attualmente implementano la traduzione dei messaggi dei file sorgenti originali in stringhe di messaggi tradotte. La classe di base usata da tutte le classi di traduzione è la `NullTranslations`; questa fornisce l'interfaccia base che potrete usare per scrivere le vostre classi di traduzione specializzate. Questi sono i metodi di `NullTranslations`:

`__init__`([*fp*])
Prende un file oggetto facoltativo *fp*, che viene ignorato dalla classe base. Inizializza istanze di variabili "protette" *_info* e *_charset*, che vengono impostate da classi derivate, come *_fallback*, che viene impostata attraverso `add_fallback`. Quindi chiama `self._parse(fp)` se *fp* non è `None`.

`_parse`(*fp*)
Non ci sono opzioni nella classe base, questo metodo prende il file oggetto *fp* e legge i dati dal file, inizializzando il suo catalogo dei messaggi. Se avete un file contenente il catalogo dei messaggi in un formato non supportato, dovreste sovrascrivere questo metodo per analizzare il vostro formato.

`add_fallback`(*fallback*)
Aggiunge *fallback* come oggetto restituito per il corrente oggetto traduzione. Un oggetto traduzione dovrebbe consultare il fallback se esso non fornisce una traduzione per il messaggio fornito.

`gettext`(*message*)
Se un fallback è stato impostato, trasmette `gettext` perché venga restituito. Altrimenti, restituisce il messaggio tradotto. Sovrascritto nelle classi derivate.

`ugettext`(*message*)
Se un fallback è stato impostato, trasmette `ugettext` perché venga restituito. Altrimenti, restituisce il messaggio tradotto come una stringa Unicode. Sovrascritto nelle classi derivate.

`ngettext`(*singular, plural, n*)
Se un fallback è stato impostato, trasmette `ngettext` perché venga restituito. Altrimenti, restituisce il messaggio tradotto. Sovrascritto nelle classi derivate.

Nuovo nella versione 2.3.

ungettext(*singular, plural, n*)
Se un fallback è stato impostato, trasmette `ungettext` perché venga restituito. Altrimenti, restituisce il messaggio tradotto come una stringa Unicode. Sovrascritto nelle classi derivate.

Nuovo nella versione 2.3.

info()
Restituisce la variabile "protetta" `_info`.

charset()
Restituisce la variabile "protetta" `_charset`.

install([*unicode*])
Se l'opzione *unicode* risulta falso, questo metodo installa `self.gettext()` nello spazio dei nomi built-in, collegandolo a '_'. Se *unicode* risulta vero, esso collega invece `self.gettext()`. In modo predefinito, *unicode* è falso.

Notate che questo è solo uno dei modi, anche se quello più conveniente, per rendere la funzione _ disponibile a tutte le vostre applicazioni. Poiché influenza in modo globale tutta l'applicazione, e specificatamente lo spazio dei nomi built-in, i moduli localizzati non dovrebbero mai installare _. Al suo posto dovrebbero invece usare questo codice per rendere _ disponibile ai loro moduli:

```
import gettext
t = gettext.translation('mymodule', ...)
_ = t.gettext
```

Questo mette soltanto _ nel modulo dello spazio dei nomi globale, influenzando così soltanto le chiamate all'interno di questo modulo.

La classe `GNUTranslations`

Il modulo `gettext` fornisce un'ulteriore classe derivata da `NullTranslations`: `GNUTranslations`. Questa classe sovrascrive il metodo `_parse()` per abilitare la lettura dei file GNU **gettext** con formato '.mo', sia in formato big-endian che little-endian. Inoltre forza sia gli ids che le stringhe dei messaggi a Unicode.

`GNUTranslations` analizza i meta-dati opzionali esterni al catalogo delle traduzioni. Questa è una convenzione con GNU **gettext** per includere i meta-dati come traduzioni di stringhe vuote. Questo meta-dato è quello dello standard definito da RFC 822, coppie `chiave: valore`, e dovrebbe contenere la chiave `Progetto-Id-Versione`. Se viene individuata la chiave `Content-Type`, allora la proprietà del `charset` viene utilizzata per inizializzare la variabile istanza "protetta" `_charset`, preimpostandola a `None` se non viene trovata. Se la codifica dell'insieme dei caratteri viene specificata, allora tutti gli ids e le stringhe messaggio vengono convertiti in Unicode secondo questa codifica. Il metodo `ugettext()` restituisce sempre uno Unicode, mentre il metodo `gettext()` restituisce una stringa codificata a 8 bit. Vengono accettati solo caratteri US-ASCII per gli argomenti degli id messaggio di entrambi i metodi, per le stringhe Unicode o le stringhe a 8-bit. Notate che la versione Unicode dei metodi (per esempio, `ugettext()` e `ungettext()`) sono le interfacce raccomandate per usare programmi Python internazionalizzati.

L'intero insieme di coppie chiave/valore viene posizionato in un dizionario ed impostato come variabile "protetta" dell'istanza `_info`.

Se il magic number del file '.mo' non è valido, o se altri problemi si verificano intervenuti durante la lettura del file, l'instanziazione della classe `GNUTranslations` può sollevare l'eccezione `IOError`.

I seguenti metodi vengono sovrascritti dall'implementazione della classe base:

gettext(*message*)
Cerca l'id del messaggio *message* nel catalogo e restituisce la corrispondente stringa messaggio, come stringa a 8 bit codificata con il carattere di codifica del messaggio, se noto. Se non c'è nessuna voce nel catalogo per l'id di *message* ed è stato impostato un fallback, la ricerca viene trasmessa al metodo fallback di `gettext()`. Altrimenti viene restituito l'id di *message*.

ugettext(*message*)
Cerca l'id di *message* nel catalogo e restituisce la corrispondente stringa messaggio, come stringa Unicode.

Se non c'è nessuna voce nel catalogo per l'id di *message*, ed è stato impostato un fallback, la ricerca viene trasmessa al metodo fallback di `ugettext()`. Altrimenti viene restituito l'id di *message*.

`ngettext`(*singular, plural, n*)
Effettua una ricerca in forma plurale dell'id di un messaggio. Il *singular* viene usato come id del messaggio per determinare quale forma plurale usare. La stringa messaggio restituita è una stringa codificata a 8-bit con la codifica del catalogo, se conosciuta.

Se l'id del messaggio non viene trovato nel catalogo ed è stato specificato un fallback, la richiesta viene trasmessa al metodo fallback di `ngettext()`. Altrimenti, quando *n* è 1 viene restituito *singular*, in tutti gli altri casi viene restituito *plural*.

Nuovo nella versione 2.3.

`ungettext`(*singular, plural, n*)
Effettua una ricerca nella forma plurale di un id di messaggio. *singular* viene usato come id del messaggio a scopo di ricerca nel catalogo, mentre *n* viene usato per determinare quale forma plurale usare. Il messaggio restituito è una stringa Unicode.

Se l'id del messaggio non viene trovato nel catalogo ed è stato specificato un fallback, la richiesta viene trasmessa al metodo fallback di `ungettext()`. Altrimenti, quando *n* è 1 viene restituito *singular*, in tutti gli altri casi viene restituito *plural*.

Ecco un esempio:

```
n = len(os.listdir('.'))
cat = GNUTranslations(somefile)
message = cat.ungettext(
    'There is %(num)d file in this directory',
    'There are %(num)d files in this directory',
    n) % {'n': n}
```

Nuovo nella versione 2.3.

Supporto del catalogo dei messaggi Solaris

Il sistemo operativo Solaris definisce un proprio formato dei file binari '.mo', ma visto che non può essere trovata documentazione su questo formato, attualmente non viene supportato.

Il costruttore del catalogo

GNOME usa una versione del modulo `gettext` di James Henstridge, ma questa versione ha una API leggermente differente. L'uso documentato era:

```
import gettext
cat = gettext.Catalog(domain, localedir)
_ = cat.gettext
print _('hello world')
```

Per compatibilità con questo modulo più vecchio, la funzione `Catalog()` è un alias alla funzione `translation()` descritta precedentemente.

Una differenza tra questo modulo e quello di Henstridge: il suo catalogo oggetti supporta l'accesso attraverso una API mappata, ma questo sembra essere inutilizzato, e quindi non viene correntemente supportato.

6.27.3 Internazionalizzazione dei vostri moduli e programmi

L'internalizzazione (I18N) si riferisce all'operazione con la quale si realizza un programma che supporti molteplici linguaggi. La localizzazione (L10N) fa riferimento all'adattamento del vostro programma, una volta che è stato

internazionalizzato, per il linguaggio e per le abitudini culturali locali. Per fornire i messaggi multilingue per i vostri programmi in Python, dovrete seguire i seguenti passi:

1. preparate il vostro programma o modulo con i contrassegni sulle stringhe da tradurre
2. eseguite un insieme di programmi sui vostri file contrassegnati per generare i cataloghi grezzi dei messaggi
3. create specifiche traduzioni nei linguaggi dei messaggi dei cataloghi
4. Usate il modulo `gettext` in modo che le stringhe dei messaggi vengano tradotte correttamente

Per preparare il vostro codice per I18N, avete bisogno di cercare tutte le stringhe nei vostri files. Ogni stringa che necessita di essere tradotta dovrebbe essere contrassegnata in una cornice `_('...')` — cioè una chiamata alla funzione `_()`. Per esempio:

```
filename = 'mylog.txt'
message = _('writing a log message')
fp = open(filename, 'w')
fp.write(message)
fp.close()
```

In questo esempio, la stringa `'writing a log message'` viene marcata come candidata alla traduzione, mentre le stringhe `'mylog.txt'` e `'w'` non lo sono.

La distribuzione Python viene fornita di due strumenti che vi aiutano a generare i cataloghi dei messaggi una volta che avrete preparato il vostro codice sorgente. Queste potrebbero essere disponibili o meno in una distribuzione binaria, ma possono essere trovate nella distribuzione sorgente, nella directory 'Tools/i18n'.

Il programma **pygettext**[4] scansiona tutto il vostro codice sorgente Python, cercando le stringhe che voi avrete precedentemente contrassegnato come traducibili. È simile al programma GNU **gettext** tranne per il fatto che comprende tutte le complessità del codice sorgente Python, ma non sa niente del codice sorgente C o C++. Non avrete bisogno di GNU **gettext** finché non vi troverete a dover tradurre codice C (come i moduli estensione in C).

pygettext genera file testuali di catalogo dei messaggi '.pot' in stile Uniforum, essenzialmente file umanamente comprensibili, che contengono ogni stringa contrassegnata nel codice sorgente, con un marcatore per le stringhe di traduzione. **pygettext** è uno script a riga di comando che supporta un'interfaccia a riga di comando simile a **xgettext**; per i dettagli sul suo uso, eseguite:

```
pygettext.py --help
```

Copie di questi files '.pot' vengono gestite dai diversi traduttori, che scrivono versioni specifiche per la loro lingua. Loro rimandano indietro la versione specifica come un file '.po'. Usando il programma **msgfmt.py**[5] (nella directory 'Tools/i18n'), prendete i file '.po' dal vostro traduttore e generate dei file di catalogo binari in formato '.mo', leggibili dalla macchina. I file '.mo' sono quelli che il modulo `gettext` usa attualmente per il processo di traduzione in run-time.

A seconda se state internazionalizzando una vostra intera applicazione o un singolo modulo, potete usare il modulo `gettext` nel vostro codice.

Localizzazione dei vostri moduli

Se state localizzando i vostri moduli, dovreste fare attenzione a non fare cambiamenti globali, ad esempio sullo spazio dei nomi built-in. Non dovreste usare l'API di GNU `gettext`, ma piuttosto l'API basata sulle classi.

[4] François Pinard ha scritto un programma chiamato **xpot** che fa un lavoro simile. È disponibile come parte del suo pacchetto **po-utils** presso http://www.iro.umontreal.ca/contrib/po-utils/HTML/.

[5] **msgfmt.py** è un binario compatibile con GNU **msgfmt**, eccetto per il fatto che fornisce una più semplice implementazione, interamente Python. Con questo e con **pygettext.py**, generalmente non dovreste avere bisogno di installare il pacchetto GNU **gettext** per internazionalizzare le vostre applicazioni Python.

Supponiamo che il vostro modulo sia chiamato “spam” ed i vari file ‘.mo’ di traduzione naturale del linguaggio risiedano in ‘/usr/share/locale’ e siano nel formato GNU **gettext**. Ecco quello che dovreste scrivere all’inizio del vostro modulo:

```
import gettext
t = gettext.translation('spam', '/usr/share/locale')
_ = t.gettext
```

Se i vostri traduttori dovessero fornirvi stringhe Unicode nei loro file ‘.po’ dovreste invece scrivere:

```
import gettext
t = gettext.translation('spam', '/usr/share/locale')
_ = t.ugettext
```

Localizzazione della vostra applicazione

Se state localizzando la vostra applicazione, potete installare globalmente la funzione `_()` nello spazio dei nomi built-in, solitamente nel file principale della vostra applicazione. Questo vi permetterà di usare tutti i vostri file specifici per le applicazioni, usando solamente `_('...')`, senza averli esplicitamente installati in ogni file.

Nel caso più semplice quindi, avrete bisogno di aggiungere solamente la seguente piccola porzione di codice al file principale della vostra applicazione:

```
import gettext
gettext.install('myapplication')
```

Se avete bisogno di impostare la directory locale o l’opzione *unicode*, potete passare questi nella funzione `install()`:

```
import gettext
gettext.install('myapplication', '/usr/share/locale', unicode=1)
```

Cambiare linguaggio al volo

Se il vostro programma necessita del supporto per molti linguaggi contemporaneamente, potreste voler creare istanze di traduzioni multiple e quindi passare da una all’altra esplicitamente, così:

```
import gettext

lang1 = gettext.translation(languages=['en'])
lang2 = gettext.translation(languages=['fr'])
lang3 = gettext.translation(languages=['de'])

# inizia usando il linguaggio 1
lang1.install()

# ... il tempo passa, l'utente seleziona il linguaggio 2
lang2.install()

# ... passa più tempo, l'utente seleziona il linguaggio 3
lang3.install()
```

Traduzioni differite

Nella maggior parte dei casi, durante la stesura del codice le stringhe vengono tradotte nel punto in cui sono state codificate. Occasionalmente tuttavia, avrete bisogno di marcare le stringhe per la traduzione, ma rinviare la traduzione ad un momento successivo. Un classico esempio è:

```
animals = [’mollusk’,
           ’albatross’,
   ’rat’,
   ’penguin’,
   ’python’,
   ]
# ...
for a in animals:
    print a
```

Qui, volete contrassegnare le stringhe nella lista `animals` perché siano tradotte, ma non volete tradurle fino a che non vengano stampate.

Ecco un modo per gestire questa situazione:

```
def _(message): return message

animals = [_(’mollusk’),
           _(’albatross’),
   _(’rat’),
   _(’penguin’),
   _(’python’),
   ]

del _

# ...
for a in animals:
    print _(a)
```

Questo funziona perché la definizione di prova di `_()` restituisce semplicemante la stringa immutata. Inoltre la definizione di prova sovrascriverà temporaneamente ogni definizione di `_()` nello spazio dei nomi built-in (fino al comando `del`). Fate attenzione e verificate di non avere una precedente definizione di _ nello spazio dei nomi locale.

Notate che il secondo uso di `_()` non identifica “a” come traducibile dal programma **pygettext**, perchè non è una stringa.

Un altro moetodo per gestire questa situazione è rappresentato dal seguente esempio:

```
def N_(message): return message

animals = [N_(’mollusk’),
           N_(’albatross’),
   N_(’rat’),
   N_(’penguin’),
   N_(’python’),
   ]

# ...
for a in animals:
    print _(a)
```

In questo caso, state contrassegnando come traducibili stringhe marcate con la funzione `N_()`,[6] che non andrà in conflitto con alcuna definizione di `_()`. Tuttavia avrete bisogno di istruire il vostro programma a cercare le stringhe marcate con `N_()`. Sia **pygettext** che **xpot** supportano questa funzionalità attraverso l'uso di switch da riga di comando.

6.27.4 Riconoscimenti

Le seguenti persone hanno contribuito alla stesura del codice, resoconti, suggerimenti per l'architettura, precedenti implementazioni e preziose esperienze la creazione di questo modulo:

- Peter Funk
- James Henstridge
- Juan David Ibáñez Palomar
- Marc-André Lemburg
- Martin von Löwis
- François Pinard
- Barry Warsaw

6.28 `logging` — Servizio di logging per Python

Nuovo nella versione 2.3. Questo modulo definisce le funzioni e le classi che implementano un sistema di logging flessibile per gli errori nelle applicazioni.

Il logging viene effettuato tramite la chiamata di metodi sulle istanze della classe `Logger` (da qui in poi chiamati *logger*). Ciascuna delle istanze ha un nome, e vengono concettualmente organizzata in uno spazio dei nomi gerarchizzato, attraverso l'uso dei punti (dots) come separatori. Per esempio, un logger chiamato scan è il padre dei logger scan.text, scan.html e scan.pdf. I logger possono assumere qualsiasi nome si voglia, e indicano l'area di una applicazione nella quale il messaggio di log viene generato.

I messaggi di log hanno anche un livello di importanza associato ad essi. I livelli predefiniti forniti sono `DEBUG`, `INFO`, `WARNING`, `ERROR` e `CRITICAL`. Per convenienza, viene indicata l'importanza di un messaggio registrato sui log chiamando un metodo appropriato di `Logger`. I metodi sono `debug()`, `info()`, `warning()`, `error()` e `critical()`, che duplicano i livelli predefiniti. Non siete costretti ad usare questi livelli: potete specificarne di propri ed usare il metodo più generale `Logger`, `log()`, che prende un esplicito argomento di livello.

I livelli possono anche essere associati ai logger, essendo impostabili sia direttamente dal programmatore, che attraverso il caricamento di una configurazione di preimpostata. Quando un metodo di logging viene chiamato su di un logger, il logger confronta il proprio livello con quello associato alla chiamata di metodo. Se il livello del logger è più alto di quello della chiamata di metodo, non viene generato nessun messaggio di log. Questo è il meccanismo di base che controlla la prolissità dei messaggi di log generati.

I messaggi di logging vengono codificati come istanze della classe `LogRecord`. Quando un logger decide di registrare un evento, un'istanza `LogRecord` viene creata dal messaggio di logging.

I messaggi di logging sono soggetti ad un meccanismo di spedizione specifico attraverso l'uso degli *handlers* (NdT: gestori), che sono istanze di classi derivate della classe `Handler`. Gli handler hanno la responsabilità di garantire che i messaggi di log (nella forma `LogRecord`) vengano posizionati in una locazione (o insieme di locazioni) definita e comoda per gli utenti a cui sono destinati (come un utente finale, lo staff di supporto tecnico, l'amministratore di sistema, gli sviluppatori). Agli Handler vengono passate delle istanze `LogRecord`, intese per particolari destinazioni. Ogni logger può avere zero, uno o più handler associati ad esso (attraverso il metodo

[6] La scelta di `N_()` qui è totalmente arbitraria; potrebbe essere facilmente fatto anche con `MarkThisStringForTranslation()`.

`addHandler()` di `Logger`). In aggiunta ad ogni handler associato ad un Logger, *tutti gli handler associati con tutti i predecessori del logger* vengono chiamati a consegnare il messaggio.

Come per i logger, anche gli handler possono acere dei livelli associati. Un livello di handler si comporta come un filtro, nello stesso modo di un livello di un logger. Se un handler decide di consegnare un evento, viene utilizzato il metodo `emit()` per inviare il messaggio alla sua destinazione. La maggior parte delle classi derivate di `Handler` definite dall'utente avranno la necessità di sovrascrivere questo `emit()`.

In aggiunta alla classe base `Handler`, vengono fornite molte utili classi derivate:

1. Le istanze `StreamHandler` inviano i messaggi di errore agli stream (oggetti simil-file).
2. Le istanze `FileHandler` inviano i messaggi di errori a file su disco.
3. Le istanze `RotatingFileHandler` inviano i messaggi di errore ai file su disco, con il supporto alla massima dimensione del file di log e di rotazione dei file di log stessi.
4. Le istanze `SocketHandler` inviano i messaggi di errore su socket TCP/IP.
5. Le istanze `DatagramHandler` inviano i messaggi di errore a socket UDP.
6. Le istanze `SMTPHandler` inviano i messaggi di errore a un indirizzo email predefinito.
7. Le istanze `SysLogHandler` inviano i messaggi di errore al demone syslog di UNIX, possibilmente su una macchina remota.
8. Le istanze `NTEventLogHandler` inviano i messaggi di errore al log eventi di Windows NT/2000/XP.
9. Le istanze `MemoryHandler` inviano messaggi di errore ad un buffer in memoria, che viene svuotato non appena incontrano uno specifico criterio.
10. Le istanze `HTTPHandler` inviano i messaggi di errore a un server HTTP usando sia la semantica '`GET`' che '`POST`'.

Sia le classi `StreamHandler` che `FileHandler` vengono definite nella porzione prinicipale del package logging. Gli altri handler vengono definiti in un modulo derivato, `logging.handlers`. (Esiste anche un altro modulo derivato, `logging.config`, per funzionalità di configurazione).

I messaggi registrati nei log vengono formattatti per la presentazione attraverso istanze della classe `Formatter`. Vengono inizializzati con una stringa di formattazione in grado di essere usata con l'operatore % ed un dizionario.

Per la formattazione di messaggi multipli in un batch, possono venire utilizzate istanze di `BufferingFormatter`. In aggiunta alla stringa di formattazione (che viene applicata ad ogni messaggio nel batch), esiste una riserva di stringhe di formato per intestazioni e pié di pagina.

Quando il filtraggio basato sui livelli di logger o sui livelli di handler non è sufficiente, possono venire aggiunte istanze di `Filter`, sia alle istanze di `Logger` che a quelle di `Handler` (tramite i loro metodi `addFilter()`). Prima di decidere di elaborare ulteriormente un messaggio, sia i logger che gli handler consultano tutti i loro filtri sui permessi. Se ogni qualche restituisce un valore falso, il messaggio non viene ulteriormente elaborato.

La funzionalità base di `Filter` consente di filtrare attraverso uno specifico nome di logger. Se questa funzionalità viene utilizzata, solo i messaggi inviati al logger indicato, ed ai suoi figli, vengono autorizzati a passare attraverso i filtri, tutti gli altri vengono scartati.

In aggiunta alle classi descritte precedentemente, esistono anche un certo numero di funzioni a livello di modulo.

`getLogger``([`*name*`])`

Restituisce un logger con lo specifico nome o, se nessun nome viene specificato, restituisce un logger che è il logger principale della gerarchia.

Tutte le chiamate a questa funzione con un dato *name*, restituiscono la stessa istanza di logger. Questo significa che l'istanza di logger non ha mai bisogno di essere passsata attraverso parti differenti di un'applicazione.

debug(*msg*[, **args*[, ***kwargs*]])
Registra un messaggio con il livello `DEBUG` nel logger principale. Il messaggio *msg* è la stringa di formato del messaggio, e gli argomenti *args* sono quelli che vengono inseriti nel messaggio. L'unico argomento chiave in *kwargs*, che viene controllato, è `exc_info()`) che, se non viene valutato come falso, provoca la stampa di informazioni su un'eccezione (attraverso una chiamata a `sys.exc_info()`) che verranno aggiunte al messaggio di log.

info(*msg*[, **args*[, ***kwargs*]])
Registra un messaggio con livello `INFO` nel log principale. Gli argomenti vengono interpretati come nel caso di `debug()`.

warning(*msg*[, **args*[, ***kwargs*]])
Registra un messaggio con livello `WARNING` nel logger principale. Gli argomenti vengono interpretati come nel caso di `debug()`.

error(*msg*[, **args*[, ***kwargs*]])
Registra un messaggio con livello `ERROR` nel log principale. Gli argomenti vengono interpretati come nel caso di `debug()`.

critical(*msg*[, **args*[, ***kwargs*]])
Registra un messaggio con livello `CRITICAL` nel log principale. Gli argomenti vengono interpretati come nel caso di `debug()`.

exception(*msg*[, **args*])
Registra un messaggio con livello `ERROR` nel log principale. Gli argomenti vengono interpretati come nel caso di `debug()`. Informazioni sull'eccezione verranno aggiunte al messaggio di log. Questa funzione deve essere chiamata solo da un handler dell'eccezione.

disable(*lvl*)
Fornisce un livello di sovrascrittura *lvl* per tutti i logger che ottengono la precedenza rispetto ai logger del proprio livello. Questa funzione torna utile quando si presenta la necessità di fermare temporaneamente l'emissione di log per l'intera applicazione.

addLevelName(*lvl, levelName*)
Associa il livello *lvl* con il testo *levelName* in un dizionario interno, che viene utilizzato per mappare i livelli numerici con una rappresentazione testuale, per esempio quando un `Formatter` formatta un messaggio. Questa funzione può essere usata anche per definire propri livelli. Gli unici obblighi sono che tutti i livelli usati devono essere registrati usando questa funzione, i livelli dovrebbero essere interi positivi crescenti all'aumentare dell'ordine di accuratezza.

getLevelName(*lvl*)
Restituisce la rappresentazione testuale del livello di logging *lvl*. Se il livello è uno tra quelli predefiniti, `CRITICAL`, `ERROR`, `WARNING`, `INFO` o `DEBUG`, otterrete la stringa corrispondente. Se sono stati associati dei livelli utilizzando `addLevelName()`, allora vengono restituiti i nomi associati a *lvl*. Altrimenti vengono restituite le stringhe Level %s % lvl.

makeLogRecord(*attrdict*)
Crea e restituisce una nuova istanza `LogRecord` i cui attributi sono definiti da *attrdict*. Questa funzione è utile per prelevare un dizionario di attributi `LogRecord` serializzato, inviato attraverso un socket, e ricostruito come istanza `LogRecord` alla fine della ricezione.

basicConfig()
Esegue la configurazione di base per il sistema di logging creando uno `StreamHandler` con un `Formatter` predefinito e aggiungendolo al logger principale. Le funzioni `debug()`, `info()`, `warning()`, `error()` e `critical()` chiameranno `basicConfig()` automaticamente se nessun handler viene definito per il logger principale.

shutdown()
Comunica al sistema di logging di eseguire uno shutdown ordinato, svuotando e chiudendo tutti gli handler.

setLoggerClass(*klass*)
Comunica al sistema di Logging di utilizzare la classe *klass* mentre si istanzia un logger. La classe dovrebbe definire `__init__()` in modo tale che venga richiesto solamente un argomento name, e `__init__()` dovrebbe chiamare `Logger.__init__()`. Questa funzione di solito viene chiamata pri-

ma che ogni logger venga istanziato dalle applicazioni che hanno la necessità di utilizzare un proprio logger personalizzato.

Vedete anche:

PEP 282, "*A Logging System*"
: La proposta che descriveva questa caratteristica per includerla nella libreria standard di Python.

Il package Python `logging` originale
: Questo è il sorgente originale del package `logging`. La versione del package è disponibile da questo sito e può essere sfruttata con Python 2.1x e 2.2.x, che non includono il package `logging` nella libreria standard.

6.28.1 Oggetti Logger

I logger possiedono i seguenti attributi e metodi. Notate che i logger non vengono mai istanziati direttamente, ma sempre attraverso la funzione a livello di modulo `logging.getLogger(name)`.

`propagate`
: Se viene valutata come falsa, i messaggi di log non vengono trasmessi da questo logger o dai logger figli al livello più alto di logger (superclasse). Il costruttore imposta questo attributo ad 1.

`setLevel`(*lvl*)
: Imposta la soglia per questo logger a *lvl*. I messaggi di logging meno accurati di *lvl* verranno ignorati. Quando viene creato un logger, il livello viene impostato a `NOTSET` (che ottiene il risultato di far elaborare al logger principale tutti i messaggi, o delegare questo compito al padre del logger, non principale).

`isEnabledFor`(*lvl*)
: Indica se un messaggio con accuratezza *lvl* può essere elaborato da questo logger. Questo metodo controlla prima il livello del modulo, impostato da `logging.disable(lvl)`, e successivamente l'effettivo livello del logger, come determinato da `getEffectiveLevel()`.

`getEffectiveLevel`()
: Indica l'effettivo livello per questo logger. Viene restituito se un valore diverso da `NOTSET` viene usato con `setLevel()`. Altrimenti la gerarchia viene percorsa all'indietro fino alla cima, fintanto che non viene trovato un valore diverso di `NOTSET`. Quindi, se il valore viene trovato, viene restituito.

`debug`(*msg*[, **args*[, ***kwargs*]])
: Registra un messaggio con il livello `DEBUG` su questo logger. *msg* è una stringa in formato messaggio, e *args* sono gli argomenti che inseriti in *msg*. Il solo argomento chiave in *kwargs* che viene analizzato è *exc_info*, che, se non valutato come falso, causa un'informazione di eccezione (attraverso una chiamata a `sys.exc_info()`) da aggiungere al messaggio di log.

`info`(*msg*[, **args*[, ***kwargs*]])
: Registra un messaggio con livello `INFO` su questo logger. Gli argomenti vengono interpretati come nel caso di `debug()`.

`warning`(*msg*[, **args*[, ***kwargs*]])
: Registra un messaggio con livello `WARNING` su questo logger. Gli argomenti vengono interpretati come nel caso di `debug()`.

`error`(*msg*[, **args*[, ***kwargs*]])
: Registra un messaggio con livello `ERROR` su questo logger. Gli argomenti vengono interpretati come nel caso di `debug()`.

`critical`(*msg*[, **args*[, ***kwargs*]])
: Registra un messaggio con livello `CRITICAL` su questo logger. Gli argomenti vengono interpretati come nel caso di `debug()`.

`log`(*lvl*, *msg*[, **args*[, ***kwargs*]])
: Registra un messaggio con livello *lvl* su questo logger. Gli argomenti vengono interpretati come nel caso di `debug()`.

`exception`(*msg*[, **args*])
: Registra un messaggio con livello `ERROR` su questo logger. Gli argomenti vengono interpretati come nel

caso di `debug()`. Una informazione di eccezione viene aggiunta al messaggio di log. Questo metodo dovrebbe essere chiamato solo da un handler dell'eccezione.

`addFilter`(*filt*)
Aggiunge lo specifico filtro *filt* a questo logger.

`removeFilter`(*filt*)
Rimuove lo specifico filtro *filt* da questo logger.

`filter`(*record*)
Applica i filtri di questo logger a *record* e restituisce un valore vero se il recordo sta per essere elaborato.

`addHandler`(*hdlr*)
Aggiunge lo specifico handler *hdlr* a questo logger.

`removeHandler`(*hdlr*)
Rimuove lo specifico handler *hdlr* da questo logger.

`findCaller`()
Cerca il nome file del sorgente del chiamante ed il numero di riga. Restituisce il nome del file ed il numero di riga come una tupla a 2 elementi.

`handle`(*record*)
Gestisce un record saltando tutti gli handler associati a questo logger ed i suoi genitori (finché non viene trovato un valore falso di *propagate*). Questo metodo viene usato per record deserializzati ricevuti da un socket, così come per quelli creati localmente. Usando `filter()` viene applicato un filtraggio a livello di logger.

`makeRecord`(*name, lvl, fn, lno, msg, args, exc_info*)
Questo è un metodo factory che può essere sovrascritto in classi derivate per creare istanze specializzate di `LogRecord`.

6.28.2 Oggetti Handler

Gli Handler posseggono i seguenti attributi e metodi. Notate che `Handler` non è mai istanziata direttamente; questa classe si comporta come una base per più utili classi derivate. Comunque, il metodo `__init__()` in classi derivate ha la necessità di chiamare `Handler.__init__()`.

`__init__`(*level=NOTSET*)
Inizializza l'istanza `Handler` impostando il suo livello, impostando la lista dei filtri a una lista vuota e creando un lock (usando `createLock()`) per serializzare gli accessi ad un meccanismo di I/O.

`createLock`()
Inizializza un lock di thread che può essere usato per serializzare gli accessi alla sottostante funzionalità di I/O che potrebbe essere non threadsafe.

`acquire`()
Acquisisce il lock del thread creato con `createLock()`.

`release`()
Rilascia il lock del thread acquisito con `acquire()`.

`setLevel`(*lvl*)
Imposta il livello di questo handler a *lvl*. Messaggi di logging meno importanti di *lvl* verranno ignorati. Quando viene creato un handler, il livello viene impostato a `NOTSET` (che si occupa di elaborare tutti i messaggi).

`setFormatter`(*form*)
Imposta il `Formatter` per questo handler a *form*.

`addFilter`(*filt*)
Aggiunge lo specifico filtro *filt* a questo handler.

`removeFilter`(*filt*)
Rimuove lo specifico filtro *filt* da questo handler.

filter(*record*)
Applica i filtri di questo handler a *record* e restituisce un valore vero se il record verrà elaborato.

flush()
Assicura che tutto il risultato del logging sia svuotato. Questa versione non fa nulla ed è intesa per essere implementata da classi derivate.

close()
Libera ogni risorsa usata dall'handler. Questa versione non fa nulla ed è intesa per essere implementata da classi derivate.

handle(*record*)
A seconda dei filtri che possono essere stati aggiunti all'handler, emette lo specifico record di logging. Sostituisce l'attuale emissione del record con una acquisizione/rilascio del lock del thread di I/O.

handleError()
Questo metodo dovrebbe essere chiamato dagli handler quando viene riscontrata un'eccezione durante una chiamata `emit()`. Il suo comportamento predefinito è non fare niente, il che significa che le eccezioni vengono silenziosamente ignorate. Questo è quanto solitamente si desidera per un sistema di logging – la maggior parte degli utenti non faranno attenzione agli errori nel sistema di logging, ma saranno più interessati agli errori dell'applicazione. Si può comunque, rimpiazzare questo metodo con un handler personale, se lo si desidera.

format(*record*)
Effettua la formattazione di un *record* – Se il formatter è impostato, lo si usi. Altrimenti si utilizzi il formatter predefinito del modulo.

emit(*record*)
Consente a chiunque la utilizzi di registrare lo specifico messaggio di log *record*. Questa versione viene intesa per essere implementata come classe derivata e quindi solleva un'eccezione `NotImplementedError`.

StreamHandler

La classe `StreamHandler` invia il risultato del logging a un flusso come *sys.stdout*, *sys.stderr* o ogni oggetto simil-file (o, più precisamente, ad ogni oggetto che supporti i metodi `write()` e `flush()`.

class StreamHandler([*strm*])
Restituisce una nuova istanza della classe `StreamHandler`. Se *strm* viene specificato, l'istanza lo utilizzerà per il risultato del logging; altrimenti verrà usato *sys.stderr*.

emit(*record*)
Se un formatter viene specificato, viene utilizzato per preparare il *record*. Il *record* viene quindi scritto sul flusso con un codice di controllo di fine riga. Se è presente un' informazione di eccezione, questa viene formattata usando `traceback.print_exception()`, ed inserita in coda al flusso.

flush()
Svuota il flusso chiamando il suo metodo `flush()`. Notate che il metodo `close()` viene ereditato dall'`Handler` e non fa nulla, quindi a volte, è necessario un esplicito `flush()`.

FileHandler

La classe `FileHandler` invia il risultato del logging ad un file su disco. Eredita la funzionalità di output da `StreamHandler`.

class FileHandler(*filename*[, *mode*])
Restituisce una nuova istanza della classe `FileHandler`. Il file specificato viene aperto ed usato come flusso per il logging. Se *mode* non viene specificato, viene adottato `'a'`. Non esiste un limite predefinito sulla dimensione del file, che può crescere indefinitamente.

close()
Chiude il file.

emit(*record*)
Invia l'output di *record* al file.

RotatingFileHandler

La classe `RotatingFileHandler` supporta la rotazione dei file di log sul disco.

class RotatingFileHandler(*filename*[, *mode*[, *maxBytes*[, *backupCount*]]])
Restituisce una nuova istanza della classe `RotatingFileHandler`. Il file specificato viene aperto ed utilizzato come flusso per il logging. Se *mode* non viene specificato, viene adottato `'a'`. Non esiste un limite predefinito sulla dimensione del file.

Potete utilizzare i valori *maxBytes* e *backupCount* per consentire al file di essere *rollover* (NdT: sostituito) con uno di dimensione predeterminata. Quando la misura sta per essere superata, il file viene chiuso ed un nuovo file viene silenziosamente aperto per la registrazione. La sostituzione avviene quando il corrente file di log è vicino, per dimensione, a *maxBytes*; se *maxBytes* vale zero, la sostituzione non avviene. Se *backupCount* è diverso da zero, il sistema salverà il vecchio file di log aggiungendo l'estensione .1, .2 etc., al nome del file. Per esempio, se *backupCount* vale 5 ed il nome base del file è 'app.log', il file in corso di scrittura è sempre 'app.log'. Quando questo file è pieno, viene chiuso e rinominato in 'app.log.1', e se il file 'app.log.1', 'app.log.2' esistono, verranno rinominati rispettivamente in 'app.log.2', 'app.log.3'.

doRollover()
Effettua una sostituzione, come descritto sopra.

emit(*record*)
Registra l'output di *record* nel file, occupandosi anche della sostituzione come descritto in `setRollover()`.

SocketHandler

La classe `SocketHandler` invia il risultato del logging a un socket di rete. La classe base usa un socket TCP.

class SocketHandler(*host, port*)
Restituisce una nuova istanza della classe `SocketHandler`, intesa per comunicare con una macchine remota il cui indirizzo è dato da *host* e *port*.

close()
Chiude un socket.

handleError()

emit()
Serializza il dizionario degli attributi del record e lo scrive nel socket in formato binario. Se c'è un errore con il socket, scarta silenziosamente il pacchetto. Se la connessione è stata precedentemente persa, la ristabilisce. Per deserializzare il record alla ricezione in un `LogRecord`, usate la funzione `makeLogRecord()`.

handleError()
Gestisce un errore che è avvenuto durante `emit()`. La causa principale è una connessione persa. Chiude il socket in modo da poter ritentare con il prossimo evento.

makeSocket()
Questo è un metodo factory che consente alle classi derivate di definire il tipo preciso di socket che esse vogliono. L'implementazione predefinita crea un socket TCP (`socket.SOCK_STREAM`).

makePickle(*record*)
Serializza il dizionario degli attributi del record in formato binario con una lunghezza precisa, e lo restituisce pronto per la trasmissione attraverso il socket.

send(*packet*)
Invia una stringa serializzata *packet* al socket. Questa funzione consente invii parziali che si possono verificare se la rete è occupata.

DatagramHandler

La classe `DatagramHandler` eredita da `SocketHandler` il supporto per l'invio di messaggi di log attraverso un socket UDP.

class `DatagramHandler`(*host, port*)
Restituisce una nuova istanza della classe `DatagramHandler` intesa per comunicare con una macchina remota il cui indirizzo è dato da *host* e *port*.

`emit`()
Serializza il dizionario di attributi del record e lo scrive nel socket in formato binario. Se si verifica un errore con il socket, scarta silenziosamente il pacchetto. Per deserializzare il record alla fine della ricezione in un `LogRecord`, usate la funzione `makeLogRecord()`.

`makeSocket`()
Il metodo factory di `SocketHandler` viene qui sovrascritto per creare un socket UDP (`socket.SOCK_DGRAM`).

`send`(*s*)
Invia una stringa serializzata ad un socket.

SysLogHandler

La classe `SysLogHandler` supporta l'invio di messaggi di logging a un syslog UNIX locale o remoto.

class `SysLogHandler`([*address*[, *facility*]])
Restituisce una nuova istanza della classe `SysLogHandler`, intesa per comunicare con una macchina UNIX remota il cui indirizzo, *address*, è indicato in forma di tupla (*host*, *port*). Se l'indirizzo non viene specificato, viene usata la tupla `('localhost', 514)`. L'indirizzo viene usato per aprire un socket UDP. Se *facility* non viene specificato, viene usato `LOG_USER`.

`close`()
Chiude il socket all'host remoto.

`emit`(*record*)
Il *record* viene formattato, e quindi inviato al server syslog. Se è presente una informazione di eccezione, *non* viene inviato al server.

`encodePriority`(*facility, priority*)
Codifica *facility* e *priority* in un intero. Si può passare una stringa o un intero – se viene passata la stringa, viene utilizzato un dizionario interno per convertirlo in un intero.

NTEventLogHandler

La classe `NTEventLogHandler` supporta l'invio di messaggi di logging a un gestore locale di log di Windows NT, Windows 2000 o Windows XP. Prima di poterlo usare, è necessario che le estensioni Win32 per Python di Mark Hammond siano installate.

class `NTEventLogHandler`(*appname*[, *dllname*[, *logtype*]])
Restituisce una nuova istanza della classe `NTEventLogHandler`. L'argomento *appname* viene utilizzato per definire il nome dell'applicazione, ed appare nell'event log. Viene creata una voce di registro appropriata utilizzando questo nome. L'argomento *dllname* deve indicare un percorso completo e qualificato di una .dll o di un .exe che contenga la definizione del messaggio da inserire nel log (se non specificato, viene utilizzato `'win32service.pyd'` – che viene installato con le estensioni Win32, e contiene alcune definizioni di messaggi standard). Notate che l'uso di queste definizioni renderà l'event log interessato di grandi dimensioni, quanto l'intera sorgente del messaggio che verrà inserita nel log. Se desiderate un log più piccolo, passate il nome della vostra .dll o .exe che conterrà la definizione del messaggio che volete usare, nell'event log). Il *logtype* è uno tra `'Application'`, `'System'` o `'Security'`, ed il suo valore predefinito viene impostato a `'Application'`.

`close`()
A questo punto, si può rimuovere il nome dell'applicazione dal registro come sorgente delle registrazioni

nel log event. Comunque, se si fa questo, si noterà che non sarà possibile vedere gli eventi come lo si desidera nel visualizzatore degli eventi dei log – che deve essere in grado di accedere al registro per avere il nome della .dll. La versione corrente non fa questo (nei fatti, non fa nulla).

emit(*record*)
Determina l'ID del messaggio, la categoria di evento e il tipo di evento, e quindi registra il messaggio nel log degli eventi di NT.

getEventCategory(*record*)
Restituisce la categoria di evento per *record*. Sostituite questa funzione se volete specificare delle vostre categorie. Questa versione restituisce 0.

getEventType(*record*)
Restituisce il tipo evento per *record*. Sostituite questa se desiderate specificare dei vostri tipi. Questa versione effettua una mappatura utilizzando l'attributo typemap dell'handler, impostato a `__init__()` in un dizionario che contiene le associazioni per `INFO`, `WARNING`, `ERROR` e `CRITICAL`. Se desiderate utilizzare dei vostri livelli, dovrete necessariamente sostituire questo metodo o inserire un dizionario adatto nell'attributo *typemap* dell'handler.

getMessageID(*record*)
Restituisce l'ID del messaggio. Se state usando dei vostri messaggi, potete fare in modo che *msg* venga trasmesso al logger con un ID assegnato diverso da una stringa di formato. Quindi, in questa situazione, potete utilizzare una ricerca di dizionario per ottenere l'ID del messaggio. Questa versione restituisce 1, che è l'ID del messaggio di base in 'win32service.pyd'.

SMTPHandler

La classe `SMTPHandler` supporta l'invio dei messaggi di loggin ad un indirizzo email via SMTP.

class SMTPHandler(*mailhost, fromaddr, toaddrs, subject*)
Restituisce una nuova istanza della classe `SMTPHandler`. L'istanza viene inizializzata con gli indirizzi from e to ed una riga di soggetto per l'email. Il *toaddrs* dovrebbe essere una lista di stringhe senza il nome del dominio (per questo è previsto *mailhost*). Per specificare una porta SMTP non standard, usate la tupla (host,port) nell'argomento *mailhost*. Se utilizzate una stringa, viene usata la porta standard SMTP.

emit(*record*)
Formatta un record e lo invia agli indirizzi specificati.

getSubject(*record*)
Se desiderate specificare una linea di soggetto che è dipendente da *record*, è necessario sovrascrivere questo metodo.

MemoryHandler

Il `MemoryHandler` supporta il buffering in memoria dei record di logging, scaricandoli periodicamente verso un handler di destinazione (NdT: *target*). Lo svuotamento avviene quando il buffer è pieno o quando un evento di una certa importanza o più grande viene registrato.

`MemoryHandler` è una classe derivata della classe più generale `BufferingHandler`, che è una classe astratta. Questa mantiene il record di logging in memoria. Viene effettuato un controllo con la chiamata `shouldFlush()` quando ogni record viene aggiunto al buffer, per verificare se il buffer deve essere svuotato. Se ciò si verifica, si attende `flush()` per eseguire il compito necessario.

class BufferingHandler(*capacity*)
Inizializza l'handler con un buffer di capacità *capacity* specificata.

emit(*record*)
Aggiunge il record al buffer. Se `shouldFlush()` restituisce un valore vero, chiama `flush()` per elaborare il buffer.

flush()
Si può sovrascrivere questo per implementare un proprio schema di svuotamento. Questa versione semplicemente azzera il buffer.

shouldFlush(*record*)
Restituisce vero se il buffer è pieno. Questo metodo può essere sovrascritto per implementare un proprio schema di svuotamento.

class MemoryHandler(*capacity*[, *flushLevel* [, *target*]])
Restituisce una nuova istanza della classe `MemoryHandler`. L'istanza viene inizializzata con un buffer di capacità *capacity*. Se *flushLevel* non viene specificato, viene utilizzato `ERROR`. Se non viene specificato *target*, sarà necessario specificarlo successivamente usando `setTarget()` prima di questo handler per fargli fare qualcosa di utile.

close()
Chiama `flush()`, imposta il target a `None` e ripulisce il buffer.

flush()
Per un `MemoryHandler`, lo svuotamento semplicemente significa spedire dei record bufferizzati verso una destinazione, sempre che ne esista una. La si sovrascrive se si ha necessità di eseguire un compito diverso.

setTarget(*target*)
Imposta l'handler di destinazione per questo handler.

shouldFlush(*record*)
Verifica se il buffer è pieno o se un record sia pari a *flushLevel* o maggiore.

HTTPHandler

La classe `HTTPHandler` supporta l'invio dei messaggi di logging a un web server utilizzando sia la semantica '`GET`' che '`POST`'.

class HTTPHandler(*host*, *url*[, *method*])
Restituisce una nuova istanza della classe `HTTPHandler`. L'istanza viene inizializzata con un indirizzo di un host, l'indirizzo url ed il metodo HTTP. Se il metodo *method* non viene specificato, viene utilizzato '`GET`'.

emit(*record*)
Invia il record al web server come un dizionario codificato in una URL.

6.28.3 Oggetti Formatter

I `Formatter` possiedono i seguenti attributi e metodi. Essi sono responsabili della conversione di un `LogRecord` in (tipicamente) una stringa che può essere interpretata sia da un essere umano che da un sistema esterno. Il `Formatter` di base permette di specificare una stringa di formattazione. Se non ne viene specificata una, il valore predefinito sarà `'%(messaggio)s'`.

Un Formatter può essere inizializzato con una stringa di formato che faccia uso della conoscenza degli attributi di `LogRecord` – come il valore predefinito menzionato precedentemente, che sfrutta il fatto che un messaggio utente e gli argomenti siano stati preformattati in un attributo *message* di `LogRecord`. Questo formato di stringa contiene le chiavi di mappa standard Python %-style. Vedete la sezione 2.3.6, "Operazioni sulla formattazione delle stringhe" per ulteriori informazioni in merito.

Attualmente, le chiavi di mappatura utili in un `LogRecord` sono:

Formato	Descrizione
`%(name)s`	Nome del logger (canale di logging).
`%(levelno)s`	Livello numerico per il logging dei mesaggi (`DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`).
`%(levelname)s`	Livello testuale per il logging dei messaggi (`'DEBUG'`, `'INFO'`, `'WARNING'`, `'ERROR'`, `'CRITICAL'`
`%(pathname)s`	Percorso completo del file sorgente da cui la chiamata di logging è stata invocata (se disponibile).
`%(filename)s`	Porzione del percorso che contiene il nome del file.
`%(module)s`	Modulo (nome della porzione del nome del file).
`%(lineno)d`	Numero riga del sorgente da cui la chiamata di logging è stata generata (se disponibile).
`%(created)f`	Ora in cui è stato creato il `LogRecord` (come restituito da `time.time()`).
`%(asctime)s`	Il formato umanamente comprensibile del momento (Ora/data) in cui viene creato `LogRecord`. La sua for
`%(msecs)d`	Porzione dei millisecondi dell'ora in cui è stato creato `LogRecord`.
`%(thread)d`	ID del Thread (se disponibile).
`%(process)d`	ID del processo (se disponibile).
`%(message)s`	Il messaggio registrato, calcolato come `msg %args`.

class `Formatter`([*fmt*[, *datefmt*]])

Restituisce una nuova istanza della classe `Formatter`. L'istanza viene inizializzata con una stringa di formato per il messaggio, insieme alla stringa di formato per la porzione data/ora del messaggio. Se *fmt* non viene specificato, allora viene usato `'%(messaggio)s'`. Se non viene specificato *datefmt*, viene usato il formato dati ISO8601.

`format`(*record*)

Il dizionario degli attributi di *record* viene utilizzato come operando per le operazioni di formattazione della stringa. Restituisce la stringa risultante. Prima della formattazione del dizionario, viene eseguita una coppia di passi preparatori. L'attributo *messaggio* del record viene calcolato utilizzando *msg* % *args*. Se la stringa di formattazione contiene `'(asctime)'`, `formatTime()` viene chiamata per l'evento time. Se c'è un'informazione sull'eccezione, viene formattata usando `formatException()` ed aggiunta al messaggio.

`formatTime`(*record*[, *datefmt*])

Questo metodo dovrebbe essere chiamato da `format()` attraverso un formatter che voglia fare uso di una stringa time formattata. Questo metodo può essere sovrascritto nei formatter per sopperire a richieste particolari, ma il compito principale resta il seguente: se *datefmt* (una stringa) viene specificata, viene usata con `time.strftime()` per formattare l'ora di creazione del record. Altrimenti, viene usato il formato ISO8601. La stringa risultante viene restituita.

`formatException`(*exc_info*)

Formatta la specifica informazione di eccezione (una tupla standard di eccezione viene restituita da `sys.exc_info()`) come una stringa. L'implementazione predefinita usa `traceback.print_exception()`. La stringa risultante viene restituita.

6.28.4 Oggetti Filter

Gli oggetti `Filter` possono essere usati dagli `Handler` e dai `Logger` per un più sofisticato filtraggio rispetto a quello fornito da questo livello. La classe di filtro base permette solo eventi che sono sotto un certo punto nella gerarchia del logger. Per esempio, A.B, A.B.C, A.B.C.D, A.B.D etc. ma non A.BB, B.A.B etc. Se inizializzato con una stringa vuota, tutti gli eventi sono autorizzati.

class `Filter`([*name*])

Restituisce l'istanza della classe `Filter`. Se *name* viene specificato, chiama un logger, che, insieme ai suoi figli, ha l'autorizzazione a passare attraverso il filtro. Se *name* non viene specificato, consente ogni evento.

`filter`(*record*)

È il record specificato da loggare? Restituisce zero per no, un valore diverso da zero per sì. Se risulta appropriato, il record può essere modificato sul posto da questo metodo.

6.28.5 Oggetti LogRecord

Le istanze `LogRecord` vengono create ogni volta che esiste qualcosa da loggare. Esse contengono tutte le informazioni pertinenti agli eventi che devono essere loggati. Le principali informazioni che vengono passate sono msg e args, che vengono combinate usando msg % args per creare un campo messaggio nel record. Il record include anche informazioni su quando il record è stato creato, la riga sorgente da cui la chiamata di logging è stata fatta, ed ogni informazione sulle eccezioni da registrare.

`LogRecord` non ha metodi: è solo un deposito di informazioni circa l'evento di logging. La sola ragione del fatto che si tratta di una classe piuttosto che di un dizionario è per facilitarne l'estensione.

class `LogRecord`(*name, lvl, pathname, lineno, msg, args, exc_info*)
: Restituisce una istanza di `LogRecord` inizializzata con interessanti informazioni. *name* è il nome del logger; *lvl* è il livello numerico; *pathname* è il percorso dei nomi assoluto del file sorgente dal quale è stata effettuata la chiamata; *lineno* è il numero di riga del file dove la chiamata di logging viene rinvenuta; *msg* è il messaggio fornito dall'utente (una stringa di formato); *args* è la tupla in cui, insieme a *msg*, viene costruito il messaggio utente; infine *exc_info* è la tupla di eccezione ottenuta attraverso la chiamata `sys.exc_info()` (o `None`, se nessuna informazione di eccezione è disponibile).

6.28.6 Sicurezza dei thread

Il modulo di logging è costruito per essere thread-safe, senza che nessuno lavoro particolare debba essere fatto dai suoi client. Questa sicurezza è ottenuta attraverso l'uso dei lock dei thread; c'è un lock per serializzare l'accesso al modulo dei dati condivisi, e ogni handler (NdT: gestore) crea un lock per serializzare l'accesso al suo sottostante I/O.

6.28.7 Configurazione

Funzioni di configurazione

Le seguenti funzioni consentono di configurare il modulo di logging. Prima di poterle utilizzare, si deve importare `logging.config`. Il loro uso è facoltativo — si può configurare il modulo di logging nella sua interezza effettuando chiamate alla API principale (definita direttamente in `logging`) e definendo degli handler dichiarati in `logging` o in `logging.handlers`.

`fileConfig`(*fname*[, *defaults*])
: Legge la configurazione del logging da un file in formato ConfigParser chiamato *fname*. Questa funzione può essere chiamata da un'applicazione diverse volte, permettendo all'utente finale di effettuare una scelta da diverse configurazioni (se lo sviluppatore fornisce un meccanismo per offrire una scelta e caricare la configurazione voluta). In modo predefinito viene passato a ConfigParser e può essere specificato nell'argomento *defaults*.

`listen`([*port*])
: Avvia un socket server sulla specifica porta, e resta in ascolto per una nuova configurazione. Se nessuna porta viene specificata, viene usata la voce predefinita presente nel modulo `DEFAULT_LOGGING_CONFIG_PORT`. La configurazione di logging viene inviata come file adatto ad essere elaborato da `fileConfig()`. Restituisce una istanza di `Thread` su cui si può chiamare `start()` per avviare il server, e in cui si può chiamare `join()` quando appropriato. Per fermare il server, si chiama `stopListening()`.

`stopListening`()
: Ferma il server di ascolto che è stato creato con la chiamata a `listen()`. Questo metodo viene solitamente chiamato prima della chiamata a `join()` sul valore restituito da `listen()`.

Formato del file di configurazione

Il formato del file di configurazione esaminato da `fileConfig()` è basato sulla funzionalità di ConfigParser. Il file deve contenere le sezioni `[loggers]`, `[handlers]` e `[formatters]` che identificano per nome le

entità di ogni tipo che è definito nel file. Per ciascuna entità, c'è una sezione separata che identifica il modo in cui l'entità viene configurata. Quindi per un logger chiamato `log01` nella sezione `[loggers]`, i dettagli rilevati vengono mantenuti nella sezione `[logger_log01]`. Similmente, un handler chiamato `hand01` avrà la propria configurazione mantenuta in una sezione chiamata `[handler_hand01]`, quando un formatter chiamato `form01` avrà nella sezione `[formatters]` la propria configurazione specificata in una sezione chiamata `[formatter_form01]`. La configurazione del logger principale deve essere inserita nella sezione chiamata `[logger_root]`.

Esempi di queste sezioni presenti nel file sono presentate qui di seguito.

```
[loggers]
keys=root,log02,log03,log04,log05,log06,log07

[handlers]
keys=hand01,hand02,hand03,hand04,hand05,hand06,hand07,hand08,hand09

[formatters]
keys=form01,form02,form03,form04,form05,form06,form07,form08,form09
```

Il logger principale deve specificare un livello e una lista di handler. Un esempio della sezione di un root logger è fornita a seguito.

```
[logger_root]
level=NOTSET
handlers=hand01
```

La voce `level` può essere una tra `DEBUG, INFO, WARNING, ERROR, CRITICAL` o `NOTSET`. Solo per il logger principale, il root logger, `NOTSET` indica che tutti i messaggi verranno loggati. I valori di livello vengono valutati (`eval()`) nel contesto dello spazio dei nomi del package di `logging`.

La voce `handlers` è una lista separata da virgole di nomi di handler, che devono apparire nella sezione `[handlers]`. Questi nomi devono apparire nella sezione `[handlers]` e devono avere una corrispondente sezione nel file di configurazione.

Per i logger diversi dal logger principale (root logger), vengono richieste alcune ulteriori informazioni. Queste vengono illustrate con l'esempio seguente:

```
[logger_parser]
level=DEBUG
handlers=hand01
propagate=1
qualname=compiler.parser
```

Le voci degli `handlers` e di `level` vengono interpretate come per il logger principale, eccezion fatta laddove il livello del logger non principale è specificato come `NOTSET`. Il sistema consulta il logger più in alto nella gerarchia per determinare l'effettivo livello del logger. La voce `propagate` viene impostata a 1 per indicare che i messaggi devono essere propagati fino all'handler più alto nella gerarchia, o 0 per indicare che i messaggi **non** si devono propagare agli handler superiori nella gerarchia. La voce `qualname` è gerarchicamente il nome del canale del logger, per esempio, il nome usato dall'applicazione per ottenere il logger.

Le sezioni che specificano la configurazione per l'handler sono esemplificate di seguito.

```
[handler_hand01]
class=StreamHandler
level=NOTSET
formatter=form01
args=(sys.stdout,)
```

La voce `class` indica la classe dell'handler (come determinata da `eval()` nello spazio dei nomi del package `logging`). `level` viene interpretato come per i logger, e `NOTSET` viene interpretato come registra ogni cosa.

La voce `formatter` indica il nome della chiave del formatter per questo handler. Se vuoto, viene utilizzato un formatter predefinito (`logging._defaultFormatter`). Se viene specificato un nome, deve apparire nella sezione `[formatters]` e possedere una corrispondente sezione nel file di configurazione.

La voce `args`, quando valutata (attraverso `eval()`), nel contesto dello spazio dei nomi del package `logging`, è la lista degli argomenti per il costruttore della classe di handler. Riferitevi ai costruttori per l'handler principale, o all'esempio qui di seguito, per vedere quali voci tipiche vengono utilizzate.

```
[handler_hand02]
class=FileHandler
level=DEBUG
formatter=form02
args=('python.log', 'w')

[handler_hand03]
class=handlers.SocketHandler
level=INFO
formatter=form03
args=('localhost', handlers.DEFAULT_TCP_LOGGING_PORT)

[handler_hand04]
class=handlers.DatagramHandler
level=WARN
formatter=form04
args=('localhost', handlers.DEFAULT_UDP_LOGGING_PORT)

[handler_hand05]
class=handlers.SysLogHandler
level=ERROR
formatter=form05
args=(('localhost', handlers.SYSLOG_UDP_PORT), handlers.SysLogHandler.LOG_USER)

[handler_hand06]
class=NTEventLogHandler
level=CRITICAL
formatter=form06
args=('Python Application', '', 'Application')

[handler_hand07]
class=SMTPHandler
level=WARN
formatter=form07
args=('localhost', 'from@abc', ['user1@abc', 'user2@xyz'], 'Logger Subject')

[handler_hand08]
class=MemoryHandler
level=NOTSET
formatter=form08
target=
args=(10, ERROR)

[handler_hand09]
class=HTTPHandler
level=NOTSET
formatter=form09
args=('localhost:9022', '/log', 'GET')
```

Le sezioni che specificano la configurazione del formatter sono presentate di seguito.

```
[formatter_form01]
format=F1 %(asctime)s %(levelname)s %(message)s
datefmt=
```

La voce `format` è la stringa di formato principale, e la voce `datefmt` è la stringa di formato `strftime()`-compatibile data/ora. Se vuota, il package lo sostituisce con data/ora formato ISO8601, che è all'incirca equivalente alla specifica stringa di formato Il formato ISO8601 specifica anche i millisecondi, che vengono aggiunti al risultato utilizzando la stringa di formato sopra, con un separatore '`,`'. Un ora d'esempio in formato ISO8601 è `2003-01-23 00:29:50,411`.

6.28.8 Utilizzo del package logging

Esempi di base – registrare in un file

Qui un semplice esempio di logging che semplicemente registra in un file. Nell'ordine, crea una istanza `Logger`, un `FileHandler` e un `Formatter`. Quindi collega il `Formatter` al `FileHandler`, ed il `FileHandler` al `Logger`. Infine, imposta un livello di debug per il logger.

```
import logging
logger = logging.getLogger('myapp')
hdlr = logging.FileHandler('/var/tmp/myapp.log')
formatter = logging.Formatter('%(asctime)s %(levelname)s %(message)s')
hdlr.setFormatter(formatter)
logger.addHandler(hdlr)
logger.setLevel(logging.WARNING)
```

Possiamo usare questo oggetto logger per scrivere voci in un file di log:

```
logger.error('We have a problem')
logger.info('While this is just chatty')
```

Se guardate nel file che è stato creato, vedrete qualcosa del genere:

```
2003-07-08 16:49:45,896 ERROR We have a problem
```

Il messaggio *info* non è stato scritto nel file; abbiamo chiamato il metodo `setLevel` per dire che si voleva tracciare solamente `WARNING`, e quindi, il messaggio *info* è scartato.

Il timestamp è nella forma "anno-mese-giorno ora:minuti:secondi,millisecondi". Da notare i 3 valori decimali nel campo dei millisecondi, purtroppo non tutti i sistemi forniscono l'ora con questa precisione.

6.29 `platform` — Accesso ai dati identificativi della piattaforma sottostante

Nuovo nella versione 2.3.

Note: Le specifiche piattaforme vengono elencate alfabeticamente, con Linux inserito nella sezione UNIX.

6.29.1 Multi piattaforma

`architecture`(*executable=sys.executable, bits='', linkage=''*)

Interroga l'eseguibile dato (il predefinito è l'interprete Python binario) per varie informazioni sull'architettura.

Restituisce una tupla `(bits, linkage)` che contiene informazioni riguardo il numero di bit usati dall'architettura per cui l'eseguibile viene preparato, ed il formato del linker usato per l'eseguibile. Entrambi i valori vengono restituiti come stringhe.

I valori che non possono venire determinati vengono restituiti come forniti dalle impostazioni predefinite del parametro. Se bits è fornito come `''`, viene usato `sizeof(pointer)` (o `sizeof(long)` in versioni < 1.5.2 di Python) come indicatore per la dimensione del puntatore supportato.

La funzione si basa sul comando di sistema 'file' per fare il lavoro. Questo è disponibile sulla maggior parte, se non su tutti, i sistemi UNIX e su diverse piattaforme non-UNIX, solamente nel caso in cui l'eseguibile punti all'interprete Python. Vengono usate ragionevoli impostazioni predefinite nel caso in cui le condizioni di cui sopra non siano soddisfatte.

`machine``()`
Restituisce il tipo di macchina, per esempio `'i386'`. Se il valore non può essere determinato, viene restituita una stringa vuota.

`node``()`
Restituisce il nome di rete del computer (può non essere fully qualified! (NdT: Un nome di dominio non completamente qualificato)). Se il valore non può essere determinato, viene restituita una stringa vuota.

`platform``(`*aliased=0, terse=0*`)`
Restituisce una singola stringa che identifica la piattaforma sottostante con il maggior numero possibile di informazioni utili.

L'output viene inteso come *umanamente comprensibile*, piuttosto che elaborabile dalla macchina. Significa che il risultato può essere diverso su differenti piattaforme.

Se *aliased* è vero, la funzione utilizzerà dei sinonimi per le varie piattaforme, che riporteranno nomi di sistema diversi dai loro nomi comuni, ad esempio SunOS sarà riportato come Solaris. Per implementare questo comportamento, viene usata la funzione `system_alias()`.

Impostando *terse* a vero si farà in modo che la funzione restituisca solamente le informazioni minime necessarie ad identificare la piattaforma.

`processor``()`
Restituisce il nome (reale) del processore, per esempio `'amdk6'`.

Se il valore non può essere determinato, viene restituita una stringa vuota. Notate che molte piattaforme non forniscono questa informazione o semplicemente restituiscono lo stesso valore di `machine()`, NetBSD si comporta in questa maniera.

`python_build``()`
Restituisce una tupla `(`*buildno*`, `*builddate*`)` che riporta la data ed il numero di compilazione di Python come stringhe.

`python_compiler``()`
Restituisce una stringa che identifica il compilatore usato per compilare Python.

`python_version``()`
Restituisce la versione di Python come stringa `'major.minor.patchlevel'`

Notate che, diversamente da Python `sys.version`, il valore restituito includerà sempre il livello di patch (il cui valore predefinito è impostato a 0).

`python_version_tuple``()`
Restituisce la versione di Python come `(`*major*`, `*minor*`, `*patchlevel*`)`, una tupla di stringhe.

Notate che, diversamente da Python `sys.version`, il valore restituito includerà sempre il livello di patch (il cui valore predefinito è impostato a 0).

`release``()`
Restituisce la versione del sistema, per esempio `'2.2.0'` oppure `'NT'`. Se il valore non può essere determinato, viene restituita una stringa vuota.

`system``()`
Restituisce il nome del sistema/OS, per esempio `'Linux'`, `'Windows'`, o `'Java'`. Se il valore non può essere determinato, viene restituita una stringa vuota.

system_alias(*system, release, version*)
Restituisce (*system, release, version*), sinonimi dei nomi commerciali usati per alcuni sistemi. Oltre a questo, riordina alcune informazioni che in qualche caso potrebbero generare confusione.

version()
Restituisce la versione rilasciata del sistema, per esempio `'#3 on degas'`. Se il valore non può essere determinato, viene restituita una stringa vuota.

uname()
Interfaccia molto portabile ad uname. Restituisce una tupla di stringhe (*system, node, release, version, machine, processor*) che identificano la piattaforma sottostante.

Notate che, diversamente dalla funzione `os.uname()`, questa riporta anche, come voci aggiuntive nella tupla, possibili informazioni sul processore.

Le voci che non possono essere determinate vengono impostate a `''`.

6.29.2 Piattaforma Java

java_ver(*release='', vendor='', vminfo=('','',''), osinfo=('','','')*)
Versione dell'interfaccia per JPython.

Restituisce una tupla (*release, vendor, vminfo, osinfo*) con *vminfo*, che a sua volta è una tupla (*vm_name, vm_release, vm_vendor*) e *osinfo*, anche questo una tupla (*os_name, os_version, os_arch*). I valori che non possono essere determinati vengono impostati a quelli predefiniti, forniti come parametri (tutti preimpostati a `''`).

6.29.3 Piattaforma Windows

win32_ver(*release='', version='', csd='', ptype=''*)
Ricava informazioni supplementari sulla versione dal Windows Registry e restituisce una tupla (*version, csd, ptype*) riferita al numero di versione, livello CSD e tipo di OS (processore singolo/multiplo).

Un suggerimento: *ptype* è `'Uniprocessor Free'` su macchine NT a processore singolo e `'Multiprocessor Free'` su macchine multiprocessore. Il *'Free'* si riferisce alla versione dell'OS che non contiene codice di debugging. Può anche riportare *'Checked'*, che indica che la versione dell'OS usa codice di debugging, per esempio codice che testa argomenti, campi, ecc.

Note: Notate che questa funzione lavora solo se è istallato il package `win32all` di Mark Hammond e (naturalmente) solo su piattaforme Win32 compatibili.

Specifiche Win95/98

popen(*cmd, mode='r', bufsize=None*)
Interfaccia portabile `popen()`. Cerca l'implementazione per popen preferendo `win32pipe.popen()`. Su WindowsNT `win32pipe.popen()` dovrebbe funzionare; su Windows 9x resta in sospeso, a causa dei bug presenti nella libreria C di MS.

6.29.4 Piattaforma Mac OS

mac_ver(*release='', versioninfo=('','',''), machine=''*)
Ottiene l'informazione sulla versione di Mac OS, e la restituisce come una tupla ((*release, versioninfo, machine*)) con *versioninfo* essendo una tupla ((*version, dev_stage, non_release_version*)).

Le voci che non possono essere determinate vengono impostate a `''`. Tutte le voci nelle tuple sono stringhe.

La documentazione per l' API sottostane `gestalt()` è disponibile online presso http://www.rgaros.nl/gestalt/.

6.29.5 Piattaforme UNIX

dist(*distname=”, version=”, id=”, supported_dists=(’SuSE’,’debian’,’redhat’,’mandrake’)*)
Cerca di determinare il nome della distribuzione del sistema operativo, restituendo la tupla (*distname, version, id*) i cui valori predefiniti degli argomenti vengono passati come parametri.

libc_ver(*executable=sys.executable, lib=”, version=”, chunksize=2048*)
Cerca di determinare la versione della libreria C verso cui il file eseguibile (il predefinito è l’interprete Python) è linkato. Viene restituita una tupla di stringhe (*lib, version*), che viene impostata in modo predefinito ai valori dei parametri forniti, nel caso un cui la ricerca fallisca.

Notate che questa funzione ha intime conoscenze di come differenti versioni della libreria C aggiungano simboli all’eseguibile, ed è probabilmente utilizzabile solo con eseguibili compilati usando **gcc**.

Il file viene letto e analizzato in porzioni di *chunksize* byte.

CAPITOLO

SETTE

Servizi facoltativi per il sistema operativo

I moduli descritti in questo capitolo forniscono interfacce per funzionalità del sistema operativo disponibili solamente per particolari sistemi operativi. Le interfacce sono generalmente progettate in conformità ad UNIX o C ma sono comunque disponibili su altri sistemi (p.es. Windows o NT). Eccone una sintesi:

signal	Imposta la gestione per eventi asincroni.
socket	Interfaccia di rete di basso livello.
select	Attesa per il completamento dell'I/O su molteplici flussi.
thread	Crea thread multipli di controllo con un interprete.
threading	Interfaccia ad alto livello per i thread.
dummy_thread	Rimpiazzamento drop-in per il modulo `thread`.
dummy_threading	Rimpiazzamento drop-in per il modulo `threading`.
Queue	Una classe coda sincronizzata.
mmap	Interfaccia per i file mappati in memoria per UNIX e Windows.
anydbm	Interfaccia generica ai moduli per database in stile DBM.
dbhash	Interfaccia stile DBM per la libreria database BSD.
whichdb	Indovina quale modulo in stile DBM ha creato un database disponibile.
bsddb	Interfaccia alla libreria per database Berkeley DB
dumbdbm	Implementazione portabile di una semplice interfaccia DBM.
zlib	Interfaccia di basso livello di routine per la compressione e decompressione, compatibile con **gzip**.
gzip	Interfaccia per il formato **gzip** di compressione e decompressione che usa file oggetti.
bz2	Procedure di interfaccia di compressione e decompressione compatibile con **bzip2**.
zipfile	Read and write ZIP-format archive files.
tarfile	Leggere e scrivere file di archivio nel formato tar.
readline	Supporto per Python alla GNU readline.
rlcompleter	Identificatore di completamento Python per la libreria GNU readline.

7.1 `signal` — Imposta i gestori per eventi asincroni

Questo modulo fornisce meccanismi atti ad usare gestori di segnali in Python. Alcune regole generali per lavorare con i segnali e i loro gestori:

- Un gestore per un segnale particolare, una volta impostato, rimane installato fino a che non venga esplicitamente resettato (Python emula lo stile delle interfacce BSD indipendentemente dall'implementazione sottostante), con l'eccezione per il gestore di `SIGCHLD`, che segue l'implementazione sottostante.
- Non c'è modo per "bloccare" segnali temporaneamente da sezioni critiche (visto che non è supportato da tutti gli UNIX).
- Benché i gestori dei segnali di Python vengano chiamati in modo asincrono finché l'utente ne è interessato, essi possono trovarsi solo fra istruzioni "atomiche" dell'interprete Python. Ciò significa che i segnali arrivando durante lunghi calcoli implementati puramente in C (come corrispondenze di espressioni regolari su grandi parti di testo) possono essere ritardati per un lasso di tempo arbitrario.

- Quando un segnale arriva durante un'operazione di I/O, è possibile che l'operazione di I/O sollevi un'eccezione dopo il ritorno del gestore del segnale. Questo dipende dalla semantica dell'implementazione del sistema UNIX sottostante in riferimento alle chiamate di sistema interrotte.

- Visto che i gestori di segnali C ritornano sempre, ha poco senso catturare errori sincroni come `SIGFPE` o `SIGSEGV`.

- Python installa un piccolo numero di gestori di segnali di norma: `SIGPIPE` è ignorato (perciò errori di scrittura su pipe e socket possono essere riportati come normali eccezioni Python) mentre `SIGINT` viene tradotto in un'eccezione `KeyboardInterrupt`. Tutte quante possono essere sovrascritte.

- Alcune attenzioni devono essere prese se sia segnali che thread vengono utilizzati dallo stesso programma. La cosa fondamentale da ricordare nell'usare segnali e thread simultaneamente è: effettuare sempre operazioni di `signal()` nel thread principale di esecuzione. Qualunque thread può eseguire un `alarm()`, `getsignal()`, o `pause()`; solo il thread principale può impostare nuovi gestori di segnale, e il thread principale sarà l'unico a ricevere segnali (questo viene forzato dal modulo Python `signal`, anche se l'implementazione dei thread sottostante supporta l'invio di segnali ai singoli thread). Ciò significa che i segnali non possono essere usati come un mezzo di comunicazione inter-thread. Usare al contrario i lock.

Le variabili definite nel modulo `signal` sono:

`SIG_DFL`
Questa è una delle due opzioni standard di gestione dei segnali; effettuerà semplicemente la funzione predefinita per il segnale. Per esempio, su molti sistemi l'azione predefinita per `SIGQUIT` è fare un core dump ed uscire, mentre l'azione predefinita per `SIGCLD` è semplicemente ignorarlo.

`SIG_IGN`
Questo è un altro gestore di segnale standard, che semplicemente ignorerà il segnale dato.

`SIG*`
Tutti i numeri di segnale sono definiti simbolicamente. Per esempio, il segnale di hangup è definito come `signal.SIGHUP`; i nomi di variabile sono identici ai nomi usati nei programmi C, come li si trova in `<signal.h>`. La pagina del manuale UNIX per '`signal()`' elenca i segnali esistenti (su alcuni sistemi si trova in *signal*(2), su altri l'elenco si trova in *signal*(7)). Attenzione al fatto che non tutti i sistemi definiscono lo stesso insieme di nomi di segnali; solo questi nomi definiti dal sistema sono definiti in questo modulo.

`NSIG`
Uno in più del più alto numero di segnale.

Il modulo `signal` definisce le seguenti funzioni:

`alarm`(*time*)
Se *time* è diverso da zero, questa funzione richiede che un segnale `SIGALRM` sia trasmesso al processo in *time* secondi. Ogni precedente allarme schedulato viene cancellato (solo un'allarme può essere schedulato in ogni momento). Il valore restituito è il numero di secondi rimanenti prima di ogni precedente allarme impostato che deve essere trasmesso. Se *time* è zero, nessun allarme è attualmente schedulato ed ogni allarme schedulato è cancellato. Il valore restituito è il numero di secondi rimanente prima di un precedente allarme schedulato. (Leggere la pagina di manuale UNIX *alarm*(2).) Disponibilità: UNIX.

`getsignal`(*signalnum*)
Restituisce il gestore di segnale corrente per il segnale *signalnum*. Il valore restituito può essere un oggetto Python chiamabile, o uno dei valori speciali `signal.SIG_IGN`, `signal.SIG_DFL` o `None`. Qui, `signal.SIG_IGN` significa che il signale è stato in precedenza ignorato, `signal.SIG_DFL` significa che era in uso la maniera predefinita per gestire il segnale, e `None` significa che il gestore di segnale precedente non è stato installato da Python.

`pause`()
Impone al processo di dormire fino a che un segnale venga ricevuto; il gestore appropriato verrà quindi chiamato. Non ritorna nulla. Non disponibile su Windows. (leggere la pagina di manuale UNIX *signal*(2).)

`signal`(*signalnum, handler*)
Imposta la gestione per il segnale *signalnum* alla funzione *handler*. *handler* può essere un oggetto Python chiamabile che prende due argomenti (vedere sotto), o uno dei valori speciali `signal.SIG_IGN`

o `signal.SIG_DFL`. Sarà restituito il precedente gestore di segnale. (leggere sopra la descrizione `getsignal()`). (Leggere la pagina di manuale UNIX *signal*(2).)

Quando i thread sono abilitati, questa funzione può essere chiamata unicamente dal thread principale; chiamandola da altri thread verrà sollevata un'eccezione `ValueError`.

L'*handler* viene chiamato con due argomenti: il numero di segnale e lo stack frame corrente (`None` od un oggetto frame; leggere il manuale di riferimento per una descrizione degli oggetti frame).

7.1.1 Esempio

Questo è un piccolo programma di esempio. Usa la funzione `alarm()` per limitare il tempo speso aspettando di aprire un file; questo è utile se il file rappresenta un dispositivo seriale che potrebbe non essere acceso, che normalmente causerebbe un'attesa indefinita alla `os.open()`. La soluzione è impostare un allarme di 5 secondi prima di aprire un file; se l'operazione richiede troppo tempo, il segnale di allarme viene inviato, ed il gestore solleva un'eccezione.

```
import signal, os

def handler(signum, frame):
    print 'Signal handler called with signal', signum
    raise IOError, "Couldn't open device!"

# Imposta il gestore di segnale per un'allarme di 5 secondi
signal.signal(signal.SIGALRM, handler)
signal.alarm(5)

# Questa open() può attendere indefinitamente
fd = os.open('/dev/ttyS0', os.O_RDWR)

signal.alarm(0)          # Disabilita l'allarme
```

7.2 `socket` — Interfaccia di rete di basso livello

Questo modulo fornisce un accesso all'interfaccia *socket* BSD. È disponibile su tutti i moderni sistemi UNIX, Windows, MacOS, BeOS, OS/2 e probabilmente altre piattaforme.

Per un'introduzione alla programmazione dei socket (in C), leggere i seguenti libri: *An Introductory 4.3BSD Interprocess Communication Tutorial*, di Stuart Sechres e *An Advanced 4.3BSD Interprocess Communication Tutorial*, di Samuel J. Leffler et al, sia il UNIX *Programmer's Manual, Supplementary Documents 1* (sezione PS1:7 e PS1:8). I materiali di riferimento specifici per piattaforma per le varie chiamate di sistema relative ai socket sono un'ottima sorgente di informazioni sui dettagli della semantica dei socket. Per UNIX, riferirsi alle pagine di manuale; per Windows, leggere le specifiche del WinSock (o WinSock 2). Per le API IPv6-ready, i lettori vorranno far riferimento all'RFC 2553 intitolato *Basic Socket Interface Extensions for IPv6*.

L'interfaccia Python è una traduzione diretta delle chiamate di sistema UNIX e l'interfaccia di libreria per i socket in stile orientato agli oggetti di Python: la funzione `socket()` restituisce un *oggetto socket* i cui metodi implementano le varie chiamate di sistema dei socket. I tipi di parametro sono qualcosa di più alto livello rispetto all'interfaccia C: simile alle operazioni `read()` e `write()` sui file Python, l'allocazione del buffer sulle operazioni di ricezione è automatica, e la lunghezza del buffer è implicita nelle operazioni di invio.

Gli indirizzi socket sono indicati come a seguito: una singola stringa è usata per la famiglia di indirizzi `AF_UNIX`. Una coppia (*host*, *porta*) è usata per la famiglia di indirizzi `AF_INET`, dove *host* è una stringa rappresentante un hostname nella notazione dei domini Internet come `'daring.cwi.nl'` o un indirizzo IPv4 come `'100.50.200.5'`, mentre *porta* è un numero di porta intero. Per la famiglia di indirizzi `AF_INET6`, viene usata una tupla quadrupla (*host*, *porta*, *flowinfo*, *scopeid*), dove *flowinfo* e *scopeid* indicano i membri `sin6_flowinfo` e `sin6_scope_id` nella struttura `struct sockaddr_in6` del C. Per i metodi del modulo `socket`, *flowinfo* e *scopeid* possono essere omessi solo per retrocompatibilità. Notare, comunque, che

l'omissione di *scopeid* può causare problemi nella manipolazione di indirizzi IPv6. Altre famiglie di indirizzi non sono attualmente sopportate. Il formato dell'indirizzo richiesto da un particolare oggetto socket è automaticamente selezionato in base alla famiglia di indirizzi specificata quando l'oggetto socket è stato creato.

Per gli indirizzi IPv4, sono accettate due forme speciali al posto di un indirizzo di un host: la stringa vuota rappresenta `INADDR_ANY`, e la stringa `'<broadcast>'` rappresenta `INADDR_BROADCAST`. Il comportamento non è disponibile con IPv6 per retrocompatibilità, quindi, si potrebbe volerle evitare se si ha l'intenzione di supportare IPv6 nei propri programmi Python.

Se si usa un hostname in un segmento di *rete* composto da indirizzi socket IPv4/v6, il programma potrebbe mostrare un comportamento non deterministico, visto che Python usa il primo indirizzo restituito dalla risoluzione DNS. L'indirizzo socket sarebbe risolto differentemente in un attuale indirizzo IPv4/v6 reale, a seconda dei risultati della risoluzione DNS e/o la configurazione dell'host. Per un comportamento deterministico usare un indirizzo numerico nel segmento di *rete*.

Tutti gli errori sollevano eccezioni. Possono essere sollevate le consuete eccezioni per tipi di argomento non validi e condizioni di out-of-memory; errori relativi ai socket o alla semantica degli indirizzi sollevano l'errore `socket.error`.

La modalità non bloccante è supportata attraverso `setblocking()`. Una sua generalizzazione basata sui timeout è supportata attraverso `settimeout()`.

Il modulo `socket` esporta le seguenti costanti e funzioni:

exception `error`
: Questa eccezione viene sollevata per errori relativi ai socket. Il valore che la accompagna è una stringa che dice cosa è andato storto oppure una coppia (*errno*, *string*) che rappresenta un errore restituito da una chiamata di sistema, simile al valore che accompagna `os.error`. Vedere il modulo `errno`, che contiene nomi per i codici di errore definiti dal sistema operativo sottostante.

exception `herror`
: Questa eccezione viene sollevata per errori relativi agli indirizzi, i.e. per funzioni che usano *h_errno* nell'API C, inclusi `gethostbyname_ex()` e `gethostbyaddr()`.

  Il valore che la accompagna è una coppia (*h_errno*, *string*) che rappresenta un errore restituito da una chiamata di libreria. *string* rappresenta la descrizione di *h_errno*, come restituito dalla funzione C `hstrerror()`. Il valore di *error* corrisponderà con una delle costanti `EAI_*` definite in questo modulo.

exception `gaierror`
: Questa eccezione è sollevata per errori relativi agli indirizzi, per `getaddrinfo()` e `getnameinfo()`. Il valore che la accompagna è una coppia (*error*, *string*) che rappresenta un errore restituito da una chiamata di libreria. *string* rappresenta la descrizione di *error*, come restituito dalla funzione C `gai_strerror()`.

exception `timeout`
: Questa eccezione viene sollevata quando avviene un timeout su un socket che aveva abilitati i timeout attraverso una precedente chiamata a `settimeout()`. Il valore che la accompagna è una stringa il cui valore corrente è sempre "timed out". Nuovo nella versione 2.3.

`AF_UNIX`
`AF_INET`
`AF_INET6`
: Queste costanti rappresentano le famiglie degli indirizzi (e protocolli), usate per il primo argomento di `socket()`. Se la constante constantAF_UNIX non è definita significa che questo protocollo non è supportato.

`SOCK_STREAM`
`SOCK_DGRAM`
`SOCK_RAW`
`SOCK_RDM`
`SOCK_SEQPACKET`
: Queste costanti rappresentano i tipi di socket, usati per il secondo argomento di `socket()`. (solo `SOCK_STREAM` e `SOCK_DGRAM` appaiono utili in genere.)

`SO_*`

SOMAXCONN
MSG_*
SOL_*
IPPROTO_*
IPPORT_*
INADDR_*
IP_*
IPV6_*
EAI_*
AI_*
NI_*
TCP_*

Molte costanti di queste forme, trattate nella documentazione UNIX sui socket e/o sul protocollo IP, sono inoltre definite nel modulo socket. Generalmente sono usate negli argomenti dei metodi `setsockopt()` e `getsockopt()` degli oggetti socket. In molti casi, solo quei simboli definiti nelle intestazioni dei file UNIX sono qui definiti; per alcuni di essi, sono forniti valori predefiniti.

has_ipv6

Questa costante contiene un valore booleano che indica se l'IPv6 è supportato su questa piattaforma. Nuovo nella versione 2.3.

getaddrinfo(*host*, *porta*[, *family*[, *socktype*[, *proto*[, *flags*]]]])

Risolve l'argomento *host*/*porta*, in una sequenza di tuple di 5 elementi contenenti tutti gli argomenti necessari per la manipolazione dei socket. *host* è il nome di un dominio, una stringa rappresentante un indirizzo IPv4/v6 oppure `None`. *porta* è una stringa contente il nome di un servizio (come `'http'`), una rappresentazione numerica del numero della porta oppure `None`.

Il resto degli argomenti sono facoltativi e devono essere numerici se specificati. Per *host* e *porta*, devono essere passate una stringa vuota o `None`, è possibile passare `NULL` all'API C. La funzione `getaddrinfo()` restituisce una lista di tuple di 5 elementi con la seguente struttura:

(*family*, *socktype*, *proto*, *canonname*, *sockaddr*)

family, *socktype* e *proto* sono tutti interi che devono essere passati alla funzione `socket()`. *canonname* è una stringa rappresentante il nome canonico dell'*host*. Può essere un indirizzo numerico IPv4/v6 quando `AI_CANONNAME` viene specificato per un *host* numerico. *sockaddr* è una tupla che descrive un indirizzo socket, come descritto sopra. Leggere i sorgenti di `httplib` ed altri moduli di libreria per un utilizzo tipico della funzione. Nuovo nella versione 2.2.

getfqdn([*nome*])

Restituisce un nome di dominio pienamente qualificato per *nome*. Se *nome* è omesso o vuoto, viene interpretato come l'host locale. Per trovare il nome pienamente qualificato, viene controllato l'hostname restituito da `gethostbyaddr()`, quindi fatto l'alias per l'host, se disponibile. Viene selezionato il primo nome contenente un punto. Nel caso non sia disponibile un nome di dominio pienamente qualificato, viene restituito l'hostname. Nuovo nella versione 2.0.

gethostbyname(*hostname*)

Traduce un host name nel formato di indirizzi IPv4. L'indirizzo IPv4 viene restituito in una stringa, come `'100.50.200.5'`. Se l'host name è esso stesso un indirizzo IPv4 viene restituito invariato. Vedere `gethostbyname_ex()` per un'interfaccia più completa. `gethostbyname()` non supporta la risoluzione dei nomi IPv6m e `getaddrinfo()` dovrebbe essere usata al suo posto per il supporto al dual stack IPv4/v6.

gethostbyname_ex(*hostname*)

Traduce un hostname in un formato di indirizzo IPv4, interfaccia estesa. Restituisce un tripla (*hostname*, *aliaslist*, *ipaddrlist*) dove *hostname* è il nome dell'host primario in risposta all'*ip_address* dato, *aliaslist* è una lista (che può essere vuota) di hostname alternativi per lo stesso indirizzo, e *ipaddrlist* è una lista di indirizzi IPv4 per la stessa interfaccia sullo stesso host (spesso ma non sempre un singolo indirizzo). `gethostbyname_ex()` non supporta la risoluzione dei nomi IPv6, e dovrebbe essere usata `getaddrinfo()` per il supporto al dual stack IPv4/v6.

gethostname()

Restituisce una stringa contenente l'hostname della macchina in cui è in esecuzione l'interprete Python. Se

si vuol sapere l'indirizzo IP corrente della macchina, si può usare `gethostbyname(gethostname())`. Questa operazione assume che ci sia una valida reciprocità tra l'indirizzo dell'host e l'host, e questo non sempre avviene. Notare: `gethostname()` non restituisce sempre il nome di dominio pienamente qualificato; usare `gethostbyaddr(gethostname())` (vedere sotto).

`gethostbyaddr`(*ip_address*)
Restituisce una tripla (*hostname*, *aliaslist*, *ipaddrlist*) dove *hostname* è il nome dell'host primario che risponde all'indirizzo *ip_address* dato, *aliaslist* è una lista (che può essere vuota) di nomi di host alternativi per lo stesso indirizzo, e *ipaddrlist* è una lista di indirizzi IPv4/v6 per la stessa interfaccia sullo stesso host (che più facilmente contiene solo un singolo indirizzo). Per trovare il nome di dominio pienamente qualificato, usare la funzione `getfqdn()`. `gethostbyaddr` supporta sia IPv4 che IPv6.

`getnameinfo`(*sockaddr, opzioni*)
Traduce un indirizzo socket *sockaddr* in una coppia (*host*, *porta*). A seconda delle impostazioni delle *opzioni*, il risultato può contenere un nome di dominio pienamente qualificato o una rappresentazione numerica dell'indirizzo in *host*. In modo simile *porta* può contenere una stringa con il nome della porta o la rappresentazione numerica del numero della porta. Nuovo nella versione 2.2.

`getprotobyname`(*protocolname*)
Traduce il nome di un protocollo Internet (per esempio, `'icmp'`) in una costante adatta per essere passata come terzo argomento (facoltativo) della funzione `socket()`. Solitamente serve solo per socket aperti in modalità "raw" (`SOCK_RAW`); per le normali modalità adottate con i socket, il protocollo corretto è scelto automaticamente se è omesso oppure zero.

`getservbyname`(*servicename, protocolname*)
Traduce il nome di un servizio Internet e il nome di un protocollo nel numero di una porta per quel servizio. Il nome del protocollo dovrebbe essere `'tcp'` o `'udp'`.

`socket`([*family*[, *type*[, *proto*]]])
Crea un nuovo socket usando i dati per la famiglia di indirizzo, *family*, il tipo di socket *type* ed il numero di protocollo *proto*. L'indirizzo della famiglia dovrebbe essere `AF_INET` (predefinito), `AF_INET6` o `AF_UNIX`. Il tipo di socket dovrebbe essere `SOCK_STREAM` (predefinito), `SOCK_DGRAM` o forse una delle altre costanti '`SOCK_`'. Il numero di protocollo è solitamente zero e può essere omesso in quel caso.

`ssl`(*sock*[, *keyfile, certfile*])
Inizializza una connessione SSL sul socket *sock*. *keyfile* è il nome di un file formattato PEM che contiene la propria chiave privata. *certfile* è un file chiave certificato e formattato PEM. In caso di successo, viene restituito un nuovo oggetto `SSLObject`.

Avvertenze: Questa funzione non effettua nessuna verifica di certificazione!

`fromfd`(*fd, family, type*[, *proto*])
Costruisce un oggetto socket da un descrittore di file esistente (un intero restituito dal metodo `fileno()` dell'oggetto file). Famiglia di indirizzo, tipo di socket e numero di protocollo sono identici a quelli della funzione `socket()` vista sopra. Il descrittore di file dovrebbe riferirsi ad un socket, ma questo non viene controllato — le operazioni susseguenti sull'oggetto potrebbero fallire se il descrittore di file non è valido. Questa funzione è raramente impiegata, ma può essere usata per ricevere o impostare opzioni sui socket su un socket passato ad un programma come standard input o output (come in un server all'avvio, dal demone inet UNIX). Il socket si assume essere bloccante. Disponibilità: UNIX.

`ntohl`(*x*)
Converte interi di 32-bit dal byte order di rete a quello di host. Su macchine dove il byte order dell'host è lo stesso di quello di rete, non avviene nulla; altrimenti, esegue un operazione di swap a 4 byte.

`ntohs`(*x*)
Converte interi di 16-bit dal byte order di rete a quello di host. Su macchine dove il byte order dell'host è lo stesso di quello di rete, non avviene nulla; altrimenti, esegue un operazione di swap a 2 bit.

`htonl`(*x*)
Converte interi di 32-bit dal byte order dell'host a quello di rete. Su macchine dove il byte order dell'host è lo stesso di quello di rete, non avviene nulla; altrimenti, esegue un operazione di swap a 4 bit.

`htons`(*x*)
Converte interi di 16-bit dal byte order dell'host a quello di rete. Su macchine dove il byte order dell'host è lo stesso di quello di rete, non avviene nulla; altrimenti, esegue un operazione di swap a 2 bit.

inet_aton(*ip_string*)
Converte un indirizzo IPv4 dalla stringa in formato punteggiato (per esempio, '123.45.67.89') in un formato binario di pacchetto a 32-bit, come una stringa lunga 4 caratteri. Ciò risulta utile quando bisogna conversare con un programma che utilizza la libreria standard C e necessita di oggetti di tipo `struct in_addr`, che è il tipo C per i pacchetti binari 32-bit che questa funzione restituisce.

Se la stringa dell'indirizzo passata a questa funzione non è valida, verrà sollevata un'eccezione `socket.error`. Notare che la validità dipende dall'implementazione sottostante in C di `inet_aton()`.

`inet_aton()` non supporta IPv6, dovrebbe essere usata al suo posto `getnameinfo()` per il supporto al dual stack IPv4/v6.

inet_ntoa(*packed_ip*)
Converte l'indirizzo di un pacchetto a 32-bit (una stringa lunga 4 caratteri) nella rappresentazione standard in quartine-puntate (per esempio, '123.45.67.89'). Ciò risulta utile quando bisogna conversare con un programma che utilizza la libreria standard C e necessita di oggetti del tipo `struct in_addr`, che è il tipo C per i pacchetti binari 32-bit che questa funzione prende per argomento.

Se la stringa passata a questa funzione non è esattamente di 4 byte, verrà sollevata un'eccezione `socket.error`. `inet_ntoa()` non supporta l'IPv6, dovrebbe essere usata al suo posto `getnameinfo()` per il supporto al dual stack IPv4/v6.

inet_pton(*address_family, ip_string*)
Converte un indirizzo IP dal suo formato specifico di famiglia in un formato di pacchetto binario. `inet_pton()` è utile quando una libreria o un protocollo di rete richiedono un oggetto del tipo `struct in_addr` (simile ad `inet_aton()`) o `struct in6_addr`.

I valori supportati per *address_family* sono al momento `AF_INET` e `AF_INET6`. Se la stringa di indirizzo IP *ip_string* non è valida, verrà sollevata un'eccezione `socket.error`. Notare che la validità dipende sia dal valore di *address_family* che dall'implementazione sottostante di `inet_pton()`.

Disponibilità: UNIX (forse non tutte le piattaforme). Nuovo nella versione 2.3.

inet_ntop(*address_family, packed_ip*)
Converte un indirizzo IP di un pacchetto (una stringa di un qualche numero di caratteri) nella sua standard, specifica per la sua famiglia, stringa rappresentativa (per esempio, `'7.10.0.5'` or `'5aef:2b::8'`)). `inet_ntop()` è utile quando una libreria o protocollo di rete restituisce un oggetto di tipo `struct in_addr` (simile a `inet_ntoa()`) o `struct in6_addr`.

I valori supportati per *address_family* sono attualmente `AF_INET` e `AF_INET6`. Se la stringa *packed_ip* non è della lunghezza corretta per la specifica famiglia di indirizzi, sarà sollevata un'eccezione `ValueError`. Per gli errori provenienti dalla chiamata a `inet_ntop()` sarà sollevata un'eccezione `socket.error`.

Disponibilità: UNIX (forse non tutte le piattaforme). Nuovo nella versione 2.3.

getdefaulttimeout()
Restituisce il timeout predefinito in secondi floating per i nuovi oggetti socket. Il valore `None` indica che i nuovi oggetti socket non hanno timeout. Quando il modulo socket è appena importato, il suo valore predefinito è `None`. Nuovo nella versione 2.3.

setdefaulttimeout(*timeout*)
Imposta il timeout predefinito in secondi (numero a virgola mobile) per i nuovi oggetti socket. Il valore `None` indica che i nuovi oggetti socket non hanno timeout. Quando il modulo socket è appena importato, il suo valore predefinito è `None`. Nuovo nella versione 2.3.

SocketType
Questo è un tipo di oggetto Python che rappresenta il tipo oggetto socket. È lo stesso tipo di `type(socket(...))`.

Vedete anche:

Modulo `SocketServer` (sezione 11.15):
Classi che semplificano la scrittura di server di rete.

7.2.1 Oggetti Socket

Gli oggetti socket hanno i seguenti metodi. Fatta eccezione per `makefile()` questi corrispondono alle chiamate di sistema UNIX applicabili ai socket.

`accept``()`
Accetta una connessione. Il socket deve essere legato ad un indirizzo ed in attesa di connessioni. Il valore restituito è una coppia (*conn*, *address*) dove *conn* è un nuovo oggetto socket da usare per inviare e ricevere dati sulla connessione, mentre *address* è l'indirizzo legato al socket dell'altro capo della connessione.

`bind``(`*address*`)`
Lega il socket ad *address*. Il socket non deve essere già legato. (Il formato di *address* dipende dalla famiglia di indirizzo — leggere sopra.) **Note:** Questo metodo storicamente accettava una coppia di parametri per gli indirizzi `AF_INET` invece di una sola tupla. Ciò non è mai stato intenzionale e non è più disponibile in Python 2.0 e successive versioni dell'interprete.

`close``()`
Chiude il socket. Tutte le future operazioni sull'oggetto socket falliranno. L'altro capo non riceverà più alcun dato (dopo che la coda dei dati sarà svuotata). I socket sarano automaticamente chiusi quando verranno raccolti dal garbage-collector.

`connect``(`*address*`)`
Connette con un socket remoto all'indirizzo *address*. (Il formato di *address* dipende dalla famiglia di indirizzo — leggere sopra.) **Note:** Questo metodo storicamente accettava una coppia di parametri per gli indirizzi `AF_INET` invece di una sola tupla. Ciò non è mai stato intenzionale e non è più disponibile su Python 2.0 e successive versioni dell'interprete.

`connect_ex``(`*address*`)`
Come `connect(`*address*`)`, ma restituisce un indicatore di errore invece di sollevare un'eccezione, per errori restituiti dalla chiamata `connect()` a livello C (altri problemi, come "host not found", possono ancora sollevare eccezioni). L'indicatore di errore è `0` se l'operazione ha avuto successo, altrimenti viene indicato della variabile `errno`. Ciò torna utile per supportare, ad esempio, connessioni asincrone. **Note:** Questo metodo storicamente accettava una coppia di parametri per gli indirizzi `AF_INET` invece di una sola tupla. Ciò non è mai stato intenzionale e non è più disponibile su Python 2.0 e successivi.

`fileno``()`
Restituisce il descrittore di file del socket (uno small integer). Utile con la `select.select()`.

Sotto Windows lo small integer ritornato da questo metodo non può essere usato dove si può usare un descrittore di file (come `os.fdopen()`). UNIX non ha questa limitazione.

`getpeername``()`
Restituisce l'indirizzo remoto al quale il socket è connesso. Torna utile per trovare il numero di porta di un socket remoto IPv4/v6, ad esempio. (Il formato dell'indirizzo restituito dipende dalla famiglia di indirizzo — leggere sopra.) Su alcuni sistemi questa funzione non è supportata.

`getsockname``()`
Restituisce l'indirizzo stesso del socket. Torna utile per trovare il numero di porta di un socket IPv4/v6, ad esempio. (Il formato di un indirizzo restituito dipende dalla famiglia di indirizzo — leggere sopra.)

`getsockopt``(`*level, optname*`[`*, buflen*`])`
Restituisce il valore dell'opzione socket data (leggere la pagina di manuale UNIX *getsockopt*(2)). Le costanti simboliche di cui ha bisogno (`SO_*` etc.) sono definite in questo modulo. Se *buflen* è assente, viene considerata un'opzione intera ed il suo valore intero viene restituito dalla funzione. Se *buflen* è presente, specifica la lunghezza massima del buffer usata per ricevere l'opzione, e questo buffer è restituito come stringa. È compito del chiamante decodificare il contenuto del buffer (leggere il modulo built-in facoltativo `struct` per un modo per decodificare le strutture C codificate come stringhe).

`listen``(`*backlog*`)`
Attende connessioni fatte al socket. L'argomento *backlog* specifica il numero massimo di connessioni accodabili e dovrebbe essere almeno 1; il valore massimo dipende dal sistema (solitamente 5).

`makefile``([`*mode*`[`*, bufsize*`]])`
Restituisce un oggetto file associato al socket. (Gli oggetti file sono descritti nella sezione 2.3.9, "Oggetti File.") L'oggetto file si riferisce ad una versione `dup()`ped del descrittore file del socket, quindi l'oggetto

file e l'oggetto socket possono essere chiusi o raccolti dal garbage-collector indipendentemente. Il socket dovrebbe essere in modalità bloccante. Gli argomenti opzionali *mode* e *bufsize* sono interpretati nello stesso modo della funzione built-in `file()`; leggere "Funzioni built-in" (sezione 2.1) per ulteriori informazioni.

`recv`(*bufsize*[, *flags*])
Riceve dati dal socket. Il valore restituito è una stringa che rappresenta i dati ricevuti. Il massimo ammontare di dati da ricevere in una volta viene specificato da *bufsize*. Leggere la pagina di manuale UNIX *recv*(2) per informazioni sul significato dell'argomento facoltativo *flags*; il valore predefinito è zero.

`recvfrom`(*bufsize*[, *flags*])
Riceve dati dal socket. Il valore restituito è una coppia (*string*, *address*) dove *string* è una stringa che rappresenta i dati ricevuti e *address* è l'indirizzo del socket che ha inviato i dati. L'argomento opzionale *flags* ha lo stesso significato della `recv()` sopra. (Il formato di *address* dipende dalla famiglia di indirizzo — leggere sopra.)

`send`(*string*[, *flags*])
Invia dati al socket. Il socket deve essere connesso ad un socket remoto. L'argomento facoltativo *flags* ha lo stesso significato della `recv()` precedente. Restituisce il numero di byte inviati. Gli applicativi sono responsabili del controllo di tutti i dati che vengono trasmessi; se solo una parte dei dati viene inviata, l'applicativo deve tentare la consegna dei dati rimanenti.

`sendall`(*string*[, *flags*])
Invia dati al socket. Il socket deve essere connesso ad un socket remoto. L'argomento facoltativo *flags* ha lo stesso significato della `recv()` precedente. Diversamente da `send()`, questo metodo continua ad inviare dati da *string* fino a che tutti i dati vengano inviati o accada un errore. Viene restituito `None` in caso di successo. In caso di errore, viene sollevata un'eccezione, e non c'è modo di determinare quanti dati siano stati effettivamente inviati.

`sendto`(*string*[, *flags*], *address*)
Invia dati al socket. Il socket non dovrebbe essere connesso ad un socket remoto, visto che il socket di destinazione è specificato da address. L'argomento facoltativo *flags* ha lo stesso significato della `recv()` precedente. Restituisce il numero di byte inviati. (Il formato di *address* dipende dalla famiglia di indirizzo — leggere sopra.)

`setblocking`(*flag*)
Imposta la modalità bloccante o non-bloccante sul socket: se *flag* è 0, il socket è impostato come non-bloccante, altrimenti è in modalità bloccante. Inizialmente tutti i socket sono in modalità bloccante. In modalità non-bloccante, se una chiamata `recv()` non trova alcun dato, o se una chiamata `send()` non può avere subito a disposizione i dati, viene sollevata un'eccezione `error`; in modalità bloccante, la chiamata si ferma fino a che non è possibile procedere. `s.setblocking(0)` equivale a `s.settimeout(0)`; `s.setblocking(1)` equivale a `s.settimeout(None)`.

`settimeout`(*value*)
Imposta un timeout sulle operazioni di bloccaggio del socket. L'argomento *value* può essere un numero in virgola mobile non negativo espresso in secondi, o `None`. Se viene passato un numero in virgola mobile, le operazioni socket seguenti solleveranno un'eccezione `timeout` se scade il periodo di timeout prima che l'operazione venga completata. Impostando un timeout a `None` vengono disabilitati i timeout su ogni operazione. `s.settimeout(0.0)` equivale a `s.setblocking(0)`; `s.settimeout(None)` equivale a `s.setblocking(1)`. Nuovo nella versione 2.3.

`gettimeout`()
Restituisce il timeout in secondi in virgola mobile associato alle operazioni socket o `None` se non è impostato alcun timeout. Ciò riflette l'ultima chiamata a `setblocking()` o `settimeout()`. Nuovo nella versione 2.3.

Alcune note sui timeout ed il bloccaggio dei socket: un oggetto socket può essere in una di queste tre modalità: bloccante, non-bloccante, o timeout. I socket vengono sempre creati in modalità bloccante. In modalità bloccante, le operazioni sono bloccate fino al loro completamento. In modalità non-bloccante, le operazioni falliscono (con un errore che è sfortunatamente dipendente dal sistema) se non possono essere completati immediatamente. In modalità timeout le operazioni falliscono se non possono essere completate durante il timeout specificato per il socket. Il metodo `setblocking()` è semplicemente una scorciatoia per alcune chiamate a `settimeout()`.

La modalità timeout imposta internamente il socket in modalità non-bloccante. Le modalità bloccante e timeout sono condivise fra descrittori di file e oggetti socket che si riferiscono alla stessa estremità della rete. Una conseguenza di questo è che gli oggetti file ritornati dal metodo `makefile()` dovrebbero essere usati solo per socket in modalià bloccante; nelle modalità timeout e non bloccanti le operazioni sui file che non possono essere completate immediatamente falliranno.

Notare che l'operazione `connect()` è soggetta alle impostazioni di timeout, ed in generale viene raccomandato di chiamare la `settimeout()` prima di chiamare la `connect()`.

`setsockopt`(*level*, *optname*, *value*)
: Imposta il valore per la data opzione socket (leggere la pagina di manuale UNIX *setsockopt*(2)). Le costanti simboliche richieste sono definite nel modulo `socket` (`SO_*` ecc.). *value* può essere un intero o una stringa rappresentante un buffer. Nell'ultimo caso è compito del chiamante assicurarsi che la stringa contenga i bit adatti (leggere il modulo facoltativo built-in `struct` per un modo per codificare strutture C come stringhe).

`shutdown`(*how*)
: Chiude una o entrambe le metà della connessione. Se *how* è `SHUT_RD`, saranno disabilitate ulteriori ricezioni. Se *how* è `SHUT_WR`, saranno disabilitati ulteriori invii. Se *how* è `SHUT_RDWR`, verranno disabilitati entrambi.

Notare che non esiste alcuna `read()` o `write()`; usare `recv()` e `send()` senza argomenti *flags* al loro posto.

7.2.2 Oggetti SSL

Gli oggetti SSL possiedono i seguenti metodi.

`write`(*s*)
: Scrive la stringa *s* sulla connessione dell'oggetto SSL. Il valore restituito è il numero di byte scritti.

`read`([*n*])
: Se viene passato *n*, legge *n* byte dalla connessione SSL, altrimenti legge fino all'EOF. Il valore restituito è una stringa dei byte letti.

7.2.3 Esempio

Ecco quattro programmi di esempio minimali che usano il protocollo TCP/IP: un server che fa l'echo di tutti i dati ricevuti (servendo un unico client), ed un client che lo sfrutta. Notare che un server deve eseguire la sequenza `socket()`, `bind()`, `listen()`, `accept()` (possibilmente ripetendo la `accept()` per servire più di un client), mentre un client necessita della sola sequenza `socket()`, `connect()`. Notare inoltre che il server non effettua le `send()`/`recv()` sul socket sul quale sta ascoltando, ma sul nuovo socket restituito dalla `accept()`.

I primi due esempi supportano solo l'IPv4.

```
# Programma server Echo
import socket

HOST = ''                 # Nome simbolico che rappresenta il nodo locale
PORT = 50007              # Porta non privilegiata arbitraria
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1)
conn, addr = s.accept()
print 'Connected by', addr
while 1:
    data = conn.recv(1024)
    if not data: break
    conn.send(data)
conn.close()
```

```
# Programma client Echo
import socket

HOST = 'daring.cwi.nl'    # Il nodo remoto
PORT = 50007              # The La stessa porta usata dal server
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
s.send('Hello, world')
data = s.recv(1024)
s.close()
print 'Received', 'data'
```

I prossimi due esempi sono identici ai due sopra, ma supportano sia IPv4 che IPv6. Il lato server ascolterà sulla prima famiglia di indirizzi disponibile (dovrebbe ascoltare con entrambe). Su molti dei sistemi IPv6-ready, IPv6 avrà la precedenza ed il server potrebbe non accettare il traffico IPv4. Il lato client proverà a connettersi a tutti gli indirizzi restituiti come risultato della risoluzione del nome, ed invierà il suo traffico alla prima connessione utile.

```
# Programma server Echo
import socket
import sys

HOST = ''                 # Nome simbolico che rappresenta il nodo locale
PORT = 50007              # Porta non privilegiata arbitraria
s = None
for res in socket.getaddrinfo(HOST, PORT, socket.AF_UNSPEC, socket.SOCK_STREAM, 0, socket.AI
    af, socktype, proto, canonname, sa = res
    try:
s = socket.socket(af, socktype, proto)
    except socket.error, msg:
s = None
continue
    try:
s.bind(sa)
s.listen(1)
    except socket.error, msg:
s.close()
s = None
continue
    break
if s is None:
    print 'could not open socket'
    sys.exit(1)
conn, addr = s.accept()
print 'Connected by', addr
while 1:
    data = conn.recv(1024)
    if not data: break
    conn.send(data)
conn.close()
```

```
# Programma client Echo
import socket
import sys

HOST = 'daring.cwi.nl'    # Il nodo remoto
PORT = 50007              # La stessa porta usata dal server
s = None
for res in socket.getaddrinfo(HOST, PORT, socket.AF_UNSPEC, socket.SOCK_STREAM):
    af, socktype, proto, canonname, sa = res
    try:
s = socket.socket(af, socktype, proto)
    except socket.error, msg:
s = None
continue
    try:
s.connect(sa)
    except socket.error, msg:
s.close()
s = None
continue
    break
if s is None:
    print 'could not open socket'
    sys.exit(1)
s.send('Hello, world')
data = s.recv(1024)
s.close()
print 'Received', 'data'
```

7.3 `select` — Attesa del completamento dell'I/O

Questo modulo fornisce l'accesso alle funzioni `select()` e `poll()` disponibili su molti sistemi operativi. Notare che su Windows, funziona solo per i socket; su altri sistemi operativi, funziona anche per altri tipi di file (in particolare, su UNIX, funziona sulle pipe). Non può essere usato su file regolari per determinare se un file è cresciuto dall'ultima lettura.

Il modulo definisce le seguenti costanti e funzioni:

exception `error`
: L'eccezione sollevata in caso di errore. Il valore che la accompagna è una coppia contenente il codice numerico dell'errore da `errno` e la stringa corrispondente, come verrebbe stampato dalla funzione C `perror()`.

`poll``()`
: (Non supportata da tutti i sistemi operativi.) Restituisce un oggetto poll, che supporta la registrazione e deregistrazione di descrittori di file, e li registra per gli eventi di I/O; leggere la sezione 7.3.1 seguente, per i metodi supportati dagli oggetti poll.

`select``(`*iwtd, owtd, ewtd*[, *timeout*]`)`
: Questa è una interfaccia diretta alla chiamata di sistema UNIX `select()`. I primi tre argomenti sono sequenze di 'oggetti in attesa': quindi interi che rappresentano descrittori di file od oggetti con un metodo senza parametri chiamato `fileno()` che restituisce un tale intero. Le tre sequenze di oggetti in attesa sono rispettivamente per input, output e 'condizioni eccezionali'. Sono permesse sequenze vuote, ma tre sequenze vuote non vengono accettate da tutte le piattaforme. (Si sa che funziona su UNIX ma non su Windows.) L'argomento facoltativo di *timeout* specifica un *timeout* come un numero in virgola mobile in secondi. Quando l'argomento *timeout* è omesso, la funzione si blocca fino a che almeno un descrittore di file non sia pronto. Un valore di *timeout* uguale a zero specifica un poll che non si blocca mai.

  Il valore restituito è una terna di liste di oggetti pronti: reimpostazioni dei primi tre argomenti. Quando viene raggiunto il timeout senza che un descrittore di file sia pronto, vengono restituite tre liste vuote.

Tra i tipi di oggetti accettati nelle sequenze ci sono gli oggetti file Python, (e.g. `sys.stdin`, od oggtti ritornati dalla `open()` o `os.popen()`), oppure oggetti socket restituiti da `socket.socket()`. . Si può anche definire una propria classe *wrapper*, a patto che possegga un appropriato metodo `fileno()` (che restituisce un descrittore di file reale, non solo un intero a caso). **Note:** Gli oggetti file su Windows non sono accettati, solo i socket. Su Windows, la funzione sottostante `select()` viene fornita dalla libreria Winsock e non gestisce descrittori di file che non siano generati dal Winsock.

7.3.1 Registrare oggetti

La chiamata di sistema `poll()`, supportata su molti sistemi UNIX, fornisce migliore scalabilità per server di rete che devono servire moltissimi client contemporaneamente. `poll()` ha una scalabilità migliore perché la chiamata di sistema richiede solo la lista dei descrittori di file di interesse, mentre `select()` costruisce una bitmap, abilita i bit per i descrittori di file di interesse, e quindi di conseguenza l'intera bitmap deve essere analizzata linearmente di nuovo. `select()` è O(file descriptor più alto), mentre `poll()` è O(numero di file descriptor).

`register`(*fd*[, *eventmask*])
Registra un descrittore di file con l'oggetto poll. Chiamate future al metodo `poll()` controlleranno se il descrittore di file possiede eventi I/O in attesa. *fd* può essere un intero od un oggetto con un metodo `fileno()` che restituisce un intero. Gli oggetti file implementano `fileno()`, in modo che possano essere utilizzati anche come argomenti.

eventmask è una bitmask opzionale che descrive il tipo di eventi che si vogliono controllare, e può essere una combinazione delle costanti `POLLIN`, `POLLPRI` e `POLLOUT`, descritte nella tabella sottostante. Se non specificato, il valore predefinito usato controllerà tutti e tre 3 i tipi di eventi.

Costante	Significato
`POLLIN`	Non ci sono dati da leggere
`POLLPRI`	Ci sono dati urgenti da leggere
`POLLOUT`	Pronto per l'output: la scrittura non bloccherà
`POLLERR`	Condizione di errore di un qualche tipo
`POLLHUP`	Attesa
`POLLNVAL`	Richiesta non valida: descrittore non aperto

Registrare un descrittore di file che è già registrato non significa un errore, ha lo stesso effetto di registrare il descrittore una volta sola.

`unregister`(*fd*)
Rimuove un descrittore di file tracciato da un oggetto poll. Esattamente come il metodo `register()`, *fd* può essere un intero o un oggetto con un metodo `fileno()` che restituisce un intero.

Tentare di rimuovere un descrittore di file che non è mai stato registrato causerà il sollevamento dell'eccezione `KeyError`.

`poll`([*timeout*])
Effettua il poll sulla lista dei descrittori di file registrati e restituisce una lista, anche vuota, contenente tuple di 2 elementi (`(`*fd*`,` *event*`)`) per i descrittori che hanno eventi o errori da riportare. *fd* è il descrittore di file, mentre *event* è una bitmask con bit impostati per gli eventi riportati per quel descrittore — `POLLIN` per input in attesa, `POLLOUT` per indicare che sul descrittore vi si può scrivere, e cosi via. Una lista vuota indica che la chiamata è andata in *timeout* e nessun descrittore di file ha alcun evento da riportare. Se viene passato, *timeout* specifica il periodo di tempo in millisecondi durante il quale sistema aspetterà degli eventi prima di terminare la chiamata. Se *timeout* è omesso, negativo o `None`, la chiamata si bloccherà fino a che non apparirà un evento per questo oggetto poll.

7.4 `thread` — Controllo multi-thread

Questo modulo fornisce primitive di basso livello per lavorare con thread multipli (a.k.a. *processi o task leggeri* — thread multipli di controllo condividono il loro spazio di dati globale. Per la sincronizzazione, vengono forniti semplici lock (a.k.a. *semafori binari* o *mutualmente esclusivi*).

Il modulo è opzionale. È supportato su Windows, Linux, SGI IRIX, Solaris 2.x, e su ogni sistema che supporta l'implementazione thread POSIX (a.k.a. "pthread"). Ai sistemi che non possiedono il modulo `thread`, viene

fornito il modulo `dummy_thread`. Che duplica l'interfaccia di questo modulo e può essere usato come un rimpiazzamento drop-in.

Questo modulo definisce le seguenti costanti e funzioni:

exception `error`
Sollevata da errori specifici dei thread.

`LockType`
Questo è il tipo per gli oggetti lock.

`start_new_thread`(*function, args*[, *kwargs*])
Inizia un nuovo thread e restituisce il suo identificatore. Il thread esegue la funzione *function* con la lista di argomenti *args* (che deve essere una tupla). L'argomento opzionale *kwargs* specifica un dizionario di argomenti a parola chiave. Quando la funzione restituisce, il thread termina silenziosamente. Quando la funzione termina con una eccezione non gestita, viene stampata una traccia dello stack ed il thread termina (ma gli altri thread continuano a lavorare).

`interrupt_main`()
Solleva una eccezione `KeyboardInterrupt` nel thread principale. Un sotto-thread può usare questa funzione per interrompere il thread principale. Nuovo nella versione 2.3.

`exit`()
Solleva l'eccezione `SystemExit`. Quando non viene catturata, il thread termina silenziosamente.

`allocate_lock`()
Restituisce un nuovo oggetto lock. I metodi dei lock sono descritti sotto. Il lock è inizialmente sbloccato.

`get_ident`()
Restituisce l'identificatore di thread' del thread corrente. Questi è un intero diverso da zero. Il suo valore non ha un significato diretto; È da intendere come un magic cookie utilizzabile ad esempio come indice per un dizionario di dati specifici per i thread. Gli identificatori dei thread possono essere riciclati quando un thread termina ed un altro thread viene creato.

Gli oggetti lock hanno i seguenti metodi:

`acquire`([*waitflag*])
Senza argomento opzionale, questo metodo acquisisce il lock incondizionatamente, se necessario aspettando fino a che non venga rilasciato da un altro thread (solo un thread alla volta può acquisire un lock — è la ragione della loro esistenza), e restituisce `None`. Se l'argomento intero *waitflag* è presente, l'azione dipende dal suo valore: se è zero, il lock è acquisito solo se può essere acquisito immediatamente senza aspettare, mentre se è diverso da zero, il lock è acquisito incondizionatamente come prima. Se è presente un argomento, il valore restituito è `True` se il lock è acquisito con successo, `False` altrimenti.

`release`()
Rilascia il lock. Il lock deve essere stato acquisito in precedenza, ma non necessariamente dallo stesso thread.

`locked`()
Restituisce lo stato del lock: `True` se nel caso sia stato acquisito dal qualche thread, `False` altrimenti.

Avvertenze:

- I thread interagiscono in modo strano con gli interrupt: l'eccezione `KeyboardInterrupt` verrà ricevuta da un thread arbitrario. (Quando è disponibile il modulo `signal`, gli interrupt saranno ricevuti sempre dal thread principale).
- Chiamare `sys.exit()` o sollevare l'eccezione `SystemExit` equivale a chiamare `exit()`.
- Non tutte le funzioni built-in che si bloccano in attesa di I/O permettono ad altri thread di funzionare. (Le più famose (`time.sleep()`, *file*`.read()`, `select.select()`), funzionano come previsto).
- Non è possibile interrompere il metodo `acquire()` su un lock — l'eccezione `KeyboardInterrupt` verrà sollevata dopo che lock sarà acquisito.
- Quando termina il thread principale, la sopravvivenza degli altri thread dipende dal sistema. Su SGI IRIX

usando l'implementazione nativa dei thread, sopravvivono. Su molti altri sistemi, vengono uccisi senza eseguire le clausole `try` ... `finally` o eseguire i distruttori degli oggetti.

- Quando il thred principale termina, non esegue nessuno dei suoi soliti cleanup (eccetto per le clausole `try` ... `finally`, e i file standard di I/O non vengono aggiornati.

7.5 `threading` — Interfaccia ad alto livello per i thread

Questo modulo realizza un interfaccia ad alto livello per i thread sulla base del modulo a basso livello `thread`.

Nel caso in cui non si possano usare i `threading` a causa della mancanza del modulo `thread`, viene fornito il modulo `dummy_threading`.

Questo modulo definisce le seguenti funzioni ed oggetti:

`activeCount``()`
Restituisce il numero di oggetti `Thread` attivi al momento. Il contatore restituito è uguale alla lunghezza della lista restituita da `enumerate()`. Una funzione che restituisce il numero di thread attivi al momento.

`Condition``()`
Una funzione factory che restituisce un nuovo oggetto di variabile di condizione. Una variabile di condizione permette ad uno o più thread di attendere fino a che non vengano notificati da altri thread.

`currentThread``()`
Restituisce l'oggetto `Thread` corrente, corrispondente al thread di controllo del chiamante. Se il thread di controllo del chiamante non è stato creato attraverso il modulo `threading`, viene restituito un oggetto thread fantoccio con funzionalità limitate.

`enumerate``()`
Restituisce una lista di tutti gli oggetti `Thread` attivi al momento. La lista include demoni thread, oggetti thread fantocci creati da `currentThread()`, ed il thread principale. Vengono esclusi i thread terminati ed i thread che non sono ancora stati lanciati.

`Event``()`
Una funzione factory che restituisce un oggetto evento. Un evento gestisce una opzione che può essere impostata a vero con il metodo `set()` e resettata a falso con il metodo `clear()`. Il metodo `wait()` si bloccca fino a che l'opzione è vera.

`Lock``()`
Una funzione factory che restituisce un nuovo oggetto lock primitivo. Una volta che viene acquisito da un thread, i successivi tentativi di acquisirlo si bloccano, fino al suo rilascio; ogni thread lo può rilasciare.

`RLock``()`
Una funzione factory che restituisce un nuovo oggetto lock rientrante. Un lock rientrante deve essere rilasciato dal thread che lo ha acquisito. Una volta che un thread ha acquisito un lock rientrante, lo stesso thread può acquisirlo di nuovo senza bloccarsi; il thread deve effettuare un rilascio per ogni acquisizione.

`Semaphore``(`[*value*]`)`
Una funzione factory che restituisce un nuovo oggetto semaforo. Un semaforo gestisce un contatore che rappresenta il numero di chiamate a `release()` meno il numero di chiamate ad `acquire()`, più un valore *value* iniziale. Il metodo `acquire()` si blocca se necessario fino a che non possa restituire senza rendere il contatore negativo. Se non viene passato, *value* è inizialmente 1.

`BoundedSemaphore``(`[*value*]`)`
Una funzione factory che restituisce un nuovo oggetto semaforo limitato. un semaforo limitato controlla che il suo valore corrente non ecceda il suo valore iniziale. Se ciò avviene, viene sollevata un'eccezione `ValueError`. In molte situazioni i semafori sono usati per vegliare su risorse con capacità limitate. Se il semaforo è rilasciato troppe volte è segno di un bug. Se non viene passato, *value* è inizialmente 1.

class `Thread`
Una classe che rappresenta un thread di controllo. Questa classe può essere derivata in maniera sicura in modo limitato.

class `Timer`
Un thread che esegue una funzione dopo che sia passato un intervallo di tempo specificato.

`settrace`(*func*)
Imposta una funzione di traccia per tutti i thread lanciati dal modulo `threading`. *func* sarà passata a `sys.settrace()` per ogni thread, prima che il suo metodo `run()` sia chiamato. Nuovo nella versione 2.3.

`setprofile`(*func*)
Imposta una funzione di profilo per tutti i thread lanciati dal modulo `threading`. *func* sarà passata a `sys.setprofile()` per ogni thread, prima che il suo metodo `run()` sia chiamato. Nuovo nella versione 2.3.

Le interfacce dettagliate per gli oggetti sono documentate sotto.

Il design di questo modulo è vagamente basato sul modello di threading di Java. Tuttavia, dove il Java pone lock e variabili di condizione come funzionamento basilare di ogni oggetto, in Python sono oggetti separati. La classe `Thread` di Python supporta un sottoinsieme del funzionamento della classe Thread di Java; al momento, non esistono priorità o gruppi di thread, i thread non possono essere distrutti, fermati, sospesi, riattivati o interrotti. I metodi statici della classe Thread di Java, quando implementati, sono progettati come funzioni a livello di modulo.

Tutti i metodi descritti sotto sono eseguiti atomicamente.

7.5.1 Oggetti Lock

Un lock primitivo è una primitiva di sincronizzazione che non è posseduta da nessun particolare thread quando è bloccata. In Python, al momento è la primitiva di sincronizzazione di più basso livello disponibile, implementata direttamente dal modulo di estensione `thread`.

Un lock primitivo si trova in uno dei due stati, "bloccato" o "sbloccato". Viene creato nello stato sbloccato. Possiede due metodi di base, `acquire()` e `release()`. Quando lo stato è sbloccato. `acquire()` modifica lo stato a bloccato, e restituisce immediatamente. Quando lo stato è bloccato, `acquire()` si blocca fino che la chiamata `release()` in un altro thread lo modifichi a sbloccato, quindi la chiamata a `acquire()` lo reimposta in modo bloccato e restituisce. Il metodo `release()` dovrebbe essere chiamato solo nello stato bloccato; modifica lo stato in sbloccato e restituisce immediatamente. Quando più di un thread è bloccato in `acquire()`, in attesa che lo stato venga cambiato in sbloccato, solo un thread procede quando una chiamata a `release()` reimposta lo stato a sbloccato; quale dei thread in attesa procederà in seguito non è definito, e può variare a seconda dell'implementazione.

Tutti i metodi sono eseguiti atomicamente.

`acquire`([*blocking = 1*])
Acquisisce un lock, bloccante o non bloccante.

Quando viene invocato senza argomenti, si blocca fino a che il lock è sbloccato, quindi lo blocca e restituisce. Non c'è alcun valore restituito in questo caso.

Quando viene invocato con l'argomento *blocking* impostato a vero, esegue le stesse cose di quando viene chiamato senza argomenti, e restituisce vero.

Quando viene invocato con l'argomento *blocking* impostato a falso, non si blocca. Se una chiamata senza un argomento si bloccherebbe, restituisce falso immediatamente; altrimenti, esegue le stesse azioni di quando viene chiamato senza argomenti, e restituisce vero.

`release`()
Rilascia un lock.

Quando un lock è bloccato, lo reimposta come sbloccato, e restituisce. Se qualunque altro thread è bloccato in attesa che il lock si sblocchi, permette solo a uno di loro di procedere.

Non chiamare questo metodo quando il lock è sbloccato.

Non esiste valore di ritorno.

7.5.2 Oggetti RLock

Un lock rientrante è una primitiva di sincronizzazione che può essere acquisita più volte dallo stesso thread. Internamente, utilizza il concetto di "thread proprietario" e "livello di ricorsione" in aggiunta agli stati bloccato/sbloccato usate dai lock primitivi. Nello stato bloccato, alcuni thread possiedono il lock; nello stato sbloccato, esso non è posseduto da alcun thread.

Per bloccare il lock, un thread chiama il suo metodo `acquire()`; questi restituisce una volta il thread proprietario del lock. Per sbloccare il lock, un thread chiama il suo metodo `release()`. La coppia di chiamate `acquire()`/`release()` può essere nidificata; solo la chiamata finale a `release()` (la `release()` della coppia più esterna) reimposta il lock a sbloccato e permette ad un altro thread bloccato in `acquire()` a procedere.

`acquire`([*blocking = 1*])

Acquisisce il lock, bloccante o non bloccante.

Quando viene invocato senza argomenti: se questo thread possiede già il lock, incrementa il livello di ricorsione di uno, e restituisce immediatamente. Altrimenti, se un altro thread possiede il lock, si blocca fino a che il lock non venga sbloccato. Una volta che il lock è sbloccato (non posseduto da alcun thread), ne ottiene la proprietà, ne imposta il valore di ricorsione ad uno e ritorna. Se più di un thread è bloccato in attesa dello sblocco del lock, solo un thread allo volta potrà ottenere il possesso del lock. Non viene restituito alcun valore in questo caso.

Quando viene invocata con l'argomento *blocking* impostato a vero, esegue le stesse azioni di quando viene chiamata senza argomenti, e restituisce vero.

Quando viene invocata con l'argomento *blocking* impostato a falso, non si blocca. Se una chiamata senza un argomento si bloccherebbe, restituisce falso immediatamente; altrimenti, fa le stesse azioni di quando viene chiamata senza argomenti, e restituisce vero.

`release`()

Rilascia il lock, decrementando il livello di ricorsione. Se, dopo il decremento, è zero, reimposta il lock a sbloccato (non posseduto da alcun thread), e se ogni altro thread è bloccato in attesa che il lock si sblocchi, permette esattamente a uno di loro a procedere. Se dopo il decremento il livello di ricorsione è ancora diverso da zero, il lock rimane bloccato e posseduto dal thread chiamante.

Chiamate questo metodo solo quando il thread chiamante possiede il lock. Non chiamate questo metodo quando il lock è sbloccato.

Non c'è alcun valore restituito.

7.5.3 Oggetti di condizione

Una variabile di condizione è sempre associata ad alcuni tipi di lock; questa può venir passata oppure creata automaticamente. (Passarne una torna utile quando alcune variabili di condizione devono condividere lo stesso lock).

Una variabile di condizione possiede i metodi `acquire()` e `release()` che chiamano i metodi corrispondenti al lock associato. Possiede anche i metodi `wait()`, `notify()` e `notifyAll()`. Questi tre devono essere chiamati solo quando il thread chiamante ha acquisito il lock.

Il metodo `wait()` rilascia il lock, e si blocca fino a che non venga risvegliato da una chiamata a `notify()` o `notifyAll()` per la stessa variabile di condizione in un altro thread. Una volta risvegliato, riacquisisce il lock e termina. È anche possibile specificare un timeout.

Il metodo `notify()` risveglia uno dei thread in attesa della variabile di condizione, se ce ne sono. Il metodo `notifyAll()` risveglia tutti i thread in attesa della variabile di condizione.

Nota: i metodi `notify()` e `notifyAll()` non rilasciano il lock; questo significa che il thread o i thread risvegliati, non restituiranno dalla loro chiamata `wait()` immediatamente, ma solo quando il thread che ha chiamato la `notify()` o la `notifyAll()` abbandona alla fine la proprietà del lock.

Consiglio: usando le variabili di condizione lo stile di programmazione tipico si serve dei lock per sincronizzare l'accesso ad alcuni stati condivisi; i thread che sono interessati in un particolare cambio di stato chiamano la `wait()` ripetutamente fino a che vedano lo stato desiderato, mentre i thread che vogliono modificare lo stato

chiamano `notify()` o `notifyAll()` in modo che possa essere uno stato desiderato da uno dei thread in attesa. Per esempio, il codice seguente è una situazione generica di produttore-consumatore con buffer di capacità illimitata:

```
# Consuma un elemento
cv.acquire()
while not an_item_is_available():
    cv.wait()
get_an_available_item()
cv.release()

# Produce un elemento
cv.acquire()
make_an_item_available()
cv.notify()
cv.release()
```

Per scegliere fra `notify()` e `notifyAll()`, considerare se un cambio di stato può essere interessato a solo uno o più thread in attesa. Per esempio in una tipica situazione di produttore-consumatore, aggiungere un oggetto al buffer necessita solo del risveglio di uno dei thread consumatori.

class `Condition`(*[lock]*)
Se l'argomento *lock* è dato e diverso da `None`, deve essere un oggetto `Lock` o `RLock`, ed è usato come il lock sottostante. Altrimenti, viene creato un nuovo oggetto `RLock` ed usato come il lock sottostante.

`acquire`(**args*)
Acquisisce il lock sottostante. Questo metodo chiama il metodo corrispondente al lock sottostante; il valore restituito è qualsiasi cosa il metodo restituisca.

`release`()
Rilascia il lock sottostante. Questo metodo chiama il metodo corrispondente al lock sottostante; non c'è alcun valore restituito.

`wait`(*[timeout]*)
Aspetta fino a che non venga notificato o sopraggiunga un timeout. Deve essere chiamato solo quando il thread chiamante ha acquisito il lock.

Il metodo rilascia il lock sottostante, e si blocca fino a che non sia risvegliato da una chiamata a `notify()` o `notifyAll()` per la stessa variabile di condizione in un altro thread, o fino a che non giunga il timeout facoltativo. Una volta risvegliato o andato in timeout, riacquisisce il lock e termina.

Quando l'argomento *timeout* è presente e diverso da `None`, dovrebbe essere un numero in virgola mobile che specifica un timeout in secondi per l'operazione (o frazioni di secondo).

Quando il lock sottostante è un `RLock`, non viene rilasciato usando il suo metodo `release()`, visto che ciò potrebbe non sbloccare realmente il lock quando questo è stato acquisito più volte ricorsivamente. Invece, viene usata un'interfaccia interna della classe `RLock`, che lo sblocca realmente anche quando è stato acquisito ricorsivamente alcune volte. Un'altra interfaccia interna è quindi usata per ripristinare il livello di ricorsione quando il lock viene riacquisito.

`notify`()
Risveglia un thread in attesa di questa condizione, se ce ne sono. Deve essere chiamata solo quando il thread chiamante ha acquisito il lock.

Questo metodo risveglia uno dei thread in attesa della variabile di condizione, se ce ne sono; non effettua alcuna operazione se non ci sono thread in attesa.

L'implementazione corrente risveglia esattamente un thread, se ce ne sono in attesa. Comunque, non è sicuro fidarsi di questo comportamento. Un'implementazione futura ottimizzata potrà occasionalmente risvegliare più di un thread.

Nota: Il thread risvegliato non restituisce realmente dalla sua chiamata `wait()` fino a che non possa riacquisire il lock. Visto che la `notify()` non rilascia il lock, lo dovrebbe fare il chiamante.

`notifyAll`()

Risveglia tutti i thread in attesa di questa condizione. Questo metodo agisce come `notify()`, ma risveglia tutti i thread in attesa invece di uno solo.

7.5.4 Oggetti Semaforo

Questa è una delle più vecchie primitive di sincronizzazione nella storia dell'informatica, inventata dallo scienziato olandese Edsger W. Dijkstra (egli usava `P()` e `V()` invece di `acquire()` e `release()`).

Un semaforo manipola un contatore interno che viene decrementato da ogni chiamata ad `acquire()` ed incrementato da ogni chiamata a `release()`. Il contatore non può mai andare sotto lo zero; quando `acquire()` lo trova a zero, si blocca, aspettando che qualche altro thread chiami `release()`.

class `Semaphore`(\[*value*\])
L'argomento facoltativo indica il valore iniziale del contatore interno; il valore predefinito è `1`.

`acquire`(\[*blocking*\])
Acquisisce un semaforo.

Quando invocato senza argomenti: se il contatore interno è più grande di zero all'entrata, lo decrementa di uno e termina immediatamente. Se è zero all'entrata, si blocca, aspettando sino a che qualche altro thread abbia chiamato `release()` per renderlo più grande di zero. Ciò viene fatto con un interlocking appropriato così che se più chiamate ad `acquire()` sono bloccate, `release()` ne risveglierà esattamente una. L'implementazione ne può prendere uno a caso, perciò non si può fare affidamento sull'ordine nel quale i thread bloccati vengono risvegliati. Non viene restituito alcun valore in questo caso.

Quando viene invocato con *blocking* impostato a vero, fa le stesse cose di quando viene chiamato senza argomenti, e restituisce vero.

Quando viene invocato con *blocking* impostato a falso, non si blocca. Se una chiamata senza un argomento si bloccherebbe, restituisce falso immediatamente; altrimenti, fa la stessa cosa di quando viene chiamato senza argomenti, e restituisce vero.

`release`()
Rilascia un semaforo, incrementando il contatore interno di uno. Quando era zero all'entrata e un altro thread sta aspettando che diventi più grande di zero di nuovo, risveglia quel thread.

Esempio `Semaphore`

I semafori sono spesso usati per gestire risorse con capacità limitate, ad esempio, un database server. In ogni situazione dove la dimensione della risorsa è fissata, si dovrebbe usare un semaforo limitato. Prima di generare ogni thread, il thread principale inizializzerebbe il semaforo:

```
maxconnections = 5
...
pool_sema = BoundedSemaphore(value=maxconnections)
```

Una volta generati, i thread chiamano i metodi acquire e release del semaforo quando devono connettersi al server:

```
pool_sema.acquire()
conn = connectdb()
... use connection ...
conn.close()
pool_sema.release()
```

L'uso di un semaforo limitato riduce la possibilità che un errore di programmazione provocato dal maggior numero di rilasci del semaforo rispetto a quello delle sue acquisizioni non venga rilevato.

7.5.5 Oggetti evento

Questo è uno dei più semplici meccanismi di comunicazione fra thread: un thread segnala un evento ed altri thread lo attendono.

Un oggetto evento manipola una opzione interna che può essere impostata a vero con il metodo `set()` e reimpostata a falso con il metodo `clear()`. Il metodo `wait()` si blocca fino a che l'opzione è vera.

class `Event()`
L'opzione interna è inizialmente falsa.

`isSet()`
Restituisce vero se e solo se l'opzione interna è vera.

`set()`
Imposta l'opzione interna a vero. Tutti i thread in attesa che diventi vera vengono risvegliati. I thread che chiamano `wait()` una volta che l'opzione è diventata vera non si bloccheranno.

`clear()`
Reimposta l'opzione interna a falso. Di conseguenza, i thread che chiamano `wait()` si bloccheranno fino a che `set()` venga chiamata per impostare l'opzione interna di nuovo a vero.

`wait(`[*timeout*]**`)`**
Si blocca fino a che l'opzione interna diventa vera. Se l'opzione interna è vera all'inizio, restituisce immediatamente. Altrimenti, si blocca fino a che un altro thread chiama `set()` per impostare l'opzione a vero o fino a che non sopravvenga il timeout facoltativo.

Quando l'argomento timeout è presente e diverso da `None`, dovrebbe essere un numero in virgola mobile che specifica un timeout per l'operazione in secondi (o frazioni di questi).

7.5.6 Oggetti thread

Questa classe rappresenta un attività che funziona in un thread di controllo separato. Esistono due modi per specificare l'attività: passando un oggetto chiamabile al costruttore o sovrascrivendo il metodo `run()` in una sottoclasse. Nessun altro metodo (eccetto per il costruttore) dovrebbe essere sovrascritto in una sottoclasse. In altre parole, sovrascrivere *solo* i metodi `__init__()` e `run()` di questa classe.

Una volta che l'oggetto thread è stato creato, la sua attività deve essere iniziata chiamando il metodo `start()` del thread. Questi invoca il metodo `run()` in un thread di controllo separato.

Una volta che l'attività del thread è iniziata, il thread è considerato 'vivo' e 'attivo' (questi concetti sono quasi, ma non proprio, gli stessi; la loro definizione è intenzionalmente qualcosa di vago). Smette di essere vivo e attivo quando il suo metodo `run()` termina – o normalmente, o sollevando un'eccezione non gestita. Il metodo `isAlive()` controlla se il thread è vivo.

Altri thread possono chiamare il metodo `join()` di un altro thread. Questo blocca il thread chiamante fino a che termini il thread il cui metodo `join()` è stato chiamato.

Un thread ha un nome. Il nome può essere passato al costruttore, impostato con il metodo `setName()` e ottenuto con il metodo `getName()`.

Un thread può essere etichettato come un "thread demone". Il significato di questa etichetta è che l'intero programma Python termina quando sono rimasti solo thread demoni. Il valore iniziale viene ereditato dal thread creatore. L'etichetta può essere impostata con il metodo `setDaemon()` ed ottenuta con il metodo `isDaemon()`.

Esiste un oggetto "thread principale"; questo corrisponde al thread iniziale di controllo nel programma Python. Non è un thread demone.

Esiste la possibilità che vengano creati "oggetti thread dummy". Essi sono oggetti thread corrispondenti a "thread alieni". Questi ultimi sono thread di controllo nati all'esterno del modulo threading, come quelli generati direttamente dal codice C. Gli oggetti thread dummy hanno funzionalità limitate; sono sempre considerati vivi, attivi e demonici, e su di essi non si può fare il `join()`. Essi non vengono mai cancellati, visto che è impossibile identificare la terminazione dei thread alieni.

class **`Thread`**(*group=None, target=None, name=None, args=(), kwargs={}*)
Questo costruttore dovrebbe essere chiamato sempre con argomenti a parole chiave. Gli argomenti sono:

group dovrebbe essere `None`; È riservato per estensioni future quando una classe `ThreadGroup` sarà implementata.

target è l'oggetto chiamabile che sarà invocato dal metodo `run()`. Predefinito è `None`, che significa che nulla viene chiamato.

name è il nome del thread. Automaticamente, viene costruito un nome univoco nella forma "Thread-*N*" dove *N* è un piccolo numero decimale.

args è la tupla di argomenti per invocare target. Il valore predefinito è `()`.

kwargs è un dizionario di argomenti a parola chiave per l'invocazione target. Il valore predefinito è `{}`.

Se la sottoclasse sovrascrive il costruttore, deve essere sicura di invocare il costruttore della classe base (`Thread.__init__()`) prima di fare qualsiasi altre cosa al thread.

`start`()
Inizia l'attività del thread.

Deve essere chiamato almeno una volta per oggetto thread. Si occupa di far invocare il metodo `run()` dell'oggetto in un thread di controllo separato.

`run`()
Metodo che rappresenta l'attività del thread.

Si può sovrascrivere questo metodo in una sottoclasse. Il metodo standard `run()` invoca l'oggetto chiamabile passato al costruttore dell'oggetto come argomento *target*, se è stato passato, con argomenti sequenziali ed a parola chiave presi rispettivamente dagli argomenti *args* e *kwargs*.

`join`([*timeout*])
Attende fino al termine del thread. Questo blocca il thread chiamante fino a che termini il thread il cui metodo `join()` è stato chiamato – normalmente, o attraverso un'eccezione non gestita – o fino a che sopravvenga il timeout facoltativo.

Quando l'argomento *timeout* è presente e diverso da `None`, dovrebbe essere un numero in virgola mobile che specifica un timeout per l'operazione in secondi (o una frazione di questi).

Su un thread può essere chiamata `join()` più volte.

Un thread non può effettuare `join()` su se stesso perché questo causerebbe un deadlock.

Tentare di fare la `join()` su un thread prima che sia stato lanciato rappresenta un errore.

`getName`()
Restituisce il nome del thread.

`setName`(*name*)
Imposta il nome del thread.

Il nome è una stringa usata per soli scopi identificativi. Non ha semantica. Thread multipli possono avere lo stesso nome. Il nome iniziale è impostato dal costruttore.

`isAlive`()
Restituisce vero se il thread è vivo.

Detto brutalmente, un thread è vivo dal momento in cui il metodo `start()` ritorna fino a che termina il suo metodo `run()`.

`isDaemon`()
Restituisce l'opzione del thread demone.

`setDaemon`(*daemonic*)
Imposta l'opzione del thread demone al valore booleano *daemonic*. Questo metodo deve essere chiamato prima che sia chiamato il metodo `start()`.

Il valore iniziale è ereditato dal thread creatore.

L'intero programma Python esce quando non è rimasto alcun thread non-demonico.

7.5.7 Oggetti timer

Questa class rappresenta un'azione che dovrebbe essere lanciata solo dopo che sia passato un certo ammontare di tempo — un timer. `Timer` è una sottoclasse di `Thread` e come tale funziona anche come esempio per la creazione di thread personalizzati.

I Timer sono lanciati, come i thread, chiamando il loro metodo `start()`. Il timer può essere fermato (prima che la sua azione sia iniziata) chiamando il metodo `cancel()`. L'intervallo che il timer attenderà prima di eseguire la sua azione può non essere esattamente lo stesso dell'intervallo specificato dall'utente.

Per esempio:

```
def hello():
    print "hello, world"

t = Timer(30.0, hello)
t.start() # dopo 30 secondi, sarà stampato "hello, world"
```

class `Timer`(*interval, function, args=[], kwargs={}***)**
Crea un timer che lancerà *function* con *args* come argomenti e *kwargs* come argomenti a parola chiave, dopo che *interval* secondi siano passati.

`cancel`()
Ferma il timer, e cancella l'esecuzione dell'azione del timer. Questo funzionerà solo se il timer è ancora nel suo momento di attesa.

7.6 `dummy_thread` — Rimpiazzamento drop-in per il modulo `thread`

Questo modulo fornisce un'interfaccia duplicata per il modulo `thread`. È pensato per essere importato quando il modulo `thread` non viene fornito su una piattaforma.

L'utilizzo consigliato è:

```
try:
    import thread as _thread
except ImportError:
    import dummy_thread as _thread
```

Fare attenzione a non usare questo modulo dove possono intervenire dei deadlock da thread creati, che si bloccano in attesa che altri thread vengano creati. Questo accade spesso con l'I/O bloccante.

7.7 `dummy_threading` — Rimpiazzamento drop-in per il modulo `threading`

Questo modulo fornisce un'interfaccia duplicata per il modulo `threading`. È pensato per essere importato quando il modulo `thread` non viene fornito su una piattaforma.

L'utilizzo consigliato è:

```
try:
    import threading as _threading
except ImportError:
    import dummy_threading as _threading
```

Fare attenzione a non usare questo modulo dove possono avvenire dei deadlock causati da un thread creato in attesa che venga creato un altro thread. Questo accade spesso con l'I/O bloccante.

7.8 Queue — Una classe coda sincronizzata

Il modulo `Queue` implementa una coda FIFO multiproduttore e multiconsumatore. È utile specialmente nella programmazione dei thread quando le informazioni devono essere scambiate in modo sicuro fra multipli thread. La classe `Queue` in questo modulo implementa tutte le semantiche dei lock richieste. Dipende dalla disponibilità del supporto thread in Python.

Il modulo `Queue` definisce le seguenti classi ed eccezioni:

class `Queue`(*maxsize*)
Costruttore per la classe. *maxsize* è un intero che imposta il limite massimo di elementi inseribili nella coda. L'inserimento verrà bloccato una volta raggiunto questo limite, fino a che gli elementi della coda siano esauriti. Se *maxsize* è minore o uguale a zero, la dimensione della coda è infinita.

exception `Empty`
Eccezione sollevata quando viene chiamata la `get()` non bloccante (o `get_nowait()`), su un oggetto `Queue` che è vuoto o bloccato.

exception `Full`
Eccezione sollevata quando viene chiamata la `put()` non bloccante (o `put_nowait()`) su un oggetto `Queue` che è pieno o bloccato.

7.8.1 Oggetti Queue

La classe `Queue` implementa gli oggetti coda e possiede i metodi descritti qui sotto. Da questa classe si può derivare in modo da ottenere altre organizzazioni di coda (e.g. stack) ma l'interfaccia ereditabile non è qui descritta. Leggere il codice sorgente per i dettagli. I metodi pubblici sono:

`qsize``()`
Restituisce la dimensione approssimativa della coda. A causa delle semantiche multithread, questo valore non è attendibile.

`empty``()`
Restituisce `True` se la coda è vuota, `False` altrimenti. A causa delle semantiche multithread, questo valore non è attendibile.

`full``()`
Restituisce `True` se la coda è piena, `False` altrimenti. A causa delle semantiche multithread, questo numero non è attendibile.

`put``(`*item*`[`, *block*`[`, *timeout*`]])`
Aggiunge *item* nella coda. Se l'argomento opzionale *block* è vero e *timeout* è None (predefinito), si blocca se necessario, finché non sia disponibile una posizione libera. Se *timeout* è un numero positivo, si blocca fino allo scadere di *timeout* secondi e solleva l'eccezione `Full` se nessuna posizione libera si è resa disponibile in quel periodo. Altrimenti (*block* è falso), aggiunge un elemento nella coda se una posizione libera è immediatamente disponibile, altrimenti solleva l'eccezione `Full` (*timeout* viene, in quel caso, ignorato).

Nuovo nella versione 2.3: il parametro timeout.

`put_nowait``(`*item*`)`
Equivalente a `put(`*item*`, False)`.

`get``([`*block*`[`, *timeout*`]])`
Rimuove e restituisce un elemento dalla coda. Se l'argomento facoltativo *block* è vero e *timeout* è None (predefinito), si blocca se necessario fino a che un elemento sia disponibile. Se *timeout* è un numero positivo, si blocca per *timeout* secondi e solleva l'eccezione `Empty` se nessun elemento si è reso disponibile in quel periodo. Altrimenti (*block* è falso), restituisce un elemento se ce n'è uno immediatamente disponibile, altrimenti solleva l'eccezione `Empty` (*timeout* viene ignorato in questo caso).

Nuovo nella versione 2.3: il parametro timeout.

`get_nowait``()`
Equivalente a `get(False)`.

7.9 `mmap` — Supporto per i file mappati in memoria

Gli oggetti file mappati in memoria si comportano sia come oggetti stringhe che come oggetti file. Diversamente dai normali oggetti stringa, questi sono mutabili. Si possono usare gli oggetti mmap in molti posti nei quali ci si aspettano delle stringhe; per esempio, si può usare il modulo `re` per ricercare all'interno di un file mappato in memoria. Visto che sono mutabili, si può modificare un singolo carattere con `obj[`*index*`] = 'a'` o modificare una sottostringa assegnandola ad una fetta: `obj[`*i1*`:`*i2*`] = '...'`. Si possono inoltre leggere e scrivere dati iniziando dalla posizione corrente nel file, ed usare `seek()` per spostarsi in posizioni differenti.

Un file mappato in memoria viene creato attraverso la funzione `mmap()`, che è differente su UNIX e Windows. In entrambi i casi si deve fornire un descrittore di file di un file aperto per la modifica. Se si vuole mappare un oggetto file Python esistente, si usi il suo metodo `fileno()` per ottenere il valore corretto del parametro *fileno*. Altrimenti, si può aprire il file usando la funzione `os.open()`, che restituisce direttamente un descrittore di file (il file necessita comunque di essere chiuso al termine).

Per entrambe le versioni UNIX e Windows della funzione, *access* può essere specificato come un parametro facoltativo a parola chiave. *access* accetta uno di questi tre valori: `ACCESS_READ`, `ACCESS_WRITE` o `ACCESS_COPY` per specificare rispettivamente sola lettura, scrittura o memoria copy-on-write. *access* può essere usato sia su UNIX che su Windows. Se *access* non viene specificato, la mmap di Windows restituisce una mappatura in scrittura. I valori iniziali di memoria per tutti i tre tipi di accesso sono presi dal file specificato. L'assegnamento ad una mappa di memoria `ACCESS_READ` solleva l'eccezione `TypeError`. L'assegnamento ad una mappa di memoria `ACCESS_WRITE` agisce sia sulla memoria che sul file sottostante. L'assegnamento ad una mappa di memoria `ACCESS_COPY` agisce unicamente sulla memoria, senza aggiornare il file sottostante.

`mmap``(`*fileno, length*`[`*, tagname*`[`*, access*`]])`
(Versione Windows) Mappa *length* byte dal file specificato dal gestore di file *fileno*, e restituisce un oggetto mmap. Se *length* è `0`, la lunghezza massima della mappa sarà la dimensione corrente del file al momento della chiamata a `mmap()`.

tagname, se specificato e diverso da `None`, è una stringa che fornisce un nome di tag per la mappatura. Windows vi permette di avere diverse mappature riferite allo stesso file. Se specificate il nome di un tag esistente, quel tag viene aperto, altrimenti viene creato un nuovo tag con questo nome. Se questo parametro viene omesso o è `None`, la mappatura viene creata senza nome. Evitando di usare il parametro tag il codice rimarrà portabile fra UNIX e Windows.

`mmap``(`*fileno, length*`[`*, flags*`[`*, prot*`[`*, access*`]]])`
(UNIX **version)** Mappa *length* byte dal file specificato dal descrittore di file *fileno*, e restituisce un oggetto mmap.

flags specifica la natura della mappatura. `MAP_PRIVATE` crea una mappatura copy-on-write privata, perciò la modifica del contenuto dell'oggetto mmap rimarrà privata a questo processo, mentre `MAP_SHARED` crea una mappatura condivisa con tutti gli altri processi che stanno mappando le stesse aree del file. Il valore predefinito è `MAP_SHARED`.

prot, se specificato, fornisce la protezione di memoria desiderata; i due valori più utili sono `PROT_READ` e `PROT_WRITE`, per specificare che le pagine possono essere lette o scritte. *prot*, il valore predefinito è `PROT_READ | PROT_WRITE`.

access può essere specificato al posto di *flags* e *prot* come un argomento a parola chiave facoltativo. Specificare *flags*, *prot* e *access* rappresenta un errore. Leggere la descrizione qui sopra di *access* per delle informazioni su come gestire questo parametro.

I file mappati in memoria supportano i seguenti metodi:

`close``()`
Chiude il file. Chiamate successive ad altri metodi dell'oggetto solleveranno un'eccezione.

`find``(`*string*`[`*, start*`])`

Restituisce l'indice più basso nell'oggetto in cui viene trovata la sottostringa *string*. Restituisce `-1` in caso di fallimento. *start* è l'indice dal quale inizia la ricerca, il valore predefinito è zero.

`flush`(*[offset, size]*)
Aggiorna le modifiche fatte alla copia in memoria del file su disco. Senza usare questa chiamata non c'è garanzia che le modifiche vengano scritte sul disco prima che l'oggetto venga distrutto. Se *offset* e *size* vengono specificati, solo le modifiche sull'intervallo dato di byte verranno scritte sul disco; altrimenti, viene aggiornata l'intera estensione della mappatura.

`move`(*dest, src, count*)
Copia *count* byte partendo da *src* all'indice di destinazione *dest*. Se la mmap è stata creata con `ACCESS_READ`, la chiamata a move solleverà l'eccezione `TypeError`.

`read`(*num*)
Restituisce una stringa contenente almeno *num* byte partendo dalla posizione corrente nel file; la posizione nel file viene aggiornata in modo da puntare dopo i byte che sono stati restituiti.

`read_byte`()
Restituisce una stringa di lunghezza 1 contenente il carattere alla posizione corrente nel file, e la fa avanzare di 1.

`readline`()
Restituisce una singola riga, partente dalla posizione corrente nel file, ed avanza alla prossima nuova riga.

`resize`(*newsize*)
Se la mmap è stata creata con `ACCESS_READ` o `ACCESS_COPY`, ridimensionare la mappa solleverà un'eccezione `TypeError`.

`seek`(*pos[, whence]*)
Imposta la posizione corrente nel file, l'argomento *whence* è facoltativo e il valore predefinito è `0` (posizione assoluta nel file); altri valori sono `1` (relativa alla posizione corrente nel file) e `2` (relativa alla fine del file).

`size`()
Restituisce la lunghezza del file, che può essere più grande della dimensione dell'area della mappa di memoria.

`tell`()
Restituisce la posizione corrente del puntatore del file.

`write`(*string*)
Scrive i byte in *string* nella memoria alla posizione corrente del puntatore del file; la posizione nel file è aggiornata per puntare dopo i byte che sono stati scritti. Se la mmap è stata creata con `ACCESS_READ`, scrivervi solleverà un'eccezione `TypeError`.

`write_byte`(*byte*)
Scrive la stringa di un solo carattere *byte* nella memoria, alla posizione corrente del puntatore nel file; la posizione nel file è avanzata di `1`. Se la mmap è stata creata con `ACCESS_READ`, scrivervi solleverà un'eccezione `TypeError`.

7.10 `anydbm` — Accesso generico ai database in stile DBM

`anydbm` è un'interfaccia generica alle varianti del database DBM – `dbhash` (necessita di `bsddb`), `gdbm` o `dbm`. Se nessuno di questi moduli è stato installato, verrà usata l'implementazione `dumbdbm` (lenta ma semplice).

`open`(*filename[, flag[, mode]]*)
Apre il file database *filename* e restituisce un oggetto corrispondente.

Se il file database già esiste, verrà usato il modulo `whichdb` per determinare il suo tipo e userà il modulo appropriato; se non esiste, verrà usato il primo modulo disponibile elencato precedentemente.

L'argomento facoltativo *flag* può essere `'r'` per aprire un database esistente in sola lettura, `'w'` per aprire un database esistente per lettura e scrittura, `'c'` per creare il database se non esiste, o `'n'`, che creerà sempre un database vuoto. Se non viene specificato, il valore predefinito sarà `'r'`.

L'argomento facoltativo *mode* è il modo UNIX del file, usato solo quando il database deve essere creato. Il valore predefinito è l'ottale `0666` (e sarà modificato dalla umask predominante).

exception `error`
Una tupla contenente le eccezioni che possono essere sollevate da ognuno dei moduli supportati, con un'unica eccezione `anydbm.error` come primo elemento — il secondo viene usato quando viene sollevata `anydbm.error`.

L'oggetto restituito da `open()` supporta molte delle stesse funzionalità dei dizionari; le chiavi ed i loro valori corrispondenti possono essere immessi, ottenuti e cancellati, sono disponibili i metodi `has_key()` e `keys()`. Chiavi e valori devono sempre essere stringhe.

Vedete anche:

Modulo `dbhash` (sezione 7.11):
interfaccia database BSD `db`.

Modulo `dbm` (sezione 8.6):
Interfaccia standard ai database UNIX.

Modulo `dumbdbm` (sezione 7.14):
Implementazione portabile dell'interfaccia `dbm`.

Modulo `gdbm` (sezione 8.7):
Interfaccia ai database GNU, basata sull'interfaccia `dbm`.

Modulo `shelve` (sezione 3.17):
Oggetto generale persistente realizzato sulla base dell'interfaccia `dbm` di Python.

Modulo `whichdb` (sezione 7.12):
Modulo di utilità usato per determinare il tipo di un database esistente.

7.11 dbhash — Interfaccia stile DBM per la libreria database BSD

Il modulo `dbhash` fornisce una funzione per aprire database usando la libreria `db` BSD. Questo modulo rispecchia l'interfaccia di altri moduli Python per i database che forniscono un accesso ai database stile DBM. È richiesto il modulo `bsddb` per usare `dbhash`.

Questo modulo fornisce un'eccezione ed una funzione:

exception `error`
Eccezione sollevata per errori dei database diversi da `KeyError`. È un sinonimo di `bsddb.error`.

`open`(*path*[, *flag*[, *mode*]])
Apre un database `db` e restituisce l'oggetto database. L'argomento *path* rappresenta il nome del file database.

L'argomento *flag* può essere `'r'` (valore predefinito), `'w'`, `'c'` (che crea il database se non esiste), o `'n'` (che crea sempre un database vuoto). Sulle piattaforme dove la libreria `db` BSD supporta il locking, può venire aggiunta una '`l`' per indicare che dovrebbe essere usato il locking.

Il parametro facoltativo *mode* viene usato per indicare i bit dei permessi UNIX che dovrebbero essere impostati in caso di creazione di un nuovo database; ciò verrà mascherato dal valore corrente della umask del processo.

Vedete anche:

Modulo `anydbm` (sezione 7.10):
Interfaccia generica ai database stile `dbm`.

Modulo `bsddb` (sezione 7.13):
Interfaccia a più basso livello per la libreria `db` BSD.

Modulo `whichdb` (sezione 7.12):
Modulo di utilità usato per determinare il tipo di un database esistente.

7.11.1 Oggetti database

Gli oggetti database restituiti da `open()` forniscono i metodi comuni a tutti i database stile DBM ed oggetti mappati. Sono disponibili i seguenti metodi in aggiunta a quelli standard.

`first``()`
È possibile eseguire cicli su ogni coppia chiave/valore nel database usando questo metodo ed il metodo `next()`. Il traversal è ordinato dai valori interni dell'hash dei database, e non verrà ordinato dai valori delle chiavi. Questo metodo restituisce la chiave iniziale.

`last``()`
Restituisce l'ultima coppia chiave/valore nel traversal di un database. Può essere usato per iniziare un traversal in ordine inverso; vedere `previous()`.

`next``()`
Restituisce la successiva coppia chiave/valore nel traversal di un database. Il codice seguente stampa ogni chiave nel database `db`, senza bisogno di creare una lista in memoria che le contenga tutte.

```
print db.first()
for i in xrange(1, len(db)):
    print db.next()
```

`previous``()`
Restituisce la precedente coppia chiave/valore in un traversal avanzato del database. In congiunzione con `last()`, può essere usato per implementare un traversal in ordine inverso.

`sync``()`
Questo metodo forza ogni dato non scritto ad essere scritto sul disco.

7.12 `whichdb` — Indovina quale modulo DBM ha creato un database

L'unica funzione in questo modulo tenta di indovinare quale dei vari semplici moduli database disponibili – `dbm`, `gdbm` o `dbhash`–dovrebbe essere usato per aprire il file fornito.

`whichdb``(`*filename*`)`
Restituisce uno dei seguenti valori: `None` se il file non può essere aperto perché illeggibile o inesistente; la stringa vuota (`''`) se il formato del file non può essere determinato; o una stringa contenente il nome del modulo richiesto, come `'dbm'` o `'gdbm'`.

7.13 `bsddb` — Interfaccia alla libreria Berkeley DB

Il modulo `bsddb` fornisce un'interfaccia alla libreria Berkeley DB. Gli utenti possono creare file di libreria basati su hash, btree o record usando le chiamate open appropriate. Gli oggetti Bsddb generalmente si comportano come dizionari. Chiavi e valori devono essere stringhe, per usare altri oggetti come chiavi o per immettere altri tipi di oggetti l'utente deve in qualche modo serializzarli, tipicamente usando marshal.dumps o pickle.dumps.

A partire da Python 2.3 il modulo `bsddb` richiede la libreria Berkeley DB versione 3.2 o successiva (viene assicurato il funzionamento dalla 3.2 alla 4.2 al momento della scrittura di questo documento).

Vedete anche:

http://pybsddb.sourceforge.net/
Sito web con la documentazione per la nuova interfaccia python a Berkeley DB che rispecchia fedelmente l'interfaccia orientata agli oggetti sleepycat fornita in Berkely DB 3 e 4.

http://www.sleepycat.com/
Sleepycat Software produce la moderna libreria Berkeley DB.

Ciò che segue è una descrizione dell'interfaccia `bsddb` ereditata, compatibile con il vecchio modulo python bsddb. Per dettagli riguardo le più moderne interfacce orientate agli oggetti Db e DbEnv vedere l'URL pybsddb menzionato sopra.

Il modulo `bsddb` definisce le seguenti funzioni le quali creano oggetti che accedono all'appropriato tipo del file Berkeley DB. I primi due argomenti di ogni funzione sono gli stessi. Per riguardo alla portabilità, solo i primi due argomenti dovrebbero essere usati nella gran parte dei casi.

`hashopen`(*filename*[, *flag*[, *mode*[, *bsize*[, *ffactor*[, *nelem*[, *cachesize*[, *hash*[, *lorder*]]]]]]]])
Apre il file di formato hash chiamato *filename*. I file che non devono essere preservati sul disco possono essere creati passando `None` come *filename*. Il parametro facoltativo varflag identifica il modo usato per aprire il file. Può essere '`r`' (sola lettura, valore predefinito), '`w`' (lettura e scrittura), '`c`' (lettura e scrittura - crea se necessario) o '`n`' (lettura e scrittura - tronca la lunghezza zero). Gli altri argomenti sono usati raramente e vengono passati solo alla funzione di più basso livello `dbopen()`. Consultare la documentazione Berkeley DB per il loro uso e la loro interpretazione.

`btopen`(*filename*[, *flag*[, *mode*[, *btflags*[, *cachesize*[, *maxkeypage*[, *minkeypage*[, *psize*[, *lorder*]]]]]]]])
Apre il file di formato btree chiamato *filename*. I file che non devono essere preservati sul disco possono essere creati passando `None` come *filename*. Il parametro facoltativo *flag* identifica il modo usato per aprire il file. Può essere '`r`' (sola lettura, valore predefinito), '`w`' (lettura e scrittura), '`c`' (lettura e scrittura - crea se necessario) o '`n`' (lettura e scrittura - tronca la lunghezza zero). Gli altri argomenti sono usati raramente e vengono passati solo alla funzione di più basso livello `dbopen()`. Consultare la documentazione Berkeley DB per il loro uso e la loro interpretazione.

`rnopen`(*filename*[, *flag*[, *mode*[, *rnflags*[, *cachesize*[, *psize*[, *lorder*[, *reclen*[, *bval*[, *bfname*]]]]]]]]])
Apre il file di formato DB record chiamato *filename*. I file che non devono essere preservati sul disco possono essere creati passando `None` come *filename*. Il parametro facoltativo *flag* identifica il modo usato per aprire il file. Può essere '`r`' (sola lettura, valore predefinito), '`w`' (lettura e scrittura), '`c`' (lettura e scrittura - crea se necessario) o '`n`' (lettura e scrittura - tronca la lunghezza zero). Gli altri argomenti sono usati raramente e vengono passati solo alla funzione di più basso livello `dbopen()`. Consultare la documentazione Berkeley DB per il loro uso e la loro interpretazione.

Vedete anche:

Modulo `dbhash` (sezione 7.11):
Modulo dbhash Interfaccia stile DBM a `bsddb`

Note: A partire dalla 2.3 alcune versioni UNIX di Python possono presentare un modulo `bsddb185`. Questi è presente *solo* per permettere la retrocompatibilità con sistemi che navigano con la vecchia libreria database Berkeley DB 1.85. Il modulo `bsddb185` non dovrebbe mai essere usato nel nuovo codice.

7.13.1 Oggetti Hash, BTree e Record

Una volta istanziati, gli oggetti hash, btree e record supportano gli stessi metodi e dizionari. In aggiunta, supportano i metodi elencati qui sotto. Modificato nella versione 2.3.1: Aggiunti i metodi dei dizionari.

`close`()
Chiude il file sottostante. Non si può più accedere all'oggetto. Visto che non esiste alcun metodo `open` per questi oggetti, per aprire nuovamente il file è necessario chiamare ancora la funzione open del modulo `bsddb`.

`keys`()
Restituisce la lista delle chiavi contenute in un file DB. L'ordine della lista non è specificato e non vi si dovrebbe fare affidamento. In particolare, l'ordine della lista restituita è differente a seconda dei formati.

`has_key`(*key*)
Restituisce `1` se il file DB contiene l'argomento come chiave.

`set_location`(*key*)
Imposta il cursore all'elemento indicato dalla chiave *key* e restituisce una tupla contenente la chiave ed il suo valore. Per i database ad albero binario (aperti usando `btopen()`), se la chiave *key* non esiste realmente nel database, il cursore punterà al prossimo elemento in maniera ordinata restituendo quella chiave e quel valore. Per altri database, verrà sollevata l'eccezione `KeyError` se la chiave non viene trovata nel database.

`first`()
Imposta il cursore al primo elemento nel file DB e lo restituisce. L'ordine delle chiavi non è specificato, eccetto nel caso di database B-Tree.

next()
Imposta il cursore al prossimo elemento nel file DB e lo restituisce. L'ordine delle chiavi non è specificato, eccetto nel caso di database B-Tree.

previous()
Imposta il cursore all'elemento precedente nel file DB e lo restituisce. L'ordine delle chiavi non è specificato, eccetto nel caso di database B-Tree. Questo metodo non è supportato nei database a tabella di hash (quelli aperti con `hashopen()`).

last()
Imposta il cursore all'ultimo elemento nel file DB e lo restituisce. L'ordine delle chiavi non è specificato. Questo metodo non è supportato nei database a tabella di hash (quelli aperti con `hashopen()`).

sync()
Sincronizza il database sul disco.

Esempio:

```
>>> import bsddb
>>> db = bsddb.btopen('/tmp/spam.db', 'c')
>>> for i in range(10): db['%d'%i] = '%d'% (i*i)
...
>>> db['3']
'9'
>>> db.keys()
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
>>> db.first()
('0', '0')
>>> db.next()
('1', '1')
>>> db.last()
('9', '81')
>>> db.set_location('2')
('2', '4')
>>> db.previous()
('1', '1')
>>> for k, v in db.iteritems():
...     print k, v
0 0
1 1
2 4
3 9
4 16
5 25
6 36
7 49
8 64
9 81
>>> '8' in db
True
>>> db.sync()
0
```

7.14 dumbdbm — Implementazione portabile di DBM

Note: Il modulo `dumbdbm` è stato pensato come ultima risorsa da richiamare per il modulo `anydbm` quando non è disponibile nessun altro modulo robusto. Il modulo `dumbdbm` non è stato scritto con riguardo alla velocità e non viene usato pesantemente come gli altri moduli database.

Il modulo `dumbdbm` fornisce un'interfaccia persistente simile ai dizionari che è scritta interamente in Python.

Diversamente da altri moduli come gdbm e bsddb, non è richiesta alcuna libreria esterna. Come altre mappature persistenti, chiavi e valori devono essere sempre stringhe.

Il modulo definisce i seguenti:

exception `error`
Viene sollevata in caso di errori specifici a dumbdbm, come errori di I/O. Per altri errori generici di mappatura come specificare una chiave non corretta, viene sollevata l'eccezione `KeyError`.

`open`(*filename*[, *flag*[, *mode*]])
Apre un database dumbdbm e restituisce un oggetto dumbdbm. L'argomento *filename* è il basename (NdT: il prefisso comune ai vari file) del file database (senza ogni specifica estensione). Quando viene creato un database dumbdbm, verranno creati anche dei file con estensione '.dat' e '.dir'.

L'argomento facoltativo *flag* attualmente è ignorato; il database viene sempre aperto per l'aggiornamento, e verrà creato se non esiste.

L'argomento facoltativo *mode* è il modo UNIX del file, usato solo quando il database deve essere creato. Il valore predefinito è l'ottale `0666` (e sarà modificato dall'umask prevalente). Modificato nella versione 2.2: L'argomento *mode* veniva ignorato nelle versioni precedenti.

Vedete anche:

Modulo `anydbm` (sezione 7.10):
Interfaccia generica ai database in stile `dbm`.

Modulo `dbm` (sezione 8.6):
Interfaccia simile alla libreria DBM/NDBM.

Modulo `gdbm` (sezione 8.7):
Interfaccia simile alla libreria GNU GDBM.

Modulo `shelve` (sezione 3.17):
Modulo persistente che immagazzina dati diversi da stringhe.

Modulo `whichdb` (sezione 7.12):
Modulo di utilità usato per determinare il tipo di un database esistente.

7.14.1 Oggetti dumbdbm

In aggiunta ai metodi forniti dalla classe `UserDict.DictMixin`, gli oggetti `dumbdbm` forniscono i seguenti metodi.

`sync`()
Sincronizza la directory sul disco ed i file di dati. Questo metodo viene chiamato dal metodo `sync` degli oggetti `Shelve`.

7.15 `zlib` — Compressione compatibile con **gzip**

Per le applicazioni che richiedono la compressione dei dati, le funzioni in questo modulo permettono la compressione e la decompressione, usando la libreria zlib. La libreria zlib ha la sua pagina web presso http://www.gzip.org/zlib/. La versione 1.1.3 è la versione più recente nel settembre 2000; usare una versione successiva se disponibile. Esistono incompatibilità note fra i moduli Python e le versioni datate della libreria zlib.

L'eccezione e le funzioni disponibili in questo modulo sono:

exception `error`
Eccezione sollevata in caso di errori di compressione e decompressione.

`adler32`(*string*[, *value*])
Computa un checksum Adler-32 di *string*. (Un checksum Adler-32 è affidabile quasi quanto un CRC32 ma può essere computato più rapidamente.) Se *value* è presente, verrà utilizzato come valore iniziale del checksum; altrimenti, verrà usato un valore predefinito fissato. Questo permette di computare un checksum funzionante sulla concatenazione di alcune stringhe di input. L'algoritmo non è crittograficamente forte,

e non dovrebbe essere usato per firme digitali o di autenticazione. Visto che l'algoritmo è stato disegnato come algoritmo di checksum, non è adattabile come algoritmo generale di hash.

`compress`(*string*[, *level*])
: Comprime i dati in *string*, restituendo una stringa contenente i dati compressi. *level* è un intero da 1 a 9 che esprime il livello di compressione; `1` è il più veloce e produce la compressione minore, `9` è il più lento e produce la massima compressione. Il valore predefinito è `6`. In caso di errore viene sollevata l'eccezione `error`.

`compressobj`([*level*])
: Restituisce un oggetto di compressione, in modo da essere usato per comprimere flussi di dati che non devono essere memorizzati in una sola volta. *level* è un intero da 1 a 9 che esprime il livello di compressione; `1` è il più veloce e produce la compressione minore, `9` è il più lento e produce la massima compressione. Il valore predefinito è `6`.

`crc32`(*string*[, *value*])
: Computa un checksum CRC (Controllo Ciclico di Ridondanza) di *string*. Se *value* è presente, verrà utilizzato come valore iniziale del checksum; altrimenti, verrà usato un valore predefinito fissato. Questo permette di computare un checksum funzionante sulla concatenazione di alcune stringhe di input. L'algoritmo non è crittograficamente forte, e non dovrebbe essere usato per firme digitali o di autenticazione. Visto che l'algoritmo è stato disegnato come algoritmo di checksum, non è adattabile come algoritmo generale di hash.

`decompress`(*string*[, *wbits*[, *bufsize*]])
: Decomprime in dati in *string*, restituendo una stringa contenente i dati decompressi. Il parametro *wbits* controlla la dimensione del window buffer. Se è stato passato *bufsize*, questi sarà usato come dimensione iniziale per il buffer di output. Solleva l'eccezione `error` in caso di errore.

  Il valore assoluto di *wbits* è il logaritmo in base due della dimensione dell'history buffer (il "window size") usato durante la compressione dei dati. Il suo valore assoluto dovrebbe trovarsi fra 8 e 15 per le versioni più recenti della libreria zlib, valori più grandi provocano una migliore compressione al costo di un maggiore utilizzo della memoria. Il valore predefinito è 15. Quando *wbits* è negativo, l'header standard di **gzip** viene soppresso; questa è una caratteristica non documentata della libreria zlib, usata per compatibilità con il formato di compressione dei file **unzip**.

  bufsize è la dimensione iniziale del buffer usato per contenere i dati decompressi. Se viene richiesto più spazio, il buffer verrà incrementato di conseguenza, perciò non c'è bisogno di impostare esattamente questo valore; la sua immissione farà solo risparmiare qualche chiamata a `malloc()`. La dimensione predefinita è 16384.

`decompressobj`([*wbits*])
: Restituisce un oggetto di decompressione, da essere usato per decomprimere flussi di dati che non starebbero tutti insieme in memoria. Il parametro *wbits* controlla la dimensione del window buffer.

Gli oggetti di compressione supportano i seguenti metodi:

`compress`(*string*)
: Comprime *string*, restituendo una stringa contenente dati compressi per almeno parte dei dati in *string*. Questi dati dovrebbero essere concatenati all'output prodotto da ogni precedente chiamata al metodo `compress()`. Alcuni input possono essere tenuti in buffer interni per un utilizzo successivo.

`flush`([*mode*])
: Tutti gli input in attesa vengono elaborati, e viene restituita una stringa contenente l'output compresso rimanente. *mode* può essere scelto fra le costanti `Z_SYNC_FLUSH`, `Z_FULL_FLUSH` o `Z_FINISH`, predefinito è `Z_FINISH`. `Z_SYNC_FLUSH` e `Z_FULL_FLUSH` permettono un'ulteriore compressione di stringhe di dati e vengono usate per permettere un parziale recupero degli errori durante la decompressione, mentre `Z_FINISH` finisce i flussi compressi e previene la compressione di altri dati. Dopo aver chiamato `flush()` con *mode* impostato a `Z_FINISH`, il metodo `compress()` non può essere chiamato di nuovo; l'unica azione sensata è cancellare l'oggetto.

Gli oggetti di decompressione supportano i seguenti metodi, e due attributi.

`unused_data`
: Una stringa che contiene ogni byte successivo alla fine dei dati compressi. In sostanza, rimane vuota " fino a che l'ultimo byte contenente dati compressi è disponibile. Se l'intera stringa è formata da dati compressi,

questa è '', ovvero una stringa vuota.

Attualmente, l'unico modo per determinare dove una stringa di dati compressi termina è decomprimerla. Questo significa che, quando i dati compressi sono una parte contenuta in un file più grande, si può trovare la fine di questa stringa leggendo i dati, ed inserendoli insieme a qualche stringa non vuota in un metodo di decompressione di oggetti `decompress`, finché l'attributo `unused_data` non risulta più una stringa vuota.

`unconsumed_tail`
Una stringa che contiene tutti i dati che non sono stati consumati dall'ultima chiamata a `decompress`, a causa del superamento del limite per il buffer dei dati non compressi. Questi dati non sono ancora stati mandati in lavorazione da parte della zlib, per cui dovete alimentarli (possibilmente con ulteriori concatenazioni) con una successiva chiamata al metodo `decompress`, in modo da ottenere l'output corretto.

`decompress`(*string*)
[max_length] Decomprime *string*, restituendo una stringa contenente i dati non compressi corrispondenti ad almeno parte dei dati in *string*. Questi dati dovrebbero essere concatenati all'ouput prodotto da ogni precedente chiamata al metodo `decompress()`. Alcuni dei dati in input possono essere preservati in buffer interni per una elaborazione successiva.

Se viene fornito il parametro facoltativo *max_length*, il valore restituito non sarà superiore di *max_length*. Questo può significare che non tutto l'input compresso può essere elaborato; ed i dati non consumati verranno messi nell'attributo `unconsumed_tail`. La stringa deve essere passata anche ad una successiva chiamata a `decompress()` se la decompressione deve continuare. Se *max_length* non viene fornita sarà decompresso l'intero input, e `unconsumed_tail` sarà una stringa vuota.

`flush`()
Tutti gli input in attesa vengono elaborati, e viene restituita una stringa contenente l'output decompresso rimanente. Dopo aver chiamato `flush()`, il metodo `decompress()` non può più essere nuovamente chiamato; l'unica azione sensata è cancellare l'oggetto.

Vedete anche:

Modulo `gzip` (sezione 7.16):
Leggere e scrivere file in formato gzip **gzip**.

http://www.gzip.org/zlib/
La pagina web di zlib.

7.16 `gzip` — Supporto per i file **gzip**

La compressione dei dati fornita dal modulo `zlib` è compatibile con quella usata dal programma GNU di compressione **gzip**. Di conseguenza, il modulo **gzip** fornisce la classe `GzipFile` per leggere e scrivere file in formato **gzip**, comprimendo e decomprimendo automaticamente i dati in modo tale che sembrino oggetti file ordinari. Notare che altri formati di file che possono essere decompressi dai programmi **gzip** e **gunzip**, come quelli prodotti da **compress** e **pack**, non sono supportati da questo modulo.

Il modulo definisce i seguenti elementi:

class `GzipFile`([*filename*[, *mode*[, *compresslevel*[, *fileobj*]]]])
Costruttore per la classe `GzipFile`, che simula gran parte dei metodi di un oggetto file, con l'eccezione dei metodi `readinto()` e `truncate()`. Almeno uno fra *fileobj* e *filename* deve avere un valore significativo.

La nuova istanza di classe è basata su *fileobj* che deve essere un file regolare, un oggetto `StringIO`, od ogni altro oggetto che simuli un file. Il valore predefinito è `None`, in tale caso, *filename* viene aperto per fornire un oggetto file.

Quando *fileobj* è diverso da `None`, l'argomento *filename* viene usato unicamente per essere incluso nell'intestazione del file **gzip**, che può includere il nome del file originario del file non compresso. Se discernibile, il valore predefinito è il nome del file di *fileobj*, altrimenti, è una stringa vuota, ed in questo caso il nome del file originario non viene incluso nell'intestazione.

L'argomento *mode* può essere uno fra `'r'`, `'rb'`, `'a'`, `'ab'`, `'w'` o `'wb'`, a seconda se il file debba

essere letto o scritto. Se visibile, il valore predefinito è il modo di *fileobj*; altrimenti, il valore predefinito è `'rb'`. Se non viene passata, l'opzione `'b'` verrà aggiunta al modo per assicurare che il file sia aperto in modalità binaria per portabilità multipiattaforma.

L'argomento *compresslevel* è un intero compreso fra `1` e `9` che indica il livello di compressione; `1` è il più rapido e produce la compressione minore, `9` è il più lento e produce la compressione maggiore. Il valore predefinito è `9`.

Chiamare il metodo `close()` di un oggetto `GzipFile` non provoca la chiusura i *fileobj*, visto che si potrebbe voler aggiungere altro materiale dopo i dati compressi. Questo permette anche di passare un oggetto `StringIO` aperto per essere scritto come *fileobj*, e recuperare il buffer di memoria risultante usando il metodo `getvalue()` dell'oggetto `StringIO`.

open(*filename*[, *mode*[, *compresslevel*]])
Questa è una scorciatoia per `GzipFile(`*filename*`, `*mode*`, `*compresslevel*`)`. È richiesto l'argomento *filename*; *mode* ha come valore predefinito `'rb'` mentre *compresslevel* ha come valore predefinito `9`.

Vedete anche:

Modulo `zlib` (sezione 7.15):
Il modulo basilare di compressione dei dati necessario per supportare il formato dei file **gzip**.

7.17 bz2 — Compressione compatibile con **bzip2**

Nuovo nella versione 2.3.

Questo modulo fornisce un completa interfaccia per la libreria di compressione bz2. Implementa un'interfaccia file completa, funzioni di (de)compressione immediate, e tipi per (de)compressioni sequenziali.

Qui c'è un riassunto dei contenuti offerti dal modulo bz2.

- La classe `BZ2File` implementa un'interfaccia file completa, incluse `readline()`, `readlines()`, `writelines()`, `seek()`, etc;
- La classe `BZ2File` implementa un supporto emulato di `seek()`;
- La classe `BZ2File` implementa `BZ2File` il supporto universale ai fine riga;
- La classe `BZ2File` offre un'iterazione di riga ottimizzata, usando l'algoritmo readahead ereditato dagli oggetti file;
- La (de)compressione sequenziale è supportata dalle classi `BZ2Compressor` e `BZ2Decompressor`;
- (De)compressione immediata, supportata dalle funzioni `compress()` e `decompress()`;
- a sicurezza dei thread utilizza meccanismi di lock individuale;
- Documentazione completa in linea.

7.17.1 (De)compressione di file

La gestione dei file compressi viene offerta dalla classe `BZ2File`.

class `BZ2File`(*filename*[, *mode*[, *buffering*[, *compresslevel*]]])
Apre un file bz2. *mode* può essere `'r'` o `'w'`, per leggere (valore predefinito) o scrivere. Quando aperto in scrittura, il file verrà creato se non esiste, troncato altrimenti. Se viene passato *buffering*, `0` significa senza buffer, numeri più grandi specificano la dimensione del buffer; il valore predefinito è `0`. Se viene passato *compresslevel*, esso deve essere un numero compreso fra `1` e `9`; il valore predefinito è `9`. Aggiungete una '`U`' a mode per aprire il file in input con il supporto universale al newline. Ogni fine riga nel file di input verrà visto da Python come un '`\n`'. Inoltre, un file così aperto ottiene l'attributo `newlines`; il valore di questo attributo è uno fra `None` (nessun fine riga ancora letto), `'\r'`, `'\n'`, `'\r\n'` o una tupla contenente tutti i tipi di fine riga visti. I fine riga universali sono disponibili solo durante la lettura. Le istanze supportano l'iterazione allo stesso modo delle normali istanze `file`.

close()
Chiude il file. Imposta l'attributo `closed` a vero. Un file chiuso non può essere usato per ulteriori operazioni di I/O. `close()` può essere chiamata più di una volta senza provocare errori.

read([*size*])
Legge al massimo *size* byte non compressi, restituendoli come stringa. Se l'argomento *size* viene omesso o è negativo, legge fino all'EOF.

readline([*size*])
Restituisce la prossima riga del file, come stringa, conservando il fine riga. Un argomento *size* non negativo limita il massimo numero di byte da restituire (può quindi venire restituita una riga incompleta). Restituisce una stringa vuota in caso di EOF.

readlines([*size*])
Restituisce una lista delle righe lette. L'argomento facoltativo *size*, se passato, è un limite approssimativo del numero di byte totale delle righe restituite.

xreadlines()
Per retrocompatibilità. L'oggetto `BZ2File` adesso include le ottimizzazioni di performance precedentemente implementate nel modulo `xreadlines`. **Deprecato dalla versione 2.3.** Esiste solo per compatibilità con il metodo con questo nome negli oggetti `file`, che è anch'esso deprecato. Usate `for line in file` al suo posto.

seek(*offset*[, *whence*])
Si sposta ad una nuova posizione nel file. L'argomento *offset* rappresenta il conteggio dei byte. L'argomento facoltativo *whence*, ha valore predefinito `0` (offset parte dall'inizio del file, offset dovrebbe essere `>= 0`); altri valori sono `1` (spostati a partire dalla posizione corrente, positivo o negativo), e `2` (spostati relativamente alla fine del file, solitamente negativo, anche se alcune piattaforme permettono il seek oltre la fine del file).

Notate che il seek nei file bz2 viene emulato, ed a seconda dei parametri l'operazione può rivelarsi estremamente lenta.

tell()
Restituisce la posizione corrente nel file, come intero (può essere anche un intero long).

write(*data*)
Scrive la stringa *data* sul file. Notate che a causa del buffering, può essere necessaria una `close()` prima che il file sul disco rifletta i dati scritti.

writelines(*sequence_of_strings*)
Scrive la sequenza di stringhe sul file. Attenzione che nessun fine riga viene aggiunto. La sequenza può essere ogni oggetto iterabile che produce stringhe. Ciò equivale a chiamare `write()` per ogni stringa.

7.17.2 (De)compressione sequenziale

La compressione e la decompressione sequenziale viene fatta usando le classi `BZ2Compressor` e `BZ2Decompressor`.

class BZ2Compressor([*compresslevel*])
Crea un nuovo oggetto compressore. Questo oggetto può essere usato per comprimere i dati sequenzialmente. Se volete comprimere i dati in un colpo solo, usate la funzione `compress()`. Il parametro *compresslevel*, se passato, deve essere un numero fra `1` e `9`; il predefinito è `9`.

compress(*data*)
Fornisce altri dati all'oggetto compressore. Restituirà pezzi di dati compressi ogni qual volta sia possibile. Quando avete finito di fornire dati da comprimere, chiamate il metodo `flush()` per terminare il processo di compressione e restituire ciò che è rimasto nei buffer interni.

flush()
Termina il processo di compressione e restituisce ciò che rimasto nei buffer interni. Non dovete usare l'oggetto compressore dopo aver chiamato questo metodo.

class BZ2Decompressor()
Crea un nuovo oggetto decompressore. Questo oggetto può essere usato per decomprimere i da-

ti sequenzialmente. Se volete invece decomprimere i dati in un colpo solo, usate la funzione `decompress()`.

`decompress`(*data*)
Fornisce altri dati all'oggetto decompressore. Restituirà grossi pezzi di dati decompressi, quando possibile. Se tentate di decomprimere i dati dopo che è stata trovata la fine del flusso, verrà sollevata l'eccezione `EOFError`. Se vengono trovati dati dopo la fine del flusso, essi verranno ignorati e salvati nell'attributo `unused_data`.

7.17.3 (De)compressione in un colpo solo

La compressione e la decompressione in un colpo solo (one-shot) viene fornita attraverso le funzioni `compress()` e `decompress()`.

`compress`(*data*[, *compresslevel*])
Comprime i dati *data* in un solo colpo. Se volete invece comprimere dati sequenzialmente, usate un'istanza di `BZ2Compressor`. Il parametro *compresslevel*, se passato, deve essere un numero compreso fra `1` e `9`; il predefinito è `9`.

`decompress`(*data*)
Decomprime i dati *data* in un solo colpo solo. Se volete invece decomprimere i dati sequenzialmente, usate un'istanza di `BZ2Decompressor`.

7.18 `zipfile` — Lavorare con gli archivi ZIP

Nuovo nella versione 1.6.

Il formato di file ZIP è un comune standard di compressione per gli archivi. Questo modulo fornisce il necessario per creare, leggere, scrivere, aggiungere ed elencare un file ZIP. Ogni utilizzo avanzato di questo modulo richiederà una conoscenza del formato, come definito nel *PKZIP Application Note*.

Questo modulo al momento non gestisce file ZIP che possiedono commenti o file ZIP multi-disco.

Gli attributi disponibili di questo modulo sono:

exception `error`
L'errore sollevato per file ZIP corrotti.

class `ZipFile`
La classe per leggere e scrivere file ZIP. Leggere "*Oggetti ZipFile*" (sezione 7.18.1) per dettagli sul costruttore.

class `PyZipFile`
Classe per creare archivi ZIP contenenti librerie Python.

class `ZipInfo`([*filename*[, *date_time*]])
Classe usata per rappresentare le informazioni concernenti il membro di un archivio. Istanze di questa classe vengono restituite dai metodi `getinfo()` e `infolist()` degli oggetti `ZipFile`. La gran parte degli utenti del modulo `zipfile` non ha bisogno di crearli, ma solo di usare quelli creati da questo modulo. *filename* dovrebbe essere il nome completo del membro dell'archivio, mentre *date_time* dovrebbe essere una tupla contenente sei campi che descrivono la data dell'ultima modifica del file; i campi sono descritti nella sezione 7.18.3, "Oggetti ZipFile".

`is_zipfile`(*filename*)
Restituisce `True` se *filename* è un file ZIP valido, basato sul suo magic number, altrimenti restituisce `False`. Questo modulo al momento non gestisce file ZIP con commenti aggiunti.

`ZIP_STORED`
La costante numerica per un membro non compresso di un archivio.

`ZIP_DEFLATED`
La costante numerica per il metodo di compressione ZIP usuale. Richiede il modulo zlib. Nessun altro metodo di compressione viene attualmente supportato.

Vedete anche:

PKZIP Application Note
(http://www.pkware.com/appnote.html)
Documentazione sul formato di file ZIP di Phil Katz, il creatore del formato e dell'algoritmo usati.

Info-ZIP Home Page
(http://www.info-zip.org/pub/infozip/)
Informazioni sui programmi e le librerie di sviluppo per gli archivi ZIP del progetto Info-ZIP.

7.18.1 Oggetti ZipFile

class `ZipFile`(*file*[, *mode*[, *compression*]])
Apre un file ZIP, dove *file* può essere il percorso di un file (una stringa) o un oggetto simil-file. Il parametro mode dovrebbe essere `'r'` per leggere un file esistente, `'w'` per troncare e scrivere un nuovo file, oppure `'a'` per aggiungere ad un file esistente. Se *mode* è `'a'` e *file* si riferisce ad un file ZIP esistente, allora altri file vengono aggiunti ad esso. Se *file* non si riferisce ad un file ZIP, allora verrà aggiunto un nuovo archivio ZIP al file. Questo è il significato di aggiungere un archivio ZIP ad un altro file, come 'python.exe'. Anche usare

```
cat myzip.zip >> python.exe
```

funziona, ed almeno **WinZip** è capace di leggere questo tipo di file. *compression* è il metodo di compressione ZIP da usare quando si scrive sull'archivio, e dovrebbe essere `ZIP_STORED` o `ZIP_DEFLATED`; valori non riconosciuti solleveranno un'eccezione `RuntimeError`. Se è specificato `ZIP_DEFLATED` ma il modulo `zlib` non è disponibile, verrà sollevata ancora una volta l'eccezione `RuntimeError`. Il predefinito è `ZIP_STORED`.

`close`()
Chiude il file archivio. Dovete chiamare `close()` prima di uscire dal vostro programma altrimenti record essenziali non verranno scritti.

`getinfo`(*name*)
Restituisce un oggetto `ZipInfo` con informazioni riguardo il membro dell'archivio *name*.

`infolist`()
Restituisce una lista contenente un oggetto `ZipInfo` per ogni membro dell'archivio. Gli oggetti appaiono nello stesso ordine in cui si trovano nel file ZIP reale sul disco, se è stato aperto un archivio esistente.

`namelist`()
Restituisce una lista dei membri dell'archivio ordinata per nome.

`printdir`()
Stampa l'indice dell'archivio verso `sys.stdout`.

`read`(*name*)
Restituisce i byte del file nell'archivio. L'archivio deve essere stato aperto per la lettura o l'aggiunta.

`testzip`()
Legge tutti i file nell'archivio e controlla il loro CRC. Restituisce il nome del primo file corrotto, altrimenti restituisce `None`.

`write`(*filename*[, *arcname*[, *compress_type*]])
Scrive il file chiamato *filename* nell'archivio, dandogli il nome d'archivio *arcname* (il nome predefinito sarà lo stesso di *filename*). Se passato, *compress_type* sovrascrive il valore passato per il parametro *compression* del costruttore per questo nuovo ingresso. L'archivio deve essere stato aperto in modalità `'w'` o `'a'`.

`writestr`(*zinfo_or_arcname, bytes*)
Scrive la stringa *bytes* nell'archivio; *zinfo_or_arcname* è il filename che avrà nell'archivio, o un'istanza `ZipInfo`. Se è un'istanza, devono essere impostati almeno filename, date e time. Se è un nome, date e time vengono impostati alla data e all'orario correnti. L'archivio deve essere stato aperto con mode `'w'` o `'a'`.

È disponibile anche il seguente attributo:

debug
Il livello di debug output da usare. Può essere impostato da `0` (valore predefinito, nessun output) a `3` (l'output maggiore). Le informazioni di debug vengono scritte su `sys.stdout`.

7.18.2 Oggetti PyZipFile

Il costruttore `PyZipFile` accetta gli stessi parametri del costruttore `ZipFile`. Le istanze possiedono un metodo in più rispetto agli oggetti `ZipFile`.

writepy(*pathname*[, *basename*])
Ricerca i file '*.py' e li aggiunge all'archivio. Il file corrispondente è un file '*.pyo' se disponibile, altrimenti un file '*.pyc', compilando se necessario. Se il percorso del nome è un file, il nome del file deve terminare con '.py', e solo questo file ('*.py[co]' corrispondente) verrà aggiunto al livello più alto (senza informazioni sul percorso). Se è una directory, e non è una package directory, allora tutti i file '*.py[co]' verranno aggiunti al livello più alto. Se è una package directory, allora tutti i '*.py[co]' verranno aggiunti sotto il nome del package come un percorso di file, e se tutte le sottodirectory sono package directory, verranno tutte aggiunte ricorsivamente. Si intende che *basename* venga usato solamente per uso interno. Il metodo `writepy()` crea archivi con nomi di file simili a questi:

```
string.pyc                          # Nome al livello più alto
test/__init__.pyc                   # Package directory
test/testall.pyc                    # Modulo test.testall
test/bogus/__init__.pyc             # Sotto-package directory
test/bogus/myfile.pyc               # Sotto-modulo test.bogus.myfile
```

7.18.3 Oggetti ZipInfo

Le istanze della classe `ZipInfo` vengono restituite dai metodi `getinfo()` e `infolist()` degli oggetti `ZipFile`. Ogni oggetto registra informazioni riguardo un singolo membro dell'archivio ZIP.

Le istanze possiedono i seguenti attributi:

filename
Nome del file nell'archivio.

date_time
L'orario e la data dell'ultima modifica sul membro dell'archivio. È una tupla di sei valori:

Index	Value
0	Anno
1	Mese (a partire da 1)
2	Giorno del mese (a partire da 1)
3	Ore (a partire da 0)
4	Minuti (a partire da 0)
5	Secondi (a partire da 0)

compress_type
Tipo di compressione per il membro dell'archivio.

comment
Commento individuale per il membro dell'archivio.

extra
Campo di dati di espansione. La *PKZIP Application Note* contiene alcuni commenti sulla struttura interna dei dati contenuti in questa stringa.

create_system
Il sistema che ha creato l'archivio ZIP.

create_version
Versione di PKZIP che ha creato l'archivio ZIP.

`extract_version`
La versione di PKZIP necessaria per estrarre l'archivio.

`reserved`
Deve essere zero.

`flag_bits`
bit di flag ZIP.

`volume`
Numero di volume dell'intestazione del file.

`internal_attr`
Attributi interni.

`external_attr`
Attributi di file esterni.

`header_offset`
Byte offset dell'intestazione del file.

`file_offset`
Byte offset dell'inizio dei dati del file.

`CRC`
CRC-32 del file non compresso.

`compress_size`
Dimensione dei dati compressi.

`file_size`
Dimensione del file non compresso.

7.19 `tarfile` — Leggere e scrivere file di archivio tar

Nuovo nella versione 2.3.

Il modulo `tarfile` rende possibile le lettura e la creazione degli archivi tar. Ecco alcune caratteristiche:

- legge e scrive archivi compressi `gzip` e `bzip2`.
- crea archivi conformi al POSIX 1003.1-1990 o compatibili con gli archivi tar della GNU.
- legge le estensioni GNU tar *longname*, *longlink* e *sparse*.
- archivia nomi di percorso di lunghezza illimitata usando le estensioni GNU tar.
- gestisce directory, file regolari, collegamenti hard, collegamenti simbolici, code, dispositivi a caratteri e a blocchi ed è capace di acquisire e registrare informazioni di file come timestamp, permessi di accesso e del proprietario.
- può gestire dispositivi a nastro magnetico.

`open`([*name*[, *mode* [, *fileobj*[, *bufsize*]]]])
Restituisce un oggetto `TarFile` per il percorso dei nomi *name*. Per informazioni dettagliate sugli oggetti `TarFile`, leggere *Oggetti TarFile* (sezione 7.19.1).

mode deve essere una stringa nella forma `'mododelfile[:compressione]'`, il valore predefinito è `'r'`. Ecco una lista completa delle combinazioni di mode:

mode	azione
`'r'`	Apre in lettura con compressione trasparente (raccomandato).
`'r:'`	Apre in lettura esclusivamente senza compressione.
`'r:gz'`	Apre in lettura con compressione gzip.
`'r:bz2'`	Apre in lettura con compressione bzip2.
`'a' or 'a:'`	Apre per l'aggiunta senza compressione.
`'w' or 'w:'`	Apre in scrittura non compressa.
`'w:gz'`	Apre in scrittura compressa gzip.
`'w:bz2'`	Apre in scrittura compressa bzip2.

Notare che `'a:gz'` o `'a:bz2'` non sono possibili. Se *mode* non è adatto per aprire un certo file (compresso) in lettura, verrà sollevata un'eccezione `ReadError`. Usare *mode* `'r'` per evitarlo. Se un certo metodo di compressione non è supportato, verrà sollevata un'eccezione `CompressionError`.

Se viene specificato *fileobj*, esso viene usato come alternativa ad un oggetto file aperto per nome *name*.

Per scopi speciali, esiste un secondo formato per *mode*: `'mododelfile|[compressione]'`. `open()` restituirà un oggetto `TarFile` che elabora i suoi dati come un flusso di blocchi. Sul file non verrà eseguito alcun seek casuale. Se passato, *fileobj* può essere qualsiasi oggetto che abbia un metodo `read()` o `write()` (a seconda della modalità prescelta *mode*). *bufsize* specifica la dimensione dei blocchi e il valore predefinito è `20 * 512` byte. Usare questa variante in combinazione con ad esempio `sys.stdin`, un oggetto file socket od un dispositivo a nastro. In ogni caso, un oggetto `TarFile` del genere è limitato per il fatto che non permette un accesso casuale, vedere "Esempi" (sezione 7.19.3). Ecco le alternative dei modi disponibili adesso:

Modalità	azione
`'r|'`	Apre un *flusso* di blocchi tar non compressi in lettura.
`'r|gz'`	Apre un *flusso* compresso gzip in lettura.
`'r|bz2'`	Apre un *flusso* compresso bzip2 in lettura.
`'w|'`	Apre un *flusso* non compresso in scrittura.
`'w|gz'`	Apre un *flusso* compresso gzip in scrittura.
`'w|bz2'`	Apre un *flusso* compresso bzip2 in scrittura.

class `TarFile`
Classe per leggere e scrivere archivi tar. Non usare questa classe direttamente, meglio invece usare `open()`. Leggere "Oggetti TarFile" (sezione 7.19.1).

`is_tarfile`(*name*)
Restituisce `True` se *name* è un file archivio tar, del tipo che può essere letto dal modulo `tarfile`.

class `TarFileCompat`(*filename*[, *mode*[, *compression*]])
Classe per l'accesso limitato agli archivi tar con interfaccia simile a `zipfile`. Consultare la documentazione di `zipfile` per ulteriori dettagli. *compression* deve essere una delle seguenti costanti:

`TAR_PLAIN`
Costante per un archivio tar non compresso.

`TAR_GZIPPED`
Costante per un archivio tar compresso con `gzip`.

exception `TarError`
Classe base per tutte le eccezioni di `tarfile`.

exception `ReadError`
Viene sollevata quando un archivio tar viene aperto, ma non può essere gestito dal modulo `tarfile` o per qualche ragione non è valido.

exception `CompressionError`
Viene sollevata quando un metodo di compressione non è supportato o quando i dati non possono essere decodificati in modo appropriato.

exception `StreamError`
Viene sollevata dalle limitazioni tipiche per gli oggetti `TarFile` simil-flusso.

exception `ExtractError`
Viene sollevata per errori non fatali durante l'utilizzo di `extract()`, ma solo se `TarFile.errorlevel == 2`.

Vedete anche:

Modulo `zipfile` (sezione 7.18):
Documentazione del modulo standard `zipfile`.

Manuale tar della GNU, Standard Section
(http://www.gnu.org/manual/tar/html_chapter/tar_8.html#SEC118)
Documentazione per i file di archivio tar, incluse le estensioni tar della GNU.

7.19.1 Oggetti TarFile

L'oggetto `TarFile` fornisce un'interfaccia ad un archivio tar. Un archivio tar è una sequenza di blocchi. Un membro d'archivio (un file immagazzinato) è composto da un blocco d'intestazione seguito dai blocchi dei dati. È possibile immagazzinare un file in un archivio tar più volte. Ogni membro dell'archivio viene rappresentato da un oggetto `TarInfo`, leggere *Oggetti TarInfo* (sezione 7.19.2) per dettagli.

class `TarFile`(\[*name* \[, *mode*\[, *fileobj* \]\]\])
Apre un archivio tar *(non compresso)* di nome *name*. *mode* è `'r'` per leggere da un archivio esistente, `'a'` per aggiungere dati ad un file esistente o `'w'` per crearne uno nuovo sovrascrivendone uno esistente. *mode* come valore predefinito ha `'r'`.

Se *fileobj* viene passato, viene usato per leggere o scrivere dati. Se può essere determinato, *mode* viene sovrascritto dal modo di *fileobj*.

Note: *fileobj* non viene chiuso quando viene chiuso `TarFile`.

`open`(...)
Costruttore alternativo. La funzione `open()` a livello modulo è in realtà una scorciatoia per questo metodo di classe. Vedere la sezione 7.19 per dettagli.

`getmember`(*name*)
Restituisce un oggetto `TarInfo` per il membro *name*. Se *name* non può essere trovato nell'archivio, viene sollevata un'eccezione `KeyError`.

Note: Se un membro appare più volte in un archivio, la sua ultima occorrenza viene considerata come la versione più aggiornata.

`getmembers`()
Restituisce i membri dell'archivio come una lista di oggetti `TarInfo`. La lista ha lo stesso ordine dei membri nell'archivio.

`getnames`()
Restituisce i membri come lista dei loro nomi. Ha lo stesso ordine della lista restituita da `getmembers()`.

`list`(*verbose=True*)
Stampa un indice verso `sys.stdout`. Se *verbose* è `False`, vengono stampati solo i nomi dei membri. Se è `True`, viene prodotto un output simile a quello di **ls -l**.

`next`()
Restituisce il prossimo membro dell'archivio come oggetto `TarInfo`, quando `TarFile` viene aperto in lettura. Restituisce `None` se non ce ne sono altri disponibili.

`extract`(*member*\[, *path* \])
Estrae un membro dall'archivio nella directory corrente, usando il suo nome completo. Le sue informazioni di file vengono estratte nel modo più accurato possibile. *member* può essere un nome di file od un oggetto `TarInfo`. É possibile specificare una directory differente usando *path*.

`extractfile`(*member*)
Estrae *member* dall'archivio come un oggetto file. *member* può essere un nome di file od un oggetto `TarInfo`. Se *member* è un file regolare, viene restituito un oggetto simil-file. Se *member* è un collegamento, viene costruito un oggetto simil-file dall'obbiettivo del collegamento. Se *member* non è nessuno dei precedenti, viene restituito `None`.

Note: L'oggetto simil-file è in sola lettura e fornisce i seguenti metodi: `read()`, `readline()`, `readlines()`, `seek()` e `tell()`.

add(*name*[, *arcname*[, *recursive*]])
Aggiunge il file *name* all'archivio. *name* può essere ogni genere di file (directory, fifo, link simbolico, etc.). Se passato, *arcname* specifica un nome alternativo per il file nell'archivio. Se non specificato, le directory vengono aggiunte ricorsivamente. Ciò può essere evitato impostando *recursive* a `False`; il suo valore predefinito è `True`.

addfile(*tarinfo*[, *fileobj*])
Aggiunge l'oggetto *tarinfo* della classe `TarInfo` all'archivio. Se *fileobj* viene passato, vengono letti *tarinfo*`.size` byte da esso e aggiunti all'archivio. Si possono creare oggetti `TarInfo` usando `gettarinfo()`.

Note: Sulle piattaforme Windows, *fileobj* dovrebbe sempre essere aperto in modalità `'rb'`, per evitare irritazioni riguardo la dimensione del file.

gettarinfo([*name*[, *arcname*[, *fileobj*]]])
Crea un oggetto `TarInfo` per il file *name* o l'oggetto file *fileobj* (usando `os.fstat()` sul suo descrittore di file). Si possono modificare alcuni attributi di `TarInfo` prima di aggiungerlo usando `addfile()`. Se passato, *arcname* specifica un nome alternativo per il file nell'archivio.

close()
Chiude il `TarFile`. In modalità di scrittura, due blocchi zero finali vengono aggiunti all'archivio.

posix=True
Se vero, crea un archivio conforme a POSIX 1003.1-1990. Le estensioni GNU non vengono usate, visto che non sono parte dello standard POSIX. Questo limita la lunghezza dei nomi dei file a 256 caratteri e i nomi dei collegamenti a 100 caratteri. Se il percorso dei nomi eccede questo limite, viene sollevata un'eccezione `ValueError`. Se falso, crea un archivio tar GNU compatibile. Non sarà compatibile con POSIX, ma potrà immagazzinare percorsi di lunghezza illimitata.

dereference=False
Se falso, aggiunge collegamenti simbolici e hard all'archivio. Se vero, aggiunge il contenuto del file obbiettivo all'archivio. Questa funzionalità non ha effetto su sistemi che non supportano i link simbolici.

ignore_zeros=False
Se falso, tratta un blocco vuoto come fine dell'archivio. Se vero, ignora i blocchi vuoti (e non validi) e prova ad ottenere più membri possibili. Questo risulta utile solo per archivi concatenati o danneggiati.

debug=0
Può essere impostato da `0` (nessun messaggio di debug) a `3` (tutti i messaggi di debug). I messaggi vengono scritti su `sys.stdout`.

errorlevel=0
Se `0`, il valore predefinito, tutti gli errori vengono ignorati durante l'utilizzo di `extract()`. Nondimeno, essi appaiono come messaggi di errore nell'output di debug, quando il debugging è abilitato. Se `1`, tutti gli errori *fatali* vengono sollevati come eccezioni `OSError` o `IOError`. Se `2`, per tutti gli errori *non fatali* vengono sollevate eccezioni `TarError`.

7.19.2 Oggetti TarInfo

Un oggetto `TarInfo` rappresenta un membro in un `TarFile`. Oltre ad immagazzinare tutti gli attributi richiesti di un file (come tipo, dimensione, orario, permessi, proprietario ecc. ecc.), fornisce alcuni metodi utili per determinarne il tipo. *Non* contiene i dati del file.

Gli oggetti `TarInfo` vengono restituiti dai metodi `getmember()`, `getmembers()` e `gettarinfo()` di `TarFile`.

class TarInfo([*name*])
Crea un oggetto `TarInfo`.

frombuf()
Crea e restituisce un oggetto `TarInfo` da un buffer di stringa.

tobuf()
Crea un buffer stringa da un oggetto `TarInfo`.

Un oggetto `TarInfo` possiede i seguenti attributi pubblici:

`name`
Nome del membro dell'archivio.

`size`
Dimensione in byte.

`mtime`
Orario dell'ultima modifica.

`mode`
Bit di permesso.

`type`
Tipo del file. *type* solitamente è una di queste costanti: `REGTYPE`, `AREGTYPE`, `LNKTYPE`, `SYMTYPE`, `DIRTYPE`, `FIFOTYPE`, `CONTTYPE`, `CHRTYPE`, `BLKTYPE`, `GNUTYPE_SPARSE`. Per determinare il tipo di un oggetto `TarInfo` in modo più conveniente, usare i metodi `is_*()` qui sotto.

`linkname`
Nome del file obiettivo, che è presente solo negli oggetti `TarInfo` del tipo `LNKTYPE` e `SYMTYPE`.

`uid`
ID dell'utente che in origine immagazzinò questo membro.

`gid`
ID del gruppo dell'utente che in origine immagazzinò questo membro.

`uname`
Nome dell'utente.

`gname`
Nome del gruppo.

Un oggetto `TarInfo` fornisce inoltre alcuni utili metodi informativi:

`isfile()`
Restituisce `True` se l'oggetto `Tarinfo` è un file regolare.

`isreg()`
Uguale a `isfile()`.

`isdir()`
Restituisce `True` se è una directory.

`issym()`
Restituisce `True` se è un collegamento simbolico.

`islnk()`
Restituisce `True` se è un collegamento hard.

`ischr()`
Restituisce `True` se è un dispositivo a caratteri.

`isblk()`
Restituisce `True` se è un dispositivo a blocchi.

`isfifo()`
Restituisce `True` se è una FIFO.

`isdev()`
Restituisce `True` se è un dispositivo a caratteri, a blocchi o una FIFO.

7.19.3 Esempi

Come creare un archivio tar non compresso da una lista di nomi di file:

```
import tarfile
tar = tarfile.open("sample.tar", "w")
for name in ["foo", "bar", "quux"]:
    tar.add(name)
tar.close()
```

Come leggere un archivio tar compresso con gzip e mostrare alcune informazioni sui membri.

```
import tarfile
tar = tarfile.open("sample.tar.gz", "r:gz")
for tarinfo in tar:
    print tarinfo.name, "is", tarinfo.size, "bytes in size and is",
    if tarinfo.isreg():
        print "a regular file."
    elif tarinfo.isdir():
        print "a directory."
    else:
        print "something else."
tar.close()
```

Come creare un archivio tar con informazioni fasulle.

```
import tarfile
tar = tarfile.open("sample.tar.gz", "w:gz")
for name in namelist:
    tarinfo = tar.gettarinfo(name, "fakeproj-1.0/" + name)
    tarinfo.uid = 123
    tarinfo.gid = 456
    tarinfo.uname = "johndoe"
    tarinfo.gname = "fake"
    tar.addfile(tarinfo, file(name))
tar.close()
```

L'*unica* maniera per estrarre un flusso tar non compresso da `sys.stdin`:

```
import sys
import tarfile
tar = tarfile.open(mode="r|", fileobj=sys.stdin)
for tarinfo in tar:
    tar.extract(tarinfo)
tar.close()
```

7.20 `readline` — Interfaccia a GNU readline

Il modulo `readline` definisce un numero di funzioni usate o direttamente o dal modulo `rlcompleter` per facilitare il completamento e la lettura/scrittura del file storico da parte dell'interprete Python.

Il modulo `readline` definisce le seguenti funzioni:

`parse_and_bind`(*string*)
Analizza ed esegue una singola riga di un file di inizializzazione readline.

`get_line_buffer`()
Restituisce i contenuti correnti del buffer di riga.

insert_text(*string*)
Inserisce del testo nella riga di comando.

read_init_file([*filename*])
Analizza un file di inizializzazione readline. Il filename predefinito è l'ultimo filename usato.

read_history_file([*filename*])
Carica un file storico della readline. Il nome del file predefinito è '~/.history'.

write_history_file([*filename*])
Salva un file storico della readline. Il nome del file predefinito è '~/.history'.

clear_history()
Cancella lo storico corrente. (Nota: questa funzione non è disponibile se la versione della GNU readline installata non la supporta.) Nuovo nella versione 2.4.

get_history_length()
Restituisce la lunghezza desiderata del file dello storico. Valori negativi implicano una dimensione illimitata del file dello storico.

set_history_length(*length*)
Imposta il numero di righe da salvare nel file dello storico. `write_history_file()` sfrutta questo valore per troncare il file dello storico durante il salvataggio. Valori negativi implicano una dimensione illimitata del file dello storico.

get_current_history_length()
Restituisce il numero di righe immagazzinate al momento nello storico. Questo è differente da `get_history_length()`, che restituisce il massimo numero di righe che possono essere scritte nel file dello storico. Nuovo nella versione 2.3.

get_history_item(*index*)
Restituisce il contenuto corrente degli elementi dello storico in *index*. Nuovo nella versione 2.3.

redisplay()
Cambia la visualizzazione sullo schermo per riflettere il contenuto corrente del buffer di riga. Nuovo nella versione 2.3.

set_startup_hook([*function*])
Imposta o rimuove la funzione di startup_hook. Se *function* viene specificata, verrà usata come nuova funzione di startup_hook; se viene omessa o è `None`, ogni funzione di aggancio già installata viene rimossa. La funzione di startup_hook viene chiamata senza argomenti appena prima che readline stampi il primo prompt.

set_pre_input_hook([*function*])
Imposta o rimuove la funzione di pre_input_hook. Se *function* viene specificata, sarà usata come nuova funzione di pre_input_hook; se è omessa o è `None`, ogni funzione di aggancio già installata viene rimossa. La funzione di startup_hook viene chiamata senza argomenti dopo che il primo prompt è stato stampato e appena prima che readline inizi la lettura dei caratteri in input.

set_completer([*function*])
Imposta o rimuove la funzione di completamento. Se *function* viene specificata, sarà usata come nuova funzione di completamento; se viene omessa o è `None`, ogni funzione di completamento già installata viene rimossa. La funzione di completamento viene chiamata come *function*(*text*, *state*), con *state* in `0`, `1`, `2`, ..., finché non restituisce un valore diverso da una stringa. Dovrebbe restituire il prossimo completamento possibile che comincia con *text*.

get_completer()
Restituisce la funzione di completamento, o `None` se non è stata impostata alcuna funzione di completamento. Nuovo nella versione 2.3.

get_begidx()
Restituisce l'indice iniziale da cui partirebbe con il completamento tramite tab, readline.

get_endidx()
Restituisce l'indice finale che aggiungerebbe con il completamento tramite tab, readline.

set_completer_delims(*string*)
Imposta i delimitatori di parola di readline per il completamento tramite tab.

get_completer_delims()
Restituisce i delimitatori di parola di readline per il completamento tramite tab.

add_history(*line*)
Aggiunge una riga allo storico nel buffer, come se fosse l'ultima riga digitata.

Vedete anche:

Modulo `rlcompleter` (sezione 7.21):
Completamento degli identificatori Python al prompt interattivo.

7.20.1 Esempio

L'esempio seguente dimostra come usare le funzioni di lettura e scrittura sullo storico del modulo `readline` per caricare e salvare automaticamente i file dello storico chiamati '.pyhist' dalla home directory dell'utente. Il codice sottostante verrebbe normalmente eseguito automaticamente durante le sessioni interattive dal file PYTHONSTARTUP dell'utente.

```
import os
histfile = os.path.join(os.environ["HOME"], ".pyhist")
try:
    readline.read_history_file(histfile)
except IOError:
    pass
import atexit
atexit.register(readline.write_history_file, histfile)
del os, histfile
```

7.21 `rlcompleter` — Funzione di completamento per la GNU readline

Il modulo `rlcompleter` definisce una funzione di completamento per il modulo `readline` completando identificatori e parole chiave Python valide.

Questo modulo è UNIX-specifico a causa della sua dipendenza dal modulo `readline`.

Il modulo `rlcompleter` definisce la classe `Completer`.

Esempio:

```
>>> import rlcompleter
>>> import readline
>>> readline.parse_and_bind("tab: complete")
>>> readline. <TAB PRESSED>
readline.__doc__          readline.get_line_buffer  readline.read_init_file
readline.__file__         readline.insert_text      readline.set_completer
readline.__name__         readline.parse_and_bind
>>> readline.
```

Il modulo `rlcompleter` è progettato per essere utilizzato con la modalità interattiva di Python. Un utente può aggiungere le linee seguenti al suo file di inizializzazione (identificato dalla variabile di ambiente PYTHONSTARTUP) per poter disporre del completamento automatico tramite `Tab`.

```
try:
    import readline
except ImportError:
    print "Module readline not available."
else:
    import rlcompleter
    readline.parse_and_bind("tab: complete")
```

7.21.1 Oggetti completer

Gli oggetti Completer possiedono il seguente metodo:

`complete`(*text*, *state*)

Restituisce il completamento *state* per *text*.

Se chiamato su un testo *text* che non include un carattere punto ('`.`'), verrà completato in base ai nomi definiti al momento in `__main__`, `__builtin__` e dalle parole chiave (come definite dal modulo `keyword`).

Se chiamato per un nome puntato, proverà a valutare ogni cosa verso l'ultima parte senza ovvi effetti indesiderati (le funzioni non verranno valutate, ma si possono generare chiamate a `__getattr__()`), sull'ultima parte, e verranno trovate corrispondenze per il resto tramite la funzione `dir()`.

CAPITOLO

OTTO

Servizi specifici per Unix

I moduli descritti in questo capitolo forniscono interfacce a caratteristiche che sono uniche del sistema operativo UNIX, o in qualche caso, a qualcuna delle molte varianti di questo. Qui c'è una panoramica:

posix	Le più comuni chiamate di sistema POSIX (normalmente usate tramite il modulo `os`).
pwd	Il database delle password (`getpwnam()` e compagnia).
grp	Il database dei gruppi (`getgrnam()` e compagnia).
crypt	La funzione `crypt()` viene usata per verificare le password UNIX.
dl	Chiamare funzioni C in oggetti condivisi.
dbm	L'interfaccia "database" standard, basata su ndbm.
gdbm	Reinterpretazione GNU di dbm.
termios	Controllo tty in stile POSIX.
TERMIOS	Costanti simboliche richieste dal modulo `termios`.
tty	Funzioni di utilità per eseguire comuni operazioni di controllo sui terminali.
pty	Gestione di pseudo terminali per SGI e Linux.
fcntl	Le chiamate di sistema `fcntl()` e `ioctl()`.
pipes	Una interfaccia Python per le pipeline della shell UNIX.
posixfile	Un oggetto simile a file con il supporto per il locking.
resource	Un'interfaccia che fornisce informazioni sull'impiego delle risorse nel processo corrente.
nis	Interfaccia alla libreria NIS di Sun (Yellow Pages).
syslog	Un'interfaccia per le procedure syslog di UNIX.
commands	Utilità per eseguire comandi esterni.

8.1 `posix` — Le più comuni chiamate di sistema POSIX

Questo modulo fornisce l'accesso alle funzionalità del sistema operativo consone allo standard C ed allo standard POSIX (un'interfaccia UNIX leggermente trasformata).

Non importate direttamente questo modulo. Importate invece il modulo `os`, che fornisce una versione *portabile* di questa interfaccia. Su UNIX, il modulo `os` fornisce un superinsieme dell'interfaccia `posix`. Su sistemi operativi non UNIX il modulo `posix` non è disponibile, ma un sotto insieme è sempre disponibile tramite l'interfaccia `os`. Una volta che `os` viene importato, *non* ci sono cali prestazionali nell'usarlo al posto di `posix`. Inoltre, `os` fornisce qualche funzionalità aggiuntiva, come chiamare automaticamente `putenv()` quando viene modificata una voce in `os.environ`.

La descrizione sottostante è molto concisa; fate riferimento alla voce corrispondente nel manuale UNIX (o nella documentazione POSIX) per maggiori informazioni. Gli argomenti chiamati *path* si riferiscono ad un percorso fornito come stringa.

Gli errori vengono riportati come eccezioni; per gli errori di tipo vengono sollevate le solite eccezioni, mentre gli errori riportati dalle chiamate di sistema sollevano l'eccezione `error` (un sinonimo per l'eccezione standard `OSError`), descritta sotto.

8.1.1 Supporto per file di grandi dimensioni

Alcuni sistemi operativi(inclusi AIX, HPUX, Irix e Solaris) forniscono supporto per file che sono più grandi di 2 Gb da un modello di programmazione C dove gli `int` ed i `long` sono valori di 32 bit. Questo viene in genere compiuto definendo la dimensione e l'offset rilevante dei tipi come valori a 64-bit. Questi file vengono a volte chiamati *large files*large files.

Il supporto per i file di grandi dimensioni viene abilitato in Python quando la dimensione di un `off_t` è più grande di un `long` ed è disponibile il tipo `long long` grande almeno come un `off_t`. I long del Python vengono quindi usati per rappresentare le dimensioni del file, così come gli offsets ed altri valori che possono eccedere la portata di un int del Python. Potrebbe essere necessario configurare e compilare Python con talune opzioni nel compilatore che abilitino questa modalità. Per esempio, viene abilitato come predefinito nella recente versione di Irix, ma con Solaris 2.6 e 2.7 avrete bisogno di qualcosa del tipo:

```
CFLAGS="‘getconf LFS_CFLAGS‘" OPT="-g -O2 $CFLAGS" \
        ./configure
```

Su sistemi Linux con capacità di gestione dei file di grandi dimensioni, questo dovrebbe funzionare:

```
CFLAGS='-D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64' OPT="-g -O2 $CFLAGS" \
        ./configure
```

8.1.2 Contenuto del modulo

Il modulo `posix` definisce il seguente dato:

`environ`
: Un dizionario che rappresenta la stringa d'ambiente al momento in cui è stato lanciato l'interprete. Per esempio `environ['HOME']` rappresenta il percorso della propria home directory, equivalente a `getenv(HOME)` in C.

  La modifica di questo dizionario non influenza la stringa d'ambiente passata da `execv()`, `popen()` o `system()`; se avete la necessità di modificare l'ambiente, passate `environ` a `execve()` oppure aggiungete gli assegnamenti delle variabili e le istruzioni export alle stringhe di comando per `system()` oppure `popen()`.

  Note: Il modulo `os` fornisce un'implementazione alternativa di `environ` che aggiorna l'ambiente in caso di modifica. Inoltre l'aggiornamento di `os.environ` renderà obsoleto questo dizionario. Viene raccomandato l'utilizzo della versione nel modulo `os` di questa funzione al posto dell'accesso diretto al modulo `posix`.

Ulteriori contenuti di questo modulo dovrebbero essere utilizzati solo tramite il modulo `os`; per ulteriori informazioni vedete la documentazione riguardante quel modulo.

8.2 pwd — Il database delle password

Questo modulo permette l'accesso agli account degli utenti ed al database delle password sotto UNIX. È disponiblie su tutte le versioni di UNIX.

Le voci nel database delle password vengono riportate come un oggetto simile ad una tupla (tipo record n.d.T.), i cui attributi corrispondono ai membri della struttura `passwd` (di seguito vengono riportati gli attributi, vedete `<pwd.h>`):

Indice	Attributo	Significato
0	`pw_name`	Nome di login
1	`pw_passwd`	Password criptata facoltativa
2	`pw_uid`	ID dell'utente in formato numerico
3	`pw_gid`	ID del gruppo in formato numerico
4	`pw_gecos`	Nome utente o campo contenente un commento
5	`pw_dir`	Home directory dell'utente
6	`pw_shell`	Interprete dei comandi dell'utente

Gli elementi uid e gid sono valori interi, gli altri sono stringhe. Nel caso una voce richiesta non possa essere trovata viene sollevata l'eccezione `KeyError`.

Note: Tradizionalmente in UNIX il campo `pw_passwd` contiene una password criptata con un algoritmo derivato da DES (vedete il modulo `crypt`). Comunque la maggior parte degli UNIX moderni utilizzano un sistema chiamato *shadow password*. Su questi Unix il campo `pw_passwd` contiene solo un aterisco (`'*'`) oppure la lettera '`x`', mentre la password criptata viene salvata nel file '/etc/shadow', che non è leggibile dagli utenti comuni.

Questo modulo definisce i seguenti elementi:

`getpwuid`(*uid*)
Restituisce la password immessa nel database relativa all'UID numerico dell'utente.

`getpwnam`(*name*)
Restituisce la password immessa nel database relativa al nome dell'utente.

`getpwall`()
Restituisce un elenco di tutte le password disponibili nel database in ordine casuale.

Vedete anche:

Modulo `grp` (sezione 8.3):
Un'interfaccia per il database dei gruppi, simile a questo.

8.3 `grp` — Il database dei gruppi

Questo modulo fornisce l'accesso al database dei gruppi sotto UNIX. È disponibile su tutte le versioni di UNIX.

Le voci nel database dei gruppi vengono riportate come un oggetto simil-tupla (tipo record n.d.T.), i cui attributi corrispondono ai membri della struttura `group` (di seguito vengono riportati gli attributi, vedete `<pwd.h>`):

Indice	Attributo	Significato
0	gr_name	Il nome del gruppo
1	gr_passwd	Password (criptata) del gruppo; spesso vuota
2	gr_gid	ID numerico del gruppo
3	gr_mem	Tutti i nomi utente dei membri del gruppo

Il gid è un valore intero, nome e password sono stringhe, mentre l'elenco dei membri è una lista di stringhe. (Notate che molti utenti non vengono elencati esplicitamente come membri di un gruppo, anche se accomunati dal database delle password. Interrrogate entrambi i database per ottenere delle informazioni di appartenenza complete.)

Questo modulo definisce i seguenti elementi:

`getgrgid`(*gid*)
Restituisce la voce nel database dei gruppi corrispondente all'ID di gruppo numerico passato. Viene sollevata l'eccezione `KeyError` se il dato richiesto non viene trovato.

`getgrnam`(*name*)
Restituisce la voce nel database dei gruppi corrispondente al nome di gruppo passato. Viene sollevata l'eccezione `KeyError` se il dato richiesto non viene trovato.

`getgrall`()
Restituisce un elenco, in ordine arbitrario, di tutte le voci dei gruppi disponibili.

Vedete anche:

Modulo pwd (sezione 8.2):
Un'interfaccia per il database degli utenti, simile a questa.

8.4 crypt — Funzione per verificare le password UNIX

Questo modulo implementa un'interfaccia della procedura *crypt*(3), che è una funzione di hash a senso unico basata su un algoritmo DES modificato; vedete anche la relativa pagina man di UNIX per ulteriori dettagli. Tra i possibili utilizzi annoveriamo la possibilità degli script Python di accettare password digitate dall'utente o quella di tentare la rottura di password UNIX tramite un dizionario.

crypt(*word, salt*)
word sarà in genere una password utente digitata al prompt o in un'interfaccia grafica. *salt* di solito è una stringa casuale di due caratteri utilizzata per perturbare l'algoritmo DES in una delle sue 4096 modalità. I caratteri in *salt* devono essere compresi nell'insieme `[./a-zA-Z0-9]`. Questo metodo restituisce l'hash della password sotto forma di stringa, che sarà composta da caratteri presi dallo stesso alfabeto di *salt* (i primi due caratteri rappresentano *salt* stesso).

Un semplice esempio ne illustra un utilizzo tipico:

```
import crypt, getpass, pwd

def login():
    username = raw_input('Python login:')
    cryptedpasswd = pwd.getpwnam(username)[1]
    if cryptedpasswd:
        if cryptedpasswd == 'x' or cryptedpasswd == '*':
            raise "Spiacente, al momento non è disponibile il supporto per le password shado
        cleartext = getpass.getpass()
        return crypt.crypt(cleartext, cryptedpasswd[:2]) == cryptedpasswd
    else:
        return 1
```

8.5 dl — Chiamare funzioni C in oggetti condivisi

Il modulo `dl` definisce un'interfaccia per la funzione `dlopen()` che è l'interfaccia più comune sulle piattaforme UNIX per gestire dinamicamente delle librerie collegate. Permette al programma di richiamare funzioni arbitrarie in queste librerie.

Note: Questo modulo non funzionerà a meno che `sizeof(int) == sizeof(long) == sizeof(char *)` Se non è questo il caso, al momento dell'import verrà sollevata un'eccezione `SystemError`.

Il modulo `dl` definisce le seguenti funzioni:

open(*name*[, *mode* = *RTLD_LAZY*])
Apre un file oggetto condiviso e restituisce un gestore. *mode* indica che il binding (NdT: collegamento, legame) verrà posticipato (`RTLD_LAZY`) oppure verrà immediatamente eseguito (`RTLD_NOW`). Il valore predefinito è `RTLD_LAZY`. Notate che alcuni sistemi non supportano `RTLD_NOW`.

Il valore restituito è un dlobject.

Il modulo `dl` definisce le seguenti costanti:

RTLD_LAZY
Utile come argomento di `open()`.

RTLD_NOW
Utile come argomento di `open()`. Notate che su sistemi che non suportano il binding immediato, questa

costante non apparirà nel modulo. Per ottenere la massima portabilità, utilizzate `hasattr()` in modo da determinare se il sistema supporta o meno il binding immediato.

Il modulo `dl` definisce le seguenti eccezioni:

exception `error`
L'eccezione sollevata quando si verifica un errore all'interno delle procedure di caricamento e collegamento dinamico.

Esempio:

```
>>> import dl, time
>>> a=dl.open('/lib/libc.so.6')
>>> a.call('time'), time.time()
(929723914, 929723914.498)
```

Questo esempio è stato provato su un sistema Debian GNU/Linux, ed è una buona prova del fatto che utilizzare questo modulo è spesso una pessima idea.

8.5.1 Oggetti dl

Gli oggetti `dl` restituiti dalla funzione `open()` precedente, possiedono i seguenti metodi:

`close``()`
Libera tutte le risore, eccetto la memoria.

`sym``(`*name*`)`
Restituisce il puntatore per la funzione chiamata *name*, sotto forma di numero, se esiste nell'oggetto condiviso referenziato, altrimenti restituisce `None`. Questo è utile in codice del tipo:

```
>>> if a.sym('time'):
...     a.call('time')
... else:
...     time.time()
```

(Notate che questa funzione restituirà un numero diverso da zero, visto che zero è il puntatore `NULL`)

`call``(`*name*`[`*, arg1*`[`*, arg2...* `]]`)
Chiama la funzione denominata *name*, nell'oggetto condiviso referenziato. Gli argomenti devono essere o interi Python, che verranno passati così come sono, o stringhe Python, alle quali verrà passato un puntatore, o `None`, che verrà passato come `NULL`. Notate che le stringhe dovrebbero venir passate alle funzioni solamente come `const char*`, visto che a Python non piace mutare le proprie strighe.

Devono esserci al massimo 10 argomenti, e gli argomenti non forniti verranno trattati come `None`. Il valore restituito dalla funzione deve essere un `long` C, che è un intero Python.

8.6 `dbm` — Semplice interfaccia "database"

Il modulo `dbm` fornisce un'interfaccia per la libreria UNIX (`n`)`dbm`. Gli oggetti `dbm` si comportano come le mappe (dizionari), eccetto per il fatto che chiavi e valori sono sempre stringhe. Stampando un oggetto `dbm` non vengono stampati le chiavi ed i valori, ed i metodi `items()` e `values()` non vengono supportati.

Questo modulo può essere usato con la "classica" interfaccia ndbm, l'interfaccia BSD di compatibilità DB, o l'interfaccia compatibile GNU GDBM. Su UNIX, lo script **configure** tenterà di localizzare il file header appropriato per semplificare la compilazione di questo modulo.

Il modulo definisce i seguenti attributi:

exception `error`
Solleva specifici errori `dbm`, come errori di I/O. L'eccezione `KeyError` viene sollevata in caso di errori generali di mappatura, come specificare una chiave non corretta.

`library`
Nome della libreria d'implementazione `ndbm` utilizzata.

`open`(*filename*[, *flag*[, *mode*]])
Apre un database dbm e restituisce un oggetto `dbm`. L'argomento *filename* è il nome del file database (senza l'estensione '.dir' o '.pag'; notate che l'implementazione dell'interfaccia DB BSD aggiungerà l'estensione '.db' e creerà solamente un file).

L'argomento facoltativo *flag* deve possedere uno di questi valori:

Valore	Significato
`'r'`	Apre un database esistente in sola lettura (predefinito)
`'w'`	Apre un database esistente in lettura e scrittura
`'c'`	Apre un database in lettura e scrittura, creandolo se non esiste
`'n'`	Crea sempre un nuovo database vuoto, aperto in lettura e scrittura

L'argomento facoltativo *mode* rappresenta la modalità UNIX del file, usata solamente quando il database deve essere creato. Il valore ottale predefinito è `0666`.

Vedete anche:

Modulo `anydbm` (sezione 7.10):
Interfaccia generica ai database in stile `dbm`.

Modulo `gdbm` (sezione 8.7):
Intefaccia simile alla libreria GNU GDBM.

Modulo `whichdb` (sezione 7.12):
Un modulo utile da utilizzare per determinare il tipo di un database già esistente.

8.7 `gdbm` — Reinterpretazione GNU di dbm

Questo modulo è piuttosto simile a `dbm`, ma utilizza al suo posto `gdbm` per fornire qualche funzionalità aggiuntiva. Notate che i formati dei file creati da `gdbm` e `dbm` sono incompatibili.

Il modulo `gdbm` fornisce un'interfaccia per la libreria GNU DBM. Gli oggetti `gdbm` si comportano come mappe (dizionari), tranne per il fatto che chiavi e valori sono sempre stringhe. Stampando un oggetto `gdbm` non vengono stampati chiavi e valori, ed i metodi `items()` e `values()` non vengono supportati.

Il modulo definisce le seguenti costanti e funzioni:

exception `error`
Solleva un errore specifico `gdbm`, come un errore di I/O. In caso di errori generali di mappatura, come lo specificare un chiave non corretta, viene sollevata un'eccezione `KeyError`.

`open`(*filename*, [*flag*, [*mode*]])
Apre un database `gdbm` e restituisce un oggetto `gdbm`. L'argomento *filename* è il nome del file del database.

L'argomento facoltativo *flag* può essere `'r'` (per aprire un database esistente in sola lettura — predefinito), `'w'` (per aprire un database esistente in lettura e scrittura), `'c'` (per creare un database se non esiste) o `'n'` (che crea sempre un nuovo database vuoto).

I seguenti caratteri addizionali possono essere inseriti a seguito dell'opzione per controllare in che modo il database venga aperto:

- `'f'` — Apre il database in modalità veloce. La scrittura nel database non sarà sincronizzata.
- `'s'` — Modalità sincronizzata. Questo provocherà l'immediata scrittura sul file delle modifiche al database.
- `'u'` — Non blocca il database.

Non tutte le opzioni sono valide per tutte le versioni di `gdbm`. La costante di modulo `open_flags` è una stringa contenente i caratteri delle opzioni supportate. Viene sollevata l'eccezione `error` nel caso venga specificata un'opzione non valida.

L'argomento facoltativo *mode* rappresenta la modalità UNIX del file, usata solamente quando il database deve essere creato. Il valore predefinito in ottale è `0666`.

In aggiunta ai soliti metodi tipici di un dizionario, gli oggetti `gdbm` possiedono i seguenti metodi:

`firstkey``()`
È possibile iterare su ogni chiave nel database usando questo metodo ed il metodo `nextkey()`. Il risultato viene ordinato a seconda dei valori interni di hash di `gdbm`, e non verranno ordinati secondo il valore delle chiavi. Questo metodo restituisce la chiave di partenza.

`nextkey``(`*key*`)`
Restituisce la chiave che segue la chiave *key* nel risultato. Il codice seguente stampa ogni chiave nel database `db`, senza dover creare una lista in memoria che le contenga tutte:

```
k = db.firstkey()
while k != None:
    print k
    k = db.nextkey(k)
```

`reorganize``()`
Se avete eseguito molte cancellazioni e vorreste restringere lo spazio usato dal file `gdbm`, questa procedura riorganizzerà il database. `gdbm` non diminuirà la dimensione di un file database, se non tramite l'utilizzo di questa riorganizzazione; altrimenti, lo spazio cancellato dal file verrà preso e riutilizzato al momento dell'aggiunta di nuove coppie (chiave,valore).

`sync``()`
Quando il database è stato aperto in modalità veloce, questo metodo forzerà la scrittura su disco di ogni dato modificato.

Vedete anche:

Modulo `anydbm` (sezione 7.10):
Interfaccia generica ai database in stile `dbm`.

Modulo `whichdb` (sezione 7.12):
Un modulo utile da utilizzare per determinare il tipo di un database esistente.

8.8 `termios` — Controllo tty in stile POSIX

Questo modulo fornisce un'interfaccia alle chiamate POSIX per il controllo I/O delle tty. Per una descrizione completa di queste chiamate, si vedano le pagine dei manuali POSIX oppure UNIX. Questo modulo è disponibile solo per quelle versioni di UNIX che supportano il controllo I/O delle tty in stile POSIX *termios* (e solo se configurato durante l'installazione).

Tutte le funzioni in questo modulo richiedono un descrittore di file *fd* come primo argomento. Esso può essere un descrittore di file sotto forma di numero intero, come quello restituito da `sys.stdin.fileno()`, oppure un oggetto file, come `sys.stdin` stesso.

Questo modulo definisce anche tutte le costanti necessarie per lavorare con le funzioni qui fornite; queste possiedono lo stesso nome delle loro controparti in C. Fate riferimento alla documentazione del vostro sistema per ulteriori informazioni sull'utilizzo di queste interfacce per il controllo dei terminali.

Il modulo definisce le seguenti funzioni:

`tcgetattr``(`*fd*`)`
Restituisce una lista contenente gli attributi tty per il descrittore di file *fd*, come i seguenti: `[`*iflag*, *oflag*, *cflag*, *lflag*, *ispeed*, *ospeed*, *cc*`]` dove *cc* è una lista dei caratteri speciali della tty (ognuno è una stringa di lunghezza 1, eccetto gli elementi con indice `VMIN` e `VTIME`, che sono interi quando questi campi vengono definiti). L'interpretazione delle opzioni e delle velocità, così come l'indicizzazione nell'array *cc* deve essere fatta usando le costanti simboliche definite nel modulo `termios`.

`tcsetattr``(`*fd*, *when*, *attributes*`)`
Imposta gli attributi tty per il descrittore di file *fd* usando *attributes*, che è una lista come quella restituita da `tcgetattr()`. L'argomento *when* determina quando debbano venire modificati gli attributi: `TCSANOW`

per modificare immediatamente, `TCSADRAIN` per modificare dopo aver trasmesso tutti gli output accodati o `TCSAFLUSH` per modificare dopo aver trasmesso tutti gli output accodati ed aver scartato tutti gli input in coda.

`tcsendbreak`(*fd, duration*)
Manda un segnale di interruzione sul descrittore di file *fd*. Una durata, *duration*, uguale a zero manda un segnale di 0.25-0.5 secondi; una *duration* diversa da zero ha un significato dipendente dal sistema.

`tcdrain`(*fd*)
Attende finché non sia stato trasmesso tutto l'output scritto sul descrittore di file *fd*.

`tcflush`(*fd, queue*)
Scarta i dati accodati sul descrittore di file *fd*. Il selettore *queue* specifica quale coda: `TCIFLUSH` per la coda di input, `TCOFLUSH` per la coda di output, `TCIOFLUSH` per entrambe.

`tcflow`(*fd, action*)
Sospende o riavvia l'input o l'output sul descrittore di file *fd*. L'argomento *action* può essere `TCOOFF` per sospendere l'output, `TCOON` per riavviare l'output, `TCIOFF` per sospendere l'input e `TCION` per riavviare l'input.

Vedete anche:

Modulo `tty` (sezione 8.10):
Funzioni comode per operazioni comuni di controllo sui terminali.

8.8.1 Esempio

Ecco una funzione che chiede una password senza mostrarne l'inserimento dei caratteri. Notate che questa tecnica utilizza una chiamata separata a `tcgetattr()` e un'istruzione `try ... finally` per assicurarsi che siano ripristinati esattamente i vecchi attributi tty, senza riguardo a quanto avviene:

```
def getpass(prompt = "Password: "):
    import termios, sys
    fd = sys.stdin.fileno()
    old = termios.tcgetattr(fd)
    new = termios.tcgetattr(fd)
    new[3] = new[3] & ~termios.ECHO          # lflags
    try:
        termios.tcsetattr(fd, termios.TCSADRAIN, new)
        passwd = raw_input(prompt)
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old)
    return passwd
```

8.9 `TERMIOS` — Costanti utilizzate col modulo `termios`

Deprecato dalla versione 2.1. Importate invece le costanti necesssarie dal modulo `termios`.

Questo modulo definisce le costanti simboliche richieste per utilizzare il modulo `termios` (vedete la sezione precedente). Vedete le pagine di manuale POSIX o UNIX per ottenere un elenco di queste costanti.

8.10 `tty` — Funzioni per il controllo dei terminali

Il modulo `tty` definisce le funzioni per impostare il tty nelle modalità cbreak e raw.

`tty` richiede il modulo `termios`, percui funzionerà solamente sui sistemi UNIX.

Il modulo `tty` definisce le seguenti funzioni:

setraw(*fd*[, *when*])
: Cambia la modalità del descrittore di file *fd* in raw. Se *when* viene omesso, il valore predefinito sarà `termios.TCAFLUSH`, e verrà passato a `termios.tcsetattr()`.

setcbreak(*fd*[, *when*])
: Cambia la modalità del descrittore di file *fd* in cbreak. Se *when* viene omesso, il valore predefinito sarà `termios.TCAFLUSH`, e verrà passato a `termios.tcsetattr()`.

Vedete anche:

Modulo `termios` (sezione 8.8):
: Interfaccia per il controllo dei terminali a basso livello.

Modulo `TERMIOS` (sezione 8.9):
: Costanti utili per operazioni di controllo sui terminali

8.11 `pty` — Utilità per pseudo terminali

Il modulo `pty` definisce delle operazioni per gestire il concetto di pseudo terminali: ovverosia lanciare un nuovo processo ed essere in grado di scrivervi e leggervi dal suo terminale di controllo il con la programmazione.

Dato che la gestione degli pseudo terminali è fortemente dipendente dalla piattaforma usata, il codice utilizzabile esiste esclusivamente per SGI e Linux. (Il codice per Linux dovrebbe funzionare su altre piattaforme, ma non è ancora stato testato.)

Il modulo `pty` definisce le seguenti funzioni:

fork()
: Esegue una fork. Connette il terminale di controllo del figlio ad uno pseudo terminale. Il valore restituito è `(`*pid*`,` *fd*`)`. Notate che il figlio ottiene *pid* 0 ed il suo *fd non è valido*. Il valore restituito dal padre è il *pid* del figlio, ed *fd* è un descrittore di file connesso al terminale di controllo del figlio (ed anche agli standard input ed output del figlio).

openpty()
: Apre una nuova coppia di pseudo terminali, usando se possibile `os.openpty()`, o il codice di emulazione per SGI e sistemi generici UNIX. Restituisce una nuova coppia di descrittori di file `(`*master*`,` *slave*`)`, rispettivamente per i terminali master e slave.

spawn(*argv*[, *master_read*[, *stdin_read*]])
: Genera un processo e collega il suo terminale di controllo con lo standard input del processo corrente. Viene spesso utilizzato per eludere i programmi che insistono sulla lettura dal terminale di controllo.

  Le funzioni *master_read* e *stdin_read* dovrebbero essere funzioni che leggono da un descrittore di file. Le funzioni predefinite di norma provano a leggere 1024 bytes ogni volta che vengono invocate.

8.12 `fcntl` — Le chiamate di sistema `fcntl()` e `ioctl()`

Questo modulo esegue controlli di file e controlli di I/O su descrittori di file. È un'interfaccia per le procedure UNIX `fcntl()` e `ioctl()`.

Tutte le funzioni in questo modulo richiedono un descrittore di file *fd* come loro primo parametro. Questo può essere un descrittore di file intero, come restituito da `sys.stdin.fileno()`, o un oggetto file, come lo stesso `sys.stdin`, che fornisce un metodo `fileno()` il quale restituisce un descrittore di file genuino.

Il modulo definisce le seguenti funzioni:

fcntl(*fd*, *op*[, *arg*])
: Esegue l'operazione richiesta sul descrittore di file *fd* (gli oggetti file che forniscono un metodo `fileno()` vengono comunque accettati). L'operazione viene definita da *op* e dipende dal sistema operativo. Questi codici si trovano anche nel modulo `fcntl`. L'argomento *arg* è facoltativo ed il suo valore predefinito è l'intero di valore `0`. Quando è presente, può assumere sia un valore intero che una stringa. Se l'argomento è assente o è un valore intero, il valore restituito da questa funzione è lo stesso della chiamata C

`fcntl()`. Quando l'argomento è una stringa, rappresenta una struttura binaria, per esempio quella creata da `struct.pack()`. Il dato binario viene copiato in un buffer il cui indirizzo viene passato alla chiamata C `fcntl()`. Il valore restituito dopo una chiamata andata a buon fine è il contenuto del buffer, convertito in un oggetto stringa. La lunghezza della stringa restituita sarà la stessa dell'argomento *arg*. Questa lunghezza è limitata a 1024 byte. Se l'informazione restituita nel buffer dal sistema operativo è più grande di 1024 byte, molto probabilmente causerà una violazione della segmentazione oppure una corruzione di dati più subdola.

Se `fcntl()` fallisce, viene sollevata un'eccezione `IOError`.

`ioctl`(*fd*, *op*[, *arg*[, *mutate_flag*]])
Questa funzione è identica a `fcntl()`, a parte il fatto che le operazioni vengono tipicamente definite nel modulo libreria `termios` e la gestione degli argomenti è persino un po' più complicata.

Il parametro *arg* può essere un intero, può essere assente (e quindi trattato come l'intero `0`), piò essere un oggetto che supporta l'interfaccia buffer in sola lettura (simile ad una stringa di testo Python) oppure un oggetto che supporta l'interfaccia buffer in lettura e scrittura.

In tutti i casi tranne l'ultimo, il comportamento è identico a quello della funzione `fcntl()`.

Se viene passato un buffer variabile, allora il comportamento di questa funzione verrà determinato dal valore del parametro *mutate_flag*.

Se è falso, la mutabilità del buffer verrà ignorata ed il comportamento sarà quello di un buffer in sola lettura, tranne che verrà evitato il limite sopra menzionato dei 1024 byte – finchè passate un buffer sufficientemente ampio da contenere ciò che il sistema operativo intende porvi, le cose dovrebbero funzionare.

Se *mutate_flag* è vero, allora il buffer viene (in effetti) passato alla chiamata di sistema sottostante `ioctl()`, l'ultimo codice restituito verrà passato alla chiamata Python ed i nuovi contenuti del buffer rifletterano l'azione della `ioctl()`. Questa è una flebile semplificazione, poiché se il buffer fornito è grande meno di 1024 byte, viene prima copiato in un buffer statico lungo 1024 bytes, poi passato alla `ioctl()` ed infine copiato di nuovo nel buffer di iniziale.

Se *mutate_flag* non viene fornito, allora in Python 2.3 il valore predefinito è falso. È in previsione di cambiare questo dettaglio nelle prossime versioni di Python: in 2.4, non fornire *mutate_flag* causerà un avvertimento, ma la funzione lavorerà normalmente, mentre in una versione successiva alla 2.5 il valore predefinito sarà true.

Un esempio:

```
>>> import array, fcntl, struct, termios, os
>>> os.getpgrp()
13341
>>> struct.unpack('h', fcntl.ioctl(0, termios.TIOCGPGRP, "  "))[0]
13341
>>> buf = array.array('h', [0])
>>> fcntl.ioctl(0, termios.TIOCGPGRP, buf, 1)
0
>>> buf
array('h', [13341])
```

`flock`(*fd*, *op*)
Compie l'operazione *op* di lock sul descrittore di file *fd* (gli oggetti file che forniscono un metodo `fileno()` vengono comunque accettati). Per i dettagli, vedete la pagina di manuale UNIX *flock(3)*(.) Su alcuni sistemi questa funzione viene emulata tramite `fcntl()`.

`lockf`(*fd*, *operation*, [*len*, [*start*, [*whence*]]])
Questo è essenzialmente un incapsulatore per le chiamate di locking `fcntl()`. *fd* è il descrittore del file su cui eseguire il lock o l'unlock mentre *operation* sarà uno dei seguenti valori:

- `LOCK_UN` – esegue l'unlock
- `LOCK_SH` – acquisisce un lock condiviso
- `LOCK_EX` – acquisisce un lock esclusivo

Quando *operation* vale `LOCK_SH` o `LOCK_EX`, può anche essere un OR bit a bit con `LOCK_NB` per evitare blocchi su acquisizioni del lock. Se viene usato `LOCK_NB` ed il lock non può essere acquisito, verrà sollevata una `IOError` e l'eccezione avra' un attributo *errno* impostato ad `EACCES` o `EAGAIN` (a seconda del sistema operativo; per portabilità, controllate entrambi i valori). Almeno su alcuni sistemi, `LOCK_EX` può essere utilizzata solo se il descrittore di file si riferisce ad un file aperto in scrittura.

length è il numero di byte su cui effettuare il lock, *start* è il byte di offset da dove inizia il lock, relativo a *whence*, mentre *whence* funziona come con `fileobj.seek()`, nello specifico:

- `0` – relativo all'inizio del file (`SEEK_SET`)
- `1` – relativo alla posizione corrente nel buffer (`SEEK_CUR`)
- `2` – relativo alla fine del file (`SEEK_END`)

Il valore predefinito per *start* è `0`, che significa partire dall'inizio del file. Il valore predefinito per *lenght* è `0` che significa effettuare il lock sino alla fine del file. Anche il valore predefinito per *whence* è `0`.

Esempi (tutti su sistemi compatibili con SVR4):

```
import struct, fcntl

file = open(...)
rv = fcntl(file, fcntl.F_SETFL, os.O_NDELAY)

lockdata = struct.pack('hhllhh', fcntl.F_WRLCK, 0, 0, 0, 0, 0)
rv = fcntl.fcntl(file, fcntl.F_SETLKW, lockdata)
```

Notate che nel primo esempio il valore restituito nella variabile *rv* conterrà un valore intero; nel secondo esempio conterrà un valore stringa. Il lay-out della struttura della variabile *lockdata* dipende dal sistema — quindi potrebbe essere meglio utilizzare la chiamata `flock()`.

Vedete anche:

Modulo `os` (sezione 6.1):
La funzione `os.open` supporta le opzioni di locking ed è disponibile per una grande varietà di piattaforme rispetto alle funzioni `lockf()` e `flock()`, fornendo una capacità di locking dei file più indipendente dalla piattaforma.

8.13 `pipes` — Interfaccia per le pipeline della shell

Il modulo `pipes` definisce una classe per astrarre il concetto di una *pipeline* — una sequenza di convertitori da un file ad un altro.

Dato che il modulo utilizza la riga di comando **/bin/sh**, viene richiesta una shell POSIX o compatibile per `os.system()` e `os.popen()`.

Il modulo `pipes` definisce la seguente classe:

class `Template()`
Un'astrazione di una pipeline.

Esempio:

```
>>> import pipes
>>> t=pipes.Template()
>>> t.append('tr a-z A-Z', '--')
>>> f=t.open('/tmp/1', 'w')
>>> f.write('hello world')
>>> f.close()
>>> open('/tmp/1').read()
'HELLO WORLD'
```

8.13.1 Oggetti Template

Gli oggetti template possiedono i seguenti metodi:

`reset``()`
Ripristina un template pipeline al suo stato iniziale.

`clone``()`
Restituisce un nuovo template pipeline, equivalente.

`debug``(`*flag*`)`
Se *flag* è vero, viene attivato il debugging. In caso contrario viene disattivato. Quando il debugging è attivo, i comandi da eseguire vengono stampati ed alla shell viene impostato il comando `set -x` per la modalità prolissa.

`append``(`*cmd, kind*`)`
Accoda una nuova azione alla fine. La variabile *cmd* deve essere un comando di bourne shell valido. La variabile *kind* consiste di due lettere.

La prima lettera può essere `'-'` (che significa che il comando legge il proprio standard input), `'f'` (che significa che il comando legge un file indicato nella riga di comando) o `'.'` (che significa che il comando non legge input, quindi deve essere il primo.)

Allo stesso modo, la seconda lettera può essere `'-'` (che significa che il comando scrive sullo standard output), `'f'` (che significa che il comando scrive su un file passato sulla riga di comando) o `'.'` (che significa che il comando non scrive niente, quindi deve essere l'ultimo).

`prepend``(`*cmd, kind*`)`
Aggiunge una nuova azione all'inizio. Vedete `append()` per la spiegazione degli argomenti.

`open``(`*file, mode*`)`
Restituisce un oggetto simile a file, aperto su *file*, ma letto o scritto tramite la pipeline. Notate che può essere passata solamente un'opzione tra `'r'` e `'w'`.

`copy``(`*infile, outfile*`)`
Copia *infile* in *outfile* attraverso la pipe.

8.14 `posixfile` — Oggetti simile a file con il supporto per il locking

Deprecato dalla versione 1.5. L'operazione di locking che questo modulo fornisce viene eseguita meglio e con maggiore portabilità dalla chiamata `fcntl.lockf()`.

Questo modulo implementa alcune funzionalità addizionali sugli oggetti file built-in. In particolare, implementa il locking (NdT: bloccaggio) dei file, il controllo sulle opzioni relative ai file ed una semplice interfaccia per duplicare gli oggetti file. Il modulo definisce un nuovo oggetto file, l'oggetto posixfile. Questi possiede tutti i metodi standard degli oggetti file ed aggiunge i metodi descritti sotto. Questo modulo funziona solamente su certi tipi di UNIX, poiché utilizza `fcntl.fcntl()` per il locking dei file.

Per istanziare un oggetto posixfile, usate la funzione `open()` del modulo `posixfile`. L'oggetto risultante somiglia e si comporta approssimativamente allo stesso modo di un oggetto file standard.

Il modulo `posixfile` definisce le seguenti costanti:

SEEK_SET
L'offset viene calcolato dall'inizio del file.

SEEK_CUR
L'offset viene calcolato dalla posizione corrente nel file.

SEEK_END
L'offset viene calcolato dalla fine del file.

Il modulo `posixfile` definisce le seguenti funzioni:

open(*filename*[, *mode*[, *bufsize*]])
Crea un nuovo oggetto posixfile in base al nome del file e la modalità passati in argomento. Gli argomenti *filename*, *mode* e *bufsize* vengono interpretati alla stessa maniera della funzione `open()`.

fileopen(*fileobject*)
Crea un nuovo oggetto posixfile a partire dall'oggetto file standard *fileobject* passato come argomento. L'oggetto risultante possiede lo stesso nome e la stessa modalità dell'oggetto file originale.

L'oggetto posixfile definisce gli ulteriori seguenti metodi:

lock(*fmt*, [*len*[, *start*[, *whence*]]])
Esegue il lock sulla sezione specificata del file alla quale l'oggetto file si riferisce. Il formato viene esposto sotto nella tabella. L'argomento *len* specifica la lunghezza della sezione sulla quale si dovrebbe eseguire il lock. Il valore predefinito è `0`. *start* specifica l'offset di partenza della sezione, il cui valore predefinito è `0`. L'argomento *whence* specifica da dove è relativo l'offset. Esso accetta una fra le costanti `SEEK_SET`, `SEEK_CUR` o `SEEK_END`. Il valore predefinito è `SEEK_SET`. Per maggiori informazioni su questi argomenti fate riferimento alla pagina del manuale *fcntl*(2) del vostro sistema.

flags([*flags*])
Imposta le opzioni specificate per il file al quale si riferisce l'oggetto file. Viene eseguito un OR tra le nuove e le vecchie opzioni, se non specificato altrimenti. Il formato viene spiegato sotto in una tabella. Senza l'argomento *flags* viene restituita una stringa che indica le opzioni correnti (lo stesso del modificatore '`?`'). Per maggiori informazioni riguardo le opzioni, fate riferimento alla pagina del manuale *fcntl*(2) del vostro sistema.

dup()
Duplica l'oggetto file, oltre al puntatore al file ed al suo descrittore sottostanti. L'oggetto risultante si comporta come se fosse stato aperto nuovamente.

dup2(*fd*)
Duplica il file oggetto, il sottostante puntatore al file, ed il descrittore di file. Il nuovo oggetto possiedera' il dato descrittore di file. Altrimenti l'oggetto risultante si comportera' come se fosse un nuovo oggetto aperto

file()
Restituisce l'oggetto file standard sul quale è basato l'oggetto posixfile. A volte questo metodo diviene necessario per funzioni che persistono su di un oggetto file standard.

Tutti i metodi sollevano un'eccezione `IOError` quando la richiesta fallisce.

I caratteri di formattazione per il metodo `lock()` possiedono i seguenti significati:

Formato	Significato
'`u`'	Esegue l'unlock sulla regione specificata
'`r`'	richiede un lock in lettura per la regione specificata
'`w`'	richiede un lock in scrittura per la regione specificata

I seguenti modificatori possono venire ulteriormente aggiunti al formato:

Modificatore	Significato	Note
'`|`'	Attende sino a quando il lock non viene assegnato	
'`?`'	Restituisce il primo lock in conflitto con il lock richiesto, o `None` se non vi è alcun conflitto	(1)

Note:

(1) Il lock restituito è nel formato (*mode*, *len*, *start*, *whence*, *pid*) dove *mode* è un carattere che rappresenta il tipo di lock ('r' or 'w'). Questo modificatore impedisce che una richiesta venga assegnata; compare solamente a scopo di interrogazione.

I caratteri di formattazione per il metodo `flags()` possiedono i seguenti significati:

Formato	Significato
'a'	opzione di sola aggiunta
'c'	opzione di chiusura dopo l'esecuzione
'n'	opzione di annullamento del ritardo (opzione chiamata anche non bloccante)
's'	opzione di sincronizzazione

Inoltre il seguenti modificatori possono venire aggiunti al formato:

Modificatore	Significato	Note
'!'	imposta le opzioni specificate in 'off', invece della predefinita 'on'	(1)
'='	rimpiazza le opzioni, invece dell'operazione predefinita 'OR'	(1)
'?'	restituisce una stringa nella quale i caratteri rappresentano le opzioni impostate.	(2)

Note:

(1) I modificatori '!' e '=' si escludono a vicenda.

(2) Questa stringa rappresenta le opzioni che si potrebbero presentare nel caso venissero alterate dalla stessa chiamata.

Esempi:

```
import posixfile

file = posixfile.open('/tmp/test', 'w')
file.lock('w|')
...
file.lock('u')
file.close()
```

8.15 `resource` — Informazioni sull'utilizzo delle risorse

Questo modulo fornisce dei meccanismi di base per misurare e controllare le risorse di sistema utilizzate da un programma.

Vengono utilizzate delle costanti simboliche per specificare risorse di sistema particolari e per richiedere informazioni sull'utilizzo riguardo il processo corrente o uno dei suoi figli.

Viene definita una singola eccezione per gli errori:

exception `error`
Le funzioni descritte sotto possono sollevare questa eccezione se la chiamata di sistema sottostante fallisce inaspettatamente.

8.15.1 Limiti della risorsa

L'utilizzo delle risorse può venire limitato tramite la funzione `setrlimit()` descritta sotto. Ogni risorsa viene controllata da una coppia di limiti: un limite soft ed uno hard. Il limite soft è il limite corrente, e può essere

eccezionalmente diminuito od aumentato da un processo. Il limite soft non può mai eccedere il limite hard. Il limite hard può venire diminuito fino ad un qualunque valore superiore a quello del limite soft, ma non può essere aumentato. (Solamente i processi con l'UID effettivo del super utente possono aumentare un limite hard.)

Il tipo di risorse che possono venire limitate dipendendono dal sistema. Vengono descritte nella pagina del manuale di *getrlimit*(2). Le risorse mostrate sotto vengono supportate quando il sistema operativo sottostante a sua volta le supporta; le risorse che non possono venire controllate o gestite dal sistema operativo non vengono definite in questo modulo su quelle piattaforme.

`getrlimit`(*resource*)
: Restituisce una tupla (*soft*, *hard*) con i limiti correnti soft ed hard della risorsa *resource*. Solleva l'eccezione `ValueError` se viene specificata una risorsa non valida, o `error` se la chiamata di sistema sottostante fallisce inaspettatamente.

`setrlimit`(*resource*, *limits*)
: Imposta nuovi limiti di consumo della risorsa *resource*. L'argomento *limits* deve essere una tupla (*soft*, *hard*) di due numeri interi che descrivono i nuovi limiti. Si può utilizzare un valore di `-1` per specificare il limite superiore massimo possibile.

  Solleva l'eccezione `ValueError` se viene specificata una risorsa non valida, se il nuovo limite soft eccede il limite hard, o se un processo tenta di aumentare il proprio limite hard (a meno che il processo non possieda un UID effettivo di super utente). Può anche sollevare l'eccezione `error` nel caso la chiamata di sistema sottostante fallisca.

Questi simboli definiscono le risorse il cui consumo può venire controllato utilizzando le funzioni `setrlimit()` e `getrlimit()` descritte sotto. I valori di questi simboli sono esattamente le costanti usate dai programmi C.

La pagina del manuale UNIX di *getrlimit*(2) elenca le risorse disponibili. Notate che non tutti i sistemi utilizzano lo stesso simbolo e lo stesso valore per denotare la medesima risorsa. Questo modulo non tenta di mascherare le differenze tra le piattaforme — i simboli non definiti per una piattaforma non saranno disponibili in questo modulo su quella piattaforma.

`RLIMIT_CORE`
: La dimensione massima (in byte) di un file core che il processo corrente può creare. Questo potrebbe causare la creazione di un file core incompleto se viene richiesto un core più grande per contenere l'intera immagine del processo.

`RLIMIT_CPU`
: L'ammontare massimo di tempo del processore (in secondi) che un processo può utilizzare. Se questo limite viene superato, viene mandato al processo un segnale `SIGXCPU`. (Vedete la documentazione del modulo `signal` per informazioni su come catturare questo segnale e fare qualcosa di utile, ad esempio aggiornare i file aperti sul disco).

`RLIMIT_FSIZE`
: La dimensione massima di un file che il processo può creare. Questo influenza solo lo stack del thread principale in un processo multi-thread.

`RLIMIT_DATA`
: La dimensione massima (in bytes) dell'heap del processo.

`RLIMIT_STACK`
: L'area massima (in bytes) dello stack delle chiamate per il processo corrente.

`RLIMIT_RSS`
: La dimensione massima della quantita‘ di memoria fisica (non-swap) utilizzata dal processo.

`RLIMIT_NPROC`
: Il numero massimo di processi che il processo corrente può creare.

`RLIMIT_NOFILE`
: Il numero massimo di descrittori di file aperti disponibili per il processo corrente.

`RLIMIT_OFILE`
: Il nome BSD per `RLIMIT_NOFILE`.

`RLIMIT_MEMLOCK`
: Il spazio d'indirizzi massimo che può essere mantenuto in memoria.

RLIMIT_VMEM
La più grande area di memoria mappata che il processo può occupare.

RLIMIT_AS
L'area massima (in bytes) dello spazio d'indirizzi che può venir presa dal processo.

8.15.2 Utilizzo delle risorse

Queste funzioni vengono utilizzate per avere informazioni sull'utilizzo delle risorse.

getrusage(*who*)
Questa funzione restituisce un oggetto che descrive le risorse consumate dal processo corrente e dai suoi figli, come specificato dal parametro *who*. Il parametro *who* dovrebbe venire specificato usando una delle costanti RUSAGE_* descritte sotto.

Ognuno dei campi del valore restituito descrive il modo in cui una particolare risorsa di sistema sia stata utilizzata, ad esempio la quantità di tempo trascorso nella modalità utente o il numero di volte in cui il processo è stato swappato fuori dalla memoria principale. Qualche valore è dipendente dal ciclo di clock interno, ad esempio la quantità di memoria che il processo sta utilizzando.

Per retrocompatibilità, i valori restituiti sono accessibili anche come una tupla di 16 elementi.

I campi ru_utime e ru_stime del valore restituito sono numeri in virgola mobile che rappresentano rispettivamente l'ammontare di tempo trascorso in modalità utente e la quantità di tempo trascorso in modalità sistema. I valori rimanenti sono interi. Consultate la pagina del manuale *getrusage*(2) per informazioni dettagliate su questi valori. Ecco qui presentato un breve sommario:

Indice	Campo	Risorsa
0	ru_utime	tempo in modalità utente (virgola mobile)
1	ru_stime	tempo in modalità sistema (virgola mobile)
2	ru_maxrss	la quantita' di memoria fisica (non-swap) utilizzata dal processo
3	ru_ixrss	dimensione della memoria condivisa
4	ru_idrss	dimensione della memoria non condivisa
5	ru_isrss	dimensione di stack non condiviso
6	ru_minflt	fault di pagina non richiedendo I/O
7	ru_majflt	fault di pagina richiedendo I/O
8	ru_nswap	numero di swap out
9	ru_inblock	blocco delle operazioni di input
10	ru_oublock	blocco delle operazioni di output
11	ru_msgsnd	messaggi inviati
12	ru_msgrcv	messaggi ricevuti
13	ru_nsignals	segnali ricevuti
14	ru_nvcsw	scambi di contesto volontari
15	ru_nivcsw	scambi di contesto involontari

Questa funzione solleverà un'eccezione ValueError se viene specificato un parametro *who* non valido. Può anche sollevare eccezioni error in circostanze insolite.

Modificato nella versione 2.3: Aggiunto l'accesso ai valori come attributo dell'oggetto restituito.

getpagesize()
Restituisce il numero di byte in una pagina di sistema. (Non è necessario che abbia la stessa dimensione della pagina hardware.) Questa funzione è utile per determinare il numero di byte di memoria che un processo sta utilizzando. Il terzo elemento della tupla restituita da getrusage() descrive l'uso della memoria in pagine; moltiplicando per la dimensione di una pagina, si ottiene il numero dei byte.

I seguenti simboli RUSAGE_* vengono passati alla funzione getrusage() per specificare quale informazione sui processi bisognerebbbe fornire.

RUSAGE_SELF
RUSAGE_SELF dovrebbe venire utilizzata per richiedere informazioni pertinenti unicamente al processo stesso.

RUSAGE_CHILDREN

Passata a `getrusage()` per richiedere informazioni di risorsa riguardanti i processi figli del processo chiamante.

`RUSAGE_BOTH`
Passata a `getrusage()` per richiedere informazioni sul consumo delle risorse sia del processo corrente che dei suoi processi figli. Potrebbe non essere disponibile su tutti sistemi.

8.16 `nis` — Interfaccia a NIS di Sun (Yellow Pages)

Il modulo `nis` fornisce un piccolo wrapper della libreria NIS, utile per l'amministrazione centralizzata di diversi host.

Dato che NIS esiste solo su sistemi UNIX, questo modulo è disponibile solamente per UNIX.

Il modulo `nis` definisce le seguenti funzioni:

`match`(*key, mapname*)
Restituisce la corrispondenza per la chiave *key* nella mappa *mapname*, oppure riporta un errore (`nis.error`) se non ve n'è nessuna. Entrambe dovrebbero essere stringhe, *key* è di 8-bit(senza caratteri speciali). Il valore restituito è un array arbitrario di byte (può contenere `NULL` o altro).

Notate che prima viene verificato che *mapname* non sia un alias ad un altro nome.

`cat`(*mapname*)
Restituisce un dizionario che mappa chiavi *key* in valori *value* in modo tale che `match(`*key*`,` *mapname*`)==`*value*. Notate che sia le chiavi che i valori del dizionario sono array di byte arbitrari.

Notate che prima viene verificato che *mapname* non sia un alias ad un altro nome.

`maps`()
Restituisce una lista di tutte le mappe valide.

Il modulo `nis` definisce la seguente eccezione:

exception `error`
Un errore sollevato quando una funzione NIS restituisce un codice d'errore.

8.17 `syslog` — Procedure della libreria syslog di Unix

Questo modulo fornisce un'interfaccia per le procedure della libreria syslog di UNIX. Fate riferimento alle pagine del manuale di UNIX per una descrizione dettagliata dell'utility `syslog`.

Il modulo definisce le seguenti funzioni:

`syslog`([*priority,*] *message*)
Spedisce la stringa *message* al logger di sistema. Se è necessario viene aggiunto un fine riga. Ogni messaggio viene marcato con una priorità composta da una *facility* e da un *level*. L'argomento facoltativo *priority*, il cui valore predefinito è `LOG_INFO`, determina la priorità del messaggio. Se *facility* non viene codificatà in (*priority*) utilizzando l'OR logico (`LOG_INFO | LOG_USER`), verrà utilizzato al suo posto il valore passato alla chiamata `openlog()`.

`openlog`(*ident*[, *logopt*[, *facility*]])
Le opzioni di logging diverse da quelle predefinite possono venire impostate aprendo esplicitamente il file di log tramite `openlog()`, prima della chiamata a `syslog()`. I valori predefiniti sono (solitamente) *ident* = `'syslog'`, *logopt* = `0`, *facility* = `LOG_USER`. L'argomento *ident* è una stringa che viene preposta a tutti i messaggi. L'argomento facoltativo *logopt* è un campo di bit - vedete sotto i possibili valori da combinare. L'argomento facoltativo *facility* dichiara il valore di facility predefinito per i messaggi che non possiedono una facility codificata esplicitamente.

`closelog`()
Chiude il file di log.

`setlogmask`(*maskpri*)

Imposta la maschera di priorità dei permessi a *maskpri* e restituisce il valore della maschera precedente. Le chiamate a `syslog()` con un livello di priorità non impostato in *maskpri* vengono ignorate. La modalità predefinita consiste nel loggare tutte le priorità. La funzione `LOG_MASK(`*pri*`)` calcola la maschera per la priorità individuale *pri*. La funzione `LOG_UPTO(`*pri*`)` calcola la maschera per tutte le priorità fino a *pri* inclusa quest'ultima.

Il modulo definisce le seguenti costanti:

Livelli di priorità (dall'alto al basso): `LOG_EMERG`, `LOG_ALERT`, `LOG_CRIT`, `LOG_ERR`, `LOG_WARNING`, `LOG_NOTICE`, `LOG_INFO`, `LOG_DEBUG`.

Facilities: `LOG_KERN`, `LOG_USER`, `LOG_MAIL`, `LOG_DAEMON`, `LOG_AUTH`, `LOG_LPR`, `LOG_NEWS`, `LOG_UUCP`, `LOG_CRON` e `LOG_LOCAL0` fino a `LOG_LOCAL7`.

Opzioni dei log: `LOG_PID`, `LOG_CONS`, `LOG_NDELAY`, `LOG_NOWAIT` e `LOG_PERROR` se definito in `<syslog.h>`.

8.18 `commands` — Utilità per eseguire comandi

Il modulo `commands` contiene funzioni che incapsulano `os.popen()` la quale accetta un comando di sistema sotto forma di stringa, restituisce ogni output generato dal comando e, facoltativamente, lo stato d'uscita.

Il modulo `commands` definisce le seguenti funzioni:

`getstatusoutput``(`*cmd*`)`
Esegue in una shell la stringa *cmd* tramite `os.popen()` e restituisce una tupla di due elementi `(`*status*`,` *output*`)`. *cmd* viene di fatto eseguito come `{`*cmd* `; }2>&1`, in modo che l'output restituito contenga l'output o i messaggi d'errore. Dall'output viene rimosso un carattere di fine riga. Lo stato d'uscita del il comando può venire interpretato in accordo con le regole della funzione C `wait()`.

`getoutput``(`*cmd*`)`
Identica a `getstatusoutput()`, eccetto per il fatto che lo stato d'uscita viene ignorato ed il valore restituito è una stringa contenente solo l'output del comando.

`getstatus``(`*file*`)`
Restituisce l'output di '`ls -ld` *file*' sotto forma di stringa. Questa funzione utilizza la funzione `getoutput()`, ed effettua in modo appropriato l'aggiunta del carattere di protezione davanti ai caratteri slash e dollaro nell'argomento.

Esempio:

```
>>> import commands
>>> commands.getstatusoutput('ls /bin/ls')
(0, '/bin/ls')
>>> commands.getstatusoutput('cat /bin/junk')
(256, 'cat: /bin/junk: No such file or directory')
>>> commands.getstatusoutput('/bin/junk')
(256, 'sh: /bin/junk: not found')
>>> commands.getoutput('ls /bin/ls')
'/bin/ls'
>>> commands.getstatus('/bin/ls')
'-rwxr-xr-x  1 root        13352 Oct 14  1994 /bin/ls'
```

CAPITOLO

NOVE

Il debugger di Python

Il modulo `pdb` definisce un debugger interattivo per il codice sorgente dei programmi Python. Supporta l'impostazione (condizionata) dei breakpoint e l'avanzamento riga per riga a livello di codice sorgente, l'ispezione degli stack frame, il listato del codice sorgente e la valutazione di codice Python arbitrario nel contesto di qualunque stack frame. Supporta inoltre il debug post mortem e può essere chiamato sotto il controllo del programma.

Il debugger è estensibile – viene di fatto definito tramite la classe `Pdb`. Questo modulo attualmente non è documentato ma può venire facilmente compreso attraverso la lettura dei sorgenti. L'interfaccia di estensione utilizza i moduli `bdb` (non documentato) e `cmd`.

Il prompt del debugger è '`(Pdb) `'. L'utilizzo tipico di questo modulo per eseguire un programma sotto il controllo del debugger è:

```
>>> import pdb
>>> import mymodule
>>> pdb.run('mymodule.test()')
> <string>(0)?()
(Pdb) continue
> <string>(1)?()
(Pdb) continue
NameError: 'spam'
> <string>(1)?()
(Pdb)
```

'pdb.py' può anche venire invocato come script per effettuare il debug di altri script. Per esempio:

```
python /usr/local/lib/python1.5/pdb.py myscript.py
```

L'utilizzo tipico per ispezionare un programma andato in crash è:

```
>>> import pdb
>>> import mymodule
>>> mymodule.test()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "./mymodule.py", line 4, in test
    test2()
  File "./mymodule.py", line 3, in test2
    print spam
NameError: spam
>>> pdb.pm()
> ./mymodule.py(3)test2()
-> print spam
(Pdb)
```

Il modulo definisce le seguenti funzioni; ciascuna avvia il debugger in un modo leggermente diverso:

run(*statement*[, *globals*[, *locals*]])
: Esegue l'istruzione *statement* (passata come stringa) sotto il controllo del debugger. Il prompt del debugger compare prima che qualunque codice venga eseguito; potete impostare dei breakpoint e digitare '`continue`', oppure potete avanzare un passo alla volta tra le dichiarazioni utilizzando '`step`' o '`next`' (tutti questi comandi vengono spiegati più avanti). Gli argomenti facoltativi *globals* e *locals* specificano l'ambiente nel quale il codice viene eseguito; se non specificato diversamente, viene utilizzato il dizionario del modulo `__main__`. (Vedete la spiegazione dell'istruzione `exec` o della funzione built-in `eval()`.)

runeval(*expression*[, *globals*[, *locals*]])
: Valuta l'espressione *expression* (fornita come stringa) sotto il controllo del debugger. Quando `runeval()` termina, questa funzione restituisce il valore dell'espressione. Altrimenti questa funzione è simile a `run()`.

runcall(*function*[, *argument, ...*])
: Esegue la funzione *function* (una funzione o il metodo di un oggetto, non una stringa) con gli argomenti forniti. Quando `runcall()` termina, la funzione restituisce qualunque cosa venga restituito dalla funzione chiamata. Il prompt del debugger appare al momento dell'entrata nella funzione.

set_trace()
: Avvia il debugger nello stack frame chiamante. Questo è utile per inserire in modo hard-code un breakpoint ad una data posizione nel programma, anche se il codice non viene comunque sottoposto a debug (per esempio quando un'asserzione fallisce).

post_mortem(*traceback*)
: Avvia il debugging post mortem dell'oggetto *traceback* fornito.

pm()
: Avvia il debugging post mortem della traceback trovata in `sys.last_traceback`.

9.1 Comandi del debugger

Il debugger riconosce i seguenti comandi. La maggior parte dei comandi possono venire abbreviati con una o due lettere; per esempio '`h(elp)`' significa che sia '`h`' che '`help`' possono venire utilizzati per avviare l'help (ma non '`he`', '`hel`', '`H`', '`Help`' o '`HELP`'). Gli argomenti dei comandi devono essere separati da caratteri di spaziatura (spazi o tab). Nella sintassi dei comandi gli argomenti facoltativi vengono racchiusi tra parentesi quadre ('`[]`'); le parentesi quadre non devono essere digitate. Le varie alternative nella sintassi dei comandi vengono separate da una barra verticale ('`|`').

Inviando una riga vuota si otterrà la ripetizione dell'ultimo comando fornito. Eccezione: se l'ultimo comando era '`list`', vengono elencate le prossime 11 righe.

I comandi che il debugger non riconosce vengono considerati come dichiarazioni Python e vengono eseguiti nel contesto del programma in corso di debug. Le istruzioni Python possono anche venire prefissate da un punto esclamativo ('`!`'). Questo è una maniera molto efficace per analizzare il programma in corso di debug; è anche possibile modificare una variabile o chiamare una funzione. Quando sovviene un'eccezione in una di queste istruzioni, viene stampato il nome dell'eccezione ma lo stato del debugger non viene alterato.

Si possono inserire comandi multipli in una singola riga, separati da '`;;`'. (Un singolo '`;`' non viene utilizzato poiché rappresenta il separatore di comandi multipli in una riga da passare al parser Python.) Non viene applicata nessuna accortezza particolare per separare i comandi; l'input viene diviso alla prima coppia di '`;;`', anche se si trova nel mezzo di una stringa racchiusa tra virgolette.

Il debugger supporta gli alias. Gli alias possono avere parametri che permettono un certo livello di adattabilità al contesto in esame.

Se esiste un file '`.pdbrc`' nella home directory dell'utente o nella directory corrente, esso viene letto ed eseguito come se fosse stato digitato al prompt del debugger. Questo è particolarmente utile per gli alias. Se entrambi i file esistono, quello nella home directory viene letto per primo e gli alias lì definiti possono venire sovrascritti dal file locale.

h(elp) [***command***] Senza argomenti, stampa la lista dei comandi disponibili. Con un comando *command* come

argomento, stampa l'help del comando. '`help pdb`' mostra l'intero file di documentazione; se viene definita la variabile d'ambiente PAGER, il file viene passato in una pipe attraverso il comando specificato. Poichè l'argomento *command* deve essere un identificatore, bisogna digitare '`help exec`' per ottenere l'help del comando '`!`'.

w(here) Stampa la traccia dello stacke, con il frame più recente in fondo. Una freccia indica il frame corrente, che determina il contesto della maggior parte dei comandi.

d(own) Sposta il frame corrente in basso di un livello nella traccia dello stack(verso un frame più recente).

u(p) Sposta il frame corrente in alto di un livello nella traccia dello stacke(verso un frame più vecchio).

b(reak) [[***filename*:**]***lineno*** | ***function*[, *condition***]] Con un argomento *lineno*, imposta in quella riga del file corrente un break. Con un argomento *function*, imposta un break alla prima istruzione eseguibile in quella funzione. Il numero di riga può essere preceduto da un nome di file seguito da un due punti, in modo da specificare un breakpoint in un'altro file (probabilmente uno che non è ancora stato caricato). Il file viene cercato in `sys.path`. Notate che ad ogni breakpoint viene assegnato un numero a cui fanno riferimento tutti gli altri comandi dei breakpoint.

Se è presente un secondo argomento, esso è un'espressione che deve venire valutata come vera prima che il breakpoint venga rispettato.

Senza argomenti, vengono elencati tutti i breakpoint, includendo per ognuno di essi il numero di volte che è stato raggiunto, il contatore corrente dei passi da ignorare e la condizione associata, se presente.

tbreak [[***filename*:**]***lineno*** | ***function*[, *condition***]] Breakpoint temporaneo che viene rimosso automaticamente quando viene raggiunto la prima volta. Gli argomenti sono gli stessi di break.

cl(ear) [***bpnumber*** [***bpnumber ...***]] Con una lista di numeri di breakpoint separata da spazi, cancella tutti questi breakpoint. Senza argomenti, cancella tutti i breakpoint (ma prima chiede conferma).

disable [***bpnumber*** [***bpnumber ...***]] Disabilita i breakpoint forniti come lista di numeri di breakpoint separati da spazi. Disabilitare un breakpoint significa che esso non può più provocare il blocco dell'esecuzione del programma, ma al contrario della cancellazione del breakpoint, esso resta nella lista dei breakpoint e può venire riabilitato.

enable [***bpnumber*** [***bpnumber ...***]] Abilita i breakpoint specificati.

ignore *bpnumber* [***count***] Imposta il contatore dei passi da ignorare per il breakpoint fornito (come numero). Se *count* viene omesso, il contatore dei passi da ignorare viene impostato a 0. Un breakpoint diventa attivo quando il contatore dei passi da ignorare diventa 0. Quando diverso da zero, il contatore viene decrementato ogni volta che il breakpoint viene raggiunto, a condizione che esso non sia disabilitato e che ogni condizione associata sia stata valutata come vera.

condition *bpnumber* [***condition***] *condition* è un'espressione che deve essere valutata come vera prima che il breakpoint venga rispettato. Se *condition* è assente, tutte le condizioni esistenti vengono rimosse; cioè, il breakpoint viene reso incondizionato.

s(tep) Esegue la riga corrente, blocca l'esecuzione alla prima occasione possibile (in una funzione chiamata o sulla prossima riga della funzione corrente).

n(ext) Continua l'esecuzione finché la prossima riga della funzione corrente non viene raggiunta o la funzione termina. (La differenza tra '`next`' e '`step`' è che '`step`' si blocca dentro una funzione chiamata, mentre '`next`' esegue la funzione chiamata a (circa) piena velocità, fermandosi solo alla prossima riga nella funzione corrente.)

r(eturn) Continua l'esecuzione fino al termine della funzione corrente.

c(ont(inue)) Continua l'esecuzione, si blocca solo quando viene raggiunto un breakpoint.

j(ump) ***lineno*** Imposta la prossima riga che verrà eseguita. Disponibile solo nel frame più in basso. Questo permette di tornare indietro per rieseguire una porzione di codice o per saltare del codice che non volete eseguire.

Attenzione che non tutti i salti sono permessi; tanto per chiarire, non è possibile saltare nel mezzo di un ciclo `for` o fuori da una clausola `finally`.

l(ist) [*first*[, *last*]] Mostra il codice sorgente del file corrente. Senza argomenti, mostra le 11 righe attorno a quella corrente o continua l'elenco precedente. Con un argomento, elenca le 11 righe attorno a quella riga. Con 2 argomenti, mostra le righe nell'intervallo fornito; se il secondo argomento è minore del primo, viene interpretato come un incremento(n.d.T: 11 e 3 indicano le righe dalla 11 alla 14).

a(rgs) Stampa la lista degli argomenti della funzione corrente.

p *expression* Valuta l'espressione *expression* nel contesto corrente e ne stampa il valore. **Note:** Si può anche utilizzare '`print`', ma non è un comando del debugger; esso esegue l'istruzione `print` di Python.

pp *expression* Come il comando '`p`', ad eccezione del fatto che il valore dell'espressione viene stampato in forma elegante utilizzando il modulo `pprint`.

alias [***name*** [**command**]] Crea un alias chiamato *name* che esegue il comando *command*. Il comando *non* deve essere racchiuso tra virgolette. I parametri sostituibili possono venire indicati da '`%1`', '`%2`' e così via, mentre '`%*`' viene sostituito da tutti i parametri. Se non viene fornito nessun comando, viene mostrato l'alias corrente di *name*. Se non viene fornito nessun argomento, vengono elencati tutti gli alias.

Gli alias possono venire annidati e possono contenere qualsiasi cosa che sia possibile digitare al prompt di pdb. Notate che i comandi interni di pdb *possono* venire sovrascritti dagli alias. Un comando sovrascritto rimane perciò nascosto finché l'alias non viene rimosso. Il meccanismo degli alias viene applicato ricorsivamente alla prima parola della riga di comando; tutte le altre parole sulla stessa riga di comando vengono lasciate invariate.

Come esempio, ecco due utili alias (specialmente quando inseriti nel file '.pdbrc'):

```
#Visualizza le variabili d'istanza (utilizzo: "pi classInst")
alias pi for k in %1.__dict__.keys(): print "%1.",k,"=",%1.__dict__[k]
#Visualizza le variabili d'istanza in self
alias ps pi self
```

unalias *name* Cancella l'alias specificato.

[**!**]***statement*** Esegue l'istruzione (monoriga) *statement* nel contesto dello stack frame corrente. Il punto esclamativo può venire omesso, a meno che la prima parola dell'istruzione sia anche un comando del debugger. Per impostare una variabile globale potete precedere, sulla stessa riga, il comando d'assegnamento con un comando '`global`', per esempio:

```
(Pdb) global list_options; list_options = ['-l']
(Pdb)
```

q(uit) Esce dal debugger. Il programma in esecuzione viene interrotto.

9.2 Come funziona

Sono state fatte alcune modifiche all'interprete:

- `sys.settrace(`*func*`)` imposta la funzione di tracciamento globale
- Ci può essere anche una funzione di tracciamento locale (vedete oltre)

Le funzioni di tracciamento possiedono tre argomenti: *frame*, *event* e *arg*. *frame* è lo stack frame corrente. *event* è una stringa: `'call'`, `'line'`, `'return'`, `'exception'`, `'c_call'`, `'c_return'` o `'c_exception'`. *arg* dipende dal tipo di evento.

La funzione di tracciamento globale viene invocata (con *event* impostato a `'call'`) ogni volta che viene inserito un nuovo scope locale; dovrebbe restituire un riferimento alla funzione locale di tracciamento da usare con quello scope o `None` se lo scope non deve essere tracciato.

La funzione di tracciamento locale dovrebbe restituire un riferimento a se stessa (o ad un'altra funzione per ulteriori tracciamenti in quello scope) o `None` per interrompere il tracciamento nello scope.

Vengono accettati anche metodi di istanza (e sono molto utili!) come funzioni di tracciamento.

Gli eventi possiedono il seguente significato:

`'call'` Viene chiamata una funzione (o altri blocchi di codice inseriti). La funzione globale di tracciamento viene chiamata; *arg* vale `None`; il valore di restituito specifica la funzione di tracciamento locale.

`'line'` L'interprete sta per eseguire una nuova riga di codice (a volte esistono più righe di evento in una stessa riga). Viene chiamata la funzione di tracciamento locale ; *arg* vale `None`; il valore restituito specifica la nuova funzione di tracciamento locale.

`'return'` Una funzione (o un'altro blocco di codice) sta per terminare. Viene chiamata la funzione di tracciamento locale; *arg* è il valore che verrà restituito. Il valore restituito dalla funzione di tracciamento viene ignorato.

`'exception'` Un'eccezione è stata sollevata. Viene chiamata la funzione di tracciamento locale; *arg* è una tupla di tre elementi (*eccezione*, *valore*, *traceback*); il valore restituito specifica la nuova funzione di tracciamento locale.

`'c_call'` Sta per essere chiamata una funzione C. Questa può essere un'estensione o una funzione built-in. *arg* è il nome della funzione C.

`'c_return'` Una funzione Cha terminato. *arg* vale `None`.

`'c_exception'` La funzione Cha sollevato un'eccezione. *arg* vale `None`.

Notate che come un'eccezione si propaga nella catena delle chiamate, allo stesso modo un evento `'exception'` viene generato ad ogni livello.

Per ulteriori informazioni su codice e oggetti frame, fate riferimento al *Manuale di riferimento di Python*.

CAPITOLO

DIECI

Il profiler di Python

Written by James Roskind.[1]

Il profiler è stato scritto dopo sole 3 settimane di programmazione in Python. Di conseguenza, è probabile che sia del pessimo codice, anche se non posso assicurarvelo visto che sono un principiante :-). Ho lavorato sodo per rendere prestante il codice, in modo che sia ragionevole effettuare la profilatura. Ho cercato di non ripetere i frammenti di codice, ma sono certo alcune volte di aver realizzato delle parti in modo veramente maldestro. Vi prego di inviare suggerimenti per il miglioramento di questo modulo a: jar@netscape.com. Non posso assicurare *alcun* supporto. ...ma apprezzerei dei commenti.

10.1 Introduzione al profiler

Un *profiler* è un programma che descrive le prestazioni in fase di esecuzione di un programma, fornendo una gran numero di statistiche. Questo documento descrive le funzionalità del profiler offerte dai moduli `profile` e `pstats`. Questo profiler realizza un profilo deterministico di qualsiasi programma Python. Esso inoltre mette a disposizione tutta una serie di strumenti per la generazione di rapporti che permettono agli utenti di esaminare rapidamente i risultati di un'operazione di profilatura.

10.2 In che modo questo profiler è differente dal precedente?

(Questa sezione esiste solo per ragioni storiche; il profiler descritto qui è quello della versione 1.1 di Python.)

I cambiamenti più consistenti dal vecchio profiler riguardano principalmente il maggior numero di informazioni ed il minor consumo di CPU.

In particolare:

[1] Aggiornato e convertito in LaTeX da Guido van Rossum. I riferimenti al vecchio profiler vengono inseriti a sinistra del testo, nonostante possano non esistere.

Bugs removed: Lo stack frame locale non viene più molestato, il tempo di esecuzione viene ora legato alle funzioni corrette.

Aumento della precisione: Il tempo di esecuzione del profiler non dipende più dal codice dell'utente, esiste ora il supporto per la calibrazione in base alla piattaforma, e le letture dei file non vengono fatte *dal* profiler *durante* la profilatura (e legate al codice dell'utente!).

Aumento di velocità: L'eccessivo costo in termini di CPU è stato ridotto da più di un fattore 2 (a volte anche 5), tutto ciò che deve essere caricato è il piccolo modulo del profiler, e durante la profilatura il modulo per la generazione dei risultati (`pstats`) non è necessario.

Supporto alle funzioni ricorsive: I tempi cumulativi nelle funzioni ricorsive vengono calcolati correttamente; viene inoltre contato il numero di chiamate ricorsive effettuate.

Grande crescita nella generazione dei risultati UI: Si possono raggruppare differenti esecuzioni del profiler per formare un rapporto più esteso; le funzioni che importano le statistiche accettano una lista arbitraria di file; il criterio di ordinamento viene ora basato sulle parole chiave (invece che su 4 opzioni intere); i risultati mostrano quali funzioni sono state analizzate, come pure a quale file del profilo si è fatto riferimento; il formato dell'output è stato migliorato.

10.3 Manuale utente istantaneo

Questa sezione esiste per coloro che "non vogliono leggere il manuale". Offre una panoramica molto veloce, permettendo ad un utente di eseguire rapidamente il profilo di un'applicazione esistente.

Ad esempio, per analizzare un'applicazione con una funzione principale `foo()`, dovete aggiungere il codice seguente al vostro modulo:

```
import profile
profile.run('foo()')
```

Il frammento di codice precedente causerà l'esecuzione di `foo()` e la stampa a video di una serie di righe informative (il profilo). Questo approccio è più utile quando si lavora con l'interprete in modalità interattiva. Se si vuole salvare il risultato di un profilo in un file per un esame successivo, si può aggiungere il nome del file come secondo argomento della funzione `run()`:

```
import profile
profile.run('foo()', 'fooprof')
```

Il file 'profile.py' può anche venire utilizzato come script per analizzare un altro script. Per esempio:

```
python /usr/local/lib/python1.5/profile.py myscript.py
```

'profile.py' accetta due argomenti facoltativi da riga di comando:

```
profile.py [-o output_file] [-s sort_order]
```

-s ha effetto solamente sullo standard output (**-o** non viene fornito). Vedete la documentazione di `Stats` per conoscere i valori validi per l'ordinamento.

Quando volete rivedere il profilo, dovrete utilizzare i metodi disponibili nel modulo `pstats`. Di solito le statistiche vengono caricate così:

```
import pstats
p = pstats.Stats('fooprof')
```

La classe `Stats` (il codice precedente crea solo un'istanza di questa classe) possiede una grande varietà di metodi per la manipolazione e la stampa dei dati che sono stati appena caricati in `p`. Quando prima avete utilizzato `profile.run()`, la stampa ottenuta era il risultato della chiamata di tre metodi:

```
p.strip_dirs().sort_stats(-1).print_stats()
```

Il primo metodo ha rimosso i percorsi estranei da tutti i nomi dei moduli. Il secondo ha ordinato tutti le voci secondo la stringa standard modulo/riga/nome che è stata stampata (per compatibilità semantica con il vecchio profiler). Il terzo metodo ha stampato tutte le statistiche. Potete provare anche le seguenti funzioni di ordinamento:

```
p.sort_stats('name')
p.print_stats()
```

La prima chiamata ordinerà la lista delle voci secondo il nome della funzione e la seconda chiamata stamperà tutte le statistiche. Quelle che segueno sono alcune chiamate interessanti con le quali fare esperimenti:

```
p.sort_stats('cumulative').print_stats(10)
```

Questo ordina il profilo secondo il tempo cumulativo in una funzione e poi stampa solo i primi dieci risultati più significativi. Il codice precedente è l'ideale se si vuole conoscere l'algoritmo che sta spendendo più tempo.

Se si vuole sapere quali funzioni stiano effettuando parecchi cicli, consumando molto tempo, dovete fare così:

```
p.sort_stats('time').print_stats(10)
```

per ordinare secondo il tempo speso dentro ogni funzione e stampando le statistiche delle prime dieci funzioni.

Provate anche questo:

```
p.sort_stats('file').print_stats('__init__')
```

Questa funzione ordinerà le statistiche per nome di file, e poi stamperà le statistiche relative solo ai metodi di inizializzazione della classe (quelli che contengono `__init__` nel proprio nome). Un ultimo esempio:

```
p.sort_stats('time', 'cum').print_stats(.5, 'init')
```

Questa riga ordina le statistiche innanzitutto in base al tempo, poi in base al tempo cumulativo ed infine stampa alcune statistiche. In particolare, la lista viene prima ridotta del 50% (`.5`) della sua dimensione originaria, vengono poi mantenute solamente le righe che contengono `init` ed infine questa sotto-sotto-lista viene stampata.

Se volete sapere quali sono le funzioni che hanno chiamato le precedenti, potete ora fare (`p` è ancora ordinato secondo gli ultimi criteri):

```
p.print_callers(.5, 'init')
```

ed otterrete la lista delle chiamanti per ognuna delle funzioni elencate.

Se volete avere a disposizione altre funzionalità, dovrete leggere il manuale, oppure indovinare il funzionamento delle seguenti funzioni:

```
p.print_callees()
p.add('fooprof')
```

Invocato come script, il modulo `pstats` è un browser statistico per la lettura e l'analisi dei risultati di un profilo. Ha una semplice interfaccia a riga di comando (implementata usando `cmd`) ed un aiuto interattivo.

10.4 Cos'è il profilo deterministico?

Un *profilo deterministico* rappresenta una situazione in cui vengono monitorate tutte le *chiamate alle funzioni*, le *chiusure dalle funzioni* e le *eccezioni*, e vengono effettuati controlli precisi negli intervalli tra questi eventi (durante i quali viene eseguito il codice dell'utente). Al contrario, un *profilo statistico* (che non viene implementato in questo modulo), analizza casualmente il puntatore effettivo dell'istruzione e deduce dove venga speso il tempo. Quest'ultima tecnica di solito causa un minore sovraccarico (visto che non c'è bisogno di predisporre il codice), ma offre solo indicazioni relative riguardo al luogo in cui il tempo viene speso.

In Python, dato che durante l'esecuzione di un programma c'è un interprete attivo, non è necessaria la presenza di codice predisposto per eseguire un profilo deterministico. Python fornisce automaticamente un *hook*(richiamo facoltativo) per ogni evento. Inoltre, la natura interpretata di Python tende ad aggiungere un tale sovraccarico all'esecuzione, che in una normale applicazione il profilo deterministico causa solo un piccolo rallentamento. Come risultato il profilo deterministico non è poi così costoso ed inoltre offre molte statistiche run time riguardo l'esecuzione di un programma Python.

Le statistiche sul conteggio delle chiamate può venire utilizzato per identificare difetti nel codice (conteggi strani) ed per individuare particolari punti in cui ci avvengono un gran numero di chiamate (conteggi molto elevati). Le statistiche sul tempo interno di esecuzione sono utili per identificare "cicli caldi"(in cui si spende molto tempo) che dovrebbero venire ottimizzati. Le statistiche sul tempo cumulativo sono utili invece nel localizzare errori di alto livello nella selezione degli algoritmi. Notate che la particolare gestione del tempo cumulativo di questo profiler permette di confrontare direttamente gli algortmi implementati ricorsivamente con quelli implementati iterativamente.

10.5 Manuale di riferimento

Il punto di entrata iniziale del profiler è la funzione globale `profile.run()`. Essa viene tipicamente utilizzata per creare tutte le informazione di profilo. I risultati vengono formattati e stampati usando i metodi della classe `pstats.Stats`. Ciò che segue è una descrizione di tutte queste funzioni e punti d'ingresso standard. Per una visione più approfondita di alcune porzioni di codice, prendete in considerazione la lettura della sezione successiva sulle Estensioni al Profiler, che spiega come derivare profiler "migliori" dalla classe qui presentata, oppure leggendo il codice sorgente di questi moduli.

`run`(*command*[, *filename*])

Questa funzione richiede un singolo argomento che possa venire passato all'istruzione `exec` ed un nome di file facoltativo. In tutti i casi questa procedura tenta di eseguire tramite `exec` il suo primo argomento, costruendo un profilo dalla sua esecuzione. Se non è presente un nome di file questa funzione stampa automaticamente a video un semplice profilo, ordinando ogni riga secondo la stringa standard (file/riga/nome-funzione). Ecco l'output tipico di questa chiamata:

```
        main()
        2706 function calls (2004 primitive calls) in 4.504 CPU seconds

  Ordered by: standard name

  ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       2    0.006    0.003    0.953    0.477 pobject.py:75(save_objects)
    43/3    0.533    0.012    0.749    0.250 pobject.py:99(evaluate)
     ...
```

La prima riga indica che questo profilo è stato generato dalla chiamata:
`profile.run('main()')` e quindi la stringa passata ad `exec` è stata `'main()'`. La seconda riga indica che sono state monitorate 2706 chiamate di funzione. Di queste, 2004 erano chiamate *primitive*. Per *primitiva* si intende una chiamata che non è stata indotta tramite ricorsione. La riga successiva: `Ordered by: standard name`, indica che l'ordinamento delle righe è avvenuto in base alla stringa presente nella colonna più a destra. Le intestazioni delle colonne includono:

ncalls il numero di chiamate,

tottime il tempo totale speso in questa funzione (escludendo il tempo trascorso per chiamare altre sotto funzioni),

percall il rapporto tra `tottime` e `ncalls`

cumtime il tempo totale speso dalla funzione corrente e da tutte le sue sotto funzioni(dall'invocazione all'uscita). Il risultato comprende *anche* le funzioni ricorsive.

percall il rapporto tra `cumtime` ed il numero di chiamate primitive

filename:lineno(function) fornisce informazioni su ogni funzione

Quando ci sono due numeri nella prima colonna (per esempio, '`43/3`'), allora il secondo è il numero di chiamate primitive, mentre il primo è il numero di chiamate reali. Notate che quando una funzione non è ricorsiva, questi due numeri sono identici, e quindi ne viene stampato solo uno.

`runctx`(*command, globals, locals*[, *filename*])
Questa funzione è simile a `profile.run()`, con argomenti aggiuntivi per supplire ai dizionari locali e globali per la stringa *command*.

L'analisi dei dati prodotti dal profiler si esegue usando la classe definita nel modulo `pstats`:

class `Stats`(*filename*[, ...])
Il costruttore di questa classe crea un'istanza di un "oggetto delle statistiche" a partire da uno o più nomi di file *filename*. Gli oggetti di tipo `Stats` posseggono dei metodi per manipolare e stampare statistiche utili.

Il file passato al costruttore deve essere stato creato dalla corrispondente versione di `profile`. In particolare, *non* viene garantita la compatibilità del formato del file con future versioni del profiler, e non c'è alcuna compatibilità con i file generati da altri profiler (come il vecchio profiler di sistema).

Se vengono passati al costruttore vari file, tutte le statistiche di una funzione vengono riunite in una sola, cosicché si possa avere in un unico rapporto la visione generale di vari processi. Se dopo la creazione dell'oggetto `Stats` si ha la necessità di combinare i suoi dati con altri file aggiuntivi, si può sempre utilizzare il metodo `add()`.

10.5.1 La classe `Stats`

Gli oggetti di tipo `Stats` possiedono i seguenti metodi:

`strip_dirs`()
Questo metodo per la classe `Stats` rimuove tutte le informazione relative alla directory nel percorso dei nomi di file. È molto utile per ridurre la larghezza dello stampato affinché si adatti al limite delle 80 colonne. Questo metodo modifica l'oggetto, e l'informazione rimossa viene definitivamente persa. Dopo l'esecuzione di questo metodo, le voci dell'oggetto si trovano in ordine "casuale", come dopo l'inizializzane ed il caricamento dell'oggetto. Se l'esecuzione di `strip_dirs()` fa sì che due funzioni diventino indistinguibili (cioè si trovano sulla stessa riga dello stesso file ed hanno lo stesso nome), allora le statistiche per queste due voci vengono riunite in una sola.

`add`(*filename*[, ...])
Questo metodo della classe `Stats` aggiunge ulteriori informazioni di profilatura a quella corrente. I suoi argomenti devono riferirirsi ai file creati dalla corrispondente versione di `profile.run()`. Statistiche per funzioni con stesso nome (vedi: file, line, name) vengono automaticamente riunite in una sola statistica.

`dump_stats`(*filename*)
Salva i dati caricati nell'oggetto `Stats` in un file chimato *filename*. Il file viene creato se non esiste, o altrimenti sovrascritto. Questo è equivalente al metodo dello stesso nome nella classe `profile.Profile`. Nuovo nella versione 2.3.

`sort_stats`(*key*[, ...])
Questo metodo modifica l'oggetto `Stats` ordinandolo secondo i criteri specificati. L'argomento è tipicamente una stringa che identifica la base dell'ordinamento (esempio: `'time'` o `'name'`).

Quando viene passato più di un argomento, le chiavi addizionali vengono utilizzate come criteri secondari quando le precedenti chiavi hanno prodotto delle righe uguali. Ad esempio, `sort_stats('nome', 'file')` ordinerà i record secondo il nome della funzione ed ordinerà le funzioni con stesso nome secondo il nome del file a cui appartengono.

Per i nomi delle chiavi si possono usare delle abbreviazioni, purché non creino ambiguità. Ecco l'elenco delle chiavi definite:

Argomento	Significato
`'calls'`	numero di chiamate
`'cumulative'`	tempo cumulativo
`'file'`	nome di file
`'module'`	nome di file
`'pcalls'`	numero di chiamate primitive
`'line'`	numero di riga
`'name'`	nome della funzione
`'nfl'`	nome/file/riga
`'stdname'`	nome standard
`'time'`	tempo interno

Notate che tutti gli ordinamenti sulle statistiche avvengono in ordine decrescente (dal più costoso in termini di tempo al meno), mentre name, file e line sono in ordine ascendente (alfabeticamente). La sottile distinzione che c'è tra `'nfl'` e `'stdname'` è che quest'ultimo è un ordinamento del nome come stampato, e quindi i numeri delle righe vengono comparati in maniera bizzarra. Per esempio, le righe 3, 20, e 40 (se appartengono allo stesso file) verranno ordinate come 20, 3, 40. Invece, `'nfl'` effettua una comparazione numerica dei numeri di riga. Infatti `sort_stats('nfl')` è la stessa cosa di `sort_stats('name', 'file', 'line')`.

Per compatibilità con il vecchio profiler, viene consentito l'uso di argomenti numerici come `-1`, `0`, `1` e `2`. Vengono interpretati rispettivamente come `'stdname'`, `'calls'`, `'time'` e `'cumulative'`. Se usate questo formato per gli argomenti, potrete usare solo un tipo di ordinamento(la chiave numerica) ed ulteriori argomenti verranno silenziosamente ignorati.

`reverse_order``()`

Questo metodo per la classe `Stats` rovescia l'ordinamento della lista di base nell'oggetto. Questo metodo viene fornito principalmente per compatibilità con il vecchio profiler. La sua utilità è dubbia adesso che l'ordinamento ascendente o descendente viene selezionato correttamente in base alla chiave scelta.

`print_stats``(`[*restriction, ...*]`)`

Questo metodo della classe `Stats` stampa un rapporto come descritto nella definizione della funzione `profile.run()`.

L'ordinamento con cui vengono stampati i risultati è basato sull'ultima operazione `sort_stats()` effettuata sull'oggetto (ed alle eventuali successive chiamate ai metodi `add()` e `strip_dirs()`).

Gli argomenti forniti possono venire utilizzati per limitare la lista alle voci più significative. Inizialmente, la lista include tutte le funzioni che sono state analizzate. Ogni restrizione può essere un intero (per selezionare una quantità di righe), o un decimale compreso tra 0.0 e 1.0 inclusi (per selezionare una percentuale di righe), o un'espressione regolare (per eseguire corrispondenze sullo standard che viene stampato; come in Python 1.5b1, si usa la sintassi per le espressioni regolari in stile Perl definita nel modulo `re`). Se si specificano varie restrizioni, queste vengono applicate sequenzialmente. Per esempio:

```
print_stats(.1, 'foo:')
```

innanzitutto limiterà lo stampato al 10% della lista e poi stamperà solamente le funzioni che appartengono ai file '.*foo:'. Invece il comando:

```
print_stats('foo:', .1)
```

prima selezionerà solo le funzioni che appartengono ai file '.*foo:' e poi stamperà solo il 10% della lista così ottenuta.

`print_callers``(`[*restriction, ...*]`)`

Questo metodo della classe `Stats` stampa una lista di tutte le funzioni che hanno chiamato le funzioni presenti nel database profilato. L'ordinamento, come anche la definizione degli argomenti di restrizione, è identico a quello fornito da `print_stats()`. Per comodità, viene anche mostrato un numero tra parentesi dopo ogni chiamante per indicare quante volte è stata effettuata questa specifica chiamata. Un secondo numero, non tra parentesi, rappresenta il tempo cumulativo speso nella funzione più a destra.

print_callees([*restriction, ...*])
Questo metodo della classe `Stats`, stampa una lista di tutte le funzioni chiamate dalla funzione indicata. A parte questa visione inversa ("ha chiamato" al posto di "è stata chiamata da"), gli argomenti e l'ordinamento sono gli stessi del metodo `print_callers()`.

ignore()
Deprecato dalla versione 1.5.1. Non è necessario nelle moderne versioni di Python.[2]

10.6 Limitazioni

Questo profiler ha fondamentalmente due limitazioni. La prima è che esso conta sull'interprete Python per poter sbrigare eventi *call*, *return* ed *exception*. Il codice C non viene interpretato e quindi è "invisibile" al profiler. Tutto il tempo speso nel codice C (comprese le funzioni built-in dell'interprete), verrà considerato relativo alla funzione Python che ha invocato quel codice. Se poi il codice C chiama a sua volta altro codice Python, allora queste chiamate verranno profilate correttamente.

La seconda limitazione riguarda l'accuratezza dell'informazione sul tempo. Esiste un problema fondamentale dei profiler deterministici riguardo la loro accuratezza. La restrizione più ovvia riguarda "l'orologio" sottostante, che tipicamente ha una precisione di 0.001 secondi. Quindi non ci possono essere misure più accurate di 0.001 secondi. Se vengono prese sufficienti misurazioni "l'errore" tende a livellarsi. Sfortunatamente però, rimuovere questo primo errore ne induce un secondo.

Il secondo problema è dovuto all'attesa tra "l'istante" in cui avviene un evento e "l'istante" in cui la chiamata del profiler per ottenere il tempo di esecuzione *acquisisce* realmente lo stato del clock. Analogamente, esiste un certo intervallo di tempo tra l'uscita del gestore dell'evento del profiler ed il momento in cui il valore del clock è stato ottenuto (che è già trascorso), fino al momento in cui torna in esecuzione il codice utente. Alla fine le funzioni che vengono chiamate molte volte, o che chiamano molte funzioni, tendono in genere ad accumulare questo errore. L'errore generato in questa maniera di solito è minore della precisione del clock (inferiore al suo intervallo), ma *può* accumularsi e diventare molto significativo. Questo profiler offre un modo per calibrarsi in base ad una data piattaforma, cosicché questo errore possa probabilmente venire rimosso. Dopo la calibrazione del profiler, esso sarà molto più accurato(in un senso lato), ma a volte potrà produrre dei numeri negativi (quando il numero delle chiamate è eccezionalmente basso, ed i vantaggi della probabilità lavorano contro di voi :-)). *Non* ci si deve preoccupare di questi numeri negativi, dovrebbero apparire *solo* se si è calibrato il profiler, e questi risultati sono sicuramente migliori che senza calibrazione.

10.7 Calibrazione

Il profiler sottrae una costante dal tempo accumulato per la gestione di un ogni evento in modo da compensare il rallentamento dovuto alla chiamata della funzione timer, mostrando immediatamente i risultati. Di predefinito, la costante vale 0. La seguente procedura può venire usata per ottenere una costante migliore su una specifica piattaforma (vedete in proposito il capitolo precedente sulle limitazioni).

```
import profile
pr = profile.Profile()
for i in range(5):
    print pr.calibrate(10000)
```

Questo metodo esegue il numero di chiamate Python fornite come argomento, prima direttamente e poi attraverso il profiler, misurando il tempo di entrambi. Poi calcola il rallentamento nascosto dovuto all'esecuzione del profiler stesso, che restituisce come numero in virgola mobile. Per esempio, su un Pentium a 800 MHz con Windows 2000 ed usando come timer la funzione Python `time.clock()`, il numero magico è circa 12.5e-6.

Lo scopo di questo esercizio è quello di ottenere un risultato giusto e consistente. Se il vostro computer è *molto*

[2] Questo metodo una volta era necessario, in tempi in cui Python avrebbe stampato qualsiasi risultato di una espressione inutilizzata che non fosse `None`. Il metodo viene ancora definito per una questione di retrocompatibilità.

veloce, o la funzione di timer ha un scarsa risoluzione, potreste avere la necessità di superare i 100000, o anche 1000000, per ottenere risultati consistenti.

Quando si ottiene una risposta adeguata, avete tre modi per usarla:[3]

```
import profile

# 1. Applicare la costante ottenuta a tutte le istanze del profiler
#    create d'ora in poi.
profile.Profile.bias = your_computed_bias

# 2. Applicare la costante ottenuta ad una specifica istanza del profiler.
pr = profile.Profile()
pr.bias = your_computed_bias

# 3. Specificare la costante ottenuta nel costruttore dell'istanza.
pr = profile.Profile(bias=your_computed_bias)
```

Se ne avete la possibilità, potreste voler scegliere una costante ancora più piccola, cosicché i vostri risultati saranno "meno spesso" affetti dalla presenza di numeri negativi nelle statistiche del profiler.

10.8 Estensioni — Derivare profiler migliori

La classe `Profile` del modulo `profile` è stata scritta con l'obiettivo di permettere l'estensione del profiler in classi derivate. I dettagli non vengono descritti qui, in quanto offrire una spiegazione proficua richiede una conoscenza esperta della struttura interna della classe `Profile`. Studiare il codice sorgente del modulo `profile` è un ottimo sistema per capire come funziona.

Se l'unico obiettivo è quello di cambiare il modo in cui viene calcolato il tempo corrente (per esempio, forzare l'utilizzo del tempo di clock del bios o il tempo speso dal processo), passate la funzione timer scelta al costruttore della classe `Profile`:

```
pr = profile.Profile(your_time_func)
```

Il profiler così ottenuto invocherà quindi `your_time_func()`. La funzione dovrebbe restituire un solo numero, o una lista di numeri la cui somma è il tempo corrente (in modo simile ad `os.times()`. Se la funzione restituisce un solo numero, o la lista di numeri restituiti ha lunghezza 2, allora otterrete una versione particolarmente veloce della procedura d'invio.

Prestate attenzione al fatto che sarebbe meglio calibrare il profiler in base alla funzione timer scelta. Su molti computer, un timer che restituisce un solo valore intero offrirà risultati migliori in termini di bassi rallentamenti durante la profilatura. (`os.times()` è *piuttosto* scarsa, dato che restituisce una tupla di valori in virgola mobile). Se volete sostituire un timer migliore nella maniera più pulita, derivate una classe e reimplementate il metodo che mostra i risultati finali in modo che gestisca meglio il timer che avete scelto, insieme all'appropriata costante di calibratura.

10.9 `hotshot` — Un profiler ad alte prestazioni per i log

Nuovo nella versione 2.2.

Questo modulo offre un'interfaccia più semplice per il modulo C `_hotshot`. Hotshot è un rimpiazzo del modulo `profile` esistente. Dato che è scritto prevalentemente in C, dovrebbe influenzare molto meno le performance generali rispetto al modulo `profile`.

[3] Prima di Python 2.2, era necessario modificare il codice sorgente del profiler per adattare la costante da utilizzare per la calibrazione. Potete sempre farlo, ma questo metodo non è descritto accuratamente perché non è più necessario.

Il profiler `hotshot` non funziona ancora bene con i thread. Sarebbe meglio utilizzare uno script non inserito in un thread per lanciare il profiler sul codice che volete misurare, sempre se sia possibile.

class **`Profile`**(*logfile*[, *lineevents*[, *linetimings*]])
L'oggetto profiler. L'argomento *logfile* è il nome di un file log da usare per registrare i dati del profilo. L'argomento *lineevents* indica se generare eventi per ogni riga di codice sorgente, o solo sulle chiamate/restituzioni delle funzioni. Il valore predefinito è `0` (registra solo le chiamate/restituzioni delle funzioni). L'argomento *linetimings* specifica se registrare anche le informazioni sul tempo. Il valore predefinito è `1` (salvare le informazioni sul tempo).

10.9.1 Oggetti Profile

Gli oggetti Profile possiedono i seguenti metodi:

`addinfo`(*key, value*)
Aggiunge un valore etichettato arbitrario nel risultato del profilo.

`close`()
Chiude il file di log e termina il profiler.

`fileno`()
Restituisce il descrittore del file di log.

`run`(*cmd*)
Analizza una stringa compatibile con `exec` nell'ambiente dello script. Tutto ciò che è globale nel modulo `__main__` viene usato sia come globale che come locale nello script.

`runcall`(*func, *args, **keywords*)
Analizza una singola chiamata di un oggetto chiamabile. Si possono passare alla funzione argomenti aggiuntivi posizionali ed a parola chiave; Viene restituito il risultato della chiamata, ed è permesso il propagarsi delle eccezioni, assicurando però che non ricadano alla fine sul profiler.

`runctx`(*cmd, globals, locals*)
Valuta una stringa compatibile con `exec` in un ambiente specifico. La stringa viene compilata prima di essere analizzata.

`start`()
Avvia il profiler.

`stop`()
Ferma il profiler.

10.9.2 Utilizzare i dati di hotshot

Nuovo nella versione 2.2.

Questo modulo carica i dati di profilo creati da hotshot negli oggetti Stats del modulo `pstats`.

`load`(*filename*)
Carica i dati di hotshot dal file *filename*. Restituisce un'istanza della classe `pstats.Stats`.

Vedete anche:

Modulo `profile` (sezione 10.5):
La classe `Stats` del modulo `profile`

10.9.3 Esempi di utilizzo

Notate che in questo esempio viene eseguito il "benchmark" pystones di python. Può richiedere molto tempo e produrre file molto grandi.

```
>>> import hotshot, hotshot.stats, test.pystone
>>> prof = hotshot.Profile("stones.prof")
>>> benchtime, stones = prof.runcall(test.pystone.pystones)
>>> prof.close()
>>> stats = hotshot.stats.load("stones.prof")
>>> stats.strip_dirs()
>>> stats.sort_stats('time', 'calls')
>>> stats.print_stats(20)
         850004 function calls in 10.090 CPU seconds

   Ordered by: internal time, call count

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    3.295    3.295   10.090   10.090 pystone.py:79(Proc0)
   150000    1.315    0.000    1.315    0.000 pystone.py:203(Proc7)
    50000    1.313    0.000    1.463    0.000 pystone.py:229(Func2)
  .
  .
  .
```

10.10 `timeit` — Misurare il tempo di esecuzione di piccole porzioni di codice

Nuovo nella versione 2.3.

Questo modulo offre un modo semplice per calcolare il tempo impiegato nell'esecuzione di piccole parti di codice Python. Può essere utilizzato da riga di comando o tramite interfacce richiamabili. Aggira un certo numero di trappole comuni nella misurazione del tempo di esecuzione. Vedete anche l'introduzione di Tim Peters al capitolo "Algoritmi", nel *Python Cookbook*, pubblicato dalla O'Reilly.

Il modulo definisce la seguente classe pubblica:

class `Timer`([*stmt=`'pass'`* [, *setup=`'pass'` [, *timer=<timer function>*]]])
Classe per calcolare la velocità di esecuzione di piccole porzioni di codice.

Il costruttore richiede come primo argomento un'istruzione da misurare, un'ulteriore istruzione da usare per il setup, ed una funzione timer. Il valore predefinito per i primi due argomenti è `'pass'`; la funzione timer è dipendente dalla piattaforma (vedete la stringa di documentazione del modulo). Le istruzioni possono contenere dei caratteri di fine riga, anche se magari non contengono caratteri di continuazione riga.

Per misurare il tempo di esecuzione della prima istruzione, usate il metodo `timeit()`. Il metodo `repeat()` invece è utile per chiamare più volte il metodo `timeit()`, restituendo quindi una lista di risultati.

`print_exc`([*file=`None`*])
Permette di stampare una traceback del codice analizzato.

Utilizzo tipico:

```
t = Timer(...)       # fuori dal try/except
try:
    t.timeit(...)    # oppure t.repeat(...)
except:
    t.print_exc()
```

Il vantaggio di questa funzione è quello di poter stampare le informazioni sugli eventuali errori verificatisi. L'argomento facoltativo *file* indica dove redirigere la stampa della traceback; la redirezione predefinita è su `sys.stderr`.

`repeat`([*repeat=3* [, *number=`1000000`*]])
Chiama più volte la funzione `timeit()`.

Questa è una funzione di comodità che chiama `timeit()` ripetutamente, restituendo una lista di tutti i risultati. Il primo argomento specifica quante volte chiamare `timeit()`. Il secondo specifica l'argomento *number* da passare a `timeit()`.

Note: Si può essere tentati di calcolare e riportare la media e la deviazione standard dalla lista ottenuta. Tuttavia, fare questo non è molto utile. In un caso tipico, il valore più basso indica il limite inferiore della velocità con cui la macchina può eseguire il codice fornito; i valori più alti non sono causati da una qualche variabilità della velocità di Python nell'eseguire il codice, ma dall'interferenza di altri processi nella precisione della vostra misurazione. Così il `min()` del risultato è il solo valore a cui dovreste essere interessati. Alla fin fine comunque, è meglio dare un'occhiata all'intera lista ed applicare il buon senso piuttosto che le leggi della probabilità statistica.

`timeit`(`[`*number=1000000*`]`)

Calcola il tempo impiegato per eseguire *number* esecuzioni dell'istruzione principale. Il codice di setup viene eseguito una sola volta, il metodo poi restituisce come numero in virgola mobile il tempo impiegato per eseguire l'istruzione principale il numero dato di volte, misurato in secondi. L'argomento è il numero di volte in cui effettuare il ciclo, il cui valore predefinito è un milione. L'istruzione principale, quella di setup e la funzione timer da usare vengono passate al costruttore.

Note: Di predefinito, `timeit()` temporaneamente inibisce il funzionamento della garbage collection durante il calcolo del tempo. Il vantaggio di questo approccio consiste nella maggior possibilità di comparare calcoli del tempo indipendenti. Lo svantaggio è che la GC può avere un ruolo fondamentale nelle prestazioni di questa funzione. Se è questo il caso, la GC può venire riabilitata come prima istruzione nella stringa *setup*. Per esempio:

```
timeit.Timer('for i in xrange(10): oct(i)', 'gc.enable()').timeit()
```

10.10.1 Interfaccia a riga di comando

Quando si invoca questo modulo dalla riga di comando, usate la seguente forma:

```
python timeit.py [-n N] [-r N] [-s S] [-t] [-c] [-h] [statement ...]
```

dove le varie opzioni hanno i seguenti significati:

-n N/--number=N quante volte eseguire l'istruzione 'statement'

-r N/--repeat=N quante volte ripetere timeit() (predefinito 3)

-s S/--setup=S l'istruzione da eseguire all'inizio una sola volta (il predefinito è `'pass'`)

-t/--time usa `time.time()` (predefinito su tutte le piattaforme tranne Windows)

-c/--clock usa `time.clock()` (predefinito su Windows)

-v/--verbose stampa ulteriori risultati dell'analisi; ripetetela per maggiore precisione

-h/--help stampa un breve messaggio di aiuto e termina

Si può passare un'istruzione multiriga fornendo ogni riga come argomento separato; l'indentazione delle righe può venire ricreata racchiudendo un argomento fra virgolette ed utilizzando opportunamente gli spazi. Opzioni **-s** multiple vengono trattate in modo simile.

Se l'opzione **-n** non viene fornita, viene calcolato un numero ragionevole di cicli provando per successive potenze di dieci finché il tempo totale non è minore di 0.2 secondi.

La funzione timer predefinita dipende dalla piattaforma. Su Windows, `time.clock()` ha la precisione dei microsecondi ma `time.time()` solo di un sessantesimo di secondo; su UNIX, `time.clock()` ha la precisione di un centesimo di secondo, mentre `time.time()` è molto più precisa. Su altre piattaforme, la funzione timer predefinita misura il tempo del clock, non il tempo della CPU. Questo significa che altri processi in esecuzione sulla stessa macchina potrebbero interferire con il calcolo del tempo. La cosa migliore da fare quando è necessaria

una grande accuratezza è fare il calcolo più volte. Questo si può fare attraverso l'opzione **-r**; il valore predefinito 3 dovrebbe essere più che sufficiente nella maggior parte dei casi. Su UNIX, si può usare `time.clock()` per misurare il tempo della CPU.

Note: Esiste un certo rallentamento di base associato all'esecuzione dell'istruzione pass. Il codice presentato non cerca di nasconderlo, ma è bene che si sappia. Questo rallentamento di base può venire misurato invocando il programma senza argomenti.

Il rallentamento di base differisce a seconda della versione di Python! Inoltre, per poter comparare correttamente i risultati di vecchie versioni di Python con la 2.3, si dovrebbe utilizzare l'opzione **-O** per le vecchie versioni dell'interprete in modo da evitare il calcolo del tempo delle istruzioni `SET_LINENO`.

10.10.2 Esempi

Ecco due sessioni d'esempio (una con la riga di comando, l'altra utilizzando l'interfaccia del modulo) che confrontano il costo di `hasattr()` contro `try`/`except` per verificare la presenza o meno degli attributi in un oggetto.

```
% timeit.py 'try:' '  str.__nonzero__' 'except AttributeError:' '  pass'
100000 loops, best of 3: 15.7 usec per loop
% timeit.py 'if hasattr(str, "__nonzero__"): pass'
100000 loops, best of 3: 4.26 usec per loop
% timeit.py 'try:' '  int.__nonzero__' 'except AttributeError:' '  pass'
1000000 loops, best of 3: 1.43 usec per loop
% timeit.py 'if hasattr(int, "__nonzero__"): pass'
100000 loops, best of 3: 2.23 usec per loop
```

```
>>> import timeit
>>> s = """\
... try:
...     str.__nonzero__
... except AttributeError:
...     pass
... """
>>> t = timeit.Timer(stmt=s)
>>> print "%.2f usec/pass" % (1000000 * t.timeit(number=100000)/100000)
17.09 usec/pass
>>> s = """\
... if hasattr(str, '__nonzero__'): pass
... """
>>> t = timeit.Timer(stmt=s)
>>> print "%.2f usec/pass" % (1000000 * t.timeit(number=100000)/100000)
4.85 usec/pass
>>> s = """\
... try:
...     int.__nonzero__
... except AttributeError:
...     pass
... """
>>> t = timeit.Timer(stmt=s)
>>> print "%.2f usec/pass" % (1000000 * t.timeit(number=100000)/100000)
1.97 usec/pass
>>> s = """\
... if hasattr(int, '__nonzero__'): pass
... """
>>> t = timeit.Timer(stmt=s)
>>> print "%.2f usec/pass" % (1000000 * t.timeit(number=100000)/100000)
3.15 usec/pass
```

Per poter fornire al modulo `timeit` l'accesso alle funzioni che avete definito, potete passare un parametro `setup` che contiene un'istruzione import:

```
def test():
    "Stupido test di funzione"
    L = []
    for i in range(100):
        L.append(i)

if __name__=='__main__':
    from timeit import Timer
    t = Timer("test()", "from __main__ import test")
    print t.timeit()
```

CAPITOLO

UNDICI

Protocolli internet e loro supporto

I moduli descritti in questo capitolo implementano i protocolli internet ed il supporto alle relative tecnologie. Sono tutte implementate in Python. Molti di questi moduli richiedono la presenza di moduli per i `socket`, in funzione del sistema in uso, che sono attualmente supportati nella maggior parte delle piattaforme. Segue un estratto:

`webbrowser`	Un controller semplice da usare per browser web.
`cgi`	Supporto alle Common Gateway Interface, usando script lato server per interpretare le form.
`cgitb`	Gestore configurabile per le traceback degli script CGI.
`urllib`	Apertura di risorse arbitrarie di rete tramite URL (richiede i socket).
`urllib2`	Una libreria estensibile per l'apertura delle URL che usa svariati protocolli
`httplib`	Client dei protocolli HTTP ed HTTPS (sono richiesti i socket).
`ftplib`	Client per protocollo FTP (richiede i socket).
`gopherlib`	Client per il protocollo Gopher (richiede i socket).
`poplib`	Client per il protocollo POP3 (richiede i socket).
`imaplib`	Client per protocollo IMAP4 (richiede i socket).
`nntplib`	Client per il protocollo NNTP (richiede i socket).
`smtplib`	Client per il protocollo SMTP (richiede i socket).
`telnetlib`	Telnet client class.
`urlparse`	Analizza le URL nei suoi componenti.
`SocketServer`	Un'infrastruttura per i server di rete.
`BaseHTTPServer`	Server HTTP di base (classe base per `SimpleHTTPServer` e `CGIHTTPServer`).
`SimpleHTTPServer`	Questo modulo fornisce un gestore di richieste di base per server HTTP.
`CGIHTTPServer`	Questo modulo fornisce un gestore di richieste per server HTTP che possono eseguire script CGI.
`Cookie`	Supporta la gestione dello stato HTTP (cookies).
`xmlrpclib`	Accesso a client XML-RPC.
`SimpleXMLRPCServer`	Implementazione di base di un server XML-RPC.
`DocXMLRPCServer`	Implementazione di un server XML-RPC di autodocumentazione.
`asyncore`	Una classe di base per sviluppare, con socket asincroni, la gestione di servizi.
`asynchat`	Supporto per protocolli asincroni di comando/risposta.

11.1 `webbrowser` — Un semplice gestore per browser web

Il modulo `webbrowser` fornisce un'interfaccia di altissimo livello per consentire la visualizzazione di documenti web agli utenti. Gli oggetti controller sono facili da usare e sono indipendenti dalla piattaforma. In molte circostanze, si otterrà un risultato semplicemente chiamando la funzione `open()` da questo modulo.

In UNIX, si privilegiano i browser grafici in ambiente X11, ma verranno usati i browser testuali se i browser grafici non sono disponibili o se non è disponibile un display X11. Se vengono usati i browser testuali, il processo chiamante verrà bloccato finché l'utente non esce dal browser.

In UNIX, se esiste la variabile d'ambiente BROWSER, verrà interpretata per sovrascrivere l'elenco predefinito dei browser per quella piattaforma, come una lista, separata da '`:`', di browser da provare in ordine. Quando il valore di uno degli elementi della lista contiene la stringa `%s`, questa viene interpretata come una opzione da riga di comando per il browser e viene utilizzata come argomento URL sostituito dal `%s`; se la parte non contiene `%s`, è semplicemente interpretata come il nome del browser da lanciare.

Per le piattaforme non UNIX, o quando i browser X11 sono disponibili in UNIX, il processo di controllo non attende che l'utente finisca con il browser, ma consente al browser di mantenere la propria finestra sul display.

Viene definita la seguente eccezione:

exception `Error`
Eccezione sollevata quando avviene un errore nel controllo del browser.

Sono definite le seguenti funzioni:

`open`(*url*[, *new=0*][, *autoraise=1*])
Mostra *url* utilizzando il browser predefinito. Se *new* è vera, verrà aperto un nuovo browser, quando possibile. Se *autoraise* è vera, la finestra è portata in primo piano, quando possibile (notare che in molti window manager, questo avviene indipendentemente dalle impostazioni di questa variabile).

`open_new`(*url*)
Apre *url* in una nuova finestra del browser predefinito, se possibile, altrimenti, apre *url* nella stessa finestra del browser.

`get`([*name*])
Restituisce un oggetto controller per il browser tipo *name*. Se *name* è vuota, restituisce, per il browser predefinito, un controller appropriato per l'ambiente del chiamante.

`register`(*name, constructor*[, *instance*])
Registra il tipo *name* del browser. Quando un tipo di browser è registrato, la funzione `get()` può restituire un controller per quel tipo di browser. Se *instance* non viene fornita, o è `None`, il costruttore *constructor* verrà chiamato senza parametri per costruire una istanza quando richiesto. Se *instance* viene passata, il costruttore non verrà mai chiamato, e potrà essere `None`.

Il punto di ingresso è utile solo se si pianifica di impostare sia la variabile BROWSER o chiamare `get` con un argomento non vuoto corrispondente al nome del gestore dichiarato.

Un certo numero di tipi di browser sono predefiniti. Questa tavola fornisce i nomi dei tipi di browser che possono venire passati alla funzione `get()` e la corrispondente istanziazione per le classi controller, tutte definite in questo modulo.

Tipo nome	Nome classe	Note
`'mozilla'`	`Netscape('mozilla')`	
`'netscape'`	`Netscape('netscape')`	
`'mosaic'`	`GenericBrowser('mosaic %s &')`	
`'kfm'`	`Konqueror()`	(1)
`'grail'`	`Grail()`	
`'links'`	`GenericBrowser('links %s')`	
`'lynx'`	`GenericBrowser('lynx %s')`	
`'w3m'`	`GenericBrowser('w3m %s')`	
`'windows-default'`	`WindowsDefault`	(2)
`'internet-config'`	`InternetConfig`	(3)

Note:

(1) "Konqueror" è il file manager per l'ambiente Desktop KDE per UNIX, e ha senso solamente se KDE è in esecuzione. Alcuni modi per rilevare correttamente KDE sarebbero apprezzati dato che la variabile d'ambiente KDEDIR non è sufficiente. Notare anche che il nome "kfm" viene usato anche quando viene eseguito il comando **konqueror** in KDE 2 — l'implementazione stabilisce la migliore strategia per eseguire konqueror.

(2) Solo sulle piattaforme Windows; richiede i moduli di estensione comuni `win32api` e `win32con`.

(3) Solo su piattaforme MacOS; richiedono il modulo standard MacPython `ic`, descritto nel manuale dei *Moduli di libreria Macintosh*.

11.1.1 Oggetti controller dei browser

I controller dei Browser forniscono due metodi che parallelamente offrono due delle funzioni a livello di modulo:

open(*url*[, *new*])
Mostra *url* usando il browser indicato da questo controller. Se *new* è vera, viene aperta una nuova finestra del browser, se possibile.

open_new(*url*)
Apre *url* in una nuova finestra del browser gestito da questo controller, se possibile, altrimenti, apre *url* nella sola finestra del browse:if expand(r.

11.2 cgi — Supporto alle Common Gateway Interface

Modulo di supporto agli script Common Gateway Interface (CGI).

Questo modulo definisce un numero di utilità da usare con gli script CGI scritti in Python.

11.2.1 Introduzione

Uno script CGI viene invocato da un server HTTP, tipicamente per elaborare un input inserito da un utente attraverso elementi HTML `<FORM>` o `<ISINDEX>`.

Molto spesso, gli script CGI risiedono nella directory speciale del server chiamata 'cgi-bin'. Il server HTTP inserisce ogni sorta di informazioni circa la richiesta (come il nome dell'host cliente, l'URL richiesta, la stringa di ricerca, e parecchie altre cose) nell'ambiente di shell dello script, esegue lo script e restituisce il risultato dello script al cliente.

L'input dello script è connesso anche al cliente, e qualche volta le informazioni della form vengono lette attraverso questa via; altre volte i dati della form passano attraverso parte della URL, con le "query string". Il modulo è costruito per prestare attenzione ai differenti casi e fornisce una semplice interfaccia per lo scripting in Python. Fornisce anche un certo numero di utilità che aiutano nell'analisi ed il debugging degli script, e l'aggiunta più recente è il supporto per l'upload del file da una form (se il browser lo supporta — Grail 3.0 e Netscape 2.0 lo fanno).

Il risultato dello script CGI consiste di due sezioni, separate da una riga vuota. La prima sezione contiene un certo numero di header che informano il cliente circa il tipo di dati che seguono. Il codice Python per generare una sezione minimale di header è simile a questa:

```
print "Content-Type: text/html"     # segue HTML
print                               # riga vuota, fine degli header
```

La seconda sezione è tipicamente HTML, che consente al software del cliente di mostrare facilmente il testo formattato con gli header, immagini in linea, etc. Segue del codice Python che stampa un semplice pezzo di HTML:

```
print "<TITLE>Output di uno script CGI</TITLE>"
print "<H1>Questo è il mio primo script CGI</H1>"
print "Ciao, mondo!"
```

11.2.2 Usare il modulo cgi

Si inizia scrivendo '`import cgi`'. Non si deve usare '`from cgi import *`' dato che il modulo definisce tutti i tipi di nomi che userà o per compatibilità con il passato, che non si vorranno nello spazio dei nomi che si sta usando.

Quando si scrive un nuovo script, occorre considerare l'aggiunta della seguente riga:

```
import cgitb; cgitb.enable()
```

Questo attiva uno speciale gestore per le eccezioni che mostra nel browser resoconti dettagliati se dovesse insorgere un errore. Se non si desidera mostrare queste informazioni all'utente del proprio script, si può decidere di salvare il resoconto in un file, con una riga simile a questa:

```
import cgitb; cgitb.enable(display=0, logdir="/tmp")
```

È molto utile utilizzare questa funzionalità durante la stesura dello script. Il resoconto prodotto da `cgitb` fornisce informazioni che possono far risparmiare tanto tempo nella ricerca dei bug. Si può semplicemente rimuovere successivamente la riga `cgitb` quando lo script è stato adeguatamente testato e si è sicuri che funzioni correttamente.

Per ottenere le informazioni inserite attraverso la form, è preferibile utilizzare la classe `FieldStorage`. Le altre classi definite in questo modulo, sono presenti soprattutto per compatibilità all'indietro. Istanziarla esattamente una sola volta, senza argomenti. In questo modo verà letto il contenuto della form dallo standard input o dalle variabili ambiente (questo dipende dal valore di diverse variabili d'ambiente, in accordo con lo standard CGI). Dato che può sprecare dello standard input, deve essere istanziata solo una volta.

L'istanza `FieldStorage` può essere indicizzata come un dizionario Python, e supporta anche i metodi standard per i dizionari `has_key()` e `keys()`. È supportata anche la funzione built-in `len()`. I campi della form che contengono stringhe vuote vengono ignorate e non appaiono nel dizionario; per catturare questi campi, si deve fornire un valore *True* al parametro chiave facoltativo *keep_blank_values* quando si crea un'istanza di classe `FieldStorage`.

Per esempio, il seguente codice (che assume che l'header Content-Type: ed una riga vuota siano già state stampate), controlla che i campi `name` e `addr` siano entrambi impostati a stringhe non vuote:

```
form = cgi.FieldStorage()
if not (form.has_key("name") and form.has_key("addr")):
    print "<H1>Errore</H1>"
    print "Per favore, inserire un nome ed un indirizzo valido nei campi."
    return
print "<p>nome:", form["name"].value
print "<p>indirizzo:", form["addr"].value
... Altre azioni da eseguire qui ...
```

Qui i campi, accessibili attraverso '`form[`*chiave*`]`', sono a loro volta istanze di `FieldStorage` (o `MiniFieldStorage`, in base alla codifica della form). L'attributo `value` dell'istanza contiene il valore stringa del campo. Il metodo `getvalue()` restituisce direttamente il valore della stringa; accetta anche un secondo argomento facoltativo come predefinito da restituire se la chiave richiesta non è presente.

Se i dati immessi nella form contengono più di un campo con il medesimo nome, l'oggetto recuperato da '`form[`*chiave*`]`' non è una istanza di `FieldStorage` o `MiniFieldStorage` ma una lista di queste istanze. Analogamente, in questa situazione, '`form.getvalue(`*chiave*`)`' potrebbe restituire una lista di stringhe. Se ci si aspetta questa possibilità (quando la propria form HTML contiene campi multipli con lo stesso nome), utilizzare la funzione built-in `isinstance()` per determinare quando si ha a che fare con singole istanze o liste di istanze. Per esempio questo codice concatena un numero imprecisato di campi nomeutente, separati da una virgola:

```
value = form.getvalue("nomeutente", "")
if isinstance(value, list):
    # Campi multipli nomeutente specificati
    usernames = ",".join(value)
else:
    # Singolo o nessun campo nomeutente specificato
    usernames = value
```

Se il campo rappresenta un file caricato via upload, l'accesso al valore tramite l'attributo `value` o attraverso il metodo `getvalue()` legge in memoria l'intero file come una stringa. Questo può non essere ciò che si vuole. Si

può verificare se si tratta di un file caricato controllando o l'attributo `filename` o l'attributo `file`. Si possono quindi leggere i dati contenuti attraverso l'attributo `file`.

```
fileitem = form["userfile"]
if fileitem.file:
    # E' un file caricato via upload; si contano le righe
    linecount = 0
    while 1:
        line = fileitem.file.readline()
        if not line: break
        linecount = linecount + 1
```

Lo standard per l'upload dei file descrive la possibilità di inviare file multipli da un singolo campo (usando una codifica ricorsiva multipart/*). Quando questo avviene, l'elemento sarà simile ad un dizionario `FieldStorage`. Questo si può determinare controllando il suo attributo `type`, che dovrebbe essere multipart/form-data (o, al massimo, un altro tipo MIME corrispondente a multipart/*). In questo caso, può essere iterato ricorsivamente semplicemente come un oggetto di alto livello della form.

Quando una form viene immessa attraverso il "vecchio" formato (come una stringa di ricerca o come un singolo elemento di dati del tipo application/x-www-form-urlencoded), l'elemento sarà attualmente una istanza della classe `MiniFieldStorage`. In questo caso gli attributi `list`, `file` e `filename` sono sempre `None`.

11.2.3 Interfaccia di alto livello

Nuovo nella versione 2.2.

La precedente sezione spiega come leggere i dati di una form CGI usando la classe `FieldStorage`. Questa sezione descrive un'interfaccia di più alto livello che è stata aggiunta a questa classe per consentire di operare in un modo più leggibile ed intuitivo. L'interfaccia non fa diventare obsolete le tecniche della sezione precedente — per esempio, restano comunque utili per elaborare gli upload dei file in modo efficiente.

L'interfaccia consiste di due semplici metodi. Usando i metodi si possono elaborare i dati di una form in un modo generico, senza la necessità di preoccuparsi se uno o più valori sono stati inseriti con lo stesso nome.

Nella precedente sezione, si è appreso come scrivere il seguente codice ogni volta che ci si aspetta che un utente immetta più di un valore con lo stesso nome:

```
item = form.getvalue("item")
if isinstance(item, list):
    # L'utente sta richiedendo piu' di un elemento.
else:
    # L'utente ha richiesto un solo elemento.
```

Questa situazione è comune, per esempio, quando una form contiene un gruppo di checkbox multiple con lo stesso nome:

```
<input type="checkbox" name="item" value="1" />
<input type="checkbox" name="item" value="2" />
```

In molte situazioni, comunque, c'è un solo campo in una form con un nome particolare e quindi ci si aspetta e si ha necessità di un solo valore associato a questo nome. Così, si scriva uno script contenente per esempio questo codice:

```
user = form.getvalue("user").upper()
```

Il problema con il codice è che non ci si deve aspettare che un cliente fornisca un input valido ai propri script. Per esempio, se un utente curioso, aggiunge un altra coppia '`user=foo`' alla stringa di query, lo script dovrebbe bloccarsi, perché in questa situazione il metodo `getvalue(user)` restituisce una lista al posto di una stringa.

Chiamare il metodo `upper()` su una lista non viene accettato (dato che le liste non hanno un metodo con questo nome) ed il risultato è un'eccezione `AttributeError`.

Comunque, il modo appropriato per leggere il valore delle form di dati è usare sempre del codice che controlli che i valori ottenuti siano un valore singolo o una lista di valori. Questo è noioso e rende meno leggibile gli script.

Un approccio più pratico consiste nell'uso dei metodi `getfirst()` e `getlist()` forniti da questa interfaccia di più alto livello.

`getfirst`(*name*[, *default*])
: Questo metodo restituisce sempre un solo valore associato al nome del campo della form. Il metodo restituisce solo il primo valore nel caso che più valori siano inseriti con il medesimo nome. Notare che l'ordine in cui i valori vengono ricevuti possono variare da browser a browser e non ci si deve fare affidamento.[1] Se non esiste nessun campo della form o nessun valore, viene restituito il valore specificato attraverso il parametro facoltativo *default*. Questo parametro *default* è `None` se non è specificato.

`getlist`(*name*)
: Questo metodo restituisce sempre una lista di valori associati al campo della form *name*. Il metodo restituisce una lista vuota se non esiste nessun valore o nessun campo della form per *name*. Restituisce una lista consistente in un elemento se esiste solamente un simile valore.

Usando questi metodi si può scrivere del codice compatto.

```
import cgi
form = cgi.FieldStorage()
user = form.getfirst("user", "").upper()  # Questo metodo è più sicuro.
for item in form.getlist("item"):
    do_something(item)
```

11.2.4 Vecchie classi

Queste classi, presenti nelle vecchie versioni del modulo `cgi`, vengono ancora supportate per compatibilità con il passato. Le nuove applicazioni devono usare la classe `FieldStorage`.

`SvFormContentDict` memorizza il contenuto dei singoli valori della form come un dizionario; assume che ogni nome di campo appaia una sola volta nella form.

`FormContentDict` memorizza contenuti di valori multipli della form come un dizionario (gli elementi della form sono liste di valori). Utile se la form contiene campi multipli con lo stesso nome.

Altre classi (`FormContent`, `InterpFormContentDict`) sono presenti per compatibilità con il passato con applicazioni estremamente vecchie. Se si usano queste e saltano fuori degli inconvenienti quando queste spariranno da una prossima versione di questo modulo, mandare una nota agli sviluppatori.

11.2.5 Funzioni

Queste sono utili se si vuole maggior controllo, o se si vogliono impiegare alcuni degli algoritmi implementati in questo modulo in altre circostanze.

`parse`(*fp*[, *keep_blank_values*[, *strict_parsing*]])
: Analizza una query nell'ambiente o da un file (il file predefinito è `sys.stdin`). I parametri *keep_blank_values* e *strict_parsing* vengono passati a `parse_qs()` inalterati.

`parse_qs`(*qs*[, *keep_blank_values*[, *strict_parsing*]])
: Analizza una stringa di una query passata come un argomento stringa (dati di tipo application/x-www-form-urlencoded). I dati vengono restituiti come un dizionario. Le chiavi del dizionario sono i nomi univoci delle variabili della query e i valori sono liste di valori per ogni nome.

[1] Notare che alcune recenti versioni delle specifiche HTML stabiliscono in quale ordine debbano essere valutati i campi, ma è necessario sapere se la richiesta è stata ricevuta da un browser conforme a queste specifiche, o perfino da quale browser è partita, ed è in realtà noioso e porta facilmente a sbagliare.

L'argomento facoltativo *keep_blank_values* è un'opzione che indica quando valori vuoti nelle query codificate in URL devono essere trattati come stringhe vuote. Un valore *True* indica che i campi vuoti devono essere considerati stringhe vuote. Il valore predefinito *False* indica che un valore vuoto deve essere ignorato e trattato come se non fosse incluso.

L'argomento facoltativo *strict_parsing* è un'opzione che indica come trattare gli errori di analisi. Se *False* (il predefinito), gli errori sono silenziosamente ignorati. Se *True*, gli errori sollevano una eccezione `ValueError`.

Usare la funzione `urllib.urlencode()` per convertire i dizionari in stringhe di query.

`parse_qsl`(*qs*[, *keep_blank_values*[, *strict_parsing*]])
Analizza una stringa di query passata come un argomento stringa (dato di tipo application/x-www-form-urlencoded). I dati vengono restituiti come una lista di coppie nome, valore.

L'argomento facoltativo *keep_blank_values* è un'opzione che indica quando i valori vuoti nelle query codificate in URL devono essere trattate come stringhe vuote. Un valore *True* indica che i campi vuoti devono essere considerati come stringhe vuote. Il valore predefinito *False* indica che i valori vuoti devono essere ignorati e trattati come se non fossero inclusi.

L'argomento facoltativo *strict_parsing* è un'opzione che indica come comportarsi con gli errori di analisi. Se *False* (il predefinito), gli errori devono essere silenziosamente ignorati. Se *True*, gli errori sollevano eccezione `ValueError`.

Usare la funzione `urllib.urlencode()` per convertire le liste di coppie in stringhe di query.

`parse_multipart`(*fp*, *pdict*)
Analizza gli input del tipo multipart/form-data (per gli upload di file). Gli argomenti sono *fp* per il file in input e *pdict* per un dizionario contenente altri parametri negli header Content-Type:.

Restituisce un dizionario come le chiavi di `parse_qs()` sono nel nome del campo, ogni valore è una lista di valori per quel campo. Questa è facile da usare ma non molto efficiente se ci si aspettano megabyte in arrivo — in questo caso, al suo posto usare la classe `FieldStorage` che risulta molto più flessibile.

Notare che questa non analizza parti multiple annidate — al suo posto usare `FieldStorage`.

`parse_header`(*string*)
Analizza una intestazione MIME (come Content-Type:) in un valore principale e un dizionario di parametri.

`test`()
Un robusto script di test CGI, utilizzabile come programma principale. Scrive il minimo necessario di intestazioni HTTP e formatta tutte le informazioni fornite dallo script nella forma HTML.

`print_environ`()
Formatta l'ambiente della shell in HTML.

`print_form`(*form*)
Formatta una form in HTML.

`print_directory`()
Formatta la directory corrente in HTML.

`print_environ_usage`()
Stampa una lista di utili variabili d'ambiente in HTML (usate dal CGI).

`escape`(*s*[, *quote*])
Converte i caratteri '&', '<' e '>' nella stringa *s* in sequenze HTML sicure. La si utilizzi quando si ha bisogno di mostrare testo che deve contenere questi caratteri in HTML. Se l'opzione facoltativa *quote* è vera, anche il carattere di doppia virgoletta ('') viene tradotto; questo aiuta per le inclusioni in un valore di attributo HTML come `<A HREF=...>`. Se il valore da racchiudere tra virgolette deve includere caratteri di singole o doppie virgolette, o entrambe, considerare invece l'uso della funzione `quoteattr()` fornita nel modulo `xml.sax.saxutils`.

11.2.6 Preoccuparsi della sicurezza

Esiste una sola regola importante: se si invoca un programma esterno (attraverso le funzioni `os.system()` o `os.popen()`, o altre con funzionalità simili), occorre prestare estrema attenzione a non far passare stringhe

arbitrarie ricevute dal client alla shell. Questa è una mancanza di sicurezza ben conosciuta dove un cracker può forzare uno script CGI debole per eseguire comandi di shell arbitrari. Ogni parte della URL o del nome del campo non può essere considerata affidabile, se la richiesta non arriva dalla propria form!

Per avere più sicurezza, si deve passare la stringa ottenuta dalla form ad un comando di shell, cercando di assicurarsi che la stringa contenga esclusivamente caratteri alfanumerici, lineette, trattini bassi e punti.

11.2.7 Installare il proprio script CGI in un sistema UNIX

Leggere la documentazione del proprio server HTTP e verificare con il proprio amministratore locale di trovare la directory dove lo script CGI deve essere installato; tipicamente questa è una directory chiamata 'cgi-bin' nell'albero delle directory del server.

Verificare che il proprio script sia leggibile e eseguibile da "altri"; la modalità prevista dai file UNIX dovrebbe essere `0755` in ottale (usate '`chmod 0755` *nomefile*'). Accertarsi che la prima riga dello script contenga `#!` posizionato alla colonna 1 seguito dal percorso dell'interprete Python, solitamente:

```
#!/usr/local/bin/python
```

Verificare che l'interprete Python esista e che lo script sia eseguibile da "altri" (NdT: in pratica eseguibile da chiunque).

Verificare che ogni file di cui lo script necessiti scrivere o leggere sia leggibile o scrivibile, rispettivamente, da "altri"; il loro modo di accesso dovrebbe essere `0644` per essere leggibile e `0666` per essere scrivibile. Questo perché, per ragioni di sicurezza, il server HTTP esegue lo script come utente "nobody", senza alcun privilegio. Può soltanto leggere (scrivere, eseguire) file che ognuno può leggere (scrivere, eseguire). La directory corrente, al momento dell'esecuzione, è anche diversa (è tipicamente la directory cgi-bin del server) e l'insieme delle variabili d'ambiente è differente da quelle che si ottengono con l'accesso di log in. In particolare, non si pensi che il percorso di ricerca della shell per gli eseguibili (PATH) o quello di ricerca moduli Python (PYTHONPATH) sia impostato su qualcosa di interessante.

Se si ha bisogno di caricare moduli da una directory che non è nel percorso predefinito di ricerca moduli di Python, si può cambiare il percorso nello script, prima di importare altri moduli. Per esempio:

```
import sys
sys.path.insert(0, "/usr/home/joe/lib/python")
sys.path.insert(0, "/usr/local/lib/python")
```

(In questo modo, la directory inserita per ultima sarà la prima in cui si effettuerà la ricerca!)

Le istruzioni per i sistemi non UNIX sono varie; consultare la documentazione del server HTTP in uso (che solitamente ha una sezione sugli script CGI).

11.2.8 Verificare il proprio script CGI

Sfortunatamente, uno script CGI, generalmente, non viene testato da riga di comando, e uno script che funziona perfettamente da riga di comando può fallire misteriosamente quando lo si esegue dal server. C'è una sola ragione per testare uno script da riga di comando: se contiene un errore di sintassi, l'interprete Python non lo vorrà assolutamente eseguire, mentre il server HTTP invierà un errore criptico al client.

Assumendo che lo script non contenga errori di sintassi e ancora non funzioni, non si ha altra scelta che leggere la prossima sezione.

11.2.9 Il debugging sugli script CGI

Prima di tutto, verificare triviali errori di installazione; leggere attentamente la sezione precedente sull'installazione degli script CGI può far risparmiare molto tempo. Se ci si meraviglia del modo in cui si è capita la corretta procedura di installazione, si provi a installare una copia di questo modulo ('cgi.py') come script CGI. Quando richiamato come script, il file mostra il suo ambiente ed il contenuto della form in formato HTML. Gli si diano i giusti diritti, etc, e gli si invii una richiesta. Se è installato nella directory standard 'cgi-bin', è possibile inviargli una richiesta inserendo una URL nel browser come se fosse una form.

```
http://yourhostname/cgi-bin/cgi.py?name=Joe+Blow&addr=At+Home
```

Se questo restituisce un errore di tipo 404, il server non trova lo script, forse va installato in una directory differente. Se restituisce un altro errore, c'e' un problema di installazione e lo si deve correggere prima di tentare qualsiasi altra cosa. Se si riceve un elenco piacevolmente formattato dell'ambiente ed il contenuto della form (in questo esempio, il campo dovrebbe essere mostrato come "addr" con valore "At Home" e "name" con valore "Joe Blow"), lo script 'cgi.py' è stato installato correttamente. Se si segue la stessa procedura per il proprio script, si è in grado di farne il debug.

Il prossimo passo potrebbe essere la chiamata dal proprio script della funzione `test()` del modulo `cgi`: sostituire il codice principale con la singola definizione:

```
cgi.test()
```

Questo dovrebbe produrre lo stesso risultato come quando si è installato il file 'cgi.py' da solo.

Quando uno script Python generico solleva una eccezione non gestita (per diverse ragioni: un errore in un nome modulo, un file che non può essere aperto, etc.), l'interprete Python stampa una traceback elegante ed esce. Mentre l'interprete Python fa la stessa cosa quando lo script CGI solleva una eccezione, più facilmente una traceback finirà in uno dei file di log del server HTTP, oppure verrà abbandonato.

Fortunatamente, dopo aver lavorato intorno allo script per eseguire *del* codice, si può facilmente inviare una traceback al browser web usando il modulo `cgitb`. Se non si è già fatto, si aggiunga semplicemente questa riga

```
import cgitb; cgitb.enable()
```

all'inizio del proprio script. Quindi si tenti nuovamente l'esecuzione; quando insorge un problema, si dovrebbe vedere un rapporto dettagliato che facilmente renderà visibile la causa del crash.

Se si sospetta che ci potrebbe essere un problema nell'importazione del modulo `cgitb`, si può utilizzare un metodo più robusto (che usa solo moduli built-in):

```
import sys
sys.stderr = sys.stdout
print "Content-Type: text/plain"
print
..il proprio codice da qui...
```

Questo permette all'interprete Python di stampare la traceback. Il tipo di contenuto del risultato viene impostato a testo puro, che disabilita l'elaborazione di HTML. Se lo script funziona, l'HTML verrà mostrato dal client. Se viene sollevata un'eccezione, meglio se dopo che le prime due righe vengono stampate, verrà mostrata una traceback. Siccome non c'è interpretazione HTML in corso, la traceback sarà leggibile.

11.2.10 Problemi comuni e soluzioni

- Molti server HTTP bufferizzano l'output dallo script CGI fino a quando lo script non viene completato. Questo significa che non è possibile mostrare un report progressivo sul monitor del client mentre lo script è in esecuzione.
- Si controllino le istruzioni di installazione viste precedentemente.
- Si controllino i file di log del server HTTP. ('`tail -f logfile`' in una finestra separata può risultare molto utile!)
- Come prima cosa si verifichi sempre uno script per gli errori di sintassi, facendo qualcosa tipo '`python script.py`'.
- Se lo script non ha nessun errore di sintassi, si provi ad aggiungere '`import cgitb; cgitb.enable()`' all'inizio dello script.
- Quando si invocano programmi esterni, ci si accerti che possano essere trovati. Tipicamente, questo significa usare il percorso assoluto; nello script CGI, PATH solitamente non viene impostata ad un valore utile.
- Quando si leggono o si scrivono file, ci si deve accertare che questi possano essere scritti dall'id utente sotto il quale lo script CGI verrà eseguito: questo è tipicamente l'id utente col quale viene eseguito il server web, oppure l'id utente esplicitamente specificato dalla funzionalità '`suexec`' del server web stesso.
- Non tentare di assegnare allo script CGI un modo set-uid. Questo non lavora sulla maggior parte dei sistemi, ed è una mancanza di sicurezza certa.

11.3 `cgitb` — Gestore delle traceback per gli script CGI

Nuovo nella versione 2.2.

Il modulo `cgitb` fornisce un gestore di eccezioni speciale per gli script Python. (Il suo nome può causare confusione. Originariamente è stato progettato per mostrare informazioni estensive sulle traceback in HTML per gli script CGI. Successivamente è stato generalizzato per mostrare anche informazioni in puro testo.) Dopo che questo modulo è stato attivato, se interviene un'eccezione non gestita, verrà mostrato un report, dettagliato e formattato. Il report include una traceback contentente gli estratti del codice sorgente per ogni livello, quanto il valore degli argomenti e delle variabili locali delle funzioni correntemente in esecuzione, per aiutare alla risoluzione del problema. Facoltativamente si possono salvare queste informazioni in un file invece di inviarle al browser.

Per abilitare questa funzionalità, semplicemente si aggiunga una linea all'inizio dello script CGI:

```
import cgitb; cgitb.enable()
```

Le opzioni della funzione `enable()` controllano quando il report viene mostrato nel browser e quando il report viene registrato in un file per un'analisi successiva.

`enable`([*display*[, *logdir*[, *context*[, *format*]]]])

Questa funzione fa sì che il modulo `cgitb` sovrascriva il gestore predefinito dell'interprete per le eccezioni, impostando il valore di `sys.excepthook`.

L'argomento facoltativo *display* viene impostato al valore predefinito `1` e può essere impostato a `0` per sopprimere l'invio delle traceback al browser. Se l'argomento *logdir* è presente, il rapporto della traceback viene scritto in un file. Il valore di *logdir* deve essere una directory dove questi file verranno posizionati. L'argomento facoltativo *context* è il numero di righe di contesto da mostrare intorno alla riga corrente del codice sorgente nella traceback; il suo valore predefinito è `5`. Se l'argomento facoltativo *format* è `html`, l'output viene formattato come HTML. Ogni altro valore forza il risultato in puro testo. Il valore predefinito è `html`.

handler([*info*])
Questa funzione gestisce un'eccezione usando le impostazioni predefinite (cioè, mostrare il report in un browser, ma non registrarle in un file). Ciò può essere usato quando è stata catturata un'eccezione e la si vuole nel report usando `cgitb`. L'argomento facoltativo *info* deve essere una tripla tupla contenente il tipo dell'eccezione, il valore dell'eccezione, e l'oggetto traceback, esattamente come la tupla restituita da `sys.exc_info()`. Se l'argomento *info* non viene fornito, l'eccezione corrente è ottenuta da `sys.exc_info()`.

11.4 `urllib` — Apertura di risorse arbitrarie tramite URL

Questo modulo fornisce un'interfaccia di alto livello per estrarre dati attraverso il World Wide Web. In particolare, la funzione `urlopen()` è simile alla funzione built-in `open()`, ma accetta Universal Resurce Locators (URL) invece dei nomi di file. Vengono applicate alcune restrizioni: può aprire solo URL in lettura, e non sono disponibili funzioni di seek.

Definisce le seguenti funzioni pubbliche:

urlopen(*url*[, *data*[, *proxies*]])
Apre un oggetto di rete denotato da una URL in lettura. Se l'URL non possiede uno schema identificativo, o se ha un 'file:' come per lo schema identificativo, apre un file locale (senza i fine riga universali); altrimenti apre un socket verso un server da qualche parte nella rete. Se la connessione non può essere instaurata, o se il server restituisce un codice di errore, viene sollevata l'eccezione `IOError`. Se tutto va bene, viene restituito un oggetto simile a file. Questo supporterà i seguenti metodi: `read()`, `readline()`, `readlines()`, `fileno()`, `close()`, `info()` e `geturl()`. Inoltre possiede un proprio supporto per il protocollo di iterazione. Un ammonimento: il metodo `read()`, se la dimensione dell'argomento viene omessa o è negativa, può non riuscire a leggere fino alla fine del flusso di dati; questa non è una soluzione adatta a determinare che l'intero flusso di dati proveniente dal socket venga letto in casi generali.

Ad eccezione di `info()` e `geturl()`, questi metodi possiedono la stessa interfaccia di un oggetto file — vedere la sezione 2.3.9 di questo manuale. (Non è un oggetto file built-in, comunque, cosìcché non può essere usato in quelle poche situazioni dove è richiesto un vero oggetto file built-in.)

Il metodo `info()` restituisce un'istanza della classe `mimetools.Message` contenente meta informazioni associate all'URL. Quando il metodo è HTTP, queste intestazioni sono quelle restituite dal server all'inizio della pagina HTML recuperata (incluso Content-Length e Content-Type). Quando il metodo è FTP restituisce la richiesta. Una intestazione Content-Type sarà presente se il tipo MIME può essere analizzato. Quando il metodo è un file locale, le intestazioni restituite includeranno una rappresentazione Date dell'ora dell'ultima modifica al file, un Content-Lenght indicante la dimensione del file, ed un Content-Type contenente un riferimento al tipo di file. Vedere anche la descrizione del modulo `mimetools`.

Il metodo `geturl()` restituisce l'URL reale della pagina. In alcuni casi, il server HTTP redirige un client verso un altra URL. La funzione `urlopen()` lo gestisce in modo trasparente, ma in alcuni casi il chiamante ha la necessità di sapere verso quale URL è stato rediretto il client. Il metodo `geturl()` può anche essere usato per posizionarsi su questa URL rediretta.

Se l'*url* utilizza lo schema identificativo 'http:' l'argomento facoltativo *data* può essere indicato per specificare una richiesta `POST` (normalmente il tipo di richiesta è `GET`). L'argomento *data* deve essere nel formato standard application/x-www-form-urlencoded; vedere sotto alla funzione `urlencode()`.

La funzione `urlopen()` lavora in modo trasparente con i proxy che non richiedono autenticazione. In un ambiente UNIX o Windows, impostare le variabili d'ambiente http_proxy, ftp_proxy o gopher_proxy alla URL che identifica il proxy server prima di avviare l'interprete Python. Per esempio (il '%' è il comando del prompt):

```
% http_proxy="http://www.someproxy.com:3128"
% export http_proxy
% python
...
```

In un ambiente Windows, se le variabili ambiente di proxy non sono impostate, l'impostazione del proxy sono ottenute dalla sezione delle impostazioni Internet del registro.

In un ambiente Macintosh, `urlopen()` recupererà le informazioni sul proxy da Internet Config.

Alternativamente, l'argomento facoltativo *proxies* può essere usato per specificare esplicitamente i proxy. Deve essere un dizionario con lo schema dei nomi delle URL del proxy mappati, un dizionario vuoto ottiene il risultato di non permettere l'uso di alcun proxy, e `None` (il valore predefinito) causa l'uso delle impostazioni proxy dell'ambiente, come discusso precedentemente. Per esempio:

```
# Usa http://www.someproxy.com:3128 come http proxy
proxies = proxies={'http': 'http://www.someproxy.com:3128'}
filehandle = urllib.urlopen(some_url, proxies=proxies)
# Non usa alcun proxy
filehandle = urllib.urlopen(some_url, proxies={})
# Usa i proxy dall'ambiente - entrambe le versioni sono equivalenti
filehandle = urllib.urlopen(some_url, proxies=None)
filehandle = urllib.urlopen(some_url)
```

La funzione `urlopen()` non supporta la specifica esplicita del proxy. Se si ha la necessità di sovrascrivere l'impostazione d'ambiente del proxy, si usi `URLopener`, o una sua sotto classe come `FancyURLopener`.

I proxy che richiedono l'autenticazione per il loro utilizzo, non sono attualmente supportati; questa è considerata una implementazione limitata.

Modificato nella versione 2.3: Added the *proxies* support.

`urlretrieve`(*url*[, *filename*[, *reporthook*[, *data*]]])

Copia un oggetto di rete denotato da una URL verso un file locale, se necessario. Se la URL punta ad un file locale, o se esiste una copia valida dell'oggetto, l'oggetto non viene copiato. Restituisce una tupla (*filename*, *headers*) dove *filename* è il nome del file locale in cui l'oggetto può essere trovato, e *headers* è qualsiasi cosa il metodo `info()` dell'oggetto restituito da `urlopen()` restituisca (per un oggetto remoto, possibilmente posizionato nella cache). Le eccezioni sono le stesse di `urlopen()`.

Il secondo argomento, se presente, specifica la posizione del file su cui copiare (se assente, la locazione sarà un file temporaneo con un nome generato). Il terzo argomento, se presente, è una funzione di aggancio che verrà chiamata una volta sulla connessione della rete ed una volta che un blocco verrà letto successivamente. L'aggancio passerà tre argomenti: un numero dei blocchi trasferiti fino a quel momento, una dimensione in byte del blocco, e la dimensione totale del file. Il terzo argomento deve essere un `-1` sui vecchi server FTP che non restituiscono la dimensione del file su di una richiesta di recupero.

Se la *url* usa lo schema identificativo 'http:', l'argomento facoltativo *data* può essere indicato per specificare una richiesta `POST` (normalmente il tipo di richiesta è `GET`). L'argomento *data* deve essere nel formato standard application/x-www-form-urlencoded; vedere sotto la funzione `urlencode()`.

`_urlopener`

Le funzioni pubbliche `urlopen()` e `urlretrieve()` creano una istanza della classe `FancyURLopener` e la usano per migliorare le loro azioni di richiesta. Per sovrascrivere queste funzionalità, i programmatori possono creare una sotto classe di `URLopener` o `FancyURLopener`, che assegna una istanza della classe alla variabile `urllib._urlopener` prima di chiamare la funzione desiderata. Per esempio, le applicazioni possono voler specificare un differente header User-Agent: rispetto a quello definito da `URLopener`. Questo può essere realizzato con il seguente codice:

```
import urllib

class AppURLopener(urllib.FancyURLopener):
    def __init__(self, *args):
        self.version = "App/1.7"
        urllib.FancyURLopener.__init__(self, *args)

urllib._urlopener = AppURLopener()
```

`urlcleanup`()

Pulisce la cache che potrebbe essere stata costruita da una precedente chiamata a `urlretrieve()`.

`quote`(*string*[, *safe*])

Sostituisce i carattere speciali all'interno della stringa *string*, usando il carattere di protezione '`%xx`'. Let-

tere, numeri ed i caratteri '_.-' non vengono mai quotati. Il parametro facoltativo *safe* specifica caratteri aggiuntivi che non dovrebbero essere quotati — il suo valore predefinito è '/'.

Esempio: quote('/~connolly/') yields '/%7econnolly/'.

quote_plus(*string*[, *safe*])
Come quote(), ma sostituisce anche gli spazi con un segno '+', come richiesto per la quotatura dei valori di form HTML. I segni '+' nella stringa originale vengono protetti finché non vengono inclusi in *safe*. Inoltre non possiede un predefinito *safe* per '/'.

unquote(*string*)
Sostituisce il carattere di protezione '%xx' con i loro equivalenti in singolo carattere.

Esempio: unquote('/%7Econnolly/') yields '/~connolly/'.

unquote_plus(*string*)
Come unquote(), ma sostituisce anche i segni '+' con spazi, come richiesto quando si toglie la quotatura ai valori delle form HTML.

urlencode(*query*[, *doseq*])
Converte un oggetto mappato o una sequenza di tuple di due elementi in una stringa "url codificata", passibile di essere passata a urlopen() come argomento facoltativo *data*. Questo è utile per passare un dizionario di campi di form ad una richiesta POST. La stringa risultante è una serie di coppie *chiave=valore* separate dal carattere '&', dove ogni *chiave* ed ogni *valore* vengono quotati usando la funzione quote_plus() indicata sopra. Se il parametro facoltativo *doseq* è presente e valutato come vero, le coppie individuali *chiave=valore* vengono generate per ogni elemento della sequenza. Quando una sequenza di tuple di due elementi viene usata come argomento di *query*, il primo elemento di ogni tupla è una chiave e il secondo è un valore. L'ordine dei parametri nella stringa codificata deve corrispondere all'ordine delle tuple dei parametri nella sequenza. Il modulo cgi fornisce le funzioni parse_qs() e parse_qsl() che vengono usate per analizzare le stringhe di interrogazione nelle strutture dati di Python.

pathname2url(*path*)
Converte dalla sintassi locale, un percorso di nomi *path* in un percorso nella forma usata nei percorsi di una URL. Questo non produce una URL completa. Il valore restituito sarà già quotato usando la funzione quote().

url2pathname(*path*)
converte il componente *path* dalla URL codificata, in una sintassi di un percorso locale. Questo non accetta una URL completa. Questa funzione usa unquote() per decodificare *path*.

class URLopener([*proxies*[, ***x509*]])
Classe di base per l'apertura e la lettura delle URL. Finché si avrà bisogno di supportare l'apertura di oggetti usando schemi diversi da 'http:', 'ftp:', 'gopher:' o 'file:', probabilmente si vorrà usare FancyURLopener.

Predefinitamente, la classe URLopener invia un'intestazioneUser-Agent: di 'urllib/*VVV*', dove *VVV* è la versione della urllib. Le applicazioni possono definire la propria intestazione User-Agent: attraverso sotto classi di URLopener o FancyURLopener ed impostando l'attributo version dell'istanza ad un appropriato valore stringa prima che il metodo open() venga chiamato.

Il parametro facoltativo *proxies* dovrebbe essere un dizionario che mappa gli schemi dei nomi in proxy URL, dove un dizionario vuoto imposta il proxy completamente ad off. Il suo valore predefinito è None, nel cui caso le impostazioni d'ambiente del proxy verranno usate se presenti, come discusso precedentemente nella definizione di urlopen().

Parametri di chiavi addizionali, raccolte in *x509*, vengono usate per l'autenticazione con lo schema 'https:'. Le chiavi *key_file* e *cert_file* vengono supportate; entrambe sono attualmente necessarie per recuperare una risorsa da una URL 'https:':

class FancyURLopener(...)
FancyURLopener (sotto classe di URLopener) fornisce il gestore predefinito per i seguenti codici di risposta HTTP: 301, 302, 303, 307 e 401. Per i codici di risposta 30x indicati sopra, l'intestazione Location: viene usata per scaricare l'URL attuale. Per i codici di risposta 401 (autenticazione richiesta), viene fornita l'autenticazione base dell'HTTP. Per i codici di risposta 30x, la ricorsione viene controllata dal valore dell'attributo *maxtries*, che ha un valore predefinito impostato a 10.

Note: In accordo alle indicazioni dell'RFC 2616, le risposte 301 e 302 a richieste POST non devono essere automaticamente redirette senza la conferma dell'utente. In realtà, i browser eseguono la redire-

zione automatica di fronte a queste risposte, cambiando il POST in GET, e `urllib` riproduce questa caratteristica.

Il parametro passato al costruttore è lo stesso di quello per `URLopener`.

Note: Quando si esegue l'autenticazione di base, un'istanza `FancyURLopener` chiama il suo metodo `prompt_user_passwd()`. L'implementazione predefinita domanda all'utente le informazioni richieste sul terminale in uso. Se necessario, una sotto classe può sovrascrivere questo metodo per supportare caratteristiche più appropriate.

Restrizioni:

- Correntemente, solo i seguenti protocolli vengono supportati: HTTP, (Versione 0.9 e 1.0), Gopher (man non Gopher-+), FTP e file locali.
- La funzionalità di caching di `urlretrieve()` è stata disabilitata fino a quando non si troverà il corretto processo per le intestazioni di "Expiration time".
- Ci dovrebbe essere una funzione per verificare quale particolare URL è nella cache.
- Per compatibilità con il passato, se una URL mostra di puntare a un file locale, ma il file non può essere aperto, la URL viene reinterpretata usando il protocollo FTP. Questo può causare in alcune occasione messaggi di errori confusi.
- Le funzioni `urlopen()` e `urlretrieve()` possono causare lunghi ritardi quando attendono che una connessione di rete venga attivata. Questo significa che è difficile costruire un client Web interattivo usando questa funzione senza l'uso dei threads.
- Il dato restituito da `urlopen()` o `urlretrieve()` è il dato grezzo restituito dal server. Questo può essere un dato binario (per esempio un'immagine), puro testo o (per esempio) HTML. Il protocollo HTTP fornisce il tipo di informazione nell'intestazione della risposta, che può essere controllato cercando l'intestazione Content-Type:. Per il protocollo , il tipo di informazione è codificata nell'URL; non c'è attualmente nessun modo facile per estrarlo. Se il dato restituito è HTML, si può usare il modulo `htmllib` per analizzarlo.
- Questo modulo non supporta l'uso di proxy che richiedono autenticazione. Questo deve essere implementato in futuro.
- Il modulo `urllib` contiene (non documentate) funzioni per l'analisi e la deanalisi delle stringhe URL, l'interfaccia raccomandata per la manipolazione è nel modulo `urlparse`.

11.4.1 Oggetti URLopener

Gli oggetti `URLopener` e `FancyURLopener` possiedono i seguenti attributi.

open(*fullurl*[, *data*])
Apre *fullurl* usando il protocollo appropriato. Questo metodo imposta la cache e le informazioni di proxy, quindi chiama l'appropriato metodo `open()` con i suoi argomenti di input. Se lo schema non viene riconosciuto, viene chiamato `open_unknown()`. L'argomento *data* ha lo stesso significato dell'argomento *data* di `urlopen()`.

open_unknown(*fullurl*[, *data*])
Interfaccia sovrascrivibile per aprire tipi di URL sconosciuti.

retrieve(*url*[, *filename*[, *reporthook*[, *data*]]])
Recupera il contenuto della *url* e la inserisce in *filename*. Il valore restituito è una tupla consistente di un nome file locale e un oggetto `mimetools.Message` contenente l'intestazione di risposta (per gli URL remoti) o `None` (per le URL locali). Il chiamante deve successivamente aprire e leggere il contenuto di *filename*. Se *filename* non viene passato, e la URL si riferisce ad file locale, viene restituito il *filename* in ingresso. Se la URL non è locale e *filename* non viene passato, *filename* è l'output di `tempfile.mktemp()` con un suffisso che corrisponde al suffisso del percorso dell'ultimo componente della URL in ingresso. Se *reporthook* viene passato, deve essere una funzione che accetta tre parametri numerici. Verrà chiamata dopo che ogni blocco di dati viene letto dal network. *reporthook* viene ignorato per gli URL locali.

Se la *url* utilizza lo schema identificativo 'http:', l'argomento facoltativo *data* deve essere passato per specificare una richiesta `POST` (normalmente il tipo di richiesta è `GET`). L'argomento *data* deve essere nel formato standard application/x-www-form-urlencoded; vedere la funzione `urlencode()` sotto.

`version`
Variabile che specifica lo user agent dell'oggetto opener. Per fare in modo che `urllib` dica ai server che si tratta di un particolare user agent, impostare questa in una sotto classe come una classe variabile o nel costruttore prima della chiamata al costruttore base.

La classe `FancyURLopener` offre un ulteriore metodo che può essere sovrascritto per fornire l'approccio più corretto:

`prompt_user_passwd`(*host*, *realm*)
Restituisce le informazioni necessarie per autenticare l'utente all'host indicato nello specifico contesto di sicurezza. Il valore restituito deve essere una tupla (*user*, *password*), che può venire usata per l'autenticazione di base.

L'implementazione chiede queste informazioni sul terminale; un'applicazione dovrebbe sovrascrivere questo metodo per usare un modello appropriato di interazione nell'ambiente locale.

11.4.2 Esempi

Di seguito un esempio di sessione che usa il metodo '`GET`' per recuperare una URL contenente dei parametri:

```
>>> import urllib
>>> params = urllib.urlencode({'spam': 1, 'eggs': 2, 'bacon': 0})
>>> f = urllib.urlopen("http://www.musi-cal.com/cgi-bin/query?%s" % params)
>>> print f.read()
```

L'esempio seguente usa il metodo '`POST`' invece del precedente '`GET`':

```
>>> import urllib
>>> params = urllib.urlencode({'spam': 1, 'eggs': 2, 'bacon': 0})
>>> f = urllib.urlopen("http://www.musi-cal.com/cgi-bin/query", params)
>>> print f.read()
```

L'esempio successivo usa un proxy HTTP specifico, sovrascrivendo le impostazioni d'ambiente:

```
>>> import urllib
>>> proxies = {'http': 'http://proxy.example.com:8080/'}
>>> opener = urllib.FancyURLopener(proxies)
>>> f = opener.open("http://www.python.org")
>>> f.read()
```

L'esempio non usa alcun proxy, sovrascrivendo le impostazioni d'ambiente:

```
>>> import urllib
>>> opener = urllib.FancyURLopener({})
>>> f = opener.open("http://www.python.org/")
>>> f.read()
```

11.5 `urllib2` — Libreria estensibile per l'apertura delle URL

Il modulo `urllib2` definisce funzioni e classi che aiutano nell'apertura di URL (in massima parte HTTP) in un mondo complesso: autenticazione di base e digest, redirezione e altro.

il modulo `urllib2` definisce le seguenti funzioni:

`urlopen`(*url*[, *data*])
Apre l'URL *url*, che può essere sia una stringa che un oggetto `Request` (attualmente il codice verifica che sia realmente un'istanza `Request`, o un'istanza di una sotto classe di `Request`).

data dovrebbe essere una stringa, che specifica ulteriori dati da inviare al server. In una richiesta HTTP, che è l'unica a supportare *data*, dovrebbe essere un buffer nel formato di application/x-www-form-urlencoded, per esempio come quello restituito da `urllib.urlencode()`.

La funzione restituisce un oggetto simile a file con due ulteriori metodi:

- `geturl()` — restituisce l'URL della risorsa recuperata
- `info()` — restituisce le meta informazioni della pagina, come un oggetto simile ad un dizionario

In caso di errore solleva l'eccezione `URLError`.

`install_opener`(*opener*)
Installa un oggetto `OpenerDirector` nel predefinito *opener*. Il codice non verifica per un `OpenerDirector` reale, ed ogni classe con l'appropriata interfaccia sarà in grado di lavorare.

`build_opener`([*handler, ...*])
Restituisce un'istanza di `OpenerDirector`, che concatena gli handler nell'ordine in cui vengono forniti. Gli *handler* possono essere sia istanze di `BaseHandler`, o sotto classi di `BaseHandler` (in tal caso è possibile chiamare il costruttore senza alcun parametro). Le istanze delle seguenti classi verranno inserite nelle prime posizioni nell'*handler*, fino a che l'*handler* stesso le conterrà, o conterrà istanze o sotto classi di loro stesse: `ProxyHandler`, `UnknownHandler`, `HTTPHandler`, `HTTPDefaultErrorHandler`, `HTTPRedirectHandler`, `FTPHandler`, `FileHandler`, `HTTPErrorProcessor`.

Se l'installazione di Python ha il supporto SSL (se `socket.ssl()` esiste), verrà aggiunto anche `HTTPSHandler`.

A partire da Python 2.3, una sotto classe di `BaseHandler` può anche cambiare la propria variabile `handler_order` per modificare la sua posizione nella lista degli handler. Dopo `ProxyHandler`, che ha un `handler_order` del valore di `100`, tutti gli handler vengono impostati a `500`.

Le seguenti eccezioni vengono sollevate quando necessario:

exception `URLError`
Gli handler (NdT: gestori) sollevano questa eccezione (o l'eccezione derivata) quando incappano in un problema. Si tratta di una sotto classe di `IOError`.

exception `HTTPError`
Una sotto classe di `URLError`, può anche funzionare restituendo un valore simile a file non particolare (la stessa cosa restituita da `urlopen()`). Questo è utile quando si gestiscono errori HTTP particolari, come una richiesta di autenticazione.

exception `GopherError`
Una sotto classe di `URLError`, questo è un errore sollevato dall'handler del Gopher.

Sono disponibili le seguenti classi:

class `Request`(*url*[, *data*[, *headers*]])
Questa classe è una astrazione di una richiesta URL.

url dovrebbe essere una stringa che rappresenta una URL valida. Per una descrizione di *data* vedere la descrizione di `add_data()`. Gli *headers* dovrebbero essere un dizionario, e verranno trattati come quando `add_header()` viene chiamato con ogni chiave e valore per argomento.

class `OpenerDirector`()
La classe `OpenerDirector` apre gli URL tramite `BaseHandler` congiuntamente. Gestisce il collegamento degli handler, ed il recupero dagli errori.

class `BaseHandler`()
Questa è una classe di base per tutti gli handler registrati; e gestisce solo i semplici meccanismi di registrazione.

class `HTTPDefaultErrorHandler`()
Una classe che definisce un handler predefinito per le risposte di errore HTTP; tutte le risposte sono convertite in eccezioni `HTTPError`.

class `HTTPRedirectHandler`()
Una classe che gestisce le redirezioni.

class `ProxyHandler`([*proxies*])
Fa sì che le richieste transitino attraverso un proxy. Se *proxies* viene indicato, deve essere un dizionario che mappi i nomi di protocollo in URL di proxy. Il predefinito è leggere la lista di proxy dalla variabili ambiente *protocol*_proxy.

class `HTTPPasswordMgr`()
Mantiene un database di (*realm*, *uri*) -> (*user*, *password*) mappate.

class `HTTPPasswordMgrWithDefaultRealm`()
Mantiene un database di (*realm*, *uri*) -> (*user*, *password*) mappati. Un insieme di `None` viene considerato un insieme catch-all (NdT: acchiappa tutto) che viene ricercato se nessun altro insieme corrisponde.

class `AbstractBasicAuthHandler`([*password_mgr*])
Questa è una classe intermedia che aiuta con l'autenticazione HTTP, sia verso host remoti che verso proxy. *password_mgr*, se passata, dovrebbe essere qualcosa di compatibile con `HTTPPasswordMgr`; fare riferimento alla sezione 11.5.6 per informazioni sull'interfaccia che deve essere supportata.

class `HTTPBasicAuthHandler`([*password_mgr*])
Gestisce l'autenticazione con l'host remoto. *password_mgr*, se passata, dovrebbe essere qualcosa di compatibile con `HTTPPasswordMgr`; fare riferimento alla sezione 11.5.6 per informazioni sulle interfacce che devono essere supportate.

class `ProxyBasicAuthHandler`([*password_mgr*])
Gestisce l'autenticazione con il proxy. *password_mgr*, se passata, dovrebbe essere qualcosa di compatibile con `HTTPPasswordMgr`; fare riferimento alla sezione 11.5.6 per informazioni sull'interfaccia deve dev'essere supportata.

class `AbstractDigestAuthHandler`([*password_mgr*])
Questa è una classe intermedia che aiuta con l'autenticazione HTTP, sia verso host remoti che verso proxy. *password_mgr*, se passata, dovrebbe essere qualcosa che sia compatibile con HTTPPasswordMgr; fare riferimento alla sezione 11.5.6 per informazioni circa l'interfaccia che deve essere supportata.

class `HTTPDigestAuthHandler`([*password_mgr*])
Gestisce l'autenticazione con l'host remoto. *password_mgr*, se passata, dovrebbe essere qualcosa di compatibile con `HTTPPasswordMgr`; fare riferimento alla sezione 11.5.6 per informazioni sull'interfaccia che deve essere supportata.

class `ProxyDigestAuthHandler`([*password_mgr*])
Gestisce le autenticazioni con il proxy. *password_mgr*, se passata, dovrebbe essere qualcosa di compatibile con `HTTPPasswordMgr`; fare riferimento alla sezione 11.5.6 per informazioni sull'interfaccia che deve essere supportata.

class `HTTPHandler`()
Una classe che gestisce l'apertura di URL HTTP.

class `HTTPSHandler`()
Una classe che gestisce l'apertura di URL HTTPS.

class `FileHandler`()
Apre file locali.

class `FTPHandler`()
Apre URL FTP.

class `CacheFTPHandler`()
Apre URL FTP, mantenendo una cache delle connessioni FTP aperte per minimizzare i ritardi.

class `GopherHandler`()
Apre URL gopher.

class `UnknownHandler`()
Una classe acchiappa tutto per gestire URL non riconosciute.

11.5.1 Oggetti Request

I seguenti metodi descrivono tutto delle interfacce pubbliche `Request`, e tutte devono essere sovrascritte in sotto classi.

`add_data`(*data*)
Imposta i dati di `Request` a *data*. Questo è ignorato da tutti gli handler eccezion fatta per gli handler HTTP; ci dovrebbe essere anche un buffer application/x-www-form-encoded, e la richiesta sarà cambiata in `POST` piuttosto che `GET`.

`get_method`()
Restituisce una stringa che indica il metodo di richiesta HTTP. Questo è significativo solo per le richieste HTTP, e correntemente accetta solamente uno dei valori (GET, POST).

`has_data`()
Ritorna se l'istanza ha un dato non `None`.

`get_data`()
Restituisce i dati dell'istanza.

`add_header`(*key, val*)
Aggiunge un'altra intestazione *header* alla richiesta. Le intestazioni vengono attualmente ignorate da tutti gli handler, fatta eccezione per quelli HTTP, dove vengono aggiunti alla lista delle intestazioni inviate al server. Notare che non ci può essere più di un'intestazione con lo stesso nome, e le chiamate successive sovrascriveranno le precedenti chiamati nel caso che *key* sia la stessa. Attualmente non ci sono perdite di funzionalità HTTP, finché tutte le intestazioni che hanno un significato quando vengono usate più di una volta hanno (un'intestazione specifica) modo di mantenere la stessa funzionalità usando una sola intestazione.

`add_unredirected_header`(*key, header*)
Aggiunge un'intestazione che non verrà aggiunta ad una richiesta rediretta. Nuovo nella versione 2.4.

`has_header`(*header*)
Restituisce entrambe le istanze che hanno l'intestazione *header* indicato (verifica sia i regolari che i non rediretti). Nuovo nella versione 2.4.

`get_full_url`()
Restituisce l'URL indicato nel costruttore.

`get_type`()
Restituisce il tipo di URL; conosciuto anche come lo schema.

`get_host`()
Restituisce l'host verso cui la connessione verrà instaurata.

`get_selector`()
Restituisce il selettore *selector*: la parte dell'URL che viene inviata al server.

`set_proxy`(*host, type*)
Prepara la richiesta connettendosi a un proxy server. *host* e *type* rimpiazzeranno quelli dell'instanza, ed il selettore dell'istanza sarà l'URL originale fornita nel costruttore.

11.5.2 Oggetti OpenerDirector

Le istanze `OpenerDirector` hanno i seguenti metodi:

add_handler(*handler*)
handler dovrebbe essere un'istanza di `BaseHandler`. I seguenti metodi vengono cercati, e aggiunti alle possibili catene.

- *protocol*`_open()` — segnala che l'handler sa come aprire le URL di *protocol*.
- *protocol*`_error_`*type*`()` — segnala che l'handler sa come gestire i *type* di errori di *protocol*.
- *protocol*`_request()` — segnala che l'handler sa come pre elaborare le richieste di *protocol*.
- *protocol*`_response()` — segnala che l'handler sa come post elaborare le risposte di *protocol*.

close()
Esplicitamente interrompe i cicli, e cancella tutti gli handler. Siccome `OpenerDirector` necessita di sapere degli handler registrati, e un handler deve sapere quale `OpenerDirector` lo ha chiamato, si verifica una referenza ciclica. Anche se versioni recenti di Python hanno collezioni cicliche, è preferibile, qualche volta, interrompere esplicitamente i cicli.

open(*url*[, *data*])
Apre l'*url* passata (che può essere un oggetto di richiesta o una stringa), facoltativamente passando *data*. Argomenti, valori restituiti ed eccezioni sollevate sono le stesse di quelle di `urlopen()` (che semplicemente chiama il metodo `open()` sull'`OpenerDirector` installato in modo predefinito).

error(*proto*[, *arg*[, ...]])
Gestisce un errore di un protocollo passato. Questo chiamerà gli handler degli errori registrati per il protocollo passato con gli argomenti anch'essi passati (che sono specifici del protocollo). Il protocollo HTTP è un caso speciale che usa i codici di risposta HTTP per determinare il gestore di errore specifico; fare riferimento ai metodi `http_error_*()` delle classi handler.

Valori restituiti ed eccezioni sollevate sono le stesse di quelle di `urlopen()`.

11.5.3 Oggetti BaseHandler

Gli oggetti `BaseHandler` forniscono un insieme di metodi che sono utili direttamente, ed altri che si intendono per essere utilizzati da classi derivate. Questi sono quelli intesi per l'uso diretto:

add_parent(*director*)
Aggiunge un director come padre.

close()
Rimuove ogni padre.

I seguenti membri e metodi dovrebbero essere usati solo da classi derivate da `BaseHandler`:

parent
Un `OpenerDirector` valido, che può essere usato per aprire usando un protocollo differente, o per gestire errori.

default_open(*req*)
Questo metodo *non* viene definito in `BaseHandler`, ma le sotto classi dovrebbero definirlo se vogliono intercettare tutte le URL.

Questo metodo, se implementato, verrà chiamato dall'`OpenerDirector` padre. Dovrebbe restituire un oggetto simile a file, come descritto nel valore restituito da `open()` di `OpenerDirector`, o `None`. Dovrebbe sollevare l'eccezione `URLError`, finché un qualcosa di realmente eccezionale accada (per esempio, `MemoryError` non dovrebbe essere mappata con `URLError`).

Questo metodo sarà chiamato prima di ogni metodo di apertura specifica del protocollo.

***protocol*_open**(*req*)
Questo metodo *non* viene definito in `BaseHandler`, ma le sotto classi dovrebbero definirlo se vogliono gestire URL con il protocollo indicato.

Questo metodo, se definito, sarà chiamato dall'`OpenerDirector` `padre`. I valori restituiti dovrebbero essere gli stessi di `default_open()`.

unknown_open(*req*)
Questo metodo *non* viene definito in `BaseHandler`, ma le sotto classi dovrebbero definirlo se vogliono intercettare tutte le URL senza uno specifico handler registrato per la sua apertura.

Questo metodo, se implementato, verrà chiamato dall'`OpenerDirector padre`. I valori restituiti dovrebbero essere gli stessi di `default_open()`.

http_error_default(*req, fp, code, msg, hdrs*)
Questo metodo *non* viene definito in `BaseHandler`, ma le sotto classi dovrebbero sovrascriverlo se intendono fornire un sistema acchiappa tutto per gli errori HTTP non altrimenti gestiti. Verrà chiamato automaticamente dall'`OpenerDirector` che riceve l'errore, e non dovrebbe essere normalmente chiamato in altre circostanze.

req sarà un oggetto `Request`, *fp* sarà un oggetto simile a file con il corpo dell'errore HTTP, *code* sarà un codice di tre cifre dell'errore, *msg* sarà la spiegazione visibile all'utente del codice e *hdrs* sarà un oggetto corrispondente alle intestazioni dell'errore.

I valori restituiti e le eccezioni sollevate dovrebbero essere le stesse di `urlopen()`.

http_error_*nnn*(*req, fp, code, msg, hdrs*)
nnn dovrebbe essere un codice di errore HTTP di tre cifre. Anche questo metodo non viene definito in `BaseHandler`, ma sarà chiamato, se esiste, su di una istanza di una sotto classe, quando interviene un errore HTTP con codice *nnn*.

Le sotto classi dovrebbero sovrascrivere questo metodo per gestire errori HTTP specifici.

Argomenti, valori restituiti ed eccezioni sollevate dovrebbero esser le stesse come per `http_error_default()`.

11.5.4 Oggetti HTTPRedirectHandler

Note: Alcune redirezioni HTTP richiedono un'azione dal codice di questo modulo cliente. Se è questo il caso, viene sollevata l'eccezione `HTTPError`. Vedere l'RFC 2616 per i dettagli del preciso significato dei vari codici di redirezione.

redirect_request(*req, fp, code, msg, hdrs*)
Restituisce un `Request` o `None` in risposta ad un redirect. Questo viene chiamato dall'implementazione predefinita dei metodi `http_error_30*()` quando una redirezione viene ricevuta dal server. Se una redirezione deve avvenire, restituisce un nuovo `Request` per permettere ad `http_error_30*()` di eseguire la redirezione. Altrimenti, viene sollevata l'eccezione `HTTPError` se nessun altro `Handler` vuole provare a gestire questa URL, o restituisce `None` se un altro `Handler` potrebbe gestirla.

Note: L'implementazione predefinita di questo metodo non rispecchia fedelmente l'RFC 2616, che dice che le risposte 301 e 302 a richieste `POST` non devono essere automaticamente redirette senza la conferma dell'utente. In realtà, i browser consentono la redirezione automatica di queste risposte, cambiando POST in `GET`, e l'implementazione predefinita riproduce questa particolarità.

http_error_301(*req, fp, code, msg, hdrs*)
Redirige verso la `Location:` URL. Questo metodo viene chiamato dall'`OpenerDirector` padre quando si riceve una risposta HTTP 'spostato permanentemente'.

http_error_302(*req, fp, code, msg, hdrs*)
Come per `http_error_301()`, ma chiamato per le risposte 'found' (NdT: trovato).

http_error_303(*req, fp, code, msg, hdrs*)
Lo stesso di `http_error_301()`, ma chiamato per risposte 'see other' (NdT: vedere altro).

http_error_307(*req, fp, code, msg, hdrs*)
Lo stesso di `http_error_301()`, ma chiamato per risposte 'temporary redirect' (temporaneamente rediretto).

11.5.5 Oggetti ProxyHandler

***protocol*_open**(*request*)
Il `ProxyHandler` avrà un metodo *protocol*`_open()` per ogni *protocol* che ha un proxy nel dizionario

dei *proxies* fornito dal costruttore. Il metodo modificherà la richiesta, per attraversare il proxy, chiamato `request.set_proxy()`, e chiamando il prossimo handler nella catena per eseguire attualmente il protocollo.

11.5.6 Oggetti HTTPPasswordMgr

Questi metodi sono disponibili con oggetti `HTTPPasswordMgr` ed oggetti `HTTPPasswordMgrWithDefaultRealm`.

`add_password`(*realm, uri, user, passwd*)
uri può essere sia una singola URI, o una sequenza di URI. *realm*, *user* e *passwd* devono essere stringhe. Queste comportano (*user, passwd*) che vengano usate come elementi di autenticazione quando è richiesta autenticazione per l'ambito corrente ed è fornito un super URI di ognuna delle URI.

`find_user_password`(*realm, authuri*)
Ottiene user/password per i realm ed URI passate, se ce ne sono. Restituirà `(None, None)` se non ci sono user/password corrispondenti.

Per oggetti `HTTPPasswordMgrWithDefaultRealm`, il *realm* `None` verrà cercato se il *realm* passato non possiede corrispondenti user/password.

11.5.7 Oggetti AbstractBasicAuthHandler

`handle_authentication_request`(*authreq, host, req, headers*)
Gestisce una richiesta di autenticazione prendendo una coppia user/password, e ritentando la richiesta. *authreq* deve essere il nome dell'intestazione dove l'informazione circa il *realm* viene inclusa nella richiesta, *host* è l'host dove autenticarsi, *req* dovrebbe essere l'oggetto (fallito) `Request`, ed *headers* dovrebbe rappresentare l'intestazione dell'errore.

11.5.8 Oggetti HTTPBasicAuthHandler Objects

`http_error_401`(*req, fp, code, msg, hdrs*)
Ritenta la richiesta con le informazioni di autenticazione, se disponibili.

11.5.9 Oggetti ProxyBasicAuthHandler Objects

`http_error_407`(*req, fp, code, msg, hdrs*)
Ritenta la richiesta con le informazioni di autenticazione, se disponibili.

11.5.10 Oggetti AbstractDigestAuthHandler

`handle_authentication_request`(*authreq, host, req, headers*)
authreq dovrebbe essere il nome dell'intestazione dove l'informazione circa il *realm* viene incluso nella richiesta, *host* dovrebbe essere l'host verso cui effettuare l'autenticazione, *req* dovrebbe essere l'oggetto `Request` (fallito), e *headers* dovrebbe essere l'intestazione dell'errore.

11.5.11 Oggetti HTTPDigestAuthHandler

`http_error_401`(*req, fp, code, msg, hdrs*)
Ritenta la richiesta con le informazioni di autenticazione, se disponibili.

11.5.12 Oggetti ProxyDigestAuthHandler

`http_error_407`(*req, fp, code, msg, hdrs*)
Ritenta la richiesta con le informazioni di autenticazione, se disponibili.

11.5.13 Oggetti HTTPHandler

`http_open`(*req*)
Invia una richiesta HTTP, che può essere sia GET che POST, dipendente da *req*`.has_data()`.

11.5.14 Oggetti HTTPSHandler

`https_open`(*req*)
Invia una richiesta HTTPS, che può essere sia GET che POST, dipendente da *req*`.has_data()`.

11.5.15 Oggetti FileHandler

`file_open`(*req*)
Apre un file localmente, se non c'è il nome dell'host, o il nome dell'host è `localhost`. Altrimenti, cambia il protocollo in `ftp`, e ritenta l'apertura usando l'oggetto `padre`.

11.5.16 Oggetti FTPHandler

`ftp_open`(*req*)
Apre il file FTP indicato da *req*. Il login viene sempre eseguito con username e password vuote.

11.5.17 Oggetti CacheFTPHandler

Gli oggetti `CacheFTPHandler` sono oggetti `FTPHandler` che possiedono i seguenti metodi addizionali:

`setTimeout`(*t*)
Imposta una pausa nella connessione a *t* secondi.

`setMaxConns`(*m*)
Imposta il numero massimo di connessioni a *m*.

11.5.18 Oggetti GopherHandler

`gopher_open`(*req*)
Apre la risorsa gopher indicata da *req*.

11.5.19 Oggetti UnknownHandler

`unknown_open`()
Solleva un'eccezione `URLError`.

11.5.20 Oggetti HTTPErrorProcessor

Nuovo nella versione 2.4.

`unknown_open`()
Elabora le risposte di errore HTTP.

Per i codici di errore 200, l'oggetto di risposta viene restituito immediatamente.

Per i codici di errore non 200, questi passa semplicemente il lavoro al metodo gestore *protocol*`_error_`*code*`()` attraverso `OpenerDirector.error()`. Eventualmente, `urllib2.HTTPDefaultErrorHandler` solleverà un'eccezione `HTTPError` se nessun'altro gestore gestisce l'errore.

11.5.21 Esempi

Questo esempio ottiene la pagina principale di python.org e mostra i suoi primi 100 byte.

```
>>> import urllib2
>>> f = urllib2.urlopen('http://www.python.org/')
>>> print f.read(100)
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<?xml-stylesheet href="./css/ht2html
```

Qui, si sta inviando un flusso di dati attraverso lo stdin di una CGI e leggendo i dati restituiti:

```
>>> import urllib2
>>> req = urllib2.Request(url='https://localhost/cgi-bin/test.cgi',
...                       data='This data is passed to stdin of the CGI')
>>> f = urllib2.urlopen(req)
>>> print f.read()
Got Data: "This data is passed to stdin of the CGI"
```

Il codice per il CGI di esempio usato nel precedente esempio è:

```
#!/usr/bin/env python
import sys
data = sys.stdin.read()
print 'Content-type: text-plain\n\nGot Data: ''%s''' % data
```

11.6 `httplib` — Client del protocollo HTTP

Questo modulo definisce le classi che implementano il lato client dei protocolli HTTP ed HTTPS. Non è normalmente usato direttamente — il modulo `urllib` li usa per gestire gli URL che fanno uso di HTTP ed HTTPS.

Note: Il supporto HTTPS è disponibile solo se il modulo `socket` è stato compilato con il supporto SSL.

Note: L'interfaccia pubblica per questo modulo è cambiata in modo sostanziale in Python 2.0. La classe HTTP viene mantenuta solo per compatibilità con la 1.5.2. Non dovrebbe essere usata in nuovo codice. Fare riferimento alla docstrings online per il suo utilizzo.

Le costanti definite in questo modulo sono:

`HTTP_PORT`
La porta predefinita per il protocollo HTTP (sempre `80`).

`HTTPS_PORT`
La porta predefinita per il protocollo HTTPS (sempre `443`).

Il modulo fornisce le seguenti classi:

class `HTTPConnection`(*host*[, *port*])
Un'istanza `HTTPConnection` rappresentante una transazione con un server HTTP. Dovrebbe essere istanziata passandogli un host ed un numero di porta facoltativo. Se non viene passato alcun numero di porta, questa viene estratta attraverso la stringa host se è nella forma *host*`:`*port*, altrimenti viene utilizzata la porta HTTP predefinita `80`). Per esempio, le seguenti chiamate creano tutte istanze che si connettono al server allo stesso host e porta:

```
>>> h1 = httplib.HTTPConnection('www.cwi.nl')
>>> h2 = httplib.HTTPConnection('www.cwi.nl:80')
>>> h3 = httplib.HTTPConnection('www.cwi.nl', 80)
```

Nuovo nella versione 2.0.

class `HTTPSConnection`(*host*[, *port, key_file, cert_file*])
Una sotto classe di `HTTPConnection` che usa SSL per le comunicazioni con server sicuri. Predefinitamente utilizza la porta `443`. *key_file* è il nome di un file formattato PEM che contiene la propria chiave privata. *cert_file* è un file contenente il certificato formattato PEM.

Avvertenze: Questo non effettua alcuna verifica del certificato!

Nuovo nella versione 2.0.

class `HTTPResponse`(*sock*[, *debuglevel=0*][, *strict=0*])
Classe la cui istanza viene restituita in caso di connessione con successo. Non viene istanziata direttamente dall'utente. Nuovo nella versione 2.0.

Le seguenti eccezioni vengono sollevate quando appropriato:

exception `HTTPException`
La classe di base di altre eccezioni di questo modulo. È una sotto classe di `Exception`. Nuovo nella versione 2.0.

exception `NotConnected`
Una sotto classe di `HTTPException`. Nuovo nella versione 2.0.

exception `InvalidURL`
Una sotto classe di `HTTPException`, sollevata quando viene fornita una porta che si riveli essere non numerica o vuota. Nuovo nella versione 2.3.

exception `UnknownProtocol`
Una sotto classe di `HTTPException`. Nuovo nella versione 2.0.

exception `UnknownTransferEncoding`
Una sotto classe di `HTTPException`. Nuovo nella versione 2.0.

exception `UnimplementedFileMode`
Una sotto classe di `HTTPException`. Nuovo nella versione 2.0.

exception `IncompleteRead`
Una sotto classe di `HTTPException`. Nuovo nella versione 2.0.

exception `ImproperConnectionState`
Una sotto classe di `HTTPException`. Nuovo nella versione 2.0.

exception `CannotSendRequest`
Una sotto classe di `ImproperConnectionState`. Nuovo nella versione 2.0.

exception `CannotSendHeader`
Una sotto classe di `ImproperConnectionState`. Nuovo nella versione 2.0.

exception `ResponseNotReady`
Una sotto classe di `ImproperConnectionState`. Nuovo nella versione 2.0.

exception `BadStatusLine`
Una sotto classe di `HTTPException`. Sollevata se un server risponde con un codice di stato HTTP che non viene capito. Nuovo nella versione 2.0.

11.6.1 Oggetti HTTPConnection

Le istanze `HTTPConnection` possiedono i seguenti metodi:

`request`(*method, url*[, *body*[, *headers*]])
Questo metodo invia una richiesta al server usando il metodo *method* di richiesta HTTP ed il selettore *url*. Se l'argomento *body* è presente, dovrebbe essere una stringa di dati da inviare dopo che gli header

sono terminati. L'header Content-Lenght viene automaticamente impostato al valore corretto. L'argomento *headers* deve essere una mappa di intestazioni HTTP supplementari da inviare con la richiesta.

getresponse()
Dovrebbe essere chiamato dopo che una richiesta è inviata per ottenere una risposta dal server. Restituisce un'istanza HTTPResponse.

set_debuglevel(*level*)
Imposta il livello di debug a *level* (il quantitativo di messaggi di debug stampati). Il livello predefinito è 0, che significa nessun messaggio di debug stampato.

connect()
Si connette al server specificato quando l'oggetto viene creato.

close()
Chiude una connessione al server.

send(*data*)
Invia i dati *data* al server. Questo dovrebbe essere usato direttamente solo dopo che il metodo endheaders() viene chiamato e prima della chiamata di getreply().

putrequest(*request, selector*[, *skip_host*[, *skip_accept_encoding*]])
Questa dovrebbe essere la prima chiamata dopo che la connessione al server è stata instaurata. Invia una riga al server consistente nella stringa *request*, la stringa *selector*, e la versione HTTP (HTTP/1.1). Per disabilitare l'invio automatico delle intestazioni Host: o Accept-Encoding: (per esempio per accettare ulteriori codifiche di contenuto), specificare *skip_host* o *skip_accept_encoding* con valori non Falsi.

Modificato nella versione 2.4: È stato aggiunto l'argomento *skip_accept_encoding*.

putheader(*header, argument*[, ...])
Invia un'intestazione in stile RFC 822 al server. Invia una riga al server consistente in una intestazione, un simbolo di due punti ':', uno spazio, ed il primo argomento. Se più argomenti vengono passati, viene inviata una riga di continuazione consistente in un 'tab' ed un argomento.

endheaders()
Invia una riga vuota al server, segnalando la fine delle intestazioni.

11.6.2 Oggetti HTTPResponse

Le istanze HTTPResponse possiedono i seguenti metodi ed attributi:

read([*amt*])
Legge e restituisce il corpo della risposta, o fino ai prossimi *amt* bytes.

getheader(*name*[, *default*])
Prende il contenuto dell'intestazione *name*, o *default* se non ci sono intestazioni corrispondenti.

msg
Una instanza mimetools.Message contenente l'intestazione di risposta.

version
Versione del protocollo HTTP utilizzato dal server. 10 per HTTP/1.0, 11 per HTTP/1.1

status
Codice di stato restituito dal server.

reason
Frase del motivo restituita dal server.

11.6.3 Esempi

Qui un esempio di sessione che utilizza il metodo 'GET':

```
>>> import httplib
>>> conn = httplib.HTTPConnection("www.python.org")
>>> conn.request("GET", "/index.html")
>>> r1 = conn.getresponse()
>>> print r1.status, r1.reason
200 OK
>>> data1 = r1.read()
>>> conn.request("GET", "/parrot.spam")
>>> r2 = conn.getresponse()
>>> print r2.status, r2.reason
404 Not Found
>>> data2 = r2.read()
>>> conn.close()
```

Qui un esempio di sessione che mostra come inviare richieste ‘POST’:

```
>>> import httplib, urllib
>>> params = urllib.urlencode({'spam': 1, 'eggs': 2, 'bacon': 0})
>>> headers = {"Content-type": "application/x-www-form-urlencoded",
...            "Accept": "text/plain"}
>>> conn = httplib.HTTPConnection("musi-cal.mojam.com:80")
>>> conn.request("POST", "/cgi-bin/query", params, headers)
>>> response = conn.getresponse()
>>> print response.status, response.reason
200 OK
>>> data = response.read()
>>> conn.close()
```

11.7 `ftplib` — Client per protocollo FTP

Questo modulo definisce la classe FTP ed una serie di elementi correlati. La classe FTP implementa la parte client del protocollo FTP. La si può utilizzare per scrivere programmi Python che eseguono una serie di compiti FTP automatizzati, come la duplicazione di altri server ftp. Viene anche utilizzato dal modulo `urllib` per gestire URL che utilizzano FTP. Per maggiori informazioni sull’FTP (Protocollo di Trasferimento File), vedere la RFC 959 Internet.

Qui un esempio di sessione che utilizza il modulo `ftplib`:

```
>>> from ftplib import FTP
>>> ftp = FTP('ftp.cwi.nl')   # Si connette all'host sulla porta predefinita
>>> ftp.login()               # utente anonymous, password anonymous@
>>> ftp.retrlines('LIST')     # elenca il contenuto della directory
total 24418
drwxrwsr-x   5 ftp-usr  pdmaint     1536 Mar 20 09:48 .
dr-xr-srwt 105 ftp-usr  pdmaint     1536 Mar 21 14:32 ..
-rw-r--r--   1 ftp-usr  pdmaint     5305 Mar 20 09:48 INDEX
 .
 .
 .
>>> ftp.retrbinary('RETR README', open('README', 'wb').write)
'226 Transfer complete.'
>>> ftp.quit()
```

Il modulo definisce i seguenti elementi:

class `FTP`([*host*[, *user*[, *passwd*[, *acct*]]]])
: Restituisce una nuova istanza della classe `FTP`. Quando viene passato *host*, il metodo chiama `connect(`*host*`)`. Quando *user* viene passato, viene ulteriormente chiamato il metodo `login(`*user*`, `*passwd*`, `*acct*`)` (dove, in modo predefinito, *passwd* e *acct* sono stringhe vuote, se non diversamente indicate).

`all_errors`
: L'insieme di tutte le eccezioni (come tuple) che i metodi dell'istanza FTP possono sollevare, come risultanza di problemi con la connessione FTP (all'opposto di errori di programmazione fatti dal chiamante). Questo insieme include le quattro eccezioni elencate sotto, come anche `socket.error` ed `IOError`.

exception `error_reply`
: Eccezione sollevata quando viene ricevuta una risposta inaspettata dal server.

exception `error_temp`
: Eccezione sollevata quando viene ricevuto un codice di errore nell'intervallo 400–499.

exception `error_perm`
: Eccezione sollevata quando viene ricevuto un codice di errore nell'intervallo 500–599.

exception `error_proto`
: Eccezione sollevata quando viene ricevuta una risposta dal server che non inizia con un numero nell'intervallo 1-5.

Vedete anche:

Modulo `netrc` (sezione 12.18):
: Analizzatore per file in formato '.netrc'. Il file '.netrc' viene tipicamente utilizzato dai client FTP per caricare le informazioni di autenticazione dell'utente prima di richiederle direttamente al prompt dell'utente stesso.

Il file 'Tools/scripts/ftpmirror.py' nel sorgente Python della distribuzione è uno script che può duplicare siti FTP, o porzioni di esso, usando il modulo `ftplib`. Può essere usato come un esempio esteso applicabile a questo modulo.

11.7.1 Oggetti FTP

Parecchi metodi sono disponibili in due versioni: uno per gestire testo ed un'altro per i file binari. Questi vengono chiamati per i comandi che vengono usati, seguiti da '`lines`' per la versione testo e '`binary`' per la versione binaria.

Le istanze `FTP` hanno i seguenti metodi:

`set_debuglevel`(*level*)
: Imposta il livello *level* di debugging dell'istanza. Questo controlla l'ammontare del quantitativo di messaggi stampati. Il valore predefinito è `0`, non produce alcun output di debugging. Un valore di `1` produce un quantitativo moderato di messaggi di output, generalmente una singola riga per richiesta. Un valore di `2` o maggiore produce il massimo quantitativo di messaggi di debugging, registrando ogni riga inviata e ricevuta sulla connessione sotto controllo.

`connect`(*host*[, *port*])
: Si connette agli *host* e *port* passati. Il numero predefinito per la porta *port* è `21`, come indicato dalle specifiche del protocollo FTP. Raramente è necessario specificare un numero differente di porta. Questa funzione dovrebbe essere chiamata solo una volta per istanza; non dovrebbe essere chiamata mai se un host è già stato fornito quando l'istanza è stata creata. Tutti gli altri metodi possono essere usati dopo che la connessione è stata stabilita.

`getwelcome`()
: Restituisce il messaggio di benvenuto inviato dal server in risposta alla connessione iniziale. Questo messaggio, qualche volta, contiene avvisi o informazioni di aiuto che possono essere rilevanti per l'utente.

`login`([*user*[, *passwd*[, *acct*]]])
: Si connette con l'utente *user* passato. I parametri *passwd* ed *acct* sono facoltativi e, in modo predefinito, sono stringhe vuote. Se non viene specificato nessun *user*, il predefinito viene impostato ad `'anonymous'`.

Se l'utente *user* è `'anonymous'`, la password predefinita è `'anonymous@'`. Questa funzione dovrebbe essere chiamata solamente una volta per ogni istanza, dopo che una connessione è stata stabilita; non dovrebbe essere chiamata mai se un host ed un utente vengono forniti quando è stata creata l'istanza. Molti comandi FTP sono consentiti solamente dopo che il client si è autenticato.

`abort``()`
Annulla un trasferimento file che è in corso. L'uso di questo metodo non sempre funziona, ma vale la pena di fare una prova.

`sendcmd``(`*command*`)`
Invia una semplice stringa di comando al server e restituisce una stringa di risposta.

`voidcmd``(`*command*`)`
Invia una semplice stringa di comando al server e gestisce la risposta. Non restituisce niente se viene ricevuto un codice di risposta nell'intervallo 200-299. Altrimenti solleva un'eccezione.

`retrbinary``(`*command, callback*`[`*, maxblocksize*`[`*, rest*`]]`)
Recupera un file usando il modo di trasferimento binario. *command* dovrebbe essere un comando appropriato '`RETR`': `'RETR` *filename*`'`. La funzione *callback* viene chiamata per ogni blocco di dati ricevuti, con un singolo argomento stringa contenente il blocco di dati. L'argomento facoltativo *maxblocksize* specifica la misura massima del blocco da leggere sull'oggetto socket di basso livello, creato per eseguire il trasferimento in corso (che sarà anche la misura massima del blocco di dati passato al *callback*). Viene scelto un ragionevole valore predefinito. *rest* ha lo stesso significato che ha nel metodo `transfercmd()`.

`retrlines``(`*command*`[`*, callback*`])`
Restituisce un elenco di file o directory in modalità di trasferimento ASCII. *command* dovrebbe essere un appropriato comando '`RETR`' (vedere `retrbinary()`) o un comando '`LIST`' (tipicamente solo la stringa `'LIST'`). La funzione *callback* viene chiamata per ogni riga, rimuovendo il carattere CRLF. La *callback*, stampa in modo predefinito la riga su `sys.stdout`.

`set_pasv``(`*boolean*`)`
Abilita la modalità "passiva" se *boolean* è vera, altrimenti disabilita la modalità passiva. In Python 2.0 e successivi, la modalità passiva veniva impostata al valore predefinito off; dalla versione 2.1 in poi, viene impostata al valore predefinito on.

`storbinary``(`*command, file*`[`*, blocksize*`])`
Memorizza un file nella modalità trasferimento binaria. *command* dovrebbe essere un appropriato comando '`STOR`': `"STOR` *nomefile*`"`. *file* è un oggetto file aperto che viene letto fino ad EOF usando il proprio metodo `read()` in blocchi di misura *blocksize* che ha il valore predefinito 8192. Modificato nella versione 2.1: è stato aggiunto un valore predefinito per *blocksize*.

`storlines``(`*command, file*`)`
Memorizza un file usando la modalità di trasferimento ASCII. *command* dovrebbe essere un appropriato comando '`STOR`' (vedere `storbinary()`). Le righe vengono lette fino all'EOF da un oggetto *file* aperto usando il proprio metodo `readline()` che deve fornire i dati da memorizzare.

`transfercmd``(`*cmd*`[`*, rest*`])`
Inizia un trasferimento sopra la connessione dati. Se il trasferimento è attivo, invia un comando '`EPRT`' o '`PORT`', il comando di trasferimento specificato da *cmd*, e accetta la connessione. Se il server lavora in modalità passiva, invia un comando '`EPSV`' o '`PASV`', si connette, e avvia il comando di trasferimento. In entrambi i casi, restituisce il socket per la connessione.

Se viene fornita l'opzione *rest*, allora viene inviato un comando '`REST`' al server, passando *rest* come un argomento. *rest* è usualmente una posizione in byte all'interno del file richiesto, chiedendo al server di ricominciare nuovamente ad inviare i byte del file dalla posizione indicata, saltando sui byte iniziali. Notare, comunque, che la RFC 959 richiede solamente che *rest* sia una stringa contenente caratteri nell'intervallo stampabile dal codice ASCII 33 al codice ASCII 126. Il metodo `transfercmd()`, comunque, converte *rest* in una stringa, ma non ne controlla la validità del contenuto. Se il server non riconosce il comando '`REST`', verrà sollevata un'eccezione `error_reply`. Se questo avviene, chiamare semplicemente `transfercmd()` senza il comando *rest*.

`ntransfercmd``(`*cmd*`[`*, rest*`])`
Come `transfercmd()`, ma restituisce una tupla della connessione dati e la dimensione attesa dei dati. Se la dimensione attesa non può essere calcolata, verrà restituito, come dimensione attesa, `None`. *cmd* e

rest hanno lo stesso significato degli equivalenti in `transfercmd()`.

`nlst`(*argument*[, ...])
Restituisce un elenco di file come restituito dal comando '`NLST`'. L'argomento facoltativo *argument* è una directory da elencare (il suo valore predefinito è la directory corrente del server). Argomenti multipli possono essere usati per inviare opzioni non standard al comando '`NLST`'.

`dir`(*argument*[, ...])
Produce un elenco del contenuto della directory come restituito dal comando '`LIST`', inviandolo allo standard output. L'argomento facoltativo *argument* è una directory da elencare (il suo valore predefinito è la directory corrente del server). Argomenti multipli possono essere usati per inviare opzioni non standard al comando '`LIST`'. Se l'ultimo argomento è una funzione, questa viene usata come funzione di *callback* come per `retrlines()`; in modo predefinito, questa funzione stampa su `sys.stdout`. Il metodo restituisce `None`.

`rename`(*fromname, toname*)
Rinomina il file sul server *fromname* in *toname*.

`delete`(*filename*)
Rimuove il file *filename* dal server. Se ha successo, restituisce il testo della risposta, altrimenti solleva l'eccezione `error_perm` su errori di permessi o `error_reply` su altri errori.

`cwd`(*pathname*)
Imposta la directory corrente sul server.

`mkd`(*pathname*)
Crea una nuova directory sul server.

`pwd`()
Restituisce il nome della directory corrente sul sul server.

`rmd`(*dirname*)
Rimuove la directory *dirname* sul server.

`size`(*filename*)
Richiede la dimensione del file *filename* sul server. In caso di successo, viene restituita la dimensione del file come un intero, altrimenti viene restituito `None`. Notare che il comando '`SIZE`' non è standardizzato, ma viene supportato da molte comuni implementazioni di server.

`quit`()
Invia un comando '`QUIT`' al server e chiude la connessione. Questo è il modo corretto di chiudere una connessione, ma può sollevare un'eccezione nel caso che il server risponda con un errore al comando '`QUIT`'. Questo richiede una chiamata al metodo `close()` che rende l'istanza FTP inutile per chiamate successive (vedere sotto).

`close`()
Chiude la connessione unilateralmente. Questo non dovrebbe essere applicato ad una connessione già chiusa come in seguito ad una chiamata `quit()` eseguita con successo. Dopo questa chiamata, l'istanza `FTP` non dovrebbe più essere usata (dopo una chiamata a `close()` o `quit()` non si può riaprire una connessione richiamando il metodo `login()`).

11.8 `gopherlib` — Client per il protocollo Gopher

Questo modulo fornisce un'implementazione minima lato client del protocollo Gopher. Viene usato dal modulo `urllib` per gestire URL che utilizzano il protocollo Gopher.

Il modulo definisce le seguenti funzioni:

`send_selector`(*selector, host*[, *port*])
Invia una stringa *selector* al server gopher al nodo *host* e *port* (il valore predefinito per la porta è `70`). Restituisce un oggetto file aperto da cui il documento restituito può essere letto.

`send_query`(*selector, query, host*[, *port*])
Invia una stringa *selector* ed una stringa *query* al server gopher presso il nodo *host* e *port* (il valore pre-

definito per la porta è `70`). Restituisce un oggetto file aperto da cui il documento restituito può essere letto.

Notare che le informazioni restituite dal server Gopher possono essere di qualsiasi tipo, a seconda del primo carattere della stringa *selector*. Se l'informazione è testo (il primo carattere del *selector* è '`0`'), le singole righe vengono terminate da CRLF ed il dato viene terminato da una riga consistente in un singolo '`.`', ed un successivo '`.`' dovrà essere rimosso dalla riga che inizia con '`..`'. L'elenco delle directory (il primo carattere della stringa *selector* è '`1`') viene trasferito usando lo stesso protocollo.

11.9 `poplib` — Client per il protocollo POP3

Questo modulo definisce una classe, `POP3`, che incapsula una connessione ad un server POP3 ed implementa il protocollo come definito nell'RFC 1725. La classe `POP3` supporta sia l'insieme di comandi minimo che la parte facoltativa. Ulteriormente, questo modulo fornisce una classe `POP3_SSL`, che fornisce il supporto per connettersi a server POP3 che usano SSL come sottostante strato di protocollo.

Notare che POP3, nonostante sia largamente supportato, è obsoleto. La qualità di implementazione dei server POP3 varia molto, ed alcuni sono un po' scadenti. Se il vostro mailserver supporta IMAP, sarebbe meglio l'uso della classe `imaplib.IMAP4`, in quanto i server IMAP tendono a essere meglio implementati.

Una singola classe viene fornita dal modulo `poplib`:

class `POP3`(*host*[, *port*])
Questa classe implementa l'attuale protocollo POP3. La connessione viene creata quando viene creata l'istanza. Se la porta non viene indicata, allora viene usata la porta POP3 standard (`110`).

class `POP3_SSL`(*host*[, *port*[, *keyfile*[, *certfile*]]])
Questa è una sotto classe di `POP3` che si connette al server tramite un socket crittato con SSL. Se la porta *port* non viene specificata, 995, viene usata la porta standard POP3-over-SSL. *keyfile* e *certfile* sono anch'essi facoltativi – possono contenere una chiave privata formattata PEM e un file contenente certificati per la connessione SSL.

Nuovo nella versione 2.4.

Una eccezione viene definita come un attributo del modulo `poplib`:

exception `error_proto`
Eccezione sollevata per ogni errore. Il motivo che l'ha causata viene passato al costruttore come stringa.

Vedete anche:

Modulo `imaplib` (sezione 11.10):
Il modulo Python standard IMAP.

FAQ – Domande frequenti su Fetchmail
(http://www.catb.org/~esr/fetchmail/fetchmail-FAQ.html)
La FAQ per il client POP/IMAP **fetchmail** raccoglie informazioni sui vari server POP3 e le RFC per i server non standard (non conformi) che possono tornare utili se si ha la necessità di scrivere una applicazione basata sul protocollo POP.

11.9.1 Oggetti POP3

Tutti i comandi POP3 sono rappresentati da metodi con lo stesso nome, in lettere minuscole; la maggior parte di questi, restituisce il testo di risposta inviato dal server.

Un'instanza `POP3` possiede i seguenti metodi:

`set_debuglevel`(*level*)
Imposta il livello di debugging dell'istanza. Questo controlla la quantità di informazioni di debug stampate. Il valore predefinito, `0`, non produce output di debug. Un valore `1` produce un quantitativo moderato di informazioni di debug, generalmente una singola riga per richiesta. Un valore di `2` o maggiore produce il massimo quantitativo di informazioni di debugging, registrando ogni riga inviata e ricevuta sulla connessione di controllo.

getwelcome()
Restituisce la stringa di benvenuto inviata dal server POP3.

user(*username*)
Invia il comando user, la riposta dovrebbe indicare che viene richiesta la password.

pass_(*password*)
Invia la password, e nella risposta c'è il numero di messaggi e la dimensione della casella di posta. Notate: la mailbox sul server è bloccata fino a quando non viene chiamato il metodo `quit()`.

apop(*user, secret*)
Utilizza il più sicuro metodo di autenticazione APOP per connettersi al server POP3.

rpop(*user*)
Usa l'autenticazione RPOP (simile agli r-command di UNIX) per connettersi al server POP3.

stat()
Ottiene lo stato della mailbox. Il risultato è una tupla di due interi: (*numero dei messaggi, dimensione della mailbox*).

list([*which*])
Richiede l'elenco dei messaggi, il cui risultato è nella forma (*risposta*, `['messaggio_num ottale', ...]`). Se *which* viene impostato, è il singolo messaggio da elencare.

retr(*which*)
Recupera l'intero messaggio con numero *which*, ed imposta l'opzione visto (NdT: seen). Il risultato è nella forma (*risposta*, `['riga', ...]`, *ottale*).

dele(*which*)
Imposta il messaggio con numero *which* per la cancellazione. Sulla maggior parte dei server, la cancellazione non viene effettuata fino all'esecuzione del comando QUIT (la più eclatante eccezione è il QPOP di Eudora, che deliberatamente viola l'RFC facendo eseguire le cancellazioni ad ogni disconnessione).

rset()
Rimuove ogni opzione di cancellazione dalla mailbox.

noop()
Non fa niente. Dovrebbe essere usata come keep-alive.

quit()
Signoff: salva i cambiamente, sblocca la mailbox, interrompe la connessione.

top(*which, howmuch*)
Recupera l'intestazione del messaggio, più *howmuch* righe del messaggio dopo l'intestazione del messaggio numerato *which*. Il valore restituito è nella forma (*response*, `['line', ...]`, *octets*).

Il comando POP3 TOP che questo metodo usa, diversamente dal comando RETR, non imposta l'opzione seen del messaggio; sfortunatamente TOP è scarsamente trattato nelle RFC e frequentemente non è funzionante nei server non standard. È consigliabile verificare questo metodo sui server che si vorranno usare prima di utilizzarlo.

uidl([*which*])
Restituisce la lista dei messaggi in formato digest (id unico). Se *which* viene specificato, il risultato contiene l'id univoco per quel messaggio nella forma *'risposta mesgnum uid*, altrimenti il risultato è la lista (*risposta*, `['mesgnum uid', ...]`, *ottetti*).

Le istanze di `POP3_SSL` non hanno metodi aggiuntivi. L'interfaccia di questa sotto classe è identica al suo oggetto padre.

11.9.2 Esempi POP3

Qui un esempio minimo (senza controllo degli errori) che apre una mailbox e recupera e stampa tutti i messaggi:

```
import getpass, poplib

M = poplib.POP3('localhost')
M.user(getpass.getuser())
M.pass_(getpass.getpass())
numMessages = len(M.list()[1])
for i in range(numMessages):
    for j in M.retr(i+1)[1]:
        print j
```

Alla fine del modulo, c'è una sezione test che contiene un esempio più esteso del suo uso.

11.10 `imaplib` — Client per protocollo IMAP4

Questo modulo definisce tre classi, `IMAP4`, `IMAP4_SSL` e `IMAP4_stream`, che incapsulano una connessione ad un server IMAP4 ed implementano una larga parta del protocollo client IMAP4rev1 come definito nell'RFC 2060. È compatibile all'indietro con i server IMAP4 (RFC 1730), notare però che il comando '`STATUS`' non è supportato in IMAP4.

Il modulo `imaplib` fornisce tre classi, `IMAP4` è la classe di base:

class `IMAP4`(\[*host*\[, *port*\]\])
Questa classe implementa l'attuale protocollo IMAP4. La connessione viene creata e la versione del protocollo (IMAP4 o IMAP4rev1) viene determinata quando l'istanza viene inizializzata. Se *host* non viene specificato, allora viene utilizzato `''` (l'host locale o localhost). Se viene omessa la porta *port*, viene usata quella standard per l'IMAP4 (143).

Nella classe `IMAP4` vengono definite tre eccezioni come attributi.

exception `IMAP4.error`
Eccezione sollevata su ogni errore. Il motivo per cui è stata sollevata l'eccezione viene passato al costruttore come stringa.

exception `IMAP4.abort`
Errori del server IMAP4 causano il sollevamento di questa eccezione. Questa è una sotto classe di `IMAP4.error`. Notare che chiudendo l'istanza ed istanziandone una nuova è solitamente possibile recuperare da questa eccezione.

exception `IMAP4.readonly`
Questa eccezione viene sollevata quando ad una mailbox scrivibile viene cambiato il suo stato dal server. Questa è una sotto classe di `IMAP4.error`. Qualche altro client ha ora il permesso in scrittura, e la mailbox deve essere riaperta per ottenere nuovamente il permesso in scrittura.

C'é anche una sotto classe per le connessioni sicure:

class `IMAP4_SSL`(\[*host*\[, *port*\[, *keyfile*\[, *certfile*\]\]\]\])
Questa è una sotto classe derivata da `IMAP4` che si connette su un socket criptato SSL (per usare questa classe avete bisogno di un modulo socket compilato con il supporto SSL). Se *host* non viene specificato, allora viene utilizzato `''` (l'host locale o localhost). Se viene omessa la porta *port*, viene usata quella standard per IMAP4-over-SSL (993). Anche *keyfile* e *certfile* sono facoltativi – possono contenere una chiava privata formattata PEM ed il file della catena del certificato per la connessione SSL.

La seconda sotto classe permette connessioni create da un processo figlio:

class `IMAP4_stream`(*command*)
Questa è una sotto classe derivata da `IMAP4` che si connette ad un descrittore di file `stdin/stdout` creato passando *command* a `os.popen2()`. Nuovo nella versione 2.3.

Sono definite le seguenti funzioni di utilità:

`Internaldate2tuple`(*datestr*)

Converte la stringa IMAP4 INTERNALDATE in UTC (Tempo Universale Coordinato). Restituisce una tupla del modulo `time`.

`Int2AP`(*num*)
Converte un intero in una rappresentazione stringa usando caratteri dall'insieme [A .. P].

`ParseFlags`(*flagstr*)
Converte una risposta IMAP4 'FLAGS' in una tupla di opzioni individuali.

`Time2Internaldate`(*date_time*)
Converte una tupla del modulo `time` in una rappresentazione IMAP4 'INTERNALDATE'. Restituisce una stringa nella forma `DD-Mmm-YYYY HH:MM:SS +HHMM` (includendo le doppie virgolette).

Si noti che i numeri di messaggi IMAP4 cambiano al modificarsi della mailbox; in particolare, dopo un comando 'EXPUNGE' che esegue le cancellazioni, i rimanenti messaggi vengono rinumerati. Così è altamente preferibile usare gli UID, con il comando UID.

Alla fine del modulo, c'é una sezione di test che contiene una serie più estesa di esempi d'uso.

Vedete anche:

Documenti che descrivono il protocollo, sorgenti e binari per server che lo implementano, possono essere tutti trovati presso l'*IMAP Information Center* dell'Università di Washington (http://www.cac.washington.edu/imap/).

11.10.1 Oggetti IMAP4

Tutti i comandi IMAP4rev1 vengono rappresentati da metodi con lo stesso nome sia in maiuscolo che minuscolo.

Tutti gli argomenti per i comandi vengono convertiti in stringhe, eccetto che per 'AUTHENTICATE', e l'ultimo argomento per 'APPEND' che viene passato come letterale IMAP4. Se necessario (la stringa contiene caratteri sensibili per il protocollo IMAP4 e non è chiuso tra parentesi o doppie virgolette) ogni stringa viene quotata. Comunque, l'argomento *password* per il comando 'LOGIN' viene sempre quotato. Se si vuole evitare di avere una stringa argomento quotata (per esempio: l'argomento *flags* per 'STORE') includere la stringa tra parentesi (per esempio: `r'(\Deleted)'`).

Ogni comando restituisce una tupla `(`*type*`, [`*data*`, ...])` in cui *type* è abitualmente `'OK'` o `'NO'`, e *data* è, sia il testo della risposta del comando, che il risultato generato dal comando. Ogni *data* è, sia una stringa, che una tupla. Se è una tupla, la prima parte è l'intestazione della risposta, e la seconda parte contiene i dati (per esempio: valore 'literal').

Una istanza `IMAP4` ha i seguenti metodi:

`append`(*mailbox, flags, date_time, message*)
Aggiunge messaggi alla casella indicata.

`authenticate`(*func*)
Comando di autenticazione — richiede l'analisi della riposta. Questo è attualmente non implementato, e solleva un'eccezione.

`check`()
Verifica la mailbox sul server.

`close`()
Chiude la mailbox correntemente selezionata. I messaggi cancellati vengono rimossi dalla mailbox scrivibile. Questo è il comando raccomandato prima del 'LOGOUT'.

`copy`(*message_set, new_mailbox*)
Copia i messaggi *message_set* alla fine di *new_mailbox*.

`create`(*mailbox*)
Crea una nuova mailbox chiamata *mailbox*.

`delete`(*mailbox*)
Cancella la vecchia mailbox chiamata *mailbox*.

`expunge`()
Rimuove permanentemente gli elementi cancellati dalla mailbox selezionata. Genera una risposta

'EXPUNGE' per ogni messaggio cancellato. Il dato restituito contiene una lista di messaggi 'EXPUNGE' numerati nell'ordine di ricezione.

fetch(*message_set, message_parts*)
Scarica (una parte) dei messaggi. *message_parts* deve essere una stringa con il nome della parte del messaggio inclusa tra parentesi, per esempio: '(UID BODY[TEXT])'. I dati restituiti sono tuple di parte dell'intestazione della busta e dei dati del messaggio.

getacl(*mailbox*)
Ottiene le 'ACL' per la *mailbox*. Questo metodo non è standard, ma è supportato dai server 'Cyrus'.

getquota(*root*)
Ottiene il 'quota' di utilizzo della risorse di *root* ed i limiti d'uso. Questo metodo è parte delle estensioni IMAP4 QUOTA definite nell'rfc2087. Nuovo nella versione 2.3.

getquotaroot(*mailbox*)
Ottiene l'elenco di 'quota' dei 'root' per la *mailbox* indicata. Questo metodo è parte delle estensioni IMAP4 QUOTA definite nell'rfc2087. Nuovo nella versione 2.3.

list([*directory*[, *pattern*]])
Elenca i nomi delle mailbox nella *directory* che corrisponde a *pattern*. La *directory* predefinita è la cartella di più alto livello ed il valore predefinito di *pattern* è verifica qualsiasi cosa. Restituisce i dati contenenti una lista di risposte 'LIST'.

login(*user, password*)
Identifica il client utilizzando una password in testo piano. La *password* verrà quotata.

login_cram_md5(*user, password*)
Forza l'utilizzo dell'autenticazione 'CRAM-MD5' quando si identifica il cliente per proteggerne la password. Lavora solo se la risposta del server 'CAPABILITY' include la frase 'AUTH=CRAM-MD5'. Nuovo nella versione 2.3.

logout()
Chiude la connessione al server. Restituisce 'BYE' come risposta del server.

lsub([*directory*[, *pattern*]])
Elenca i nomi delle mailbox sottoscritte in *directory* corrispondenti a *pattern*. La directory predefinita è quella di livello massimo e il *pattern* predefinito corrisponde con tutte le mailbox. I dati restituiti sono tuple contenenti la parte dell'intestazione della busta del messaggio ed i dati del messaggio stesso.

noop()
Invia 'NOOP' al server.

open(*host, port*)
Apre un socket sulla porta *port* dell'*host*. L'oggetto connessione, stabilito con questo metodo, verrà utilizzato nei metodi read, readline, send e shutdown. Questo metodo può essere sovrascritto.

partial(*message_num, message_part, start, length*)
Scarica la parte troncata di un messaggio. Il dato restituito è una tupla della parte dell'intestazione della busta del messaggio e parte del corpo del messaggio stesso.

proxyauth(*user*)
Esegue l'autenticazione come utente *user*. Permette ad un amministratore autorizzato di collegarsi alla casella di un utente. Nuovo nella versione 2.3.

read(*size*)
Legge *size* bytes da un server remoto. Si può sovrascrivere questo metodo.

readline()
Legge una riga da un server remoto. Si può sovrascrivere questo metodo.

recent()
Invita il server per un aggiornamento. I dati restituiti sono None se non ci sono nuovi messaggi, altrimenti il valore della risposta 'RECENT'.

rename(*oldmailbox, newmailbox*)
Rinomina la mailbox chiamata *oldmailbox* in *newmailbox*.

response(*code*)
Restituisce i dati per il codice *code* di risposta quando ricevuto, altrimenti restituisce `None`. Restituisce il codice passato, al posto del tipo abituale.

search(*charset, criterion*[, ...])
Cerca la mailbox per il messaggio corrispondente. I dati restituiti contengono una lista separata da uno spazio dei numeri del messaggio rispondente. *charset* può essere `None`, nel cui caso nessun '`CHARSET`' verrà specificato nella richiesta al server. Il protocollo IMAP richiede che al massimo venga specificato un criterio; verrà sollevata un'eccezione quando il server restituisce un errore.

Esempio:

```
# M è un'istanza di IMAP4 connessa...
msgnums = M.search(None, 'FROM', '"LDJ"')

# o:
msgnums = M.search(None, '(FROM "LDJ")')
```

select([*mailbox*[, *readonly*]])
Seleziona una mailbox. Il dato restituito è il conteggio dei messaggi nella *mailbox* (risposta '`EXISTS`'). La *mailbox* predefinita è `'INBOX'`. Se l'opzione *readonly* viene impostata, non sono autorizzate modifiche alla mailbox.

send(*data*)
Invia i dati `data` al server remoto. Questo metodo può essere sovrascritto.

setacl(*mailbox, who, what*)
Imposta una '`ACL`' alla *mailbox*. Il metodo non è standard, ma è supportato dal server '`Cyrus`'.

setquota(*root, limits*)
Imposta il limite di '`quota`' *limits* delle risorse di *root*. Questo metodo fa parte delle estensioni IMAP4 QUOTA definite nell'rfc2087. Nuovo nella versione 2.3.

shutdown()
Chiude una connessione stabilita con `open`. Questo metodo può essere sovrascritto.

socket()
Restituisce l'istanza socket usata per la connessione al server.

sort(*sort_criteria, charset, search_criterion*[, ...])
Il comando `sort` è una variante di `search` con la semantica del sorting per il risultato. I dati restituiti contengono una lista separata da spazi dei numeri del messaggi corrispondenti.

Sort ha due argomenti prima dell'argomento/i *search_criterion*; una lista di *sort_criteria* tra parentesi ed il *charset* di ricerca. Notare che, diversamente da `search`, il parametro *charset* è obbligatorio. Esiste anche un comando `uid sort` che corrisponde a `sort`, alla stessa maniera di come `uid search` corrisponde a `search`. Il comando `sort`, prima cerca la mailbox per i messaggi che verificano il criterio di ricerca indicato usando l'argomento *charset* per l'interpretazione delle stringhe del criterio di ricerca. Restituisce il numero di messaggi corrispondenti.

Questo comando è previsto dall' estensione '`IMAP4rev1`'.

status(*mailbox, names*)
Richiede la condizione di stato named per la *mailbox*.

store(*message_set, command, flag_list*)
Altera le opzioni di disposizione per i messaggi nella mailbox.

subscribe(*mailbox*)
Sottoscrive una nuova mailbox.

thread(*threading_algorithm, charset, search_criterion*[, ...])
Il comando `thread` è una variante della ricerca `search` dei risultati, con semantica threading. I dati restituiti contengono una lista separata da spazi dei membri del thread.

I membri thread consistono in zero o più messaggi delimitati da spazi, che indicano successivi genitori e figli.

Thread possiede due argomenti prima dell'argomento/i *search_criterion*: un *threading_algorithm* ed il *charset* di ricerca. Notare che, diversamente da `search`, l'argomento di ricerca *charset* è obbligatorio. Esiste anche un comando `uid thread` corrispondente a `thread`, alla stessa maniera in cui `uid search` corrisponde a `search`. Il comando `thread` cerca per prima cosa le mailbox i cui messaggi trovino corrispondenza con il criterio di ricerca indicato usando l'argomento *charset* per l'interpretazione delle stringhe nel criterio di ricerca. Quindi restituisce i corrispondenti messaggi threaded in accordo con l'algoritmo di threading passato.

Questo è un comando di estensione '`IMAP4rev1`'. Nuovo nella versione 2.4.

`uid`(*command*, *arg*[, ...])
Esegue i comandi in argomento con i messaggi identificati da UID, diversamente dal numero di messaggio. Restituisce la risposta appropriata al comando. Al massimo può essere indicato un argomento; se non ne vengono forniti, il server restituirà un errore e verrà sollevata un'eccezione.

`unsubscribe`(*mailbox*)
Effettua la cancellazione di una sottoscrizione di una vecchia mailbox.

`xatom`(*name*[, *arg*[, ...]])
Consente semplici comandi di estensioni notificati dal server nella risposta '`CAPABILITY`'.

Le istanze di `IMAP4_SSL` hanno solamente un metodo aggiuntivo:

`ssl`()
Restituisce l'istanza SSLObject usata per le connessioni sicure al server.

I seguenti attributi vengono definiti sull'istanza di `IMAP4`:P4:

`PROTOCOL_VERSION`
Il protocollo più recente supportato nella risposta '`CAPABILITY`' del server.

`debug`
Valore intero per controllare la quantità di messaggi di debug. Il valore iniziale è preso dalla variabile `Debug` del modulo. Valori più grandi di 3 tracciano ogni comando.

11.10.2 Esempio IMAP4

Qui un esempio minimale (senza controllo degli errori) che apre una mailbox, recupera e stampa tutti i messaggi:

```
import getpass, imaplib

M = imaplib.IMAP4()
M.login(getpass.getuser(), getpass.getpass())
M.select()
typ, data = M.search(None, 'ALL')
for num in data[0].split():
    typ, data = M.fetch(num, '(RFC822)')
    print 'Message %s\n%s\n' % (num, data[0][1])
M.logout()
```

11.11 `nntplib` — Client per il protocollo NNTP

Il modulo definisce la classe `NNTP` che implementa la parte client del protocollo NNTP. Può essere usata per implementare un lettore o un generatore, o un processore automatico di news. Per ulteriori informazioni su NNTP (Network News Transfer Protocol), vedere la Internet RFC 977.

Qui ci sono due piccoli esempi di come può essere usato. Mostra alcune statistiche circa un newsgroup e stampa il soggetto degli ultimi 10 articoli.

```
>>> s = NNTP('news.cwi.nl')
>>> resp, count, first, last, name = s.group('comp.lang.python')
>>> print 'Group', name, 'has', count, 'articles, range', first, 'to', last
Group comp.lang.python has 59 articles, range 3742 to 3803
>>> resp, subs = s.xhdr('subject', first + '-' + last)
>>> for id, sub in subs[-10:]: print id, sub
...
3792 Re: Removing elements from a list while iterating...
3793 Re: Who likes Info files?
3794 Emacs and doc strings
3795 a few questions about the Mac implementation
3796 Re: executable python scripts
3797 Re: executable python scripts
3798 Re: a few questions about the Mac implementation
3799 Re: PROPOSAL: A Generic Python Object Interface for Python C Modules
3802 Re: executable python scripts
3803 Re: \POSIX{} wait and SIGCHLD
>>> s.quit()
'205 news.cwi.nl closing connection.  Goodbye.'
```

Per postare un articolo da un file (questo assume che l'articolo possieda intestazioni valide):

```
>>> s = NNTP('news.cwi.nl')
>>> f = open('/tmp/article')
>>> s.post(f)
'240 Article posted successfully.'
>>> s.quit()
'205 news.cwi.nl closing connection.  Goodbye.'
```

Il modulo definisce i seguenti elementi:

class `NNTP`(*host*[, *port* [, *user*[, *password* [, *readermode*]]]])
Restituisce una nuova istanza della classe NNTP, rappresentando una connessione al server NNTP eseguito sull'host *host*, in ascolto sulla porta *port*. Il valore predefinito di *port* è 119. Se vengono forniti i facoltativi *user* e *password*, o se sono presenti valide credenziali in ' /.netrc', i comandi '`AUTHINFO USER`' ed '`AUTHINFO PASS`' vengono usati per identificare ed autenticare l'utente al server. Se l'opzione facoltativa *readermode* è vera, viene inviato un comando '`mode reader`' prima che sia effettuata l'autenticazione. Reader mode è necessario occasionalmente se vi state connettendo ad un server NNTP sulla macchina locale ed intendete chiamare comandi specifici del lettore, come '`group`'. Se ottenete un inaspettato `NNTPPermanentError`, è possibile che dobbiate impostare *readermode*. Il valore predefinito di *readermode* è `None`.

class `NNTPError`()
Derivata dall'eccezione standard `Exception`, questa è la classe di base per tutte le eccezioni sollevate dal modulo `nntplib`.

class `NNTPReplyError`()
Eccezione sollevata quando viene ricevuta una risposta inaspettata dal server. Per compatibilità con il passato, l'eccezione `error_reply` è equivalente a questa classe.

class `NNTPTemporaryError`()
Eccezione sollevata quando viene ricevuto un codice di errore compreso nell'intervallo 400-499. Per compatibilità con il passato, l'eccezione `error_temp` è equivalente a questa classe.

class `NNTPPermanentError`()
Eccezione sollevata quando viene ricevuto un codice di errore compreso nell'intervallo 500-599. Per compatibilità all'indietro, l'eccezione `error_perm` è equivalente a questa classe.

class `NNTPProtocolError`()
Eccezione sollevata quando viene ricevuta una risposta dal server che non inizia con un numero compreso

nell'intervallo 1-5. Per compatibilità con il passato, l'eccezione `error_proto` è equivalente a questa classe.

class `NNTPDataError`()
Eccezione sollevata quando c'è qualche errore nei dati di risposta. Per compatibilità con il passato, l'eccezione `error_data` è equivalente a questa classe.

11.11.1 Oggetti NNTP

Le istanze NNTP possiedono i seguenti metodi. Le risposte *response* che vengono restituite come primo elemento nella tupla restituita da quasi tutti i metodi sono le risposte del server: una stringa che inizia con un codice di tre valori numerici. Se la risposta del server indica un errore, il metodo solleva una delle eccezioni precedenti.

`getwelcome`()
Restituisce un messaggio di benvenuto, inviato dal server in risposta alla connessione iniziale. Questo messaggio, qualche volta, contiene un avviso o delle informazioni di aiuto che possono essere rilevanti per l'utente.

`set_debuglevel`(*level*)
Imposta il livello di debugging dell'istanza. Questo controlla l'ammontare delle informazioni di debug stampate. Il valore predefinito, `0`, non produce alcun output. Un valore di `1` produce un moderato quantitativo di messaggi di debugging, generalmente una riga singola per richiesta o risposta. Un valore di `2` o maggiore, produce il massimo ammontare di messaggi di debugging, registrando ogni riga inviata e ricevuta sulla connessione (incluso il testo del messaggio).

`newgroups`(*date, time,* [*file*])
Invia un comando '`NEWGROUPS`'. L'argomento *date* deve essere un stringa nel formato `'`*yymmdd*`'`, indicante la data, e *time* dovrebbe essere una stringa nella forma `'`*hhmmss*`'`, indicante l'orario. Restituisce una coppia (*risposta*, *gruppi*) dove *gruppi* è una lista di nomi dei gruppi che risultano nuovi rispetto all'orario ed alla data indicate. Se il *file* dei parametri viene fornito, il risultato del comando '`NEWGROUPS`' viene memorizzato in un file. Se *file* è una stringa, il metodo aprirà un oggetto file con quel nome, ci scriverà sopra e lo chiuderà. Se *file* è un oggetto file, inizierà chiamando`write()` su di esso per memorizzare le righe del risultato del comando. Se *file* viene fornito, la *lista* restituita sarà una lista vuota.

`newnews`(*group, date, time,* [*file*])
Invia un comando '`NEWNEWS`'. Qui *group* è un nome di gruppo o `'*'` e *date* e *time* hanno lo stesso significato come per `newgroups()`. Restituisce una coppia (*risposta*, *articoli*) dove *articoli* è una lista di id degli articoli. Se il parametro *file* viene indicato, il risultato del comando '`NEWNEWS`' viene memorizzato in un file. Se *file* è una stringa, il metodo aprirà un file oggetto con quel nome, vi scriverà sopra e lo chiuderà. Se *file* è un oggetto file, il metodo inizierà chiamando `write()` su di esso per memorizzare le righe del risultato del comando. Se *file* viene indicato, la *list* restituita sarà una lista vuota.

`list`([*file*])
Invia un comando '`LIST`'. Restituisce una coppia (*risposta*, *lista*) dove *lista* è una lista di tuple. Ogni tupla ha la forma (*gruppo*, *ultimo*, *primo*, *opzione*), dove *gruppo* è un nome di gruppo, *ultimo* e *primo* sono l'ultimo ed il primo numero dell'articolo (come stringa) ed *opzione* è `y` se il posting è autorizzato, `n` se no, e `m` se il newsgroup è moderato. Notare l'ordine: *ultimo*, *primo*. Se il parametro *file* viene indicato, il risultato del comando '`LIST`' viene inserito in un file. Se il file è una stringa, il metodo aprirà un file oggetto con quel nome, scriverà e quindi lo chiuderà. Se *file* è un oggetto file, allora verrà chiamato il metodo `write()` per memorizzare le righe del risultato del comando. Se *file* viene indicato, la lista *list*, restituita è una lista vuota.

`group`(*name*)
Invia un comando '`GROUP`', dove *name* è il nome del gruppo. Restituisce una tupla (*risposta*, *conteggio*, *primo*, *ultimo*, *nome*) dove *conteggio* è il numero (stimato) di articoli nel gruppo, *primo* è il primo numero dell'articolo nel gruppo, *ultimo* è l'ultimo numero dell'articolo nel gruppo, e *nome* è il nome del gruppo. I numeri vengono restituiti come stringhe.

`help`([*file*])
Invia un comando '`HELP`'. Restituisce una coppia (*risposta*, *lista*) dove *lista* è la lista di stringhe di aiuto. Se il parametro *file* viene indicato, il risultato del comando '`HELP`' viene memorizzato in un file. Se *file* è

una stringa, il metodo aprirà un oggetto file con quel nome, vi scriverà e quindi lo chiuderà. Se *file* è un oggetto file, il metodo chiamerà `write()` per memorizzare le righe del risultato del comando. Se *file* viene indicato, la lista restituita è una lista vuota.

`stat`(*id*)
Invia un comando '`STAT`', dove *id* è l'id del messaggio (racchiuso tra '`<`' e '`>`') o il numero di un articolo (come stringa). Restituisce una tripla (*risposta, numero, id*) dove *numero* è il numero dell'articolo (come stringa) ed *id* è l'id dell'articolo (racchiuso tra '`<`' e '`>`').

`next`()
Invia un comando '`NEXT`'. Restituisce gli stessi valori di `stat()`.

`last`()
Invia un comando '`LAST`'. Restituisce gli stessi valori di `stat()`.

`head`(*id*)
Invia un comando '`HEAD`', dove *id* possiede lo stesso significato che in `stat()`. Restituisce una tupla (*risposta, numero, id, lista*) dove i primi tre sono gli stessi di `stat()`, e *lista* è una lista di intestazioni di articolo (una lista non interpretata di righe, senza il carattere di fine riga).

`body`(*id*,[*file*])
Invia un comando '`BODY`', dove *id* possiede lo stesso significato di `stat()`. Se il parametro *file* viene indicato, allora il corpo del messaggio restituito viene memorizzato in un file. Se *file* è una stringa, allora il metodo aprirà un oggetto file con quel nome, vi scriverà e lo chiuderà. Se *file* è un oggetto file, verrà chiamato `write()` per memorizzare le righe del corpo del testo. Restituisce gli stessi valori di `head()`. Se *file* viene indicato, la lista restituita è una lista vuota.

`article`(*id*)
Invia un comando '`ARTICLE`', dove *id* ha lo stesso significato di `stat()`. Restituisce gli stessi parametri di `head()`.

`slave`()
Invia un comando '`SLAVE`'. Restituisce la risposta del server *response*.

`xhdr`(*header, string,* [*file*])
Invia un comando '`XHDR`'. Questo comando non viene definito nelle rfc ma è un'estensione comune. L'argomento *header* è una chiave di intestazione, per esempio `'subject'`. L'argomento *string* deve possedere la forma `'`*primo*`-`*ultimo*`'` dove *primo* e *ultimo* sono il primo e l'ultimo numero dell'articolo da cercare. Restituisce una coppia (*risposta, lista*), dove *list* è una lista di coppie (*id, testo*), dove *id* è l'id di un'articolo (come una stringa) e *test* è il testo dell'intestazione richiesta per quell'articolo. Se il parametro *file* viene indicato, il risultato del comando '`XHDR`' viene memorizzato in un file. Se *file* è una stringa, il metodo aprirà un oggetto file con quel nome, vi scriverà e lo chiuderà. Se *file* è un oggetto file, si inizierà chiamando il metodo `write()` per memorizzare le righe del risultato del comando. Se *file* viene indicato, la lista restituita è una lista vuota.

`post`(*file*)
Pubblica un articolo usando il comando '`POST`'. L'argomento *file* è un oggetto file aperto che viene letto fino all'EOF con il proprio metodo `readline()`. Dovrebbe essere un articolo di news correttamente costruito, incluse le intestazioni richieste. Il metodo `post()` inserisce automaticamente l'escape per le linee che iniziano con '`.`'.

`ihave`(*id, file*)
Invia un comando '`IHAVE`'. Se la riposta non è un errore, tratta *file* esattamente come il metodo `post()`.

`date`()
Restituisce una tripla (*risposta, data, orario*), contenente la data e l'ora corrente in un formato adatto ai metodi `newnews()` e `newgroups()`. Questa è un'estensione facoltativa NNTP e potrebbe non essere supportata da tutti i server.

`xgtitle`(*name,* [*file*])
Processa un comando '`XGTITLE`', restituendo una coppia (*risposta, lista*), dove *lista* è una lista di tuple contenenti (*nome, titolo*). Se il parametro *file* viene indicato, il risultato del comando '`XGTITLE`' viene memorizzato in un file. Se *file* è una stringa, il metodo aprirà un oggetto file con quel nome, vi scriverà e lo chiuderà. Se *file* è un oggetto file, inizierà chiamando `write()` sull'oggetto stesso per registrare le righe

del risultato del comando. Se *file* viene indicato, la lista restituita è una *lista* vuota. Questa è un'estensione NNTP facoltativa, e potrebbe non essere supportata da tutti i server.

xover(*start, end,* [*file*])
Restituisce una coppia (*risposta, lista*). *lista* è una lista di tuple, una per ogni articolo nell'ambito delimitato dagli intervalli del numero corrispondente all'articolo con *start* ed *end*. Ogni tupla è nel formato (*article numero, soggetto, poster, data, id, riferimenti, dimensione, righe*). Se il parametro *file* viene indicato, il risultato del comando 'XOVER' viene memorizzato in un file. Se *file* è una stringa, il metodo aprirà un oggetto file con quel nome, vi scriverà sopra e quindi lo chiuderà. Se *file* è un oggetto file, si inizierà chiamando il metodo `write()` per memorizzare le righe del risultato del comando. Se *file* viene indicato, la lista restituita sarà una *lista* vuota. Questa è un'estensione NNTP facoltativa, e potrebbe non essere supportata da tutti i server.

xpath(*id*)
Restituisce una coppia (*risposta, percorso*), dove *percorso* è la directory dell'articolo indicato dall'ID *id*. Questa è un'estensione NNTP facoltativa, e potrebbe non essere supportata da tutti i server.

quit()
Invia un comando 'QUIT' e chiude la connessione. Una volta che questo metodo è stato chiamato, non dovrebbe essere chiamato nessun'altro metodo sull'oggetto NNTP.

11.12 smtplib — Client per il protocollo SMTP

Il modulo `smtplib` definisce l'oggetto di una sessione smtp (lato Client) che può essere usata per inviare mail ad ogni macchina internet con demone SMTP o ESMTP in ascolto. Per i dettagli delle operazioni SMTP o ESMPT, consultare l'RFC 821 (*Simple Mail Transfer Protocol*) e la RFC 1869 (*SMTP Service Extensions*).

class SMTP([*host*[, *port*[, *local_hostname*]]])
Un'istanza `SMTP` che incapsula una connessione SMTP. Possiede metodi che supportano un gran repertorio di operazioni SMTP ed ESMTP. Se i parametri facoltativi *host* e *port* vengono indicati, il metodo SMTP `connect()` viene chiamato durante l'inizializzazione con questi parametri. Viene sollevata un'eccezione `SMTPConnectError` se l'host specificato non risponde correttamente.

Per un uso normale, vengono richiesti solamente i metodi di inizializzazione/connessione, `sendmail()` e `quit()`. Un esempio viene proposto di seguito.

Segue una selezione di eccezioni:

exception SMTPException
Classe dell'eccezione di base per tutte le eccezioni sollevate da questo modulo.

exception SMTPServerDisconnected
L'eccezione viene sollevata quando il server si disconnette in modo inaspettato, o quando viene fatto un tentativo di usare un'istanza `SMTP` prima che ci si connetta al server.

exception SMTPResponseException
Classe di base per tutte le eccezioni che includono un codice di errore SMTP. Queste eccezioni vengono generate in alcune istanze quando il server SMTP restituisce un codice di errore. Il codice di errore viene memorizzato nell'attributo dell'errore `smtp_code`, e l'attributo `smtp_error` viene impostato con il messaggio di errore.

exception SMTPSenderRefused
L'indirizzo del Sender (NdT: mittente) è stato rifiutato. In aggiunta agli attributi impostati da tutte le eccezioni `SMTPResponseException`, questa imposta 'sender' alla stringa che il server SMTP ha rifiutato.

exception SMTPRecipientsRefused
Tutti gli indirizzi dei destinatari sono stati rifiutati. Gli errori per ogni destinatario sono accessibili attraverso l'attributo `recipients`, che è un dizionario esattamente dello stesso tipo di quello restituito da `SMTP.sendmail()`.

exception SMTPDataError
Il server SMTP rifiuta di accettare il contenuto del messaggio.

exception `SMTPConnectError`
Un errore avviene durante l'attivazione di una connessione con il server.

exception `SMTPHeloError`
Il server rifiuta il nostro messaggio '`HELO`'.

Vedete anche:

RFC 821, "*Simple Mail Transfer Protocol*"
Definizione del protocollo SMTP. Questo documento riguarda il modello, le procedure operative ed i dettagli del protocollo SMTP.

RFC 1869, "*SMTP Service Extensions*"
Definisce le estensioni ESMTP per l'SMTP. Questo descrive la struttura per estendere l'SMTP con nuovi comandi, supportando la scoperta dinamica dei comandi forniti dal server, e definisce una serie di comandi addizionali.

11.12.1 Oggetti SMTP

Un'istanza `SMTP` possiede i seguenti metodi:

`set_debuglevel`(*level*)
Imposta il livello di informazioni di debug. Un valore reale per *level* riporta tutte le informazioni di connessione per tutti i messaggi inviati e ricevuti dal server.

`connect`([*host*[, *port*]])
Si connette ad un *host* su di una porta *port* indicata. Predefinitamente si connette all'host locale sulla porta standard SMTP (25). Se l'hostname termina con i due punti ('`:`') seguiti da un numero, questo suffisso verrà estrapolato ed il numero interpretato come la porta da utilizzare. Questo metodo viene automaticamente invocato dal costruttore se un host viene specificato durante la creazione dell'istanza.

`docmd`(*cmd*, [, *argstring*])
Invia un comando *cmd* al server. L'argomento opzionale *argstring* è semplicemente concatenato al comando, separato da uno spazio.

Restituisce una doppia tupla composta da un codice di risposta numero e una riga della risposta attuale (risposte multiriga sono sommate in una singola lunga riga).

Nelle normali operazioni non dovrebbe essere necessario chiamare questo metodo esplicitamente. Viene usato per implementare altri metodi e può tornare utile per testare estensioni private.

Se viene persa la connessione al server durante l'attesa della risposta viene sollevata l'eccezione `SMTPServerDisconnected`.

`helo`([*hostname*])
Identifica sé stesso al server SMTP utilizzando '`HELO`'. L'argomento predefinito dell'hostname è il nome di dominio completo e pienamente qualificato dell'host locale.

Nelle normali operazioni non dovrebbe essere necessario chiamare questo metodo esplicitamente. Viene implicitamente chiamato da `sendmail()` quando necessario.

`ehlo`([*hostname*])
Identifica sé stesso ad un server ESMTP utilizzando '`EHLO`'. L'argomento predefinito dell'hostname è il nome di dominio completo e qualificato dell'host locale. Esamina la risposta per le opzioni ESMTP e le memorizza per utilizzarle poi con `has_extn()`.

Finché si desidera usare `has_extn()` prima di inviare mail, non dovrebbe essere necessario usare questo metodo esplicitamente. Verrà chiamato implicitamente da `sendmail()` quando necessario.

`has_extn`(*name*)
Restituisce `True` se *name* è nell'insieme delle estensioni del servizio SMTP restituito dal server, altrimenti `False`. Le differenze tra maiuscole e minuscole verranno ignorate.

`verify`(*address*)
Verifica la validità dell'indirizzo sul server usando il comando SMTP '`VRFY`'. Restituisce una tupla consistente nel codice 250 ed un indirizzo perfettamente rispondente all'RFC 822 (compreso il nome umano)

quando l'indirizzo dell'utente è valido. Altrimenti restituisce un codice di errore SMTP 400 o maggiore, ed una stringa di errore.

Note: Molti siti disabilitano il comando SMTP 'VRFY' per evitarne l'utilizzo da parte degli spammers.

login(*user, password*)
Accede ad un server SMTP che richiede l'autenticazione. Gli argomenti sono username e password con cui autenticarsi. Se in questa sessione, precedentemente, non c'è stato un comando 'EHLO' o 'HELO', il metodo prova prima il comando ESMTP 'EHLO'. Questo metodo abitualmente terminerà correttamente se l'autenticazione ha avuto successo, o solleverà le seguenti eccezioni:

SMTPHeloError Il server non risponde adeguatamente all'invito 'HELO'.

SMTPAuthenticationError Il server non accetta la combinazione nome-utente/password.

SMTPError Non sono stati trovati metodi di autenticazione accettabili.

starttls([*keyfile*[, *certfile*]])
Mette la connessione SMTP in modalità TLS (Transport Layer Security). Tutti i comandi SMTP che seguono saranno criptati. Dovrete chiamare `ehlo()` nuovamente.

Se vengono forniti *keyfile* e *certfile*, questi vengono passati alla funzione `ssl()` del modulo `socket`.

sendmail(*from_addr, to_addrs, msg*[, *mail_options, rcpt_options*])
Invia la mail. Gli argomenti richiesti sono una stringa RFC 822 from-address, una lista di indirizzi RFC 822 to-address, ed una stringa col messaggio. Il chiamante può passare una lista di opzioni ESMTP (come '8bitmime') da usare nel comando 'MAIL FROM' come *mail_options*. Le opzioni ESMPT (come il comando 'DSN') che dovrebbero essere usate con tutti i comandi 'RCPT' possono essere passate come *rcpt_options*. Se avete bisogno di usare opzioni ESMTP differenti per ciascun destinatario, dovete usare un metodo di più basso livello come `mail`, `rcpt` e `data` per inviare il messaggio.

Note: I parametri *from_addr* e *to_addrs* vengono usati per costruire l'intestazione del messaggio usata dall'agente di trasporto. La classe `SMTP` non modifica le intestazioni del messaggio in nessun modo.

Se in questa sessione non ci sono stati precedenti comandi 'EHLO' o 'HELO', questo metodo prova per prima cosa il comando ESMTP 'EHLO'. Se il server supporta ESMTP, la dimensione del messaggio ed ognuna delle opzioni specificate gli verranno passate (se l'opzione è nell'insieme di particolarità specificate dalle indicazioni inviate dal server stesso). Se 'EHLO' fallisce, verrà tentato 'HELO' ed il supporto ESMTP verrà soppresso.

Questo metodo terminerà regolarmente se la mail viene accettata per almeno un destinatario. Altrimenti solleverà un'eccezione. In parole povere, se non c'è un'eccezione, qualcuno riceverà la mail. Se questo metodo non solleva un'eccezione, restituirà un dizionario, con una riga per ogni destinatario rifiutato. Ogni riga contiene una tupla del codice di errore SMTP ed il messaggio di errore corrispondente inviato dal server.

Questo metodo può sollevare le seguenti eccezioni:

SMTPRecipientsRefused Tutti i destinatari sono stati rifiutati. Nessuno riceverà mail. L'attributo `recipients` dell'oggetto dell'eccezione è un dizionario contenente le informazioni circa i destinatari rifiutati (come quello restituito quando almeno uno dei destinatari è stato accettato).

SMTPHeloError Il server non risponde propriamente al comando 'HELO'.

SMTPSenderRefused Il server non accetta *from_addr*.

SMTPDataError Il server risponde con un codice di errore inatteso (diverso da quello di un destinatario rifiutato).

Diversamente da quanto altrimenti fatto notare, la connessione resterà aperta dopo che un'eccezione è stata sollevata.

quit()
Termina la sessione SMTP e chiude la connessione.

Vengono anche supportati i metodi di basso livello che corrispondono ai comandi standard SMTP/ESMTP 'HELP', 'RSET', 'NOOP', 'MAIL', 'RCPT' e 'DATA'. In condizioni normali non c'è bisogno di chiamare questi metodi direttamente, e quindi non sono qui documentati; consultare il codice del modulo.

11.12.2 Esempi SMTP

Questo esempio chiede all'utente gli indirizzi necessari nell'intestazione del messaggio (indirizzi 'To' e 'From') ed il messaggio da consegnare. Notare che le intestazioni che devono essere incluse nel messaggio devono essere inserite durante l'inserimento del messaggio; questo esempio non esegue alcun tipo di processo sulle intestazioni RFC 822. In particolare, gli indirizzi 'To' e 'From' devono essere inclusi esplicitamente nelle intestazioni del messaggio.

```
import smtplib

def prompt(prompt):
    return raw_input(prompt).strip()

fromaddr = prompt("From: ")
toaddrs  = prompt("To: ").split()
print "Inserire un messaggio, terminarlo con ^D (Unix) o ^Z (Windows):"

# Aggione all'inizio gli header From e To!
msg = ("From: %s\r\nTo: %s\r\n\r\n"
       % (fromaddr, ", ".join(toaddrs)))
while 1:
    try:
        line = raw_input()
    except EOFError:
        break
    if not line:
        break
    msg = msg + line

print "La lunghezza del messaggio risulta essere: " + repr(len(msg))

server = smtplib.SMTP('localhost')
server.set_debuglevel(1)
server.sendmail(fromaddr, toaddrs, msg)
server.quit()
```

11.13 `telnetlib` — Client Telnet

Il modulo `telnetlib` fornisce una classe `Telnet` che implementa il protocollo Telnet. Fare riferimento all'RFC 854 per i dettagli del protocollo. In aggiunta, supporta costanti simboliche per i caratteri del protocollo (vedere successivamente), e per le opzioni di telnet. I nomi simbolici delle opzioni telnet seguono la definizione riportata in `arpa/telnet.h`, con il carattere vuoto `TELOPT_` rimosso. Per i nomi simbolici delle opzioni che non vengono tradizionalmente incluse in `arpa/telnet.h`, vedere direttamente i sorgenti del modulo.

Le costanti simboliche per i comandi telnet sono: IAC, DONT, DO, WONT, WILL, SE (Subnegotiation End, fine della sottonegoziazione), NOP (No Operation, nessuna operazione), DM (Data Mark, marcatore dati), BRK (Break, interruzione), IP (Interrupt process, processo di interruzione), AO (Abort output, annulla emissione), AYT (Are You There, c'è qualcuno di là!), EC (Erase Character, cancella carattere), EL (Erase Line, cancella riga), GA (Go Ahead, và avanti), SB (Subnegotiation Begin, inizio della sottonegoziazione).

class `Telnet`([*host*[, *port*]])

`Telnet` rappresenta una connessione ad un server Telnet. L'istanza non è inizialmente connessa in modo predefinito; il metodo `open()` deve essere usato per stabilire una connessione. Alternativamente, il nome dell'host e facoltativamente il numero della porta (port) possono essere passati al costruttore, nel cui caso, la connessione al server verrà stabilita prima che il costruttore termini il suo compito.

Non si deve riaprire un'istanza già connessa.

Questa classe dispone di parecchi metodi `read_*()`. Fare attenzione che alcuni di questi sollevano ec-

cezioni `EOFError` quando viene letta la fine della connessione, in quanto possono restituire una stringa vuota in altre occasioni. Vedere le descrizioni individuali riportate di seguito.

Vedete anche:

RFC 854, "*Specifiche per il protocollo Telnet*"
Definizioni del protocollo telnet.

11.13.1 Oggetti Telnet

Le istanze `Telnet` hanno i seguenti metodi:

`read_until`(*expected*[, *timeout*])
Legge fino a che una data stringa, *expected*, viene incontrata, o fino a che non sono trascorsi i secondi relativi al *timeout*.

Quando non viene trovata alcuna corrisponda, restituisce al suo posto qualsiasi cosa disponibile, possibilmente la stringa vuota. Solleva l'eccezione `EOFError` se la connessione è chiusa e non ci sono dati elaborati disponibili.

`read_all`()
Legge tutti i dati fino all'EOF; resta in attesa fino alla chiusura della connessione.

`read_some`()
Legge come minimo un byte di dati elaborati fino al raggiungimento dell'EOF. Restituisce `''` se viene trovato un EOF. Blocca se non ci sono dati immediatamente disponibili.

`read_very_eager`()
Legge qualsiasi cosa che ci sia senza il blocco dell' I/O (zelante).

Solleva l'eccezione `EOFError` se la connessione viene chiusa e non ci sono dati elaborati disponibili. Altrimenti restituisce `''` se non vi sono dati disponibili. Non blocca finché è nel mezzo di una sequenza IAC.

`read_eager`()
Legge tutti i dati disponibili e leggibili.

Solleva l'eccezione `EOFError` se la connessione è chiusa e non ci sono dati elaborati disponibili. Altrimenti restituisce `''` se non ci sono dati elaborati disponibili. Non blocca finché è nel mezzo di una sequenza IAC.

`read_lazy`()
Processa e restituisce i dati già nella coda (pigro).

Solleva l'eccezione `EOFError` se la connessione è chiusa e non ci sono dati disponibili. Altrimenti restituisce `''` se non ci sono dati elaborati. Non blocca finché è nel mezzo di una sequenza IAC.

`read_very_lazy`()
Restituisce ogni dato disponibile nella coda di elaborazione (molto pigro).

Solleva l'eccezione `EOFError` se la connessione è chiusa e non ci sono dati disponibili. Restituisce `''` se altrimenti non ci sono dati elaborati disponibili. Questo metodo non è mai bloccante.

`read_sb_data`()
Restituisce i dati raccolti attraverso una coppia SB/SE (sotto opzione Inizio/Fine). La chiamata dovrebbe accedere a questi dati quando viene invocata attraverso un comando `SE`. Questo metodo non è mai bloccante.

Nuovo nella versione 2.3.

`open`(*host*[, *port*])
Si connette all'host. Il secondo argomento facoltativo è il numero della porta, che ha il valore predefinito della porta standard Telnet (23).

Non tentare di riaprire un'istanza già connessa.

`msg`(*msg*[, **args*])
Stampa un messaggio di debug quando il livello di debug è > di 0. Se sono presenti argomenti extra, vengono sostituiti nel messaggio utilizzando l'operatore standard di formattazione delle stringhe.

set_debuglevel(*debuglevel*)
Imposta il livello di debug. Maggiore è il valore di *debuglevel* e maggiore sarà il numero di informazioni di debug ottenute (su `sys.stdout`).

close()
Chiude la connessione.

get_socket()
Restituisce l'oggetto socket utilizzato internamente.

fileno()
Restituisce il descrittore di file dell'oggetto socket utilizzato internamente.

write(*buffer*)
Scrive una stringa nel socket, raddoppiando ogni carattere IAC. Questo sistema si può bloccare se la connessione è bloccata. Può sollevare l'eccezione `socket.error` se la connessione è chiusa.

interact()
Funzione di interazione. Emula un vero client Telnet stupido.

mt_interact()
Versione multithread di `interact()`.

expect(*list*[, *timeout*])
Legge finché c'è una corrispondenza con una delle espressioni regolari nella lista *list*.

Il primo argomento è una lista di espressioni regolari, sia compilate (istanza `re.RegexObject`) che non compilate (stringhe). Il secondo argomento, facoltativo, è un timeout in secondi; il suo comportamento predefinito è bloccare indefinitamente.

Restituisce una tupla di tre elementi: l'indice nella lista della prima espressione regolare che corrisponde; l'oggetto corrispondente restituito; il testo letto fino alla corrispondenza da verificare, inclusa la parte corrispondente.

Se viene trovata la fine del file e non è stato letto del testo, viene sollevata l'eccezione `EOFError`. Altrimenti, quando non corrisponde niente, restituisce (`(-1, None,` *text*`)`) dove *text* è il testo ricevuto (che potrebbe essere una stringa vuota se è intervenuto un timeout).

Se un'espressione regolare termina con una elemento di corrispondenza multipla (come un `.*`) o più di un'espressione che possa avere corrispondenza con lo stesso input, i risultati sono indeterminati e possono dipendere dalle temporizzazioni I/O.

set_option_negotiation_callback(*callback*)
Ogni volta che un'opzione telnet viene letta nel flusso in arrivo, questo *callback* (se impostato) viene chiamato con i seguenti parametri: callback(telnet socket, comando (DO/DONT/WILL/WONT), opzione). Nessun'altra azione viene accettata da telnetlib successivamente.

11.13.2 Esempi di Telnet

Un semplice esempio che ne illustra un uso tipico:

```
import getpass
import sys
import telnetlib

HOST = "localhost"
user = raw_input("Inserisci il tuo account remoto: ")
password = getpass.getpass()

tn = telnetlib.Telnet(HOST)

tn.read_until("login: ")
tn.write(user + "\n")
if password:
    tn.read_until("Password: ")
    tn.write(password + "\n")

tn.write("ls\n")
tn.write("exit\n")

print tn.read_all()
```

11.14 `urlparse` — Analizza le URL nei suoi componenti

Questo modulo definisce un'interfaccia standard per suddividere le Uniform Resource Locator (URL) in componenti (schema di indirizzamento, indirizzo di rete, percorso etc. etc.), per ricombinare i componenti in una stringa URL, e per convertire una "URL relativa" in una URL assoluta indicando una "URL di base".

Il modulo è stato progettato per essere rispondente alle RFC Internet sulle Relative Uniform Resource Locators (e scoprire un bug in una versione recente).

Definisce le seguenti funzioni:

`urlparse`(*urlstring*[, *default_scheme*[, *allow_fragments*]])
: Analizza una URL in 6 componenti, restituendo una 6-tupla: (schema di indirizzamento, indirizzo di rete, percorso, parametri, richiesta, identificatore di frammento). Questo corrisponde alla struttura generale di una URL: *scheme*`://`*netloc*`/`*path*`;`*parameters*`?`*query*`#`*fragment*. Ogni elemento della tupla è una stringa possibilimente vuota. Il componente non viene suddiviso in parti più piccole (per esempio l'indirizzo di rete è una singola stringa) ed i caratteri di escape % non vengono espansi. I delimitatori, come mostrato sopra, non sono parte degli elementi della tupla, eccetto per un carattere barra nel componente dell'indirizzo *path*, che, se presente, viene mantenuto.

  Esempio:

  ```
  urlparse('http://www.cwi.nl:80/%7Eguido/Python.html')
  ```

  contiene la tupla

  ```
  ('http', 'www.cwi.nl:80', '/%7Eguido/Python.html', '', '', '')
  ```

  Se l'argomento *default_scheme* viene specificato, fornisce lo schema di indirizzamento predefinito, per essere usato solamente se la stringa URL non ne specifica una. Il valore predefinito per questo argomento è una stringa vuota.

  Se l'argomento *allow_fragments* ha valore `0`, gli identificatori di frammento non sono permessi, anche se lo schema di indirizzamento dell'URL normalmente li supporta. Il valore predefinito per questo argomento è `1`.

`urlunparse`(*tuple*)
: Costruisce una stringa URL da una tupla come quella restituita da u`urlparse()`. Il risultato potrebbe

essere lievemente differente, ma equivalente, se le URL che erano state originariamente utilizzate avevano delimitatori ridondanti, per esempio ? con una interrogazione vuota (il loro stadio intermedio è equivalente).

`urlsplit`(*urlstring*[, *default_scheme*[, *allow_fragments*]])
Questo è simile a `urlparse()`, ma non divide i parametri della URL. Questo metodo dovrebbe essere usato generalmente al posto di `urlparse()` quando la sintassi più moderna delle URL consente parametri che possono essere applicati ad ogni segmento della porzione di percorso della URL (vedete la RFC 2396). È necessaria una funzione a parte per separare i segmenti del percorso ed i parametri. Questa funzione restituisce una tupla di 5 elementi: (schema di indirizzamento, posizione di rete, path, query, identificatore di frammento). Nuovo nella versione 2.2.

`urlunsplit`(*tuple*)
Combina gli elementi di una tupla come quella restituita da `urlsplit()` in una URL completa come se si trattasse di una stringa. Nuovo nella versione 2.2.

`urljoin`(*base, url*[, *allow_fragments*])
Costruisce una URL completa ("assoluta") attraverso la combinazione di una "URL base" (*base*) con una "URL relativa" (*url*). Al suo interno, questa funzione usa i componenti della URL base, in particolar modo lo schema di indirizzamento, la posizione di rete e (solo una parte) del percorso, per fornire i componenti mancanti nella URL relativa.

Esempio:

```
urljoin('http://www.cwi.nl/%7Eguido/Python.html', 'FAQ.html')
```

contiene la stringa

```
'http://www.cwi.nl/%7Eguido/FAQ.html'
```

L'argomento *allow_fragments* ha lo stesso significato di quello visto in `urlparse()`.

`urldefrag`(*url*)
Se *url* contiene un identificatore di frammento, restituisce una versione modificata della *url* senza l'identificatore di frammento, e l'identificatore di frammento in una stringa separata. Se in *url* non c'è identificatore di frammento, restituisce la *url* non modificata ed una stringa vuota.

Vedete anche:

RFC 1738, "*Uniform Resource Locators (URL)*"
Questa specifica la sintassi formale e la semantica delle URL assolute.

RFC 1808, "*Relative Uniform Resource Locators*"
Questa RFC (Richiesta di commento) include le regole per unire una URL assoluta con una relativa, includendo un certo numero di "esempi anormali" che coprono il trattamento di casi limite.

RFC 2396, "*Uniform Resource Identifiers (URI): Generic Syntax*"
Il documento descrive il requisito generico sintattico sia per Uniform Resource Names (URN) che per Uniform Resource Locators (URL).

11.15 `SocketServer` — Un'infrastruttura per i server di rete

Il modulo `SocketServer` semplifica il compito di scrivere server di rete.

Esistono quattro classi di base per i server: `TCPServer`, che utilizza il protocollo Internet TCP, che fornisce un flusso continuo di dati tra il client e il server. `UDPServer`, che utilizza i datagrammi, che sono pacchetti discreti di informazioni che possono arrivare fuori ordine o possono essere persi durante la trasmissione. I meno utilizzati `UnixStreamServer` e `UnixDatagramServer`, che sono simili, ma utilizzano i socket domain di UNIX; non sono disponibili su piattaforme non UNIX. Per maggiori dettagli sulla programmazione di rete, consultare un testo come lo *UNIX Network Programming* di W.Richard Steven o il *Win32 Network Programming* di Ralph Davis.

Queste quattro classi elaborano richieste in modalità *sincrona*; ogni richiesta deve essere completata prima che la richiesta successiva possa iniziare. Questo non è adatto se ogni richiesta richiede molto tempo per essere

completata, perché neccessita di parecchio calcolo, o perché restituisce molti dati ed il client è lento ad elaborarli. La soluzione è quella di creare un processo separato o un thread per gestire ogni richiesta; Le classi mix-in `ForkingMixIn` e `ThreadingMixIn` possono essere usate per supportare compiti asincroni.

Creare un server richiede diversi passaggi. Per prima cosa, si deve creare una classe handler (Ndt. gestore) di richiesta derivandola da `BaseRequestHandler` e sovrascrivendo il metodo `handle()`; questo metodo elaborerà le richieste in ingresso. Secondo, si devono istanziare una o più classi di server, passandogli l'indirizzo del server e le classi degli handler di richiesta. Infine, si deve chiamare il metodo `handle_request()` o `serve_forever()` dell'oggetto server per elaborare una o molte richieste.

Quando si eredita da `ThreadingMixIn` per avere delle connessioni in thread, si deve esplicitamente dichiarare come si vuole il proprio thread per evitare di ottenere una brusca interruzione. La classe `ThreadingMixIn` definisce un attributo *daemon_threads*, che indica se il server deve attendere un'interruzione del thread oppure no. Si deve impostare l'opzione esplicitamente se si vuole che i thread siano autonomi; il valore predefinito è `False`, che significa che Python non uscirà finché tutti i thread creati da `ThreadingMixIn` non siano terminati.

Le classi server hanno gli stessi metodi ed attributi esterni, indipendentemente dal protocollo che utilizzano.

`fileno``()`
: Restituisce un descrittore file intero per il socket su cui il server è in ascolto. La funzione è abitualmente passata a `select.select()`, per consentire il monitoraggio di server multipli nello stesso processo.

`handle_request``()`
: Elabora una richiesta singola. Questa funzione chiama i seguenti metodi nell'ordine: `get_request()`, `verify_request()` e `process_request()`. Se il metodo indicato, fornito dall'utente `handle()` della classe handler solleva un'eccezione, verrà chiamato il metodo `handle_error()` del server.

`serve_forever``()`
: Gestisce un infinito numero di richieste. Questo semplicemente chiama `handle_request()` all'interno di un ciclo infinito.

`address_family`
: La famiglia di protocolli a cui il socket del server appartiene. `socket.AF_INET` e `socket.AF_UNIX` sono due possibili valori.

`RequestHandlerClass`
: la classe handler fornita dall'utente; un'istanza di questa classe viene creata per ogni richiesta.

`server_address`
: L'indirizzo su cui il server è in ascolto. Il formato degli indirizzi varia in base alla famiglia del protocollo; vedere la documentazione del modulo socket per i dettagli. Per i protocolli internet, questo è una tupla contenente una stringa indicante l'indirizzo ed il numero di porta come intero: per esempio: `('127.0.0.1', 80)`.

`socket`
: L'oggetto socket su cui il server sarà in ascolto per le richieste in arrivo.

La classe server supporta le seguenti variabili di classe:

`allow_reuse_address`
: Quando il server consente di riutilizzare un indirizzo. Il suo valore predefinito è `False` e può essere impostato in una sotto classe per cambiare la policy.

`request_queue_size`
: La dimensione della coda di richiesta. Se il server richiede parecchio tempo per elaborare una singola richiesta, ogni richiesta che arriva quando il server è occupato viene inserita in una coda, fino al raggiungimento di `request_queue_size`. Una volta che la coda è piena, successive richieste dal client riceveranno un errore "Connection denied" (Ndt. Connessione negata). Il valore predefinito tipicamente è `5`, ma questo può essere sovrascritto con una sotto classe.

`socket_type`
: Il tipo di socket usato dal server; `socket.SOCK_STREAM` e `socket.SOCK_DGRAM` sono due possibili valori.

Ci sono diversi metodi della classe server che possono essere sovrascritti da sotto classi delle classi server come `TCPServer`; questi metodi non sono utili per le utenze esterne all'oggetto server.

finish_request()
Attualmente elabora la richiesta istanziando `RequestHandlerClass` e chiamandone il metodo `handle()`.

get_request()
Deve accettare una richiesta dal socket, e restituire una tupla di due elementi contenente il *nuovo* oggetto socket da usare per comunicare con il client e l'indirizzo del client.

handle_error(*request, client_address*)
La funzione viene chiamata se il metodo `handle()` di `RequestHandlerClass` solleva un'eccezione. L'azione predefinita è stampare la traceback sullo standard output e continuare a gestire le richieste successive.

process_request(*request, client_address*)
Chiama `finish_request()` per creare un'istanza di `RequestHandlerClass`. Se lo si desidera, questa funzione può creare un nuovo processo o un nuovo thread per gestire la richiesta; le classi `ForkingMixIn` e `ThreadingMixIn` fanno questo.

server_activate()
Chiamato dal costruttore del server per attivare il server. Può essere sovrascritto.

server_bind()
Chiamato dal costruttore del server per gestire il socket all'indirizzo desiderato. Può essere sovrascritto.

verify_request(*request, client_address*)
Deve restituire un valore Booleano; se il valore è vero, la richiesta verrà elaborata, e se è falsa, la richiesta verrà negata. La funzione può essere sovrascritta per implementare il controllo degli accessi al server. L'implementazione predefinita restituisce sempre vero.

La classe handler di richiesta deve definire un nuovo metodo `handle()` e può sovrascrivere uno dei seguenti metodi. Una nuova istanza viene creata per ogni richiesta.

finish()
Chiamato dopo il metodo `handle()` per eseguire ogni azione di pulizia richiesta. L'implementazione predefinita non fa nulla. Se `setup()` o `handle()` sollevano un'eccezione, questa funzione non verrà chiamata.

handle()
Questa funzione deve fare tutto il lavoro richiesto per servire una richiesta. Molti attributi dell'istanza sono disponibili; la richiesta è disponibile come `self.request`; l'indirizzo del client è `self.client_address`; e l'istanza del server come `self.server`, in caso avesse bisogno di accedere ad informazioni del server.

Il tipo di `self.request` è differente per i datagrammi o flussi di servizi. Per servizi di flussi, `self.request` è un oggetto socket; per i servizi di datagrammi, `self.request` è una stringa. Comunque, questo può essere nascosto usando le classi handler di richiesta mix-in `StreamRequestHandler` o `DatagramRequestHandler`, che sovrascrivono i metodi `setup()` e `finish()`, e forniscono gli attributi `self.rfile` e `self.wfile`. `self.rfile` e `self.wfile` possono essere letti o scritti, rispettivamente, per acquisire i dati richiesti o restituirli al client.

setup()
Chiamato prima del metodo `handle()` per eseguire ogni richiesta della sequenza di inizializzazione. L'implementazione predefinita non fa nulla.

11.16 `BaseHTTPServer` — Server HTTP di base

Questo modulo definisce due classi per implementare server HTTP (server Web). Solitamente, questo modulo non viene utilizzato direttamente, ma viene utilizzato come base per costruire server Web funzionanti. Vedere i moduli `SimpleHTTPServer` e `CGIHTTPServer`.

La prima classe, `HTTPServer`, è una sotto classe `SocketServer.TCPServer`. Crea ed ascolta su un socket HTTP, trasferendo la richiesta ad un gestore. Il codice per creare ed eseguire il server somiglia a questo:

```
def run(server_class=BaseHTTPServer.HTTPServer,
        handler_class=BaseHTTPServer.BaseHTTPRequestHandler):
    server_address = ('', 8000)
    httpd = server_class(server_address, handler_class)
    httpd.serve_forever()
```

class `HTTPServer`(*server_address, RequestHandlerClass*)
Questa classe costruisce sulla classe `TCPServer` memorizzando l'indirizzo del server come variabile di istanza chiamata `server_name` e `server_port`. Il server è accessibile attraverso il gestore, tipicamente attraverso la variabile `server` dell'istanza del gestore.

class `BaseHTTPRequestHandler`(*request, client_address, server*)
Questa classe viene usata per gestire le richieste HTTP che arrivano al server. Da solo, non è in grado di rispondere a nessuna richiesta HTTP; deve essere derivato in una sotto classe per gestire ogni metodo di richiesta (GET o POST). `BaseHTTPRequestHandler` fornisce numerose classi, variabili d'istanza e metodi per l'uso attraverso le sotto classi.

Il gestore analizzerà la richiesta e le intestazioni, quindi chiamerà un metodo specifico per il tipo di richiesta. Il nome del metodo viene costruito in base alla richiesta. Per esempio, per la richiesta con metodo '`SPAM`', il metodo `do_SPAM()` verrà chiamato senza alcun argomento. Tutte le informazioni rilevanti vengono memorizzate in variabili d'istanza del gestore. Le sotto classi non devono necessariamente sovrascrivere o estendere il metodo `__init__()`.

`BaseHTTPRequestHandler` ha le seguenti variabili d'istanza:

`client_address`
Contiene una tupla nella forma (*host*, *porta*) che si riferisce all'indirizzo del client.

`command`
Contiene il comando (tipo di richiesta). Per esempio `'GET'`.

`path`
Contiene il percorso richiesto.

`request_version`
Contiene la stringa della versione nella richiesta. Per esempio, `'HTTP/1.0'`.

`headers`
Mantiene le istanze della classe specificata dalla variabile di classe `MessageClass`. L'istanza analizza e gestisce le intestazioni della richiesta HTTP.

`rfile`
Contiene uno flusso dell'input, posizionato all'inizio di eventuali dati in input.

`wfile`
Contiene il flusso dell'output per rispondere ad una richiesta del client. Occorre mantenersi aderenti al protocollo HTTP quando si scrive in questo flusso.

`BaseHTTPRequestHandler` possiede le seguenti variabili di classe:

`server_version`
Specifica la versione del software del server. Questa variabile si può sovrascrivere. Il formato è una stringa separata da spazi multipli, dove ogni stringa è nella forma nome[/versione]. Per esempio, `'BaseHTTP/0.2'`.

`sys_version`
Contiene la versione del sistema Python in una forma usabile attraverso il metodo `version_string` e la variabile di classe `server_version`. Per esempio, `'Python/1.4'`.

`error_message_format`
Specifica una stringa di formato per costruire un errore di risposta al client. Utilizza specificatori di formato immodificabili e racchiusi tra parentesi, cosicché l'operando di formato dovrà essere un dizionario. La chiave di *code* dovrebbe essere un intero che specifica il valore del codice numerico di errore HTTP. *message* dovrebbe essere una stringa contenente un (dettagliato) messaggio di errore di ciò che è accaduto e *explain* dovrebbe essere una spiegazione del numero attribuito al codice di errore. Il messaggio *message* predefinito

e le spiegazioni *explain* possono essere rinvenuti nelle variabili di classe delle risposte *responses*.

`protocol_version`
Questa è la specifica versione del protocollo HTTP utilizzato nelle risposte. Se impostato a `'HTTP/1.1'`, il server consentirà connessioni persistenti; il server deve comunque includere un'accurata intestazione `Content-Length` (utilizzando `send_header()`) in tutte le sue risposte ai client. Per retrocompatibilità, l'impostazione predefinita è `'HTTP/1.0'`.

`MessageClass`
Specifica una classe corrispondente all'`rfc822.Message` per analizzare le intestazioni HTTP. Tipicamente, questa non viene sovrascritta ed il suo valore predefinito è `mimetools.Message`.

`responses`
Questa variabile contiene una mappa tra gli interi dei codici di errori e una serie di tuple di due elementi contenenti un breve e un lungo messaggio. Per esempio, `{`*code*`: (`*shortmessage*`, `*longmessage*`)}`. Il messaggio breve viene usato di solito per la chiave del messaggio in una risposta di errore, e il messaggio lungo come la chiave di spiegazione (vedere la variabile di classe `error_message_format`).

Un'istanza `BaseHTTPRequestHandler` possiede i seguenti metodi:

`handle``()`
Chiama una volta `handle_one_request()` (o, se la connessione persistente è abilitata, più volte) per gestire le richieste HTTP in arrivo. Non ci sono motivi per sovrascrivere questo metodo; al suo posto si possono implementare i più appropriati metodi `do_*()`.

`handle_one_request``()`
Questo metodo analizza e invia la richiesta all'appropriato metodo `do_*()`. Non si deve sovrascrivere questo metodo.

`send_error``(`*code*`[`*, message*`])`
Invia e registra come log una risposta completa di errore al client. Il codice numerico *code* indica il codice di errore HTTP, con *message* facoltativo, contenente testo maggiormente specifico. Un insieme completo di intestazioni viene inviato seguito dal testo composto usando la variabile di classe `error_message_format`.

`send_response``(`*code*`[`*, message*`])`
Invia un'instestazione di risposta e registra come log le richieste accettate. La riga HTTP di risposta viene inviata, seguita dalle intestazioni *Server* e *Date*. Il valore per queste due intestazioni viene prelevato rispettivamente dai metodi `version_string()` e `date_time_string()`.

`send_header``(`*keyword, value*`)`
Scrive un'intestazione HTTP specifica sul flusso di uscita. *keyword* dovrebbe specificare la chiave dell'intestazione, con *value* che specifica il valore.

`end_headers``()`
Invia una riga vuota, che indica la fine delle intestazioni HTTP nella risposta.

`log_request``([`*code*`[`*, size*`]])`
Registra come log ed accetta (con successo) la richiesta. *code* dovrebbe specificare il codice HTTP numerico associato alla risposta. Se la dimensione della riposta è disponibile, dovrebbe venir passata come parametro *size*.

`log_error``(...)`
Registra come log un errore quando una richiesta non può essere pienamente soddisfatta. Il suo comportamento predefinito è passare il messaggio a `log_message()`, in modo da acquisire gli stessi argomenti (*format* ed ulteriori valori).

`log_message``(`*format, ...*`)`
Registra come log un messaggio arbitrario sul `sys.stderr`. Questo metodo viene tipicamente sovrascritto per creare meccanismi di logging di errori personalizzati. L'argomento *format* è una stringa di formato standard in stile printf, dove gli argomenti addizionali per `log_message()` vengono applicati come input nella formattazione. L'indirizzo del client, data ed ora correnti vengono inseriti all'inizio di ogni messaggio registrato nei log.

version_string()
Restituisce la stringa con la versione del software del server. Questa è una combinazione delle variabili di classe `server_version` e `sys_version`.

date_time_string()
Restituisce la data e l'ora corrente, formattata per l'intestazione di un messaggio.

log_data_time_string()
Restituisce la data e l'ora corrente, formattata per il logging.

address_string()
Restituisce l'indirizzo del client, formattato per il logging. Una ricerca del nome viene eseguita sull'indirizzo IP del client.

Vedete anche:

Modulo `CGIHTTPServer` (sezione 11.18):
Gestore di richieste esteso, che supporta gli scripts CGI.

Modulo `SimpleHTTPServer` (sezione 11.17):
Gestore di richieste di base che limita le risposte ai file attuamente nella directory principale.

11.17 `SimpleHTTPServer` — Semplice gestore di richieste HTTP

Il modulo `SimpleHTTPServer` definisce una classe per la gestione di richieste ed un'interfaccia compatibile con `BaseHTTPServer.BaseHTTPRequestHandler` che serve file solo dalla directory di base.

Il modulo `SimpleHTTPServer` definisce le seguenti classi:

class SimpleHTTPRequestHandler(*request*, *client_address*, *server*)
Questa classe viene usata per fornire file dalla directory corrente e discendenti, mappando direttamente la struttura delle directory alla richiesta HTTP.

Una buona parte del lavoro viene eseguito dalla classe di base `BaseHTTPServer.BaseHTTPRequestHandler`, come l'analisi della richiesta. Questa classe implementa le funzioni `do_GET()` e `do_HEAD()`.

La classe `SimpleHTTPRequestHandler` definisce le seguenti variabili membro:

server_version
Questa sarà `SimpleHTTP/ + __version__`, dove `__version__` viene definita nel modulo.

extensions_map
Un dizionario che mappa i suffissi con i tipi MIME. Il valore predefinito è rappresentato da una stringa vuota, ed è considerato `text/plain`. La mappa viene usata senza il distinguo maiuscolo/minuscolo e dovrebbe contenere solo chiavi in minuscolo.

La classe `SimpleHTTPRequestHandler` definisce i seguenti metodi:

do_HEAD()
Questo metodo risponde alle richieste di tipo `'HEAD'`: invia le intestazioni equivalenti alla richiesta `GET`. Vedere il metodo `do_GET()` per una più completa spiegazione sulle intestazioni possibili.

do_GET()
La richiesta viene mappata con un file locale attraverso l'interpretazione della richiesta come un percorso relativo alla directory di lavoro corrente.

Se la richiesta viene mappata con una directory, viene inviata una risposta `403`, seguita dalla spiegazione `'Directory listing not supported'`. Ogni eccezione `IOError` durante l'apertura del file richiesto viene mappata ad un errore `404`, `'File not found'`. Altrimenti il tipo di contenuto viene elaborato usando la variabile *extensions_map*.

Un'intestazione `'Content-type:'` con il tipo di contenuto presunto è l'output inviato, insieme ad una riga vuota, che si traduce nella fine delle intestazioni e quindi viene il contenuto del file. Il file viene sempre aperto in modalità binaria.

Per un esempio sull'uso vedere l'implementazione della funzione `test()`.

Vedete anche:

Modulo `BaseHTTPServer` (sezione 11.16):
Implementazione della classe di base per server Web e gestori di richieste.

11.18 `CGIHTTPServer` — Gestore di richieste HTTP con supporto CGI

Il modulo `CGIHTTPServer` definisce una classe di gestione (handler), con interfaccia compatibile con `BaseHTTPServer.BaseHTTPRequestHandler`, eredita le sue caratteristiche da `SimpleHTTPServer.SimpleHTTPRequestHandler` potendo anche eseguire script CGI.

Note: Questo modulo può eseguire script CGI su sistemi UNIX e Windows; su sistemi Mac OS sarà in grado solo di eseguire script Python all'interno del suo stesso processo.

Il modulo `CGIHTTPServer` definisce le seguenti classi:

class `CGIHTTPRequestHandler`(*request, client_address, server*)
Questa classe viene usata per servire sia file che l'output di script CGI dalla directory corrente e sottostante. Notare che la mappatura della struttura gerarchica HTTP alla locale struttura di directory è esattamente la stessa che in `SimpleHTTPServer.SimpleHTTPRequestHandler`.

La classe comunque, esegue lo script CGI, invece di servirlo come file, se viene informata che l'oggetto è uno script CGI. Vengono utilizzati solo CGI basati su directory — la configurazione dei server comuni è di trattare le estensioni speciali considerandole come script CGI.

Le funzioni `do_GET()` e `do_HEAD()` vengono modificate per eseguire gli script CGI e servire l'output, invece di servire i file, se la richiesta viene è fatta all'interno dei percorsi `cgi_directories` sottostanti.

La classe `CGIHTTPRequestHandler` definisce i seguenti dati membri:

`cgi_directories`
L'impostazione predefinita è `['/cgi-bin', '/htbin']` e descrive le directory in cui i file devono essere considerati script CGI.

La classe `CGIHTTPRequestHandler` definisce i seguenti metodi:

`do_POST`()
Questo metodo serve le richieste di tipo `'POST'`, permesse solo per gli script CGI. Viene generato l'errore 501, Can only POST to CGI scripts quando si cerca di usare questa funzione con url non CGI.

Notare che gli script CGI verranno eseguiti con UID dell'utente nobody, per motivi di sicurezza. Problemi con gli script CGI verranno convertiti in errore 403.

Per un esempio sull'utilizzo, si veda l'implementazione della funzione `test()`.

Vedete anche:

Modulo `BaseHTTPServer` (sezione 11.16):
Implementazione della classe base per server WEB e gestori di richieste.

11.19 `Cookie` — gestione dello stato HTTP

Il modulo `Cookie` definisce le classi per astrarre il concetto di cookies, un meccanismo di gestione dello stato HTTP. Supporta cookie semplici basati su stringhe e fornisce una astrazione per inserire ogni tipo di dato serializzabile come valore cookie.

Il modulo si attiene strettamente alle regole di analisi descritte nelle specifiche RFC 2109 e RFC 2068. È anche stato scoperto che MSIE 3.0x non segue le regole sui caratteri indicate in queste specifiche. Come risultato, l'analisi delle regole usate è un pò meno stringente.

exception `CookieError`

Eccezione riferita a fallimenti per una non validità rispetto all'RFC 2109: attributi non corretti, intestazioni Set-Cookie: non corrette, etc..

class `BaseCookie`([*input*])
Questa classe è un oggetto simile ai dizionari le cui chiavi sono stringhe e i cui valori sono istanze `Morsel`. Notare che dopo aver impostato la chiave in un valore, il valore viene prima convertito in un `Morsel` contenente una chiave ed il valore.

Se *input* viene indicato, viene passato al metodo `load()`.

class `SimpleCookie`([*input*])
Questa classe deriva da `BaseCookie` e sovrascrive i metodi `value_decode()` e `value_encode()` per divenire rispettivamente identity e `str()`.

class `SerialCookie`([*input*])
Questa classe deriva da `BaseCookie` e sovrascrive `value_decode()` e `value_encode()` per divenire `pickle.loads()` e `pickle.dumps()`.

Deprecato dalla versione 2.3. La lettura di valori pickled da dati cookie non affidabili è un potenziale grave buco di sicurezza, in quanto le stringhe serializzate possono essere modificate in modo che codice arbitrario possa essere eseguito sul server. Viene supportato solo per compatibilità con le versioni precedenti ed in futuro potrebbe essere eliminato definitivamente.

class `SmartCookie`([*input*])
Questa classe deriva da `BaseCookie`. Sovrascrive `value_decode()` per divenire `pickle.loads()` se c'e' un valido dato serializzato o altrimenti il suo stesso valore. Sovrascrive anche `value_encode()` in `pickle.dumps()` finché è una stringa, nel cui caso restituisce il valore stesso.

Deprecato dalla versione 2.3. Lo stesso avviso di sicurezza di `SerialCookie` viene ribadito anche qui.

Una altra nota sulla sicurezza è necessaria: per compatibilità con il passato, il modulo `Cookie` esporta una classe chiamata `Cookie` che è solo un alias per `SmartCookie`. Questo è probabilmente un errore e verrà quanto prima rimosso. Non si deve utilizzare la classe `Cookie` nelle proprie applicazioni, per lo stesso motivo per cui non si deve utilizzare la classe `SerialCookie`.

Vedete anche:

RFC 2109, "*HTTP State Management Mechanism*"
Queste sono le specifiche per la gestione dello stato implementate da questo modulo.

11.19.1 Oggetti Cookie

`value_decode`(*val*)
Restituisce un valore decodificato da una rappresentazione di stringa. Il valore restituito può essere di qualsiasi tipo. Questo metodo non fa nulla in `BaseCookie` — esiste per essere sovrascritto.

`value_encode`(*val*)
Restituisce un valore codificato. *val* può essere di qualsiasi tipo, ma il valore restituito deve essere una stringa. Questo metodo non fa nulla in `BaseCookie` — esiste per essere sovrascritto.

In generale, dovrebbe essere la situazione in cui `value_encode()` e `value_decode()` vengono invertiti nell'intervallo di *value_decode*.

`output`([*attrs*[, *header*[, *sep*]]])
Restituisce una rappresentazione sotto forma di stringa adatta per essere inviata come intestazione HTTP. *attrs* ed *header* vengono inviati ad ogni metodo di `output()` `Morsel`. *sep* viene utilizzato per unire le intestazioni assieme, ed il suo valore predefinito è un carattere di fine riga.

`js_output`([*attrs*])
Restituisce uno snippet JavaScript, che, se eseguito in un browser che supporta JavaScript, si comporta nello stesso modo di quando viene inviata un'intestazione HTTP.

Il significato di *attrs* è lo stesso che in `output()`.

`load`(*rawdata*)
Se *rawdata* è una stringa, la analizza come un `HTTP_COOKIE` ed aggiunge il valore trovato come `Morsel`. Se è un dizionario, è equivalente a:

```
for k, v in rawdata.items():
    cookie[k] = v
```

11.19.2 Oggetti Morsel

class `Morsel`()
Estrae una coppia chiave/valore, che possiede alcuni attributi RFC 2109.

Morsel è un oggetto equivalente ad un dizionario, dove l'insieme di chiavi è costante – gli attributi RFC 2109 validi si compongono di:

- `expires`
- `path`
- `comment`
- `domain`
- `max-age`
- `secure`
- `version`

Le chiavi non sono sensibili alle differenza maiuscole/minuscole.

`value`
Il valore del cookie.

`coded_value`
Il valore codificato del cookie — questo dovrebbe essere ciò che è stato inviato.

`key`
Il nome del cookie.

`set`(*key, value, coded_value*)
Imposta i membri di *key*, *value* e *coded_value*.

`isReservedKey`(*K*)
Dove *K* è un membro dell'insieme di chiavi di un `Morsel`.

`output`([*attrs*[, *header*]])
Restituisce una rappresentazione sotto forma di stringa del Morsel, adatta ad essere inviata come intestazione HTTP. In modo predefinito vengono inclusi tutti gli attributi, al contrario, se viene passato *attrs*, deve essere una lista di attributi da usare. Il valore predefinito per *header* è `Set-Cookie:`.

`js_output`([*attrs*])
Restituisce un frammento di JavaScript inglobabile, che, se eseguito in un browser che supporta JavaScript, si comporta allo stesso modo di un invio di un'intestazione HTTP.

Il significato di *attrs* è lo stesso di quello in `output()`.

`OutputString`([*attrs*])
Restituisce una rappresentazione sotto forma di stringa del Morsel, senza alcun HTTP o JavaScript di contorno.

Il significato di *attrs* è lo stesso di quello in `output()`.

11.19.3 Esempio

Il seguente esempio mostra come utilizzare il modulo `Cookie`.

```
>>> import Cookie
>>> C = Cookie.SimpleCookie()
>>> C = Cookie.SerialCookie()
>>> C = Cookie.SmartCookie()
>>> C["fig"] = "newton"
>>> C["sugar"] = "wafer"
>>> print C # genera le intestazioni HTTP
Set-Cookie: sugar=wafer;
Set-Cookie: fig=newton;
>>> print C.output() # stessa cosa
Set-Cookie: sugar=wafer;
Set-Cookie: fig=newton;
>>> C = Cookie.SmartCookie()
>>> C["rocky"] = "road"
>>> C["rocky"]["path"] = "/cookie"
>>> print C.output(header="Cookie:")
Cookie: rocky=road; Path=/cookie;
>>> print C.output(attrs=[], header="Cookie:")
Cookie: rocky=road;
>>> C = Cookie.SmartCookie()
>>> C.load("chips=ahoy; vienna=finger") #  Carica da una
                                        #+ stringa (intestazione HTTP)
>>> print C
Set-Cookie: vienna=finger;
Set-Cookie: chips=ahoy;
>>> C = Cookie.SmartCookie()
>>> C.load('keebler="E=everybody; L=\\"Loves\\"; fudge=\\012;";')
>>> print C
Set-Cookie: keebler="E=everybody; L=\"Loves\"; fudge=\012;";
>>> C = Cookie.SmartCookie()
>>> C["oreo"] = "doublestuff"
>>> C["oreo"]["path"] = "/"
>>> print C
Set-Cookie: oreo=doublestuff; Path=/;
>>> C = Cookie.SmartCookie()
>>> C["twix"] = "none for you"
>>> C["twix"].value
'none for you'
>>> C = Cookie.SimpleCookie()
>>> C["number"] = 7 # equivalente a C["number"] = str(7)
>>> C["string"] = "seven"
>>> C["number"].value
'7'
>>> C["string"].value
'seven'
>>> print C
Set-Cookie: number=7;
Set-Cookie: string=seven;
>>> C = Cookie.SerialCookie()
>>> C["number"] = 7
>>> C["string"] = "seven"
>>> C["number"].value
7
>>> C["string"].value
'seven'
>>> print C
Set-Cookie: number="I7\012.";
Set-Cookie: string="S'seven'\012p1\012.";
>>> C = Cookie.SmartCookie()
>>> C["number"] = 7
>>> C["string"] = "seven"
>>> C["number"].value
7
>>> C["string"].value
'seven'
>>> print C
Set-Cookie: number="I7\012.";
Set-Cookie: string=seven;
```

11.20 `xmlrpclib` — accesso a client XML-RPC

Nuovo nella versione 2.2.

XML-RPC è un metodo per Chiamate a Procedure Remote che utilizza come trasporto l'XML trasferito via HTTP. Con questo metodo, un client può chiamare metodi con parametri su server remoti (il server viene indicato da una URI) e gli vengono restituiti dei dati strutturati. Questo modulo supporta la scrittura di codice client XML-RPC; gestisce tutti i dettagli sulla traduzione tra oggetti Python conformi e XML durante le operazioni.

class `ServerProxy`(*uri*[, *transport*[, *encoding*[, *verbose*[, *allow_none*]]]])

Un'istanza `ServerProxy` è un oggetto che gestisce le comunicazioni con un server remoto XML-RPC. Il primo argomento richiesto è una URI (Uniform Resource Indicator) e normalmente è l'URL del server. Il secondo argomento (facoltativo) è un'istanza di base per il trasporto; il modo predefinito è un'istanza interna per HTTPS `SafeTransport`; altrimenti URL ed istanza interna HTTP `Transport`. Il terzo argomento (facoltativo) è una codifica, il cui valore predefinito è UTF-8. Il facoltativo quarto argomento è un'opzione di debugging. Se *allow_none* è vera, la costante Python `None` verrà tradotta in XML; il comportamento predefinito per `None` è il sollevamento di un'eccezione `TypeError`. Questa è un'estensione comunemente usata per la specifica XML-RPC, ma non viene supportata da tutti i client e da tutti server; vedere http://ontosys.com/xml-rpc/extensions.html per una descrizione.

Entrambe le modalità di trasporto, HTTP ed HTTPS, supportano le estensioni della sintassi del'URL per l'autenticazione di base HTTP: `http://user:pass@host:port/path`. La porzione `user:pass` viene codificata in base64 come un'intestazione HTTP 'Authorization', ed inviata al server remoto come parte del processo di connessione quando si richiama un metodo XML-RPC. Si ha bisogno di usare questo meccanismo solo quando il server remoto richiede una autenticazione di base con utente e password.

L'istanza restituita è un oggetto proxy con metodi che possono essere usati per invocare le corrispondenti chiamate RPC sul server remoto. Se il server remoto supporta l'API per l'introspezione, il proxy può essere usato anche per richiedere al server remoto quali metodi supporta (NdT: scoperta servizi – service discovery) ed analizzare altri metadata associati al server.

I metodi dell'istanza `ServerProxy` utilizzano come argomenti tipi base di Python ed oggetti, e restituiscono tipi base di Python e classi. I tipi si conformano (per esempio possono essere serializzati attraverso XML), inclusi i seguenti (ed eccetto dove indicato, vengono deserializzati come alcuni tipi di Python):

Nome	Significato
`boolean`	Le costanti `True` e `False`
`integers`	Passati direttamente
`floating-point numbers`	Passati direttamente
`strings`	Passati direttamente
`arrays`	Ogni tipo di sequenza Python contenente elementi conformabili. Gli Array vengono res
`structures`	Un dizionario Python. La chiave deve essere una stringa, il valore può essere qualsiasi t
`dates`	in secondi a partire da epoch; passa in un'istanza della classe wrapper `DateTime`
`binary data`	passa in un'istanza della classe wrapper `Binary`

Questo è l'intero insieme dei tipi di dati supportati da XML-RPC. Chiamate di metodi possono anche sollevare una speciale istanza di `Fault`, usata per segnalare errori a server XML-RPC, o `ProtocolError` usato per segnalare un errore nello strato di trasporto HTTP/HTTPS. Fare attenzione che a partire da Python 2.2 si possono derivare tipi built-in, e che il modulo xmlrpclib non supporta attualmente la serializzazione di istanze di queste sotto classi.

Quando si passa una stringa, caratteri speciali per XML come '<', '>' e '&' sono automaticamente preceduti dal carattere di protezione (NdT: la barra rovesciata). Comunque è nella responsabilità del chiamante sincerarsi che la stringa sia priva di caratteri non consentiti in XML, come i caratteri di controllo ASCII che vanno dal valore 0 al 31; ignorando questo particolare si ottengono richieste XML-RPC che non sono ben formate per XML. Se si devono passare stringhe arbitrarie attraverso XML-RPC, usare la classe wrapper `Binary` descritta sotto.

`Server` è considerato come un alias per `ServerProxy` per compatibiltà con il passato. Il nuovo codice dovrebbe usare `ServerProxy`.

Vedete anche:

XML-RPC HOWTO

(http://xmlrpc-c.sourceforge.net/xmlrpc-howto/xmlrpc-howto.html)
: Una buona descrizione delle operazioni XML e programmi client in diversi linguaggi. Contiene tutto quanto uno sviluppatore di client XML-RPC deve conoscere.

XML-RPC-Hacks page
(http://xmlrpc-c.sourceforge.net/hacks.php)
: Estensioni per varie librerie open-source per supportare introspezione e chiamate multiple.

11.20.1 Oggetti ServerProxy

Un'istanza `ServerProxy` ha un metodo corrispondente per ogni chiamata di procedura remota accettata dal server XML-RPC. La chiamata al metodo esegue un RPC, comunicato sia dal nome che da una firma di argomento (per esempio lo stesso nome di metodo può essere sovrascritto con firme di argomenti multipli). L'RPC finisce restituendo un valore, che dovrebbe essere o il dato restituito come tipo conforme o un oggetto `Fault` o `ProtocolError` che indicano un errore.

I server che supportano le API di introspezione XML, supportano anche alcuni metodi comuni raggruppati sotto la voce membri `system` riservati:

`system.listMethods``()`
: Questo metodo restituisce una lista di stringhe, una per ogni (non di sistema) metodo supportato dal server XML-RPC.

`system.methodSignature``(`*name*`)`
: Questo metodo prende un parametro, il nome di un metodo implementato dal server XML-RPC. Restituisce un array di possibili firme per questo metodo. Una firma è un array di tipi. Il primo di questi tipi è il tipo restituito dal metodo, il resto sono parametri.

  Siccome sono permesse firme multiple (come la sovrascrittura), questo metodo restituisce una lista di firme diverse anziché una sola.

  Le firme, sono ristrette ai parametri del livello principale che si attendono dal metodo. Per esempio, se un metodo attende un array di strutture come parametro, e restituisce una stringa, la sua firma è semplicemente string, array. Se attende tre interi e restituisce una stringa, la sua firma è string, int, int, int.

  Se non sono definite firme per questo metodo, viene restituito un valore non-array. In Python questo significa che il tipo del valore restituito è qualcosa di diverso da quello lista.

`system.methodHelp``(`*name*`)`
: Questo metodo prende un parametro, il nome di un metodo implementato dal server XML-RPC. Restituisce una docstring che descrive l'uso di quel metodo. Se non è disponibile alcuna stringa, viene restituita una stringa vuota. La docstring può contenere marcatori HTML.

I metodi di introspezione vengono correntemente supportati dai server scritti in PHP, C e Microsoft .NET. Il supporto parziale all'introspezione è incluso nei recenti update di UserLand Frontier. Il supporto per l'introspezione per Perl, Python e Java è disponibilie nelle pagine di Hacking di XML-RPC.

11.20.2 Oggetti Boolean

Questa classe può essere inizializzata da ogni valore Python; l'istanza restituita dipende solo dal suo valore reale. Supporta vari operatori Python attraverso i metodi `__cmp__()`, `__repr__()`, `__int__()` e `__nonzero__()`, tutti implementati con le solite modalità.

Questo oggetto possiede anche i seguenti metodi, supportati principalmente per uso interno dal codice di deserializzazione:

`encode``(`*out*`)`
: Scrive la codifica XML-RPC di questo elemento booleano sull'output dell'oggetto flusso.

11.20.3 Oggetti DateTime

Questa classe può essere inizializzata dalla data in secondi da epoch, una tupla del tempo, o una stringa ora/data ISO8601. Dispone dei seguenti metodi, supportati principalmente per uso interno dal codice di serializzazione/deserializzazione:

`decode`(*string*)
Accetta una stringa come nuova istanza di valore tempo.

`encode`(*out*)
Scrive la codifica XML-RPC di questo elemento DateTime sull'output dell'oggetto flusso.

Supporta anche alcuni operatori built-in di Python attraverso i metodi `__cmp__` e `__repr__`.

11.20.4 Oggetti Binary

Questa classe può essere inizializzata dalla stringa di dati (che può includere valori NUL). Il primo sistema di accesso al contenuto di un oggetto `Binary` viene fornito da un attributo:

`data`
il dato binario *data* incapsulato dall'istanza `Binary`. *data* viene fornito come una stringa 8-bit.

Gli oggetti `Binary` hanno i seguenti metodi, supportati principalmente per uso interno dal codice di serializzazione/deserializzazione:

`decode`(*string*)
Accetta una stringa base64 e la decodifica come nuovi dati dell'istanza.

`encode`(*out*)
Scrive la codifica base64 XML-RPC di questo elemento binario sull'output dell'oggetto flusso.

Supporta anche alcuni operati built-in di Python come il metodo `__cmp__()`.

11.20.5 Oggetti Fault

Un oggetto `Fault` incapsula il contenuto di un tag fault XML-RPC. Gli oggetti Fault dispongono dei seguenti membri:

`faultCode`
Una stringa che indica il tipo di fault.

`faultString`
Una stringa contenente un messaggio di diagnostica associato al fault.

11.20.6 Oggetti ProtocolError

Un oggetto `ProtocolError` descrive un errore di protocollo in uno strato di trasporto sottostante (come un errore 404 'not found' se il server indicato dalla URI non esiste). Dispone dei seguenti membri:

`url`
L'URI o l'URL che ha scatenato l'errore.

`errcode`
Il codice di errore.

`errmsg`
Il messaggio di errore o la stringa con la diagnostica.

`headers`
Una stringa contenente l'intestazione della richiesta HTTP/HTTPS che ha causato l'errore.

11.20.7 Oggetti MultiCall

Nuovo nella versione 2.4.

All'indirizzo http://www.xmlrpc.com/discuss/msgReader$1208, viene mostrato un metodo di approccio per incapsulare chiamate multiple verso un server remoto in una singola richiesta.

class `MultiCall`(*server*)
Crea un oggetto usato dalla chiamata del metodo boxcar. *server* è l'eventuale bersaglio della chiamata. Le chiamate possono essere fatte all'oggetto risultante, ma devono immediatamente restituire `None` e memorizzare solo il nome chiamato ed i parametri nell'oggetto `MultiCall`. Chiamare l'oggetto direttamente comporta che tutte le chiamate memorizzate vengano trasmesse come una singola richiesta `system.multicall`. Il risultato di questa chiamata è un generatore; iterando su questo generatore si ottengono i risultati individuali.

Un esempio di utilizzo di questa classe è:

```
multicall = MultiCall(server_proxy)
multicall.add(2,3)
multicall.get_address("Guido")
add_result, address = multicall()
```

11.20.8 Funzioni utili

`boolean`(*value*)
Converte ogni valore Python in una costante booleana XML-RPC, `True` o `False`.

`binary`(*data*)
Converte brutalmente ogni stringa Python in un oggetto `Binary`.

`dumps`(*params*[, *methodname*[, *methodresponse*[, *encoding*[, *allow_none*]]]])
Converte *params* in una richiesta XML-RPC o in una risposta se *methodresponse* viene impostata al valore vero. *params* può essere sia una tupla di argomenti che un'istanza della classe di eccezione `Fault`. Se *methodresponse* viene impostata al valore vero, può essere restituito solamente un singolo valore, il che significa che *params* deve essere di lunghezza `1`. *encoding*, se viene indicato, è la codifica da usare per generare XML; il suo valore predefinito è UTF-8. Il valore `None` di Python non può essere usato in XML-RPC standard; per permetterne l'uso tramite un'estensione, fornite un valore vero al parametro *allow_none*.

`loads`(*data*)
Converte una richiesta o una risposta XML-RPC in un oggetto Python, un (*parametri*, *nomemetodo*). *parametri* è una tupla di argomenti; *nomemetodo* è una stringa, o `None` se non è presente alcun nome di metodo nel pacchetto. Se il pacchetto XML-RPC rappresenta una condizione di errore, questa funzione solleverà un'eccezione `Fault`.

11.20.9 Esempi di utilizzo del client

```
# semplice programma di test (dalle specifiche XML-RPC)

# server = ServerProxy("http://localhost:8000") # server locale
server = ServerProxy("http://betty.userland.com")

print server

try:
    print server.examples.getStateName(41)
except Error, v:
    print "ERROR", v
```

11.21 SimpleXMLRPCServer — Server basilare XML-RPC

Il modulo SimpleXMLRPCServer fornisce una struttura di server di base per server XML-RPC scritti in Python. I server possono essere autonomi, usando SimpleXMLRPCServer, o inglobati in un ambiente CGI, usando CGIXMLRPCRequestHandler.

class SimpleXMLRPCServer(*addr*[, *requestHandler*[, *logRequests*]])
Crea una nuova istanza di server. Il parametro *requestHandler* deve essere un valore factory per istanze di gestori di richieste. il suo valore predefinito è SimpleXMLRPCRequestHandler. I parametri *addr* e *requestHandler* vengono passati al costruttore SocketServer.TCPServer. Se *logRequests* viene impostato al valore vero (il predefinito), le richieste verranno registrate; impostando questo parametro a falso, si disattiverà la registrazione dei log. Questa classe fornisce metodi per registrare le funzioni che saranno poi chiamate dal protocollo XML-RPC.

class CGIXMLRPCRequestHandler()
Crea una nuova istanza per gestire richiesta XML-RPC in un ambiente CGI. Nuovo nella versione 2.3.

class SimpleXMLRPCRequestHandler()
Crea una nuova istanza di gestione delle richieste. Questo gestore di richieste supporta le richieste POST e modifica la registrazione del log cosicché il parametro *logRequests* del costruttore SimpleXMLRPCServer venga onorato.

11.21.1 Oggetti SimpleXMLRPCServer

La classe SimpleXMLRPCServer è basata su SocketServer.TCPServer e permette la creazione di semplici, autonomi server XML-RPC.

register_function(*function*[, *name*])
Registra una funzione che può rispondere ad una richiesta XML-RPC. Se *name* viene indicato, sarà il nome del metodo associato a *function*, altrimenti verrà usata *function*.__name__. *name* può essere una stringa normale o Unicode, e può contenere caratteri non validi negli identificatori Python, incluso il carattere della virgola.

register_instance(*instance*)
Registra un oggetto usato per esporre i nomi dei metodi che sono stati registrati usando register_function(). Se *instance* non possiede un metodo _dispatch(), viene cercato un attributo che corrisponde al nome del metodo richiesto; se il nome del metodo contiene virgole, ogni componente del nome del metodo viene cercato individualmente, con l'effetto che viene eseguita una semplice ricerca gerarchica. Il valore trovato da questa ricerca viene quindi chiamato con il parametro della richiesta ed il valore restituito viene rispedito indietro al client.

register_introspection_functions()
Registra la funzione di introspezione XML-RPC system.listMethods, system.methodHelp e system.methodSignature. Nuovo nella versione 2.3.

register_multicall_functions()
Registra le funzioni multichiamata XML-RPC system.multicall.

Esempio:

```
class MyFuncs:
    def div(self, x, y) : return div(x,y)


server = SimpleXMLRPCServer(("localhost", 8000))
server.register_function(pow)
server.register_function(lambda x,y: x+y, 'add')
server.register_introspection_functions()
server.register_instance(MyFuncs())
server.serve_forever()
```

11.21.2 CGIXMLRPCRequestHandler

La classe `CGIXMLRPCRequestHandler` può essere usata per gestire richieste XML-RPC inviate a script CGI Python.

`register_function`(*function*[, *name*])
Registra una funzione che può rispondere ad una richiesta XML-RPC. Se *name* viene passato, sarà il nome del metodo associato alla funzione, altrimenti verrà usata *function.__name__*. *name* può essere una normale stringa Unicode, e può contenere caratteri non legali negli identificatori Python, incluso un carattere virgola.

`register_instance`(*instance*)
Registra un oggetto che viene usato per esporre nomi di metodi che non sono stati registrati usando `register_function()`. Se *istance* contiene un metodo `_dispatch()`, viene chiamato con il nome del metodo richiesto ed il parametro della richiesta; il valore restituito viene inviato al client come risultato. Se *instance* non ha un metodo `_dispatch()`, viene cercato un attributo che corrisponde al nome del metodo richiesto; se il nome del metodo richiesto contiene virgole, ogni componente del nome del metodo viene cercato individualmente eseguendo una semplice ricerca gerarchica. Il valore trovato da questa richiesta viene quindi chiamato con il parametro indicato dalla richiesta ed il valore restituito viene rispedito al client.

`register_introspection_functions`()
Registra le funzioni di introspezione XML-RPC `system.listMethods`, `system.methodHelp` e `system.methodSignature`.

`register_multicall_functions`()
Registra la funzione multichiamata XML-RPC `system.multicall`.

`handle_request`([*request_text = None*])
Gestisce una richiesta XML-RPC. Se *request_test* viene indicato, deve essere il dato POST fornito dal server HTTP, altrimenti viene utilizzato il contenuto di stdin.

Esempio:

```
class MyFuncs:
    def div(self, x, y) : return div(x,y)


handler = CGIXMLRPCRequestHandler()
handler.register_function(pow)
handler.register_function(lambda x,y: x+y, 'add')
handler.register_introspection_functions()
handler.register_instance(MyFuncs())
handler.handle_request()
```

11.22 `DocXMLRPCServer` — Server XML-RPC di autodocumentazione

Nuovo nella versione 2.3.

Il modulo `DocXMLRPCServer` estende le classi trovate in `SimpleXMLRPCServer` per fornire documentazione HTML in risposta ad una richiesta HTTP GET. I server possono essere standalone, usando `DocXMLRPCServer`, o integrati in un ambiente CGI, usando `DocCGIXMLRPCRequestHandler`.

class `DocXMLRPCServer`(*addr*[, *requestHandler*[, *logRequests*]])
Crea una nuova istanza di server. Tutti i parametri hanno lo stesso significato che si trova in `SimpleXMLRPCServer.SimpleXMLRPCServer`; il *requestHandler* predefinito è `DocXMLRPCRequestHandler`.

class `DocCGIXMLRPCRequestHandler`()

Crea una nuova istanza per gestire richieste XML-RPC in un ambiente CGI.

class `DocXMLRPCRequestHandler`()
Crea una nuova istanza di gestore di richieste. Questo gestore di richieste supporta le richieste XML-RPC POST, richieste GET di documentazione, e modifica il gestore di log così che il parametro *logRequests* del costruttore `DocXMLRPCServer` sia onorato.

11.22.1 Oggetti DocXMLRPCServer

La classe `DocXMLRPCServer` deriva da `SimpleXMLRPCServer.SimpleXMLRPCServer` e fornisce un significato alla creazione di server XML-RPC autonomi e con gestione dell'autodocumentazione. Le richieste HTTP POST vengono gestite come chiamate di metodo XML-RPC. Le richieste HTTP GET generano documentazione HTML in stile pydoc. Questo consente ad un server di fornire la propria documentazione web.

`set_server_title`(*server_title*)
Imposta il titolo usato nella documentazione HTML generata. Questo titolo verrà usato all'interno dell'elemento HTML title.

`set_server_name`(*server_name*)
Imposta il nome usato nella documentazione HTML generata. Questo nome appare al principio della documentazione generata all'interno di un elemento h1.

`set_server_documentation`(*server_documentation*)
Imposta la descrizione usata nella documentazione HTML generata. La descrizione apparirà nella documentazione come un paragrafo, sotto il nome del server.

11.22.2 DocCGIXMLRPCRequestHandler

La classe `DocCGIXMLRPCRequestHandler` deriva da `SimpleXMLRPCServer.CGIXMLRPCRequestHandler` e fornisce un meccanismo per la creazione di autodocumentazione a script CGI XML-RPC. Le richieste HTTP GET vengono gestite attraverso la generazione di documentazione HTML in stile pydoc. Questo consente a un server di generare autonomamente la propria documentazione web.

`set_server_title`(*server_title*)
Imposta il titolo usato nella documentazione HTML generata. Questo titolo verrà usato dentro l'elemento HTML title.

`set_server_name`(*server_name*)
Imposta il nome usato nella documentazione HTML generata. Questo nome apparirà all'inizio della documentazione dentro l'elemento h1.

`set_server_documentation`(*server_documentation*)
Imposta la descrizione usata nella documentazione HTML generata. Questa descrizione apparirà nella documentazione come un paragrafo, sotto il nome del server.

11.23 `asyncore` — Gestione di socket asincroni

Questo modulo fornisce l'infrastruttura di base per la scrittura di servizi client e server con socket asincroni.

Ci sono solo due sistemi per avere un programma che su di un singolo processore faccia "più di una cosa nello stesso tempo". La programmazione multithread è il più semplice ed il più popolare metodo per fare ciò, ma esiste anche un'altra tecnica, che consente di avere praticamente tutti i vantaggi del multithreading, senza utilizzare thread multipli. È applicabile praticamente solo se il vostro programma richiede un I/O massiccio. Se il programma richiede solo calcolo, la schedulazione di thread preemptive è probabilmente la via migliore. I server di rete, comunque, raramente fanno un uso pesante del processore.

Se il vostro sistema operativo supporta nella propria libreria di I/O la chiamata di sistema `select()` (e tutto quanto ne consegue), la si può utilizzare per gestire canali multipli di comunicazione immediati; svolgendo altri lavori mentre il vostro I/O sta procedendo in "background". Curiosamente, questa strategia può sembrare strana e complessa, specialmente all'inizio, in realtà è sotto molti aspetti facile da capire e da controllare, contrariamente

alla programmazione multithread. Il modulo `asyncore` risolve molti dei problemi difficili, svolgendo il compito di costruire server e client di rete ad alte e sofisticate prestazioni in un attimo. Per applicazioni e protocolli di "conversazione" il modulo di supporto `asynchat` è insostituibile.

L'idea di base dietro ad entrambi i moduli è di creare uno o più *canali* di rete, istanze delle classi `asyncore.dispatcher` e `asynchat.async_chat`. La creazione dei canali li aggiunge ad una mappa globale, usata dalla funzione `loop()` nel caso non si fornisca una propria *mappa*.

Una volta che il canale o i canali è/sono creato/i, la chiamata alla funzione `loop()` attiva il servizio del canale, che continua finché l'ultimo canale (includendo tutti quelli aggiunti alla mappa durante i servizi asincroni) non viene chiuso.

`loop`(\[*timeout*\[, *use_poll*\[, *map*\]\]\])
: Inserisce un ciclo di controllo che termina solo dopo che tutti i canali sono stati chiusi. Tutti gli argomenti sono facoltativi. L'argomento *timeout* imposta il parametro del tempo limite per le chiamate `select()` o `poll()` appropriate, misurato in secondi; il valore predefinito è `30` secondi. Il parametro *use_poll*, se vero, indica che `poll()` deve essere usato come preferenza rispetto a `select()` (il valore predefinito è `False`). Il parametro *map* è un dizionario i cui elementi sono i canali da controllare. Alla chiusura dei canali, questi vengono rimossi da *map*. Se *map* viene omessa, viene utilizzata una mappa globale (il metodo di classe predefinito `__init__()` aggiorna questa mappa – ci si deve assicurare di estendere `__init__()` piuttosto che sovrascriverlo se si vuole mantenere questo comportamento).

  I canali (istanze di `asyncore.dispatcher`, `asynchat.async_chat` e relative sotto classi) possono essere liberamente inserite nella mappa.

class `dispatcher`()
: La classe `dispatcher` è un involucro leggero intorno all'oggetto socket di basso livello. Per renderlo più efficace, possiede una serie di metodi per gestire gli eventi che vengono chiamati dal ciclo asincrono. Altrimenti, può essere trattato come un normale oggetto socket non bloccante.

  Due attributi di classe che possono essere modificati, per aumentare le prestazioni oppure per conservare la memoria.

  `ac_in_buffer_size`
  : La dimensione del buffer di input asincrono (il suo valore predefinito è `4096`).

  `ac_out_buffer_size`
  : La dimensione del buffer di output asincrono (il suo valore predefinito è `4096`).

  Lo scopo degli eventi di basso livello in momenti precisi o in particolari stati della connessione dicono al ciclo asincrono che determinate azioni di alto livello devono prendere il via. Per esempio, se si è chiesto ad un socket di connettersi a un altro host, si sa che la connessione è stata stabilita quando il socket diviene scrivibile per la prima volta (a questo punto si è in grado di scrivere con la certezza di avere successo). Gli eventi di alto livello coinvolti sono:

Evento	Descrizione
`handle_connect()`	Coinvolto dal primo evento di scrittura
`handle_close()`	Coinvolto da un evento di lettura senza che ci siano dati disponibili
`handle_accept()`	Coinvolto da un evento di lettura su di un socket in ascolto

  Durante un processo asincrono, ogni metodo `readable()` e `writable()` dei canali mappati viene usato per determinare se il socket del canale deve essere aggiunto alla lista dei canali selezionati (`select()`) o interrogati (`poll()`) per eventi di lettura e scrittura.

Comunque, l'insieme degli eventi dei canali è maggiore degli eventi di base del socket. L'intero insieme dei metodi che possono essere sovrascritti da nuove classi sono:

`handle_read`()
: Chiamato quando il ciclo asincrono registra una chiamata a `read()` sul socket del canale.

`handle_write`()
: Chiamato quando il ciclo asincrono rileva che un socket scrivibile può essere scritto. Spesso questo metodo implementa il necessario buffer per migliorare le prestazioni. Per esempio:

```
def handle_write(self):
    sent = self.send(self.buffer)
    self.buffer = self.buffer[sent:]
```

handle_expt()
Chiamato quando si è oltre il limite (OOB (NdT: out of band – fuori banda)) dei dati per una connessione su socket. Questo non dovrebbe mai accadere, ma l'OOB viene ancora supportato anche se usato raramente.

handle_connect()
Chiamato quando il socket dell'opener attivo crea una connessione. Può inviare, per esempio, un banner di "benvenuto", o iniziare una negoziazione di protocollo con il punto remoto.

handle_close()
Chiamato quando il socket viene chiuso.

handle_error()
Chiamato quando viene sollevata un'eccezione non altrimenti gestita. La versione predefinita stampa un traceback condensato.

handle_accept()
Chiamato sul canale in ascolto (opener passivo) quando una connessione può essere stabilita con un nuovo destinatario che ha invocato la chiamata `connect()` per il suo punto locale.

readable()
Chiamato ogni volta che durante il ciclo asincrono si deve determinare se un socket di un canale debba essere aggiunto alla lista su cui gli eventi di lettura possono verificarsi. Il metodo predefinito restituisce semplicemente `True`, indicando che, per definizione, tutti i canali verranno interessati all'evento di lettura.

writable()
Chiamato ogni volta che durante il ciclo asincrono si deve determinare se il socket del canale debba essere aggiunto alla lista in cui avvengono gli eventi in scrittura. Il metodo predefinito restituisce semplicemente `True`, indicando per definizione, che tutti i canali verranno interessati dagli eventi di scrittura.

In aggiunta, ogni canale delega o estende molti dei metodi dei socket. Molti di questi sono praticamente identici ai loro corrispondenti nei socket.

create_socket(*family, type***)**
Questo è identico alla creazione di un normale socket e usa le stesse opzioni per la sua creazione. Ci si riferisca alla documentazione del modulo `socket` per le informazioni su come creare socket.

connect(*address***)**
Come con un normale oggetto socket, *address* è una tupla il cui primo elemento è l'host a cui connettersi ed il secondo il numero della porta.

send(*data***)**
Invia *data* al punto remoto del socket.

recv(*buffer_size***)**
Legge tanti byte fino a *buffer_size* dal punto remoto del socket. Una stringa vuota indicata che il canale è stato chiuso dall'altra parte.

listen(*backlog***)**
Ascolta le connessioni fatte sul socket. L'argomento *backlog* specifica il numero massimo di connessioni in coda e deve essere almeno `1`; il numero massimo applicabile è dipendente dal sistema (solitamente `5`).

bind(*address***)**
Associa il socket a *address*. Il socket non deve già essere associato. (Il formato di *address* (NdT: indirizzo) dipende dalla famiglia dell'indirizzo stesso — vedere più avanti.)

accept()
Accetta una connessione. Il socket deve essere associato ad un'indirizzo ed essere in ascolto per le connessioni. Il valore restituito è una coppia `(`*connessione*`,` *indirizzo*`)` dove *connessione* è un nuovo oggetto socket utilizzabile per inviare e ricevere dati sulla connessione e *indirizzo* è l'indirizzo associato al socket all'altra estremità della connessione.

close()
Chiude il socket. Tutte le successive operazioni sull'oggetto socket falliranno. Il punto remoto non riceverà più dati (dopo che i dati in coda verranno scaricati). I socket vengono automaticamente chiusi quando sono riciclati attraverso il meccanismo della garbage-collected.

11.23.1 Esempio di client HTTP di base con asyncore

Come esempio di base, viene mostrato un client HTTP molto semplice che utilizza una classe `dispatcher` per implementare la propria gestione del socket:

```
class http_client(asyncore.dispatcher):
    def __init__(self, host,path):
        asyncore.dispatcher.__init__(self)
        self.path = path
        self.create_socket(socket.AF_INET, socket.SOCK_STREAM)
        self.connect( (host, 80) )
        self.buffer = 'GET %s HTTP/1.0\r\n\r\n' % self.path

    def handle_connect(self):
        pass

    def handle_read(self):
        data = self.recv(8192)
        print data

    def writable(self):
        return (len(self.buffer) > 0)

    def handle_write(self):
        sent = self.send(self.buffer)
        self.buffer = self.buffer[sent:]
```

11.24 `asynchat` — Gestore di comando/risposta su socket asincroni

Questo modulo poggia sull'infrastruttura di `asyncore`, semplificando client e server asincroni e rendendo facile la gestione di protocolli i cui elementi siano terminati da stringhe arbitrarie, o di lunghezza variabile. `asynchat` definisce la classe astratta `async_chat` che sarà la vostra sotto classe, e fornirà l'implementazione dei metodi `collect_incoming_data()` e `found_terminator()`. Utilizza lo stesso ciclo asincrono di `asyncore`, e i due tipi di canale, `asyncore.dispatcher` e `asynchat.async_chat` possono essere liberamente mescolati nella mappa dei canali. Tipicamente un canale di server `asyncore.dispatcher` genera un nuovo oggetto canale `asynchat.async_chat` appena riceve una richiesta di connessione in ingresso.

class `async_chat`()

Questa classe è una classe astratta di `asyncore.dispatcher`. Per farne un uso pratico nel codice, deve essere una sotto classe di `async_chat`, che fornirà funzionali metodi `collect_incoming_data()` e `found_terminator()`. Il metodo `asyncore.dispatcher` può essere usato, anche se non tutto ha senso in un contesto messaggio/risposta.

Come `asyncore.dispatcher`, `async_chat` definisce un insieme di eventi che vengono generati da una analisi delle condizioni del socket dopo una chiamata a `select()`. Una volta che il ciclo di controllo è stato avviato, i metodi dell'oggetto `async_chat` vengono chiamati dalla struttura di gestione eventi senza alcuna azione da parte del programmatore.

Diversamente da `asyncore.dispatcher`, `async_chat` permette di definire una fifo dei *producers*. Un producer necessita di un solo metodo, `more()`, che dovrà restituire dati che saranno trasmessi sul canale. Il producer indica esaurimento (*per esempio* non ci sono più dati) con il proprio metodo `more()` che restituisce una stringa vuota. A questo punto l'oggetto `async_chat` rimuove il producer dalla coda fifo e si avvia usando il successivo producer, se è disponibile. Quando la coda fifo del producer è vuota il metodo `handle_write()` non fa nulla. Utilizzare il metodo `set_terminator()` dell'oggetto canale per descrivere come riconoscere la sua fine, o un importante punto di interruzione, su una trasmissione in arrivo da un punto remoto.

Per costruire una sotto classe `async_chat` funzionale, i metodi `collect_incoming_data()` e

`found_terminator()` devono gestire i dati che il canale riceve in modo asincrono. I metodi vengono descritti più avanti.

`close_when_done``()`
Inserisce un `None` nella coda fifo del producer. Quando questo producer viene estratto dalla coda fifo causerà la chiusura del canale.

`collect_incoming_data``(`*data*`)`
Chiamato con *data*, contiene un'arbitraria quantità di dati ricevuti. Il metodo predefinito, che deve essere sovrascritto, solleva un'eccezione `NotImplementedError`.

`discard_buffers``()`
In emergenza, questo metodo scarica ogni dato mantenuto nei buffer di input e/o output e nella coda della fifo producer.

`found_terminator``()`
Chiamato quando il flusso di dati in entrata corrisponde alla condizione di terminazione imposta da `set_terminator`. Il metodo predefinito, che deve essere sovrascritto, solleva un'eccezione `NotImplementedError`. I dati in input bufferizzati, devono essere disponibili attraverso un attributo dell'istanza.

`get_terminator``()`
Restituisce il terminatore corrente per il canale.

`handle_close``()`
Chiamato quando il canale è chiuso. Il metodo predefinito chiude in modo trasparente e discreto il socket del canale.

`handle_read``()`
Chiamato quando si scatena un evento di lettura sul socket del canale nel ciclo asincrono. Il metodo predefinito verifica la condizione di conclusione stabilita con `set_terminator()`, che può essere sia l'apparire di una particolare stringa nel flusso in entrata o la ricezione di un particolare numero di caratteri. Quando viene rinvenuta la stringa di conclusione, `handle_read` chiama il metodo `found_terminator()` dopo la chiamata a `collect_incoming_data()`, con tutti i dati restanti precedenti la condizione di conclusione.

`handle_write``()`
Chiamato quando l'applicazione deve scrivere dati sul canale. Il metodo predefinito chiama il metodo `initiate_send()`, che, di volta in volta chiama `refill_buffer()` per raccogliere i dati dal buffer fifo del producer associato al canale.

`push``(`*data*`)`
Crea un oggetto `simple_producer` (vedere sotto) contenente i dati e lo inserisce nel `producer_fifo` del canale per assicurare la sua trasmissione. Questo è tutto quello che si deve fare per far si che il canale scriva i dati fuori, verso la rete, altrimenti è possibile utilizzare un proprio producer in un più complesso schema per implementare per esempio criptazione e frammentazione.

`push_with_producer``(`*producer*`)`
Prende un oggetto producer e lo aggiunge alla coda fifo del producer associata con il canale. Quando tutti i producer attualmente inseriti sono esauriti il canale consuma i dati di questo producer chiamando il proprio metodo `more()` ed invia i dati al punto remoto.

`readable``()`
Dovrebbe restituire `True` per il canale che deve essere incluso nell'insieme dei canali testati dal ciclo di `select()` per leggibilità.

`refill_buffer``()`
Rifornisce il buffer di uscita chiamando il metodo `more()` del producer in testa al buffer fifo. Se è esaurito, il producer viene estratto dalla fifo ed il producer successivo viene attivato. Se il producer corrente è, o diviene `None`, il canale viene chiuso.

`set_terminator``(`*term*`)`
Imposta la condizione di conclusione in modo che sia riconosciuta dal canale. `term` può essere uno dei tre tipi di valore, corrispondenti ai tre differenti modi di gestire i dati del protocollo in arrivo.

term	Descrizione
string	Chiamerà `found_terminator()` quando la stringa viene trovata nel flusso in entrata
integer	Chiamerà `found_terminator()` quando saranno stati ricevuti il numero di caratteri indicato
`None`	Il canale continua a raccogliere dati per sempre

Notare che ogni dato seguito dalla stringa di conclusione sarà disponibile per la lettura da parte del canale dopo che verrà chiamato `found_terminator()`.

`writable``()`
Dovrebbe restituire `True` tanto a lungo quanto l'elemento resta nella coda fifo del producer, o il canale è connesso ed il buffer di uscita del canale non è vuoto.

11.24.1 asynchat - Classi ausiliarie e funzioni

class `simple_producer``(`*data*`[`*, buffer_size=512*`])`
Un `simple_producer` prende un segmento di dati ed un parametro facoltativo della dimensione del buffer. Chiamate ripetute al proprio metodo `more()` prelevano successivi blocchi di dati non più larghi di *buffer_size*.

`more``()`
Produce il prossimo blocco di informazioni dal producer, o restituisce la stringa vuota.

class `fifo``([`*list=None*`])`
Ogni canale mantiene un propria coda `fifo` che mantiene i dati che sono stati inseriti dall'applicazione ma non ancora inseriti per la scrittura nel canale. Un `fifo` è una lista usata per mantenere dati e/o producer finché non vengono richiesti. Se l'argomento *list* viene indicato, deve contenere elementi producer o elementi di dati che devono essere scritti nel canale.

`is_empty``()`
Restituisce `True` se la coda fifo è vuota.

`first``()`
Restituisce l'elemento più recente inserito (NdT: `push()`ed) dalla coda fifo.

`push``(`*data*`)`
Aggiunge il dato indicato *data* (che può essere una stringa o un oggetto producer) nella coda fifo del producer.

`pop``()`
Se la coda fifo non è vuota, restituisce `True, first()`, cancellando l'elemento estratto. Restituisce `False, None` per una coda fifo vuota.

Il modulo `asynchat` definisce anche una funzione utile, che può essere usata in rete o in operazioni di analisi testuali.

`find_prefix_at_end``(`*haystack, needle*`)`
Restituisce `True` se la stringa *haystack* termina con nessun prefisso non-vuoto di stringhe *needle*.

11.24.2 Esempi asynchat

Il seguente, parziale, esempio mostra come richieste HTTP possano essere lette con `async_chat`. Un server web deve creare un oggetto `http_request_handler` per ogni connessione client in arrivo. Notare che inizialmente la stringa di conclusione del canale viene impostato per verificare la riga vuota alla fine delle intestazioni HTTP, ed un'opzione indica che le intestazioni sono in fase di lettura.

Quando le intestazioni sono state lette, se la richiesta è di tipo POST (indicante che successivi dati sono presenti nel flusso in ingresso), l'intestazione `Content-Length:` viene usata per impostare una stringa di conclusione numerica per leggere il giusto quantitativo di dati in arrivo dal canale.

Il metodo `handle_request()` viene chiamato una sola volta quando tutti gli input rilevanti sono stati serializzati, subito dopo aver impostato la stringa di conclusione del canale a `None` per assicurarsi che ogni dato estraneo inviato dal cliente web sia ignorato.

```
class http_request_handler(asynchat.async_chat):

    def __init__(self, conn, addr, sessions, log):
        asynchat.async_chat.__init__(self, conn=conn)
        self.addr = addr
        self.sessions = sessions
        self.ibuffer = []
        self.obuffer = ""
        self.set_terminator("\r\n\r\n")
        self.reading_headers = True
        self.handling = False
        self.cgi_data = None
        self.log = log

    def collect_incoming_data(self, data):
        """Il buffer dei dati"""
        self.ibuffer.append(data)

    def found_terminator(self):
        if self.reading_headers:
            self.reading_headers = False
            self.parse_headers("".join(self.ibuffer))
            self.ibuffer = []
            if self.op.upper() == "POST":
                clen = self.headers.getheader("content-length")
                self.set_terminator(int(clen))
            else:
                self.handling = True
                self.set_terminator(None)
                self.handle_request()
        elif not self.handling:
            self.set_terminator(None) # browsers sometimes over-send
            self.cgi_data = parse(self.headers, "".join(self.ibuffer))
            self.handling = True
            self.ibuffer = []
            self.handle_request()
```

CAPITOLO

DODICI

Gestione dei dati internet

Questo capitolo descrive i moduli che supportano la gestione dei dati nel formato comunemente usato su internet.

formatter	Formattatore generico per l'output e interfaccia dispositivo.
email.Iterators	Iterare su un albero di oggetti message.
mailcap	Gestione di file Mailcap.
mailbox	Gestione dei vari tipi di mailbox.
mhlib	Manipolare mailbox MH da Python.
mimetools	Strumenti per analizzare il corpo dei messaggi in stile MIME.
mimetypes	Mappa l'estensione dei nomi dei file al tipo MIME.
MimeWriter	Scrittore generico di file MIME.
mimify	Mimificazione e demimificazione di messaggi di posta.
multifile	Supporto per la lettura di file contenenti parti distinte, come alcuni tipi di dati.
rfc822	Analizza i messaggi di posta elettronica in stile RFC 2822.
base64	RFC 3548: codifiche di dati Base16, Base32, Base64.
binascii	Strumento per convertire tra binario e varie rappresentazioni di codifiche binarie ASCII.
binhex	Codifica e decodifica per file in formato binhex4.
quopri	Codifica e decodifica di dati usando la codifica MIME quoted-printable.
uu	Codifica e decodifica file in formato uuencode.
xdrlib	Codifica e decodifica dati per Rappresentazione di dati Esterni (XDR).
netrc	Caricamento di file '.netrc'.
robotparser	Carica il file 'robots.txt' che risponde ad interrogazioni circa il prelevamento di altre URL.
csv	Lettura e scrittura di dati formattati in tabelle e da file con delimitazioni.

12.1 `formatter` — Formattatore generico per l'output

Questo modulo supporta definizioni di interfacce, ciascuna con implementazioni multiple. L'interfaccia di *formatter* viene utilizzata dalla classe `HTMLParser` del modulo `htmllib` e l'interfaccia *writer* viene richiesta dall'interfaccia di formatter.

Gli oggetti formattatori trasformano un flusso astratto di eventi di formattazione in uno specifico evento di scrittura sugli oggetti scrittori. I formattatori gestiscono molteplici strutture stack per permettere a varie proprietà di un oggetto scrittore di essere modificate e ripristinate; gli scrittori non necessitano di poter gestire i cambiamenti relativi né nessun tipo di operazione "change back" (NdT. modifiche all'indietro). Proprietà specifiche degli scrittori che devono essere controllate dagli oggetti formattatori sono allineamento orizzontale, font ed indentazione del margine sinistro. Un meccanismo fornito supporta la fornitura arbitraria, di stili non esclusivi di configurazioni anche in uno scrittore. Interfacce addizionali facilitano gli eventi di formattazione che non sono reversibili, come la separazione dei paragrafi.

Gli oggetti scrittori incapsulano le interfacce dei dispositivi. I dispositivi astratti, come formati di file, vengono supportati così come i dispositivi fisici. Tutte le implementazione fornite lavorano con i dispositivi astratti. L'interfaccia rende disponibili dei meccanismi per impostare le proprietà che gli oggetti formattatori gestiscono e l'inserimento di dati nell'output.

12.1.1 L'interfaccia formatter

Le interfacce per creare i formattatori dipendono dalla specifica classe di formattatori che sta per essere istanziata. Le interfacce descritte di seguito sono quelle che tutti i formattatori devono supportare una volta inizializzati.

Un elemento dato è definito a livello di modulo:

`AS_IS`
Valore che può essere utilizzato nella specifica dei font passati al metodo `push_font()` descritto di seguito o come nuovo valore di ogni altro metodo `push_`*property*`()`. Eseguire un push del valore `AS_IS` permette al corrispondente metodo `pop_`*property*`()` di essere invocato senza dover rintracciare l'eventuale cambiamento della proprietà.

I seguenti attributi sono definiti per le istanze degli oggetti formatter:

`writer`
Istanza che scrive, con la quale iteragisce il formattatore.

`end_paragraph`(*blanklines*)
Chiude ogni paragrafo aperto ed inserisce almeno una *blanklines* (NdT: riga vuota) prima del prossimo paragrafo.

`add_line_break`()
Aggiunge un'interruzione di riga se non ne esiste ancora una. Questo non interrompe il paragrafo logico.

`add_hor_rule`(**args, **kw*)
Inserisce un separatore orizzontale (NdT: horizontal rule). Un separatore viene inserito se ci sono dei dati nel paragrafo corrente, ma non viene interrotto il paragrafo logico. Gli argomenti e le chiavi vengono passate al metodo `send_line_break()` dello scrittore.

`add_flowing_data`(*data*)
Fornisce *data*, che deve essere formattato collassando i caratteri di spaziatura. I caratteri di spaziatura delle chiamate precedenti e successive a `add_flowing_data()` vengono considerati come se il collassamento sia stato effettuato. Si suppone che i dati passati a questo metodo siano divisi in base alle parole dal dispositivo di output. Notare che ogni divisione sulle parole deve ancora essere effettuata dall'oggetto scrittore, questo perché si deve fare affidamento sulle informazioni su dispositivi e font.

`add_literal_data`(*data*)
Fornisce *data*, che deve essere passato allo scrittore senza alterazioni. I caratteri di spaziatura, incluso i caratteri di fine riga e tabulazione, vengono considerati legali per il valore di *data*.

`add_label_data`(*format, counter*)
Inserisce un'etichetta che deve essere posizionata a sinistra del margine corrente. Questo può essere utilizzato per costruire liste o liste numerate. Se il valore di *format* è una stringa, viene interpretato come il formato specifico per *counter*, che deve essere un intero. Il risultato di questa formattazione diventa il valore dell'etichetta; se *format* non è una stringa, viene utilizzato direttamente come valore dell'etichetta. Il valore dell'etichetta viene passata come unico argomento del metodo `send_label_data()` dello scrittore. L'interpretazione di etichette con valore non corrispondente a stringhe dipende dallo scrittore associato.

Le specifiche di formattazione sono stringhe che, in combinazione con un contatore *counter* dei valori, vengono utilizzate per elaborare il valore dell'etichetta. Ogni carattere nella stringa di formattazione viene copiato nel valore dell'etichetta, con qualche carattere riconosciuto per indicare una trasformazione sul valore di *counter*. In modo specifico, il carattere '`1`' rappresenta il valore del contatore come numero arabo, i caratteri '`A`' e '`a`' corrispondono alla rappresentazione alfabetica del valore del contatore in maiuscolo o miniscolo, rispettivamente, e '`I`' ed '`i`' rappresentano il valore del contatore in numeri romani, in maiuscolo o miniscolo. Notare che le trasformazioni alfabetiche e romane richiedono che il valore del contatore siano maggiori di zero.

`flush_softspace`()
Invia ogni carattere di spaziatura in sospeso bufferizzato da una precedente chiamata a `add_flowing_data()` all'oggetto scrittore associato. Questo deve essere chiamato prima di ogni manipolazione diretta dell'oggetto scrittore.

`push_alignment`(*align*)
Esegue una push (NdT. inserimento) di una nuova impostazione di allineamento nello stack degli allinea-

menti. Questo può essere AS_IS se non si desidera nessun cambiamento. Se il valore dell'allineamento viene modificato dalla precedente impostazione, il metodo new_alignment() dello scrittore viene chiamato con il valore di *align*.

pop_alignment()
Ripristina l'allineamento precedente.

push_font(*size, italic, bold, teletype*)
Cambia alcune o tutte le proprietà dei font di un oggetto scrittore. Le proprietà che non sono impostate ad AS_IS vengono impostate al valore passato, mentre le altre vengono mantenute alle impostazioni attuali. Il metodo new_font() dello scrittore viene chiamato con la specifica del font completamente risolta.

pop_font()
Ripristina i font precedenti.

push_margin(*margin*)
Incrementa il numero di indentazione del margine sinistro di uno, associando il tag del margine logico, *margin*, alla nuova indentazione. Il livello iniziale del margine è 0. I cambiamenti del valore del tag logico devono essere dei valori veri; valori falsi diversi da AS_IS non sono sufficienti per cambiare il margine.

pop_margin()
Ripristina il precedente margine.

push_style(**styles*)
Esegue una push di qualunque numero di specifiche arbitrarie di stili. Tutti gli stili vengono inseriti nello stack in ordine. Una tupla rappresentante l'intero stack, incluso il valore AS_IS, viene passata al metodo new_styles() dello scrittore.

pop_style([*n = 1*])
Estrae l'ultima *n*-esima specificazione di stile passata a push_style(). Una tupla rappresentante lo stack rivisto, incluso il valore AS_IS, viene passata al metodo new_styles() dello scrittore.

set_spacing(*spacing*)
Imposta lo stile di spaziatura per lo scrittore.

assert_line_data([*flag = 1*])
Informa il formatter che i dati sono stati aggiunti al paragrafo corrente fuori banda. Questo deve essere utilizzato quando lo scrittore è stato manipolato direttamente. L'argomento facoltativo *flag* può essere impostato a falso se la manipolazione dello scrittore produce un'interruzione di riga alla fine dell'output.

12.1.2 Implementazione del formattatore

Questo modulo fornisce due implementazioni degli oggetti formattatori. La maggior parte delle applicazioni possono utilizzare una di queste classi senza modificazioni o subclassamento.

class NullFormatter([*writer*])
Un formattatore che non fa niente. Se *writer* viene omesso, viene creata un'istanza di NullWriter. Nessun metodo dello scrittore viene chiamato dall'istanza di NullFormatter. Le implementazioni devono ereditare da questa classe se implementano un'istanza di scrittore, ma non necessitano di ereditare nessuna implementazione.

class AbstractFormatter(*writer*)
Il formattatore standard. Questa implementazione ha dimostrato un'alta applicabilità verso molti scrittori e può essere utilizzata direttamente in molte circostanze. È stata utilizzata per implementare un browser WWW completo.

12.1.3 L'interfaccia dello scrittore

Le interfacce per creare gli scrittori dipendono dalla specifica classe di scrittore che viene istanziata. Le interfacce descritte di seguito sono quelle che tutti gli scrittori devono supportare una volta inizializzati. Notare che mentre molte applicazioni possono usare la classe AbstractFormatter come formattatore, lo scrittore deve tipicamente essere fornito dall'applicazione.

flush()
Svuota ogni output o evento di controllo dei dispositivi bufferizzati.

new_alignment(*align*)
Imposta lo stile di allineamento. Il valore di *align* può essere un oggetto, ma per convenzione è una stringa o `None`, dove `None` indica che deve essere utilizzato l'allineamento "preferito" dello scrittore. I valori convenzionali degli allineamenti sono `'left'`, `'center'`, `'right'` e `'justify'`.

new_font(*font*)
Imposta lo stile del font. Il valore di *font* sarà `None`, ad indicare che deve essere utilizzato il valore predefinito per il font, o una tupla nella forma (*size*, *italic*, *bold*, *teletype*). *size* sarà una stringa ad indicare la dimensione del font che dovrà essere utilizzato; le stringhe specifiche e la loro interpretazione deve essere definita dall'applicazione. I valori *italic*, *bold*, e *teletype* sono valori booleani, a specificare quale di questi attributi del font deve essere utilizzato.

new_margin(*margin, level*)
Imposta il livello del margine all'intero *level* e il tag logico a *margin*. L'interpretazione del tag logico è lasciata alla discrezione dello scrittore; l'unica restrizione sul valore del tag logico è che non può essere un valore falso per valori diversi da zero di *level*.

new_spacing(*spacing*)
Imposta lo stile di spaziatura a *spacing*.

new_styles(*styles*)
Imposta gli stili aggiuntivi. Il valore degli stili *styles* è una tupla di valori arbitrari; il valore `AS_IS` deve essere ignorato. La tupla degli stili può essere interpolata sia come un insieme o come uno stack, in funzione dei requisiti dell'applicazione e dell'implementazione dello scrittore.

send_line_break()
Interrompe la linea corrente.

send_paragraph(*blankline*)
Produce una separazione di paragrafo di almeno una riga vuota, *blankline*, o l'equivalente. Il valore di *blankline* è un intero. Notare che l'implementazione riceverà una chiamata a `send_line_break()` prima di questa chiamata se un'interruzione di linea è necessaria; questo metodo non deve includere la fine dell'ultima linea del paragrafo. È solo responsabile della spaziatura verticale tra i paragrafi.

send_hor_rule(**args, **kw*)
Mostra un separatore orizzontale (horizontal rule) sul dispositivo di output. Gli argomenti di questo metodo sono interamente specifici dell'applicazione e dello scrittore e devono essere interpretati con cura. L'implementazione del metodo può presupporre che un'interruzione di linea sia già stata pubblicata da `send_line_break()`.

send_flowing_data(*data*)
Manda in output i dati di tipo carattere, che possono essere divisi in base alla parola e rimandati come flusso come necessario. All'interno di qualsiasi sequenza di chiamate di questo metodo, lo scrittore può assumere che caratteri di spaziatura multipli siano stati collassati in un solo carattere.

send_literal_data(*data*)
Manda in output i dati di tipo carattere che sono gia stati formattati per essere mostrati. Generalmente, questo deve essere interpretato per indicare che le interruzioni di linea indicate dai caratteri di fine riga debbano essere preservati e nessuna nuova interruzione debba essere introdotta. I dati possono contenere fine riga o caratteri di tabulazione, al contrario dei dati forniti all'interfaccia `send_formatted_data()`.

send_label_data(*data*)
Imposta i dati *data* a sinistra del margine sinistro corrente, se possibile. Il valore di *data* non è ristretto; il trattamento dei valori che non sono stringhe è interamente dipendente dall'applicazione e dallo scrittore. Questo metodo verrà chiamato solamente all'inizio di una riga.

12.1.4 Implementazione dello scrittore

In questo modulo vengono fornite, come esempio, tre implementazioni dell'interfaccia dell'oggetto scrittore. La maggior parte delle applicazioni dovrà derivare nuove classi scrittori dalla classe `NullWriter`.

class `NullWriter`()
Uno scrittore che fornisce solo la definizione dell'interfaccia; nessuna azione viene effettuata in nessun metodo. Questo deve essere la classe base per tutti gli scrittori che non devono ereditare nessuna implementazione dei metodi.

class `AbstractWriter`()
Uno scrittore che può essere usato nei formattatori di debug, ma non per molto altro. Ogni metodo semplicemente si annuncia stampando il proprio nome e gli argomenti sullo standard output.

class `DumbWriter`([*file*[, *maxcol* = *72*]])
Semplice classe di scrittore che scrive l'output sull'oggetto di tipo file passato come *file* o, se *file* viene omesso, sullo standard output. L'output è semplicemente stampato separando le parole in base al numero di colonne specificato da *maxcol*. Questa classe è utilizzabile per riformattare una sequenza di paragrafi.

12.2 `email` — Un package per gestire email e MIME

Nuovo nella versione 2.2.

Il package `email` è una libreria per gestire i messaggi di email, inclusi MIME e gli altri documenti basati sulla RFC 2822. Include la maggior parte delle funzionalità di parecchi altri vecchi moduli della libreria standard, come `rfc822`, `mimetools`, `multifile` ed altri package non standard, come `mimecntl`. *Non* è specificamente progettato per nessun invio di messaggi di posta tramite server SMTP (RFC 2821); questa è la funzione del modulo `smtplib`. Il package `email` prova ad essere il più possibile compatibile con le RFC, supportando oltre alla RFC 2822, alcune RFC collegata a MIME, come RFC 2045-RFC 2047 e RFC 2231.

La caratteristica primaria che distingue il package `email` dagli altri è che divide l'analisi e la generazione del messaggio di email dal *modello oggetto* di rappresentazione interno dell'email. Le applicazioni che utilizzano il package `email` trattano principalmente con oggetti; potete aggiungere sotto oggetti ai messaggi, rimuovere sotto oggetti dai messaggi, riarrangiare completamente il contenuto, etc. etc.. Ci sono sia un parser (NdT: analizzatore) che un generatore, separati, che gestiscono le trasformazioni dal testo al modello dell'oggetto, e poi di nuovo in testo. Sono presenti anche utili sotto classi per alcuni tipi di oggetti MIME comuni, ed alcune utilità generiche che aiutano con alcuni compiti comuni come l'estrazione e l'analisi del valore dei campi, creare la data in formato compatibile con le RFC, etc. etc..

Le seguenti sezioni descrivono le funzionalità del package `email`. L'ordinamento segue una progressione che deve essere comune nelle applicazioni: un messaggio di email viene letto come testo da un file o un'altra fonte, il testo viene analizzato per produrre la struttura dell'oggetto del messaggio di email, questa struttura viene manipolata ed infine renderizzata nuovamente in puro testo.

È perfettamente fattibile creare la struttura dell'oggetto costruendosela, per esempio completamente da zero. Da qui, una progressione simile può essere usata come sopra.

Include anche le specifiche dettagliate di tutte le classi ed i moduli che vengono fornite dal package `email`, le eccezioni che potete incontrare utilizzando il package, alcune facilitazioni ausiliarie e alcuni esempi. Per gli utilizzatori del vecchio package `mimelib`, o le precedenti versioni del package `email`, viene fornita una sezione sulle differenze ed il porting.

Vedete anche:

Modulo `smtplib` (sezione 11.12):
Client per il protocollo SMTP

12.2.1 Rappresentazione di un messaggio email

La classe centrale nel package `email` è la classe `Message`; è la classe base per il modello dell'oggetto `email`. `Message` fornisce le funzionalità basilari per impostare ed interrogare i campi delle intestazioni e per accedere al corpo del messaggio.

Concettualmente, un oggetto `Message` è composto da *headers* (NdT: *intestazioni*) e *payloads* (NdT: *carico utile*). Le intestazioni hanno nomi dei campi e valori in stile RFC 2822 dove i nomi dei campi ed i valori sono separati da un due punti. Il due punti non è né parte del nome del campo né del suo valore.

Le intestazioni vengono immagazzinate e restituite in una forma che ne mantiene la sensibilità a maiuscole e minuscole, ma sono fatte corrispondere in modo non sensibile a queste differenze. Può anche esserci una singola intestazione della busta, anche nota come l'intestazione *Unix-From* o l'intestazione `From_`. Il carico utile può essere sia una stringa, in caso di semplici oggetti messaggio, sia una lista di oggetti `Message` per documenti che contengono MIME (per esempio multipart/* e messaggi/rfc822).

Gli oggetti `Message` forniscono un'interfaccia in stile mappa per accedere alle intestazioni dei messaggi ed una specifica interfaccia per accedere sia alle intestazioni che al carico utile. Fornisce metodi utili per generare una rappresentazione dell'albero degli oggetti di messaggio, per accedere ai parametri delle intestazioni acceduti comunemente e per navigare ricorsivamente nell'albero degli oggetti.

Questi sono i metodi della classe `Message`:

class `Message`()
Il costruttore non accetta argomenti.

`as_string`([*unixfrom*])
Restituisce l'intero messaggio in una stringa. Quando l'argomento facoltativo *unixfrom* è `True`, l'intestazione della busta viene inclusa nella stringa restituita. Il valore predefinito di *unixfrom* è `False`.

Notare che questo metodo viene fornito per comodità ma non può sempre formattare il messaggio nel modo voluto. Per avere maggiore flessibilità, istanziare un'istanza di `Generator` ed utilizzare direttamente il suo metodo `flatten()`. Per esempio:

```
from cStringIO import StringIO
from email.Generator import Generator
fp = StringIO()
g = Generator(fp, mangle_from_=False, maxheaderlen=60)
g.flatten(msg)
text = fp.getvalue()
```

`__str__`()
Equivalente a `as_string(unixfrom=True)`.

`is_multipart`()
Restituisce `True` se il carico utile del messaggio è una lista di oggetti sotto `Message`, altrimenti restituisce `False`. Quando `is_multipart()` restituisce `False`, il carico utile deve essere un oggetto di tipo stringa.

`set_unixfrom`(*unixfrom*)
Imposta l'intestazione della busta ad *unixfrom*, che deve essere una stringa.

`get_unixfrom`()
Restituisce l'intestazione della busta del messaggio. Il valore predefinito è `None` se nessuna intestazione della busta è mai stata impostata.

`attach`(*payload*)
Aggiunge il carico utile, *payload*, fornito al carico utile corrente, che deve essere `None` o una lista di oggetti `Message` prima della chiamata. Dopo la chiamata, il carico utile sarà sempre una lista di oggetti di tipo `Message`. Se si vuole impostare il carico utile a un oggetto scalare (tipo una stringa), utilizzare invece `set_payload()`.

`get_payload`([*i*[, *decode*]])
Restituisce un riferimento al carico utile attuale, che deve essere una lista di oggetti `Message` quando `is_multipart()` è `True` o una stringa quando `is_multipart()` è `False`. Se il carico utile è una lista e viene modificato l'oggetto lista si modifica anche il carico utile dell'oggetto.

Con l'argomento facoltativo *i*, `get_payload()` restituisce l'*i*-esimo elemento del carico utile, contando da zero, se `is_multipart()` è `True`. Viene sollevata l'eccezione `IndexError` se *i* è minore di 0 o maggiore o uguale al numero di oggetti nel carico utile. Se il carico utile è una stringa (`is_multipart()` è `False`) e *i* viene fornito, viene sollevata l'eccezione `TypeError`.

L'opzione facoltativa *decode* indica se il carico utile deve essere decodificato o no, in accordo con l'intestazione Content-Transfer-Encoding:. Quando il valore è `True` ed il messaggio non è multipart, il carico utile viene decodificato se il valore dell'intestazione è '`quoted-printable`' o '`base64`'. Se altre codifiche vengono utilizzate o l'intestazione Content-Transfer-Encoding: manca o il carico utile ha dati base64 falsi, il

carico utile viene restituito così com'è (non decodificato). Se il messaggio è multipart ed l'opzione *decode* è impostata a `True`, viene restituito `None`. Il valore predefinito per *decode* è `False`.

`set_payload`(*payload*[, *charset*])
Imposta l'intero carico utile dell'oggetto messaggio a *payload*. È responsabilità del client assicurare l'invarianza del carico utile. Il *charset* facoltativo imposta il charset predefinito per il messaggio; vedere `set_charset()` per i dettagli.

Modificato nella versione 2.2.2: Aggiunto l'argomento *charset*.

`set_charset`(*charset*)
Imposta il charset del carico utile a *charset*, che può essere sia un'istanza di `Charset` (vedere `email.Charset`), sia una stringa che nomina un charset o `None`. Se è una stringa, verrà convertita ad un'istanza di `Charset`. Se *charset* è `None`, il parametro *charset* viene rimosso dall'intestazione Content-Type:. Qualunque altra cosa solleva l'eccezione `TypeError`.

Il messaggio si assume essere di tipo text/* codificato con `charset.input_charset`. Verrà convertito in `charset.output_charset` e codificato opportunamente, se necessario, quando verrà generata la rappresentazione in testo del messaggio. Le intestazioni MIME (MIME-Version:, Content-Type:, Content-Transfer-Encoding:) verranno aggiunte quando necessario.

Nuovo nella versione 2.2.2.

`get_charset`()
Restituisce l'istanza di `Charset` associata al carico utile del messaggio. Nuovo nella versione 2.2.2.

I seguenti metodi implementano un'interfaccia in stile mappa per accedere alle intestazioni RFC 2822 del messaggio. Si noti che ci sono alcune differenze semantiche tra questi metodi e l'interfaccia delle normali mappe (dizionari). Per esempio, in un dizionario non ci sono chiavi duplicate, ma qui ci possono essere intestazioni duplicate. Inoltre, nei dizionari non c'è garanzia di ordinamento per le chiavi restituite da `keys()`, ma in un oggetto `Message`, le intestazioni vengono sempre restituite nell'ordine in cui appaiono nel messaggio originale, o aggiunte al messaggio in seguito. Ogni intestazione cancellata e poi reinserita viene sempre aggiunta alla fine della lista delle intestazioni.

Queste differenze semantiche sono intenzionali e sono polarizzate verso la massima convenienza.

Notare che in tutti i casi, ogni intestazione della busta presente nel messaggio non viene inclusa nella mappa di interfaccia.

`__len__`()
Restituisce il numero totale delle intestazioni, incluse quelle duplicate.

`__contains__`(*name*)
Restituisce *True* se l'oggetto messaggio ha un campo chiamato *name*. La corrispondenza viene effettuata in modo non sensibile alle differenze tra maiuscole e minuscole e *name* non deve includere i due punti terminali. Utilizzato per l'operatore `in`, per esempio:

```
if 'message-id' in myMessage:
    print 'Message-ID:', myMessage['message-id']
```

`__getitem__`(*name*)
Restituisce il valore del campo dell'intestazione nominata. *name* non deve includere il separatore di campi, i due punti. Se l'intestazione non è presente, viene restituito `None`; l'eccezione `KeyError` non viene mai sollevata.

Notare che se il campo nominato appare più di una volta nelle intestazioni del messaggio, non è definito quale di questi verrà restituito. Utilizzare il metodo `get_all()` per ottenere i valori di tutte le intestazioni nominate.

`__setitem__`(*name*, *val*)
Aggiunge una intestazione al messaggio con il campo chiamato *name* ed il valore *val*. Il campo viene aggiunto alla fine della lista dei campi del messaggio esistenti.

Notare che questo *non* sovrascrive o cancella nessuna intestazione esistente con lo stesso nome. Se si vuole l'assicurazione che la nuova intestazione sia l'unica presente nel messaggio con quel nome di campo, cancellare il campo prima dell'inserimento, per esempio:

```
del msg['subject']
msg['subject'] = 'Python roolz!'
```

__delitem__(*name*)
Cancella tutte le occorrenze del campo con il nome *name* dalle intestazioni del messaggio. Non viene sollevata alcuna eccezione se il nome del campo non è presente nelle intestazioni.

has_key(*name*)
Restituisce *True* se il messaggio contiene nell'intestazione un campo chiamato *name*, altrimenti restituisce *False*.

keys()
Restituisce una lista di tutti i nomi dei campi delle intestazioni del messaggio.

values()
Restituisce una lista di tutti i valori dei campi del messaggio.

items()
Restituisce una lista di tuple di 2 elementi contenenti tutte le intestazioni ed i valori dei campi del messaggio.

get(*name*[, *failobj*])
Restituisce il valore dei campi delle intestazioni nominate. Questo è identico a `__getitem__()` ad eccezione che l'argomento facoltativo *failobj* viene restituito se non è presente l'intestazione nominata (il cui valore predefinito è `None`).

Qui di seguito altri metodi utili:

get_all(*name*[, *failobj*])
Restituisce una lista di tutti i valori per il campo chiamato *name*. Se non c'è questa intestazione nel messaggio, viene restituito *failobj* (il cui valore predefinito è `None`).

add_header(*_name, _value, **_params*)
Impostazione avanzata delle intestazioni. Questo metodo è simile a `__setitem__()` ad eccezione che i parametri addizionali dell'intestazione possono essere forniti come argomenti chiave. *_name* è il campo dell'intestazione da aggiungere e *_value* è il valore *primario* per l'intestazione.

Per ogni oggetto nel dizionario degli argomenti di tipo keyword *_params*, la chiave viene presa come il nome del parametro, con i caratteri di trattino basso convertiti in punti (poiché i punti sono illegali come identificatori Python). Normalmente, il parametro viene aggiunto come `chiave=valore` a meno che *value* non sia `None`, in tal caso solo la chiave verrà aggiunta.

Ecco un esempio:

```
msg.add_header('Content-Disposition', 'attachment', filename='bud.gif')
```

Questo aggiunge un'intestazione che avrà questa forma:

```
Content-Disposition: attachment; filename="bud.gif"
```

replace_header(*_name, _value*)
Sostituisce un header. Sostituisce la prima intestazione trovata nel messaggio che corrisponde a *_name*, mantenendo l'ordine delle intestazioni e maiuscole/minuscole dei nomi dei campi. Se non ci sono intestazioni corrispondenti, viene sollevata l'eccezione `KeyError`.

Nuovo nella versione 2.2.2.

get_content_type()
Restituisce il content type (NdT: tipologia del contenuto) del messaggio. La stringa restituita viene convertita in minuscolo nella forma maintype/subtype. Se nel messaggio non c'è l'intestazione Content-Type:, viene restituito il tipo predefinito, come specificato da `get_default_type()`. In accordo con l'RFC 2045, i messaggi hanno sempre un content type predefinito, per questo `get_content_type()` restituisce sempre un valore.

La RFC 2045 definisce il tipo predefinito del messaggio come text/plain (NdT: puro testo), a meno che non appaia all'interno di un contenitore multipart/digest, in tal caso sarà message/rfc822. Se l'intestazione Content-Type: ha una specifica errata per il tipo, la RFC 2045 stabilisce che il tipo predefinito sia text/plain.

Nuovo nella versione 2.2.2.

get_content_maintype()
Restituisce il content type principale del messaggio. Questa è la parte maintype della stringa restituita da `get_content_type()`.

Nuovo nella versione 2.2.2.

get_content_subtype()
Restituisce il sotto content type del messaggio. Questa è la parte subtype della stringa restituita da `get_content_type()`.

Nuovo nella versione 2.2.2.

get_default_type()
Restituisce il content type predefinito. La maggior parte dei messaggi hanno un content type predefinito text/plain, ad eccezione dei messaggi che sono sotto parti di un contenitore multipart/digest. Questa sotto parte ha un content type predefinito message/rfc822.

Nuovo nella versione 2.2.2.

set_default_type(*ctype*)
Imposta il content type predefinito. *ctype* deve essere text/plain o message/rfc822, altrimenti non viene fatto rispettare. Il content type predefinito non viene immagazzinato nell'intestazione Content-Type:.

Nuovo nella versione 2.2.2.

get_params([*failobj*[, *header*[, *unquote*]]])
Restituisce il parametro Content-Type: del messaggio come una lista. Gli elementi della lista restituita sono delle tuple di 2 elementi costituiti da coppie chiave/valore, derivate dallo split dei valori separati dal segno '='. La parte a sinistra del segno '=' è la chiave, la parte a destra è il valore. Se non c'è il segno '=' nel parametro, il valore è una stringa vuota, altrimenti il valore è come descritto da `get_param()` ed è non quotato se l'argomento facoltativo *unquote* è `True` (il valore predefinito).

L'argomento facoltativo *failobj* è l'oggetto restituito se non c'è l'intestazione Content-Type:. L'argomento facoltativo *header* è l'intestazione da ricercare al posto di Content-Type:.

Modificato nella versione 2.2.2: aggiunto l'argomento *unquote*.

get_param(*param*[, *failobj*[, *header*[, *unquote*]]])
Restituisce il valore del parametro *param* dell'intestazione Content-Type: come una stringa. Se il messaggio non ha un'intestazione Content-Type: o se non c'è questo parametro, viene restituito *failobj* (il valore predefinito è `None`).

Se viene fornito l'argomento facoltativo *header*, specifica l'intestazione del messaggio da utilizzare al posto di Content-Type:.

Le chiavi dei parametri sono sempre confrontate in modo non sensibile alle differenze tra maiuscole e minuscole. Il valore restituito può essere una stringa o una tupla di 3 elementi se il parametro è codificato secondo la RFC 2231. Quando è una tupla di 3 elementi, gli elementi del valore sono nella forma `(CHARSET, LANGUAGE, VALUE)`. Notare che sia`CHARSET` che `LANGUAGE` possono essere `None`, in qual caso doveva considerare `VALUE` come codificato nel charset `us-ascii`. Si può generalmente ignorare `LANGUAGE`.

Un'applicazione deve essere pronta a gestire i valori restituiti come tuple di 3 elementi, e deve poter convertire il parametro in una stringa Unicode, tipo:

```
param = msg.get_param('foo')
if isinstance(param, tuple):
    param = unicode(param[2], param[0] or 'us-ascii')
```

In ogni caso, il parametro *value* (sia restituito come stringa o l'oggetto VALUE nella tupla di 3 elementi) è sempre non quotato, a meno che *unquote* non sia impostato a `False`.

Modificato nella versione 2.2.2: aggiunto l'argomento *unquote* e la tupla di 3 elementi restituisce i valori possibili.

set_param(*param*, *value*[, *header*[, *requote*[, *charset*[, *language*]]]])
Imposta un parametro nell'intestazione Content-Type:. Se il parametro esiste nell'intestazione, il valore

viene sostituito con *value*. Se l'intestazione Content-Type: non è ancora stata definita per questo messaggio, viene impostata a text/plain, ed il nuovo parametro viene aggiunto in coda come per l'RFC 2045.

L'*header* facoltativo specifica un'intestazione alternativa a Content-Type: e tutti i parametri verranno quotati come necessario a meno che il facoltativo *requote* sia `False` (il valore predefinito è `True`).

Se il facoltativo *charset* viene specificato, il parametro verrà codificato secondo l'RFC 2231. Il facoltativo *language* specifica la lingua secondo la RFC 2231, il valore predefinito è la stringa vuota. Sia *charset* che *language* devono essere stringhe.

Nuovo nella versione 2.2.2.

`del_param`(*param*[, *header*[, *requote*]])
Rimuove completamente il parametro fornito dall'intestazione Content-Type:. L'intestazione viene riscritta in linea senza il parametro o il suo valore. Tutti i valori verranno quotati se necessario, a meno che *requote* sia `False` (il valore predefinito è `True`). Il facoltativo *header* specifica un'alternativa a Content-Type:.

Nuovo nella versione 2.2.2.

`set_type`(*type*[, *header*][, *requote*])
Imposta il tipo principale e il sotto tipo per l'intestazione Content-Type:. *type* dev'essere una stringa nella forma maintype/subtype, altrimenti viene sollevata un'eccezione `ValueError`.

Questo metodo sostituisce l'intestazione Content-Type:, mantenendo tutti i parametri in linea. Se *requote* è `False`, questo lascia la quotatura esistente delle intestazioni invariata, altrimenti il parametro verrà quotato (il comportamento predefinito).

Un'intestazione alternativa può essere specificata nell'argomento *header*. Quando l'intestazione Content-Type: viene impostata, viene anche aggiunta un'intestazione MIME-Version:.

Nuovo nella versione 2.2.2.

`get_filename`([*failobj*])
Restituisce il valore del parametro `filename` dell'intestazione Content-Disposition: del messaggio o *failobj* se l'intestazione è assente o non ha un parametro `filename`. Alla stringa restituita viene sempre rimossa la quotatura come da `Utils.unquote()`.

`get_boundary`([*failobj*])
Restituisce il valore del parametro `boundary` (NdT: limite) dell'intestazione Content-Type: del messaggio o *failobj* se l'intestazione è assente o non ha un parametro `boundary`. Alla stringa restituita viene sempre rimossa la quotatura come da `Utils.unquote()`.

`set_boundary`(*boundary*)
Imposta il parametro `boundary` dell'intestazione Content-Type: a `boundary`. `set_boundary()` quoterà sempre `boundary` se necessario. Viene sollevata l'eccezione `HeaderParseError` se l'oggetto del messaggio non contiene l'intestazione Content-Type:.

Notare che usare questo metodo è sottilmente differente che non cancellare la vecchia intestazione Content-Type: ed aggiungerne una nuova con il nuovo `boundary` tramite `add_header()`, poiché `set_boundary()` mantiene l'ordine dell'intestazione Content-Type: nella lista delle intestazioni. Comunque, *non* mantiene alcuna soluzione di continuità che poteva essere presente nell'intestazione Content-Type: originale.

`get_content_charset`([*failobj*])
Restituisce il parametro `charset` dell'intestazione Content-Type: convertito in minuscolo. Se non c'è un'intestazione Content-Type: o se questa intestazione non ha un parametro `charset` viene restituito *failobj*.

Notare che questo metodo differisce da `get_charset()` che restituisce l'istanza di `Charset` per la codifica predefinita per il corpo del messaggio.

Nuovo nella versione 2.2.2.

`get_charsets`([*failobj*])
Restituisce una lista contenente il nome dell'insieme di caratteri nel messaggio. Se il messaggio è multipart, la lista conterrà un elemento per ogni sotto parte del carico utile, altrimenti restituirà una lista di lunghezza 1.

Ogni elemento nella lista sarà una stringa corrispondente al valore del parametro `charset` dell'intestazione Content-Type: per la sotto parte rappresentata. Comunque, se la sotto parte non ha un header Content-Type:,

non ha un parametro `charset` o non è il text del tipo MIME principale, l'elemento nella lista restituita sarà *failobj*.

`walk``()`

Il metodo `walk()` è un generatore generico che può essere utilizzato per iterare su tutte le parti e le sotto parti dell'albero degli oggetti del messaggio, in ordine di attraversamento in profondità. Tipicamente si utilizzerà `walk()` come iteratore in un ciclo `for`; ogni iterazione restituisce la prossima sotto parte.

Ecco un esempio che stampa il tipo di MIME di tutte le parti di una struttura di messaggio multipart:

```
>>> for part in msg.walk():
>>>     print part.get_content_type()
multipart/report
text/plain
message/delivery-status
text/plain
text/plain
message/rfc822
```

Gli oggetti `Message` possono, facoltativamente, contenere due attributi delle istanze, che possono essere utilizzate quando si genera la versione testuale di un messaggio MIME.

`preamble`

Il formato di un documento MIME permette che in alcuni testi possano essere presenti righe vuote che seguono le intestazioni, e la prima stringa del limite multipart. Normalmente, questo testo non è visibile in una lettore di mail capace di comprendere i MIME, poiché cade al di fuori dello standard MIME. Comunque, quando si visualizza il testo grezzo del messaggio o quando si visualizza il messaggio in un lettore che non comprende i MIME, questo testo può diventare visibile.

L'attributo *preamble* contiene questo testo iniziale, fuori dagli standard, per il documento MIME. Quando l'analizzatore (`Parser`) scopre del testo dopo le intestazioni ma prima della prima stringa di limite, assegna questo testo all'attributo *preamble*. Quando il generatore `Generator` sta scrivendo la rappresentazione in testo del messaggio MIME, e vede che il messaggio ha un attributo *preamble*, scriverà questo testo nell'area tra le intestazioni ed il primo limite. Vedere `email.Parser` ed `email.Generator` per i dettagli.

Notare che se l'oggetto messaggio non ha un preambolo, l'attributo *preamble* varrà `None`.

`epilogue`

L'attributo *epilogue* agisce nello stesso modo dell'attributo *preamble*, ad eccezione che contiene il testo che compare tra l'ultimo limite e la fine del messaggio.

Una nota: quando si genera la versione testuale di un messaggio multipart che non ha *epilogue* (utilizzando la classe standard `Generator`), nessun fine riga viene aggiunto dopo la riga del limite di chiusura. Se nell'oggetto del messaggio è presente *epilogue* e questo valore non inizia con un fine riga, un fine riga viene stampato dopo il limite di chiusura. Questo parrebbe un po' malcostruito, ma è la cosa più logica. Il risultato è che se si vuole assicurare che un fine riga sia stampato dopo il limite di chiusura del multipart, si deve impostare *epilogue* ad una stringa vuota.

Metodi deprecati

I seguenti metodi sono deprecati nella versione 2 di `email`. Sono documentati per completezza.

`add_payload``(`*payload*`)`

Aggiunge *payload* al carico utile esistente dell'oggetto messaggio. Se, prima di chiamare questo metodo, il carico utile dell'oggetto era `None` (per esempio non era ancora stata impostata), dopo la chiamata di questo metodo, il carico utile sarà l'argomento *payload*.

Se il carico utile dell'oggetto era già una lista (per esempio `is_multipart()` restituisce 1), *payload* verrà aggiunto alla fine della lista esistente.

Per ogni altro tipo di carico utile esistente, `add_payload()` trasformerà il nuovo carico utile in una lista consistente del vecchio carico utile e *payload*, ma solo se il documento è già del tipo MIME multipart. Questa condizione viene soddisfatta se il tipo principale dell'intestazione Content-Type: del messaggio

è multipart o non c'è un'intestazione Content-Type:. In tutti gli altri casi, viene sollevata un'eccezione `MultipartConversionError`.

Deprecato dalla versione 2.2.2. Utilizzare invece il metodo `attach()`.

`get_type`([*failobj*])
Restituisce il content type del messaggio, come una stringa nella forma maintype/subtype come dall'intestazione Content-Type:. La stringa restituita viene convertita in minuscolo.

Se non c'è un'intestazione Content-Type: nel messaggio, viene restituito *failobj* (il valore predefinito è `None`).

Deprecato dalla versione 2.2.2. Utilizzare invece il metodo `get_content_type()`.

`get_main_type`([*failobj*])
Restituisce il content type *principale* del messaggio. Questo essenzialmente restituisce la parte *maintype* della stringa restituita da `get_type()`, con la stessa semantica per *failobj*.

Deprecato dalla versione 2.2.2. Utilizzare invece il metodo `get_content_maintype()`.

`get_subtype`([*failobj*])
Restituisce il sotto content type del messaggio. Questo essenzialmente restituisce la parte *subtype* della stringa restituita da `get_type()`, con la stessa semantica per *failobj*.

Deprecato dalla versione 2.2.2. Utilizzare invece il metodo `get_content_subtype()`.

12.2.2 Analisi di un messaggio email

La struttura degli oggetti messaggio può essere creata in uno o due modi: possono essere realizzati dall'interno, istanziando oggetti di tipo `Message` e sequenziandoli tramite chiamate di `attach()` e `set_payload()`, o possono essere creati analizzando una rappresentazione in puro testo di un messaggio email.

Il package `email` fornisce un parser (NdT: analizzatore) standard che comprende la maggior parte della struttura dei documenti email, inclusi i documenti MIME. Si può passare al parser una stringa o un oggetto di tipo file ed il parser restituirà l'istanza principale di `Message` della struttura dell'oggetto. Per messaggi semplici, non MIME, il carico utile di questo oggetto principale sarà simile ad una stringa contenente il testo del messaggio. Per messaggi MIME, l'oggetto principale restituirà `True` tramite il suo metodo `is_multipart()` e le sotto parti saranno accessibili tramite i metodi `get_payload()` e `walk()`.

Notare che il parser può essere esteso in modo illimitato ed ovviamente è possibile implementare un parser partendo da zero. Non ci sono connessioni magiche tra il parser allegato al package `email` e la classe `Message`, quindi un parser personalizzato può creare alberi di oggetti messaggio in ogni modo si ritenga necessario.

Il parser primario è `Parser`, che analizza sia l'intestazione che il carico utile del messaggio. Nel caso di messaggi multipart, analizzerà ricorsivamente il corpo del messaggio contenitore. Sono supportate due modalità di analisi, quella *rigorosa*, che generalmente rigetterà ogni messaggio non compatibile con l'RFC, e quella *lasca*, che proverà a sistemare i problemi di formattazione di MIME più comuni.

Il modulo `email.Parser` fornisce anche una seconda classe, chiamata `HeaderParser` che può essere utilizzata se si è solo interessati nelle intestazioni del messaggio. `HeaderParser` può essere molto più veloce in queste situazioni, poiché non prova ad analizzare il corpo del messaggio, in sostituzione imposta il carico utile al corpo del messaggio grezzo, come una stringa. `HeaderParser` ha la stessa API della classe classParser.

API per la classe Parser

class `Parser`([*_class*[, *strict*]])
Il costruttore per la classe `Parser` riceve un argomento facoltativo *_class*. Questo deve essere interno al costruttore e chiamabile (come una funzione o una classe) e viene utilizzato quando l'oggetto sotto messaggio deve essere creato. Il valore predefinito è `Message` (vedere `email.Message`). La funzionalità interna verrà chiamata senza argomenti.

L'opzione facoltativa *strict* specifica se dovrà essere effettuata un'analisi rigorosa o lasca. Normalmente, quando cose come i limiti di MIME mancano o quando i messaggi hanno problemi di formattazione, `Parser` solleva un'eccezione `MessageParseError`. Comunque, quando viene abilitata l'analisi lasca, `Parser` prova a lavorare su questi errori di formattazione per generare una struttura del messaggio

utilizzabile (questo non significa che `MessageParseError` non verrà mai sollevata; alcuni messaggi formattati male non possono semplicemente essere analizzati). Il valore predefinito dell'opzione *strict* è `False`, poichè l'analisi lasca fornisce generalmente un comportamento più conveniente.

Modificato nella versione 2.2.2: È stata aggiunta l'opzione *strict*.

Gli altri metodi pubblici di `Parser` sono:

`parse`(*fp*[, *headersonly*])
Legge tutti i dati dall'oggetto di tipo file *fp*, analizza il testo risultante e restituisce l'oggetto principale del messaggio. *fp* deve supportare i metodi `readline()` e `read()` sugli oggetti simile a file.

Il testo contenuto in *fp* deve essere formattato come un blocco di intestazioni in stile RFC 2822 e righe di continuazione delle intestazioni, facoltativamente precedute da un'intestazione della busta. Il blocco delle intestazioni viene terminato dalla fine dei dati o da una riga vuota. A seguire il blocco delle intestazioni c'è il corpo del messaggio (che può contenere sotto parti codificate come MIME).

L'opzione facoltativa *headersonly* è come con il metodo `parse()`.

Modificato nella versione 2.2.2: È stata aggiunta l'opzione *headersonly*.

`parsestr`(*text*[, *headersonly*])
Simile al metodo `parse()`, ad eccezione che riceve un oggetto stringa invece che un oggetto simile a file. Chiamare questo metodo su una stringa è esattamente equivalente ad incapsulare *text* in un'istanza `StringIO` prima e chiamare `parse()`.

L'opzione facoltativa *headersonly* specifica se interrompere l'analisi dopo la lettura delle intestazioni o no. Il valore predefinito è `False`, a significare che analizzerà l'intero contenuto del file.

Modificato nella versione 2.2.2: È stata aggiunta l'opzione *headersonly*.

Poiché la realizzazione di una struttura di un oggetto di tipo messaggio da una stringa o da un file è una cosa molto comune, vengono fornite, per convenienza, due funzioni. Sono disponibili nello spazio dei nomi di livello più alto del package `email`.

`message_from_string`(*s*[, *_class*[, *strict*]])
Restituisce un oggetto struttura del messaggio da una stringa. Questo è esattamente equivalente a `Parser().parsestr(s)`. Gli argomenti facoltativi *_class* e *strict* vengono interpretati come nel costruttore della classe `Parser`.

Modificato nella versione 2.2.2: È stata aggiunta l'opzione *strict*.

`message_from_file`(*fp*[, *_class*[, *strict*]])
Restituisce una struttura ad albero di oggetti messaggio da un oggetto file aperto. Questo è esattamente equivalente a `Parser().parse(fp)`. Gli argomenti facoltativi *_class* e *strict* vengono interpretati come nel costruttore della classe `Parser`.

Modificato nella versione 2.2.2: È stata aggiunta l'opzione *strict*.

Ecco un esempio di come si può utilizzare questo in una sessione interattiva di Python:

```
>>> import email
>>> msg = email.message_from_string(myString)
```

Note aggiuntive

Ecco alcune note sulla semantica di analisi:

- La maggior parte dei messaggi di tipo non multipart vengono elaborati come un singolo oggetto messaggio con un carico utile di tipo stringa. Questi oggetti restituiscono `False` per `is_multipart()`. Il loro metodo `get_payload()` restituisce un oggetto di tipo stringa.
- Tutti i messaggi di tipo multipart vengono elaborati come un oggetto di tipo contenitore con una lista di oggetti sotto messaggio per il loro carico utile. Il messaggio contenitore esterno restituirà `True` per `is_multipart()` ed il loro metodo `get_payload()` restituirà la lista della sotto parti di `Message`.

- La maggior parte dei messaggi con content type di message/* (per esempio message/deliver-status e message/rfc822) verranno anche analizzati come oggetti contenitori contenenti una lista di carico utile di lunghezza 1. Il metodo `is_multipart()` restituirà `True`. Il singolo elemento nella lista di carico utile sarà un oggetto di tipo sub-message.

12.2.3 Generare documenti MIME

Una delle operazioni più comuni è generare la versione testuale del messaggio di email rappresentato da una struttura di oggetti messaggio. È necessario farlo per inviare il messaggio attraverso i moduli `smtplib` o `nntplib`, o stampare il messaggio sulla console. Prendere la struttura degli oggetti messaggio e generare un documento testuale è il lavoro della classe `Generator`.

Ancora, come con il modulo `email.Parser`, non si è limitati alle funzionalità del generatore incluso; è possibile scriverne uno da zero. Comunque il generatore incluso sa come generare la maggior parte delle email in un modo compatibile con lo standard, dovrebbe gestire messaggi MIME e non MIME in modo adeguato ed è progettato in modo che la trasformazione da testo a struttura del messaggio attraverso la classe `Parser` e di nuovo in testo sia ugualmente adeguata (l'input è identico all'output).

Questi sono i metodi pubblici della classe `Generator`:

class `Generator`(*outfp*[, *mangle_from_*[, *maxheaderlen*]])

Il costruttore della classe `Generator` riceve in input un oggetto di tipo file chiamato *outfp* come argomento. *outfp* deve supportare il metodo `write()` ed essere utilizzabile come file di output in un'istruzione print di Python.

Il parametro facoltativo *mangle_from_* è un'opzione che, quando `True`, inserisce il carattere '`>`' di fronte a tutte le linee nel corpo dell'email che iniziano esattamente con '`From `', per esempio `From` seguito da uno spazio all'inizio della riga. Questo è il solo modo garantito come portabile per evitare che queste righe siano scambiate per un separatore dell'intestazione della busta in formato mailbox Unix (vedere WHY THE CONTENT-LENGTH FORMAT IS BAD per i dettagli). Il valore predefinito di *mangle_from_* è `True`, ma si potrebbe volerlo impostare a `False` se non si sta scrivendo un file in formato UNIX mailbox.

L'argomento facoltativo *maxheaderlen* specifica la lunghezza massima per un'intestazione non continuativa. Quando una riga di intestazione è più lunga di *maxheaderlen* (in caratteri, con i caratteri di tabulazione espansi ad 8 spazi) l'intestazione viene separata come definito nella classe `email.Header`. Impostare a zero per disabilitare la separazione delle intestazioni. Il valore predefinito è 78, come raccomandato (ma non richiesto) dall'RFC 2822.

Gli altri metodi pubblici di `Generator` sono:

`flatten`(*msg*[, *unixfrom*])

Stampa la rappresentazione testuale della struttura degli oggetti messaggio originati da *msg* al file in output specificato quando l'istanza `Generator` è stata creata. Le sotto parti sono visitate in profondità e il testo risultante verrà propriamente codificato in MIME.

L'argomento facoltativo *unixfrom* è un'opzione che forza la stampa del delimitatore dell'intestazione della busta prima dell'intestazione RFC 2822 dell'oggetto messaggio principale. Se l'oggetto principale non ha un'intestazione di busta, ne viene messa una standard. Il valore predefinito è `False`, per inibire la stampa del delimitatore di busta.

Notare che per le sotto parti, non viene stampata alcuna intestazione di busta.

Nuovo nella versione 2.2.2.

`clone`(*fp*)

Restituisce un clone indipendente dell'istanza di `Generator`, con le stesse identiche opzioni.

Nuovo nella versione 2.2.2.

`write`(*s*)

Scrive la stringa *s* nell'oggetto file sottostante, per esempio *outfp* passato al costruttore di `Generator`. Questo fornisce un'interfaccia sufficientemente simile a file a `Generator`, da poter essere usata in un istruzione print estesa.

Per convenienza, vedere i metodi `Message.as_string()` e `str(aMessage)`, a.k.a.

`Message.__str__()`, che semplificano la generazione di una stringa formattata che rappresenta un oggetto messaggio. Per maggiori dettagli, vedere `email.Message`.

Il modulo `email.Generator` fornisce anche una classe derivata, chiamata `DecodedGenerator`, che è simile alla classe base `Generator`, ad accezione del fatto che le sotto parti non text vengono sostituite con una stringa di formattazione rappresentante la parte.

class `DecodedGenerator`(*outfp*[, *mangle_from_*[, *maxheaderlen*[, *fmt*]]])

Questa classe, derivata da `Generator`, attraversa tutte le sotto parti del messaggio. Se la sotto parte è principalmente text, stampa il carico utile decodificato della sotto parte. I parametri facoltativi *_mangle_from_* e *maxheaderlen* sono esattamente come nella classe base `Generator`.

Se la sotto parte non è principalmente text, l'argomento facoltativo *fmt* è una stringa di formattazione che viene usata in sostituzione del carico utile del messaggio. *fmt* viene espanso con le seguenti parole chiave, '`%(keyword)s`':

- `type` – Il tipo MIME completo della parte non text
- `maintype` – Tipo MIME principale della parte non text
- `subtype` – Tipo Sub-MIME della parte non text
- `filename` – Nome del file della parte non text
- `description` – Descrizione associata alla parte non text
- `encoding` – Codifica del trasferimento dei contenuti della parte non text

Il valore predefinito di *fmt* è `None`, a significare

```
[Non-text (%(type)s) part of message omitted, filename %(filename)s]
```

Nuovo nella versione 2.2.2.

Metodi deprecati

I seguenti metodi sono deprecati nella versione 2 di `email`. Vengono documentati di seguito per completezza.

`__call__`(*msg*[, *unixfrom*])

Questo metodo è identico al metodo `flatten()`.

Deprecato dalla versione 2.2.2. Utilizzare invece il metodo `flatten()`.

12.2.4 Creare oggetti email e MIME da zero

Normalmente, si può ottenere una struttura di oggetti messaggio passando un file o del testo al parser, che analizza il testo e restituisce l'oggetto message principale. Comunque si può anche costruire una struttura di messaggi completamente da zero, o anche oggetti `Message` individuali a mano. In pratica, si può anche prendere una struttura esistente e aggiungere nuovi oggetti `Message`, spostarli, etc. etc.. Questo rende molto conveniente l'interfaccia per effettuare lo slicing-and-dicing dei messaggi MIME.

É possibile creare una nuova struttura di oggetti creando un'istanza `Message`, aggiungendo gli allegati e tutte le intestazioni appropriate a mano. Per i messaggi MIME però il package `email` fornisce alcune sotto classi convenienti per rendere il tutto più facile. Ogni sotto classe deve essere importata da un modulo con lo stesso nome della classe, dal package `email`. Per esempio:

```
import email.MIMEImage.MIMEImage
```

o

```
from email.MIMEText import MIMEText
```

Ecco la classe:

class `MIMEBase`(*_maintype*, *_subtype*, ***_params*)

Questa è la classe base per tutte le sotto classi di `Message` specifiche per MIME. Ordinariamente non si creeranno istanze specifiche di `MIMEBase`, anche se lo si potrebbe comunque fare. `MIMEBase` viene fornita principalmente come classe base utile per alcune specifiche sotto classi MIME-consapevoli.

_maintype è il tipo principale di Content-Type: (per esempio, text o image) e *_subtype* è il tipo secondario in Content-Type: (per esempio plain o gif). *_params* è un dizionario chiave/valore passato direttamente a `Message.add_header()`.

La classe `MIMEBase` aggiunge sempre un'intestazione Content-Type: (basata su *_maintype*, *_subtype* e *_params*) ed un'intestazione MIME-Version: (sempre impostata a `1.0`).

class `MIMENonMultipart`()

Una sotto classe di `MIMEBase`, questa è una classe base intermedia per messaggi MIME che non sono multipart. Lo scopo principale di questa classe è di prevenire l'uso del metodo `attach()`, che ha senso solo per messaggi multipart. Se `attach()` viene chiamata, viene sollevata un'eccezione di tipo `MultipartConversionError`.

Nuovo nella versione 2.2.2.

class `MIMEMultipart`([*subtype*[, *boundary*[, *_subparts*[, *_params*]]]])

Una sotto classe di `MIMEBase`, questa è una classe base intermedia per messaggi MIME che non sono multipart. L'argomento facoltativo *_subtype* ha valore predefinito a mixed, ma può essere utilizzato per specificare il sotto tipo del messaggio. Un'intestazione Content-Type: del tipo multipart/*_subtype* viene aggiunta all'oggetto del messaggio. Viene anche aggiunta un'intestazione MIME-Version:.

L'argomento facoltativo *boundary* è la stringa boundary multipart. Quando è `None` (il valore predefinito), il boundary (NdT: limite) viene calcolato quando necessario.

_subparts è una sequenza della sotto parte iniziale del carico utile. Deve essere possibile convertire questa sequenza in una lista. Si possono sempre allegare nuove sotto parti al messaggio utilizzando il metodo `Message.attach()`.

Parametri addizionali per l'intestazione Content-Type: vengono presi dall'argomento parola chiave o passati nell'argomento *_params*, che è un dizionario di parole chiave.

Nuovo nella versione 2.2.2.

class `MIMEAudio`(*_audiodata*[, *_subtype*[, *_encoder*[, ***_params*]]])

Una sotto classe di `MIMENonMultipart`, la classe `MIMEAudio` viene utilizzata per creare oggetti messaggi MIME di tipo principale audio. *_audiodata* è una stringa contenente i dati audio grezzi. Se questi dati possono essere decodificati dal modulo standard di Python `sndhdr`, il sotto tipo viene automaticamente incluso nell'intestazione Content-Type:. Altrimenti si può esplicitamente specificare il sotto tipo audio attraverso il parametro *_subtype*. Se il tipo secondario non può essere determinato e *_subtype* non viene fornito, verrà sollevata un'eccezione di tipo `TypeError`.

Il parametro facoltativo *_encoder* è un oggetto eseguibile (per esempio una funzione) che effettuerà la vera codifica dei dati audio per il trasporto. Questo eseguibile prende un solo argomento, che è l'istanza di `MIMEAudio`. Si può utilizzare il metodo `get_payload()` e `set_payload()` per cambiare il carico utile alla forma codificata. Deve anche aggiungere le intestazioni Content-Transfer-Encoding: o altro che sia necessario, nell'oggetto messaggio. La codifica predefinita è base64. Vedere il modulo `email.Encoders` per una lista degli encoder built-in.

_params vengono passati direttamente attraverso il costruttore della classe base.

class `MIMEImage`(*_imagedata*[, *_subtype*[, *_encoder*[, ***_params*]]])

Una sotto classe di `MIMENonMultipart`, la classe `MIMEImage` viene utilizzata per creare oggetti MIME, principalmente del tipo image. *_imagedata* è la stringa che contiene i dati grezzi dell'immagine. Se questi dati possono essere decodificati dal modulo standard di Python `imghdr`, il sotto tipo viene automaticamente incluso nell'intestazione Content-Type:. Altrimenti potete esplicitamente specificare il sotto tipo dell'immagine tramite il parametro *_subtype*. Se il tipo secondario non può essere ricavato e *_subtype* non viene fornito, viene sollevata l'eccezione `TypeError`.

Il parametro facoltativo *_encoder* è un eseguibile (per esempio una funzione) che effettuerà la reale codifica dei dati dell'immagine per il trasporto. Questo eseguibile prende un argomento, che è l'istanza di `MIMEImage`. Deve utilizzare `get_payload()` e `set_payload()` per cambiare il carico utile nella forma codificata. Deve anche aggiungere ogni intestazione Content-Transfer-Encoding: o altro necessario

all'oggetto messaggio. La codifica predefinita è base64. Vedere il modulo `email.Encoders` per una lista di tutti gli encoder built-in.

_params vengono passati direttamente al costruttore `MIMEBase`.

class `MIMEMessage`(*_msg*[, *_subtype*])
Una sotto classe di `MIMENonMultipart`, la classe `MIMEMessage` viene utilizzata per creare oggetti MIME, principalmente di tipo message. *_msg* viene utilizzato come carico utile e deve essere un'istanza della classe `Message` (o una sotto classe), altrimenti viene sollevata l'eccezione `TypeError`.

L'argomento facoltativo *_subtype* imposta il sotto tipo del messaggio; il valore predefinito è rfc822.

class `MIMEText`(*_text*[, *_subtype*[, *_charset*[, *_encoder*]]])
Una sotto classe di `MIMENonMultipart`, la classe `MIMEText` viene utilizzata per creare oggetti MIME, principalmente di tipo text. *_text* è la stringa per il carico utile. *_subtype* è il tipo secondario ed il valore predefinito è plain. *_charset* è il charset del testo e viene passato come parametro al costruttore di `MIMENonMultipart`; il valore predefinito è `us-ascii`. Non viene effettuata alcuna analisi per determinare la codifica del testo.

Deprecato dalla versione 2.2.2. L'argomento *_encoding* è considerato deprecato. La codifica adesso viene implicitamente basata sull'argomento *_charset*.

12.2.5 Intestazioni internazionalizzate

La RFC 2822 è lo standard base che descrive il formato dei messaggi email. Deriva dallo standard della vecchia RFC 822 che era ampiamente usata quando la maggior parte delle email erano composte esclusivamente da caratteri ASCII. La RFC 2822 è una specificazione scritta assumendo che le email contenessero solo caratteri ASCII a 7 bit.

Naturalmente, come le email si sono diffuse a livello mondiale, sono diventate internazionalizzate, così che charset specifici per i vari linguaggi possono oggi essere utilizzati nei messaggi di email. Lo standard base richiede ancora che i messaggi di email siano trasferiti utilizzando solo caratteri ASCII a 7 bit, perciò varie RFC sono state scritte per descrivere come codificare mail contenenti caratteri non ASCII in un formato compatibile con la RFC 2822. Queste RFC includono RFC 2045, RFC 2046, RFC 2047 e RFC 2231. Il package `email` supporta tutti questi standard nei moduli `email.Header` e `email.Charset`.

Se si vuole includere caratteri non ASCII nelle intestazioni e nei campi Subject: o To:, si deve utilizzare la classe `Header` ed assegnare il campo nell'oggetto `Message` ad un'istanza di `Header` anziché utilizzare una stringa per il valore dell'intestazione. Per esempio:

```
>>> from email.Message import Message
>>> from email.Header import Header
>>> msg = Message()
>>> h = Header('p\xf6stal', 'iso-8859-1')
>>> msg['Subject'] = h
>>> print msg.as_string()
Subject: =?iso-8859-1?q?p=F6stal?=
```

Come fare se si vuole che il campo Subject: contenga un carattere non ASCII? Lo si può fare creando un'istanza di `Header` e passando l'insieme dei caratteri in cui la stringa è codificata. Quando l'istanza di `Message` viene convertita in testo, il campo Subject: è stato propriamente codificato secondo la RFC 2047. I lettori di email MIME-consapevoli mostreranno questa intestazione utilizzando l'insieme dei caratteri ISO-8859-1.

Nuovo nella versione 2.2.2.

Di seguito la descrizione della classe `Header`:

class `Header`([*s*[, *charset*[, *maxlinelen*[, *header_name*[, *continuation_ws*[, *errors*]]]]]])
Crea un'intestazione MIME compatibile che può contenere stringhe in differenti insiemi di charset.

L'argomento facoltativo *s* è il valore iniziale per l'intestazione. Se `None` (il valore predefinito), non viene

impostato il valore iniziale dell'intestazione. Lo si può aggiungere successivamente all'intestazione utilizzando il metodo `append()`. *s* può essere una stringa di byte o una stringa Unicode, ma si veda la documentazione del metodo `append()` per la semantica.

L'argomento facoltativo *charset* serve per due scopi: ha lo stesso significato dell'argomento *charset* per il metodo `append()`. Imposta anche il carattere predefinito per tutte le seguenti chiamate `append()` che omettono l'argomento *charset*. Se *charset* non viene fornito nel costruttore (il caso predefinito), viene utilizzato l'insieme dei caratteri `us-ascii` sia come charset iniziale per *s*, che per le seguenti chiamate ad `append()`.

La lunghezza massima della riga può essere specificata esplicitamente tramite *maxlinelen*. Per separare la prima riga ad un valore minore (per gestire il campo intestazione che non è inclusa in *s*, per esempio Subject:) passare nel nome del campo in *header_name*. Il valore perdefinito per varmaxlinelen è 76 ed il valore predefinito per *header_name* è `None`, a significare che non viene preso in considerazione per la prima riga di una lunga intestazione separata.

Facoltativamente *continuation_ws* dovrebbe essere compatibile con la RFC 2822 per quanto riguarda gli spazi vuoti, ossia spazi e codici di controllo come tabulazioni. Questi caratteri saranno considerati come indicativi per la continuazione di una riga.

L'argomento facoltativo *errors* viene passato direttamente al metodo `append()`.

append(*s*[, *charset*[, *errors*]])
Aggiunge la stringa *s* all'intestazione MIME.

L'argomento facoltativo *charset*, se fornito, deve essere un'istanza di `Charset` (vedere `email.Charset`) o il nome di un insieme di caratteri, che verrà convertito in un'istanza di `Charset`. Un valore `None` (il caso predefinito) significa che verrà utilizzato il *charset* fornito al costruttore.

s deve essere una stringa di byte o una stringa Unicode. Se è una stringa di byte (cioè `isinstance(s, str)` è vero), il *charset* è la codifica di questa stringa di byte e l'eccezione `UnicodeError` viene sollevata se la stringa non può essere decodificata con quel charset.

Se *s* è una stringa Unicode, il *charset* è un suggerimento che specifica l'insieme dei caratteri nella stringa. In questo caso, quando si genera un'intestazione compatibile con la RFC 2822, utilizzando le regole della RFC 2047, la stringa Unicode verrà codificata utilizzando i seguenti charset in ordine: `us-ascii`, il *charset* suggerito, `utf-8`. Viene utilizzato il primo insieme di caratteri che non solleva un'eccezione `UnicodeError`.

L'argomento facoltativo *errors* viene passato attraverso ogni chiamata `unicode()` o `ustr.encode()`, ed il valore predefinito è "strict".

encode([*splitchars*])
Codifica un'intestazione di un messaggio in un formato compatibile con le RFC, possibilmente dividendo le linee lunghe ed incapsulando le parti non ASCII in base64 o in codifica quoted-printable. L'argomento facoltativo *splitchars* è una stringa contenente i caratteri su cui dividere le linee ASCII lunghe, nel supporto approssimativo all'interruzione sintattica di alto livello specificata dalla RFC 2822. Questo non riguarda le linee codificate secondo l'RFC 2047.

La classe `Header` fornisce anche un numero di metodi per supportare gli operatori standard e funzioni built-in.

__str__()
Un sinonimo per `Header.encode()`. Utile per `str(aHeader)`.

__unicode__()
Un aiuto per la funzione built-in `unicode()`. Restituisce l'intestazione come stringa Unicode.

__eq__(*other*)
Questo metodo permette di confrontare due istanze di `Header` per l'eguaglianza.

__ne__(*other*)
Questo metodo permette di confrontare due istanze di `Header` per la disuguaglianza.

Il modulo `email.Header` fornisce anche le seguenti utili funzioni.

decode_header(*header*)
Decodifica il valore di un'intestazione del messaggio senza convertire l'insieme dei caratteri. Il valore dell'intestazoine è *header*.

Questa funzione restituisce una lista di coppie `(decoded_string, charset)` contenenti ciascuna delle parti dell'intestazione decodificata. *charset* è `None` per le parti non codificate dell'intestazione, altrimenti una stringa in caratteri minuscoli contenente il nome dell'insieme dei caratteri specificato nella codifica della stringa.

Ecco un esempio:

```
>>> from email.Header import decode_header
>>> decode_header('=?iso-8859-1?q?p=F6stal?=')
[('p\\xf6stal', 'iso-8859-1')]
```

`make_header`(*decoded_seq*[, *maxlinelen*[, *header_name*[, *continuation_ws*]]])
Crea un'istanza di `Header` da una sequenza di coppie restituite da `decode_header()`.

`decode_header()` riceve un valore di intestazione e restituisce una sequenza di coppie nel formato `(decoded_string, charset)` dove *charset* è il nome dell'insieme dei caratteri.

Questa funzione prende una di queste coppie di sequenze e restituisce un'istanza di `Header`. Gli argomenti facoltativi *maxlinelen*, *header_name* e *continuation_ws* sono come nel costruttore di `Header`.

12.2.6 Rappresentare l'insieme dei caratteri

Questo modulo fornisce una classe `Charset` per rappresentare l'insieme dei caratteri e le conversioni tra l'insieme di caratteri nei messaggi email, sia un registro dei charset ed alcuni convenienti metodi per manipolare questo registro. Le istanze di `Charset` vengono utilizzate in vari altri moduli nel package `email`.

Nuovo nella versione 2.2.2.

class `Charset`([*input_charset*])
Mappa dell'insieme dei caratteri di email.

Questa classe fornisce informazioni circa i requisiti imposti alle email per uno specifico insieme di caratteri. Fornisce anche utili routine per convertire tra diversi insiemi di caratteri, fornendo la disponibilità dei codec applicabili. Fornendo un insieme di caratteri, farà del suo meglio per fornire informazioni su come utilizzare quel charset in un messaggio di email in modo confacente alle RFC.

Alcuni insiemi di caratteri devono essere codificati in quoted-printable o base64 quando utilizzati in intestazioni o nel corpo del messaggio. Alcuni insiemi di caratteri devono essere convertiti completamente e non sono ammessi nelle email.

Il parametro facoltativo *input_charset* viene descritto di seguito; viene sempre convertito in minuscolo. Dopo essere stato 'alias normalized' viene anche utilizzato per dare un'occhiata nel registro degli insiemi dei caratteri per trovare la codifica dell'intestazione, del corpo del messaggio ed il codec di conversione da utilizzare per l'insieme di caratteri. Per esempio, se *input_charset* è `iso-8859-1`, l'intestazione ed il corpo del messaggio verranno codificati utilizzando quoted-printable e non sarà necessario alcun codec di conversione per l'output. Se *input_charset* è `euc-jp`, l'intestazione verrà codificata in base64, il corpo non verrà codificato, ma il testo di output verrà convertito da `euc-jp` a `iso-2022-jp`.

Le istanze `Charset` hanno i seguenti attributi:

`input_charset`
L'insieme di caratteri specificato inizialmente. Gli alias comuni vengono convertiti nel loro nome *ufficiale* (per esempio, `latin_1` viene convertito in `iso-8859-1`). Il valore predefinito è `us-ascii` a 7 bit.

`header_encoding`
Se l'insieme dei caratteri deve essere codificato prima che possa essere utilizzato nell'intestazione dell'e-mail, questo attributo verrà impostato a `Charset.QP` (per quoted-printable), `Charset.BASE64` (per la codifica base64) o `Charset.SHORTEST` per la codifica più breve tra QT e BASE64. Altrimenti varrà `None`.

`body_encoding`
Lo stesso di *header_encoding*, ma descrive la codifica per il corpo del messaggio, che effettivamente può essere differente dall'intestazione del messaggio. `Charset.SHORTEST` non è ammesso per *body_encoding*.

output_charset
Alcuni insiemi di caratteri devono essere convertiti prima di poterli utilizzare nelle intestazioni o nel corpo delle email. Se *input_charset* è uno di questi, questo attributo conterrà il nome dell'insieme dei caratteri in cui verrà convertito l'output. Altrimenti viene impostato a `None`.

input_codec
Il nome del codec Python utilizzato per convertire *input_charset* in Unicode. Se non è necessario alcun codec di conversione, questo attributo varrà `None`.

output_codec
Il nome del codec Python utilizzato per convertire Unicode in *output_charset*. Se non è necessario alcun codec di conversione, questo attributo avrà lo stesso valore di *input_codec*.

Le istanze `Charset` hanno anche i seguenti metodi:

get_body_encoding()
Restituisce la codifica del trasferimento dei contenuti utilizzata per il corpo del messaggio.

Questo può essere sia la stringa '`quoted-printable`' che '`base64`', dipendentemente dalla codifica utilizzata, o è una funzione, nel qual caso si dovrà chiamare la funzione con un singolo argomento, l'oggetto Message che dovrà essere codificato. La funzione deve quindi impostare l'intestazione Content-Transfer-Encoding: ad un valore appropriato.

Restituisce la stringa '`quoted-printable`' se *body_encoding* è `QP`, restituisce la stringa '`base64`' se *body_encoding* è `BASE64`, altrimenti restituisce la stringa '`7bit`'.

convert(*s*)
Converte la stringa *s* da *input_codec* a *output_codec*.

to_splittable(*s*)
Converte una possibile stringa multibyte in un formato che possa essere diviso senza problemi. *s* è la stringa da dividere.

Utilizza *input_codec* per provare a convertire la stringa in Unicode, perciò può essere divisa senza problemi sui caratteri delimitativi (anche per caratteri multibyte).

Restituisce la stringa così com'è, se non sa come convertire *s* in Unicode con *input_charset*.

I caratteri che non possono essere convertiti in Unicode verranno sostituiti con il carattere Unicode di sostituzione '`U+FFFD`'.

from_splittable(*ustr*[, *to_output*])
Converte una stringa divisibile in una stringa di codifica. *ustr* è la stringa unicode da "ricomporre".

Questo metodo utilizza il codec adeguato per provare a convertire la stringa da Unicode in un formato codificato. Restituisce la stringa così com'è se non è Unicode o se non può essere convertita da Unicode.

I caratteri che non possono essere convertiti da Unicode verranno sostituiti con un carattere appropriato (generalmente '`?`').

Se *to_output* è `True` (il caso predefinito), utilizza *output_codec* per convertire in un formato codificato. Se *to_output* è `False`, utilizza *input_codec*.

get_output_charset()
Restituisce l'insieme dei caratteri di output.

Questo è l'attributo *output_charset* se non vale `None`, altrimenti è *input_charset*.

encoded_header_len()
Restituisce la lunghezza della stringa di intestazione codificata, propriamente calcolata per le codifiche quoted-printable o base64.

header_encode(*s*[, *convert*])
L'intestazione codificata della stringa *s*.

Se *convert* è `True`, la stringa verrà convertita dall'insieme dei caratteri di input all'insieme dei caratteri di output in modo automatico. Questo non è utile per insiemi di caratteri multibyte, che hanno il problema della lunghezza delle righe (i charset multibyte devono essere divisi sul carattere, non un limite di byte); utilizzare la classe di alto livello `Header` per gestire questi problemi (vedere `email.Header`). Il valore predefinito di *convert* è convert `False`.

Il tipo di codifica (base64 o quoted-printable) sarà basata sull'attributo *header_encoding*.

body_encode(*s*[, *convert*])
Il corpo del messaggio codificato della stringa *s*.

Se *convert* è `True` (il caso predefinito), la stringa verrà automaticamente convertita dall'insieme dei caratteri di input all'insieme di caratteri di output. Al contrario di `header_encode()`, non ci sono problemi con i limiti di byte e gli insiemi di caratteri multibyte nel corpo dei messaggi, perciò è generalmente abbastanza sicuro.

Il tipo di codifica (base64 o quoted-printable) verrà basato sull'attributo *body_encoding*.

La classe `Charset` fornisce anche un numero di metodi per supportare le operazioni standard e funzioni built-in.

__str__()
Restituisce *input_charset* come una stringa convertita in minuscolo. `__repr__()` è un alias per `__str__()`.

__eq__(*other*)
Questo metodo permette di confrontare due istanze di `Charset` per l'uguaglianza.

__ne__(*other*)
Questo metodo permette di confrontare due istanze di `Charset` per la disuguaglianza.

Il modulo `email.Charset` fornisce anche le seguenti funzioni per aggiungere nuovi elementi all'insieme di caratteri globale, alias e registri dei codec:

add_charset(*charset*[, *header_enc*[, *body_enc*[, *output_charset*]]])
Aggiunge le proprietà dei caratteri al registro globale.

charset è il l'insieme dei caratteri di input e deve essere il nome canonico di un insieme di caratteri.

Gli argomenti facoltativi *header_enc* e *body_enc* possono essere sia `Charset.QP` per quoted-printable, `Charset.BASE64` per la codifica base64, `Charset.SHORTEST` per la codifica più breve tra quoted-printable e base64 oppure `None` per nessuna codifica. `SHORTEST` è valida solo per *header_enc*. Il valore predefinito è `None`, ad indicare che non si desidera nessuna codifica.

Il parametro opzionale *output_charset* è l'insieme di caratteri in cui deve essere l'output. Le conversioni procederanno dall'insieme dei caratteri di input verso Unicode, verso l'insieme dei caratteri di output quando viene chiamato il metodo `Charset.convert()`. Il valore predefinito genera l'output nello stesso insieme di caratteri dell'input.

Sia *input_charset* che *output_charset* devono avere delle voci dei codec Unicode nella mappa di associazione dell'insieme dei caratteri nel modulo; utilizzare `add_codec()` per aggiungere codec di cui il modulo non è a conoscenza. Vedere la documentazione del modulo `codecs` per ulteriori informazioni.

Il registro degli insiemi dei caratteri globale viene mantenuto nel dizionario globale a livello di modulo `CHARSETS`.

add_alias(*alias*, *canonical*)
Aggiunge un alias all'insieme dei caratteri. *alias* è il nome dell'alias, per esempio `latin-1`. *canonical* è il nome canonico dell'insieme dei caratteri, per esempio `iso-8859-1`.

Il registro globale degli alias degli insiemi dei caratteri viene mantenuto nel dizionario globale `ALIASES`.

add_codec(*charset*, *codecname*)
Aggiunge un codec che mappa i caratteri nel charset fornito da e verso Unicode.

charset è il nome canonico dell'insieme dei caratteri. *codecname* è il nome del codec Python, come appropriato per il secondo argomento della funzione built-in `unicode()`, o per il metodo `encode()` di una stringa Unicode.

12.2.7 Encoders – Codificatori

Quando si crea da zero un oggetto di tipo `Message`, spesso si ha bisogno di codificare il carico utile per il trasporto attraverso server email compatibili. Questo è vero specialmente per messaggi di tipo image/* e text/* che contengono dati binari.

Il package `email` fornisce alcuni codificatori pratici nel suo modulo `Encoders`. Questi codificatori sono realmente utilizzati dai costruttori delle classi `MIMEImage` e `MIMEText` per fornire le codifiche predefinite. Tutte le funzioni di codifica ricevono esattamente un argomento, l'oggetto di tipo messaggio da codificare. Generalmente estraggono il carico utile, lo codificano e reimpostano il carico utile a questo nuovo valore. Devono anche impostare l'intestazione Content-Transfer-Encoding: in modo appropriato.

Ecco le funzioni di codifica fornite:

`encode_quopri`(*msg*)
: Codifica il carico utile nella forma quoted-printable ed imposta l'intestazione Content-Transfer-Encoding: a `quoted-printable`[1]. Questa è una buona codifica da utilizzare quando la maggior parte del carico utile sono normali dati stampabili ma contengono pochi caratteri non stampabili.

`encode_base64`(*msg*)
: Codifica il carico utile nella forma base64 ed imposta l'intestazione Content-Transfer-Encoding: a base64. Questa è una buona codifica da utilizzare quando la maggior parte del carico utile è composto da dati non stampabili poiché è più compatto della forma quoted-printable. Lo svantaggio della codifica base64 è che rende il testo in formato non leggibile da un utente umano ma da una macchina.

`encode_7or8bit`(*msg*)
: Questo non modifica realmente il carico del messaggio, ma imposta l'intestazione Content-Transfer-Encoding: a `7` o `8 bit` come appropriato, in base ai dati del carico utile.

`encode_noop`(*msg*)
: Questo non fa niente; non imposta nemmeno l'intestazione Content-Transfer-Encoding:

12.2.8 Classi per le eccezioni

Le seguenti classi di eccezione sono definite nel modulo `email.Errors`:

eccezione `MessageError`()
: Questa è la classe base per tutte le eccezioni che il package `email` può sollevare. È derivata dalla classe standard `Exception` e non definisce alcun ulteriore metodo.

eccezione `MessageParseError`()
: Questa è la classe base per le eccezioni lanciate dalla classe `Parser`. Deriva da `MessageError`.

eccezione `HeaderParseError`()
: Sollevata sotto alcune condizioni di errore quando si analizzano le intestazioni RFC 2822 del messaggio, questa classe è derivata da `MessageParseError`. Può essere sollevata dai metodi `Parser.parse()` o `Parser.parsestr()`.

  Le situazioni in cui può venir sollevata includono il trovare un'intestazione della busta dopo il primo header RFC 2822 del messaggio, trovare una riga di continuazione prima che venga trovato il primo header RFC 2822 o trovare una riga nelle intestazioni che non sia né un'intestazione né una riga di continuazione.

eccezione `BoundaryError`()
: Sollevata sotto alcune condizioni di errore quando si analizzano gli header RFC 2822 del messaggio, questa classe è derivata da `MessageParseError`. Può essere sollevata dai metodi `Parser.parse()` o `Parser.parsestr()`.

  Le situazioni in cui può venir sollevata includono il non essere in grado di trovare l'inizio o la fine di una delimitazione in un messaggio multipart/* quando viene utilizzata l'analisi rigorosa.

eccezione `MultipartConversionError`()
: Sollevata quando un carico utile viene aggiunto ad un oggetto `Message` utilizzando `add_payload()`, ma il carico utile era già uno scalare ed il tipo principale di Content-Type: del messaggio non sia multipart o è mancante. `MultipartConversionError` eredita da `MessageError` e dal built-in `TypeError`.

  Poiché `Message.add_payload()` è deprecato, questa eccezione viene raramente sollevata in pratica. Comunque l'eccezione può anche essere sollevata se il metodo `attach()` viene chiamato su un'istanza di una classe derivata da `MIMENonMultipart` (per esempio `MIMEImage`).

[1] Notare che questa codifica, con `encode_quopri()` codifica anche tutte le tabulazioni e gli spazi vuoti presenti nei dati.

12.2.9 Utilità varie

Ci sono varie utilità molto comode fornite con il package `email`.

`quote`(*str*)
Restituisce una nuova stringa con i caratteri di backslashes ed i doppi apici in *str* preceduti da una (ulteriore) backslashes.

`unquote`(*str*)
Restituisce una nuova stringa che è la versione *non quotata* di *str*. Se *str* termina ed inizia con dei doppi apici, questi vengono eliminati. In modo simile, se *str* termina ed inizia con delle parentesi angolari, queste vengono eliminate.

`parseaddr`(*address*)
Analizza l'indirizzo, che deve essere il valore di alcuni campi contenenti indirizzi, come To: o Cc:, nelle sue parti costituenti, *nome reale* ed *indirizzo email*. Restituisce una tupla di queste informazioni, a meno che l'analisi non fallisca, nel qual caso viene restituita la tupla `("", "")`.

`formataddr`(*pair*)
L'inverso di `parseaddr()`, questa riceve una tupla di 2 elementi nella forma `(nome_reale, indirizzo_email)` e restituisce un valore adatto per un header To: o Cc:. Se il primo degli elementi di *pair* (NdT: una coppia) è falso, viene restituito il secondo elemento non modificato.

`getaddresses`(*fieldvalues*)
Questo metodo restituisce una lista di tuple di 2 elementi nella forma restituita da codeparseaddr(). *fieldvalues* è una sequenza di valori di campi di intestazione come può essere restituita da `Message.get_all()`. Ecco un semplice esempio che prende tutti i recipienti di un messaggio:

```
from email.Utils import getaddresses

tos = msg.get_all('to', [])
ccs = msg.get_all('cc', [])
resent_tos = msg.get_all('resent-to', [])
resent_ccs = msg.get_all('resent-cc', [])
all_recipients = getaddresses(tos + ccs + resent_tos + resent_ccs)
```

`parsedate`(*date*)
Prova ad analizzare una data in accordo con le regole contenuto nella RFC 2822. Comunque, alcuni programmi di posta non seguono questo formato come specificato, perciò `parsedate()` prova ad indovinare correttamente in questi casi. *date* è una stringa contenente una data secondo la RFC 2822, come `Mon, 20 Nov 1995 19:12:08 -0500`. Se ha successo nell'analisi, `parsedate()` restituisce una tupla di 9 elementi che può essere passata direttamente a `time.mktime()`; altrimenti viene restituito `None`. Notare che i campi 6, 7 e 8 della tupla restituita non sono utilizzabili.

`parsedate_tz`(*date*)
Effettua la stessa funzione di `parsedate()`, ma restituisce `None` o una tupla di 10 elementi; i primi 9 elementi formano una tupla che può essere passata direttamente a `time.mktime()` e la decima è l'offset del timezone della data da UTC (che è il nome ufficiale di GMT, ovvero il Tempo medio di Greenwich)[2]. Se la stringa in ingresso non ha una timezone, l'ultimo elemento della tupla restituita vale `None`. Notare che i campi 6, 7 e 8 del risultato non sono utilizzabili.

`mktime_tz`(*tuple*)
Converte una tupla di 10 elementi come restituita da `parsedate_tz()` in un timestamp UTC. Se l'oggetto timezone nella tupla vale `None`, si assume come valore il tempo locale. Differenza minore: `mktime_tz()` interpreta i primi 8 elementi della tupla *tuple* come un tempo locale e poi compensa per la differenza di timezone. Questo può causare un piccolo errore intorno ai cambiamenti nell'ora legale, anche se non ci si deve preoccupare nell'uso comune.

`formatdate`([*timeval*[, *localtime*]])
Restituisce una data come stringa come indicato dalla RFC 2822, per esempio:

[2] Notare che l'indicazione dell'offset della timezone è l'opposto del segno che indica la variabile `time.timezone` per la stessa timezone; la seconda variabile segue lo standard POSIX finché questo modulo è aderente alla RFC 2822.

```
Fri, 09 Nov 2001 01:08:47 -0000
```

Il parametro facoltativo *timeval*, se passato, è un valore in virgola mobile come accettato da `time.gmtime()` e `time.localtime()`, altrimenti viene utilizzato il tempo corrente.

Il parametro facoltativo *localtime* è un'opzione che, quando `True`, interpreta *timeval* e restituisce una data relativa al tempo locale invece di UTC, tenendo in debita considerazione l'ora legale. Il valore predefinito è `False`, a significare che viene utilizzato UTC.

`make_msgid`([*idstring*])
Restituisce una stringa utilizzabile per un'intestazione Message-ID: compatibile con la RFC 2822. Il parametro facoltativo *idstring*, se passato, è una stringa utilizzata per rinforzare l'unicità dell'id del messaggio.

`decode_rfc2231`(*s*)
Decodifica una stringa *s* in accordo con la RFC 2231.

`encode_rfc2231`(*s*[, *charset*[, *language*]])
Codifica la stringa *s* in accordo con la RFC 2231. I parametri facoltativi *charset* e *language*, se passati, sono l'insieme dei caratteri ed il nome della lingua da utilizzare. Se nessuno dei 2 viene passato, *s* viene restituita così com'è. Se *charset* viene passato, ma non *language*, la stringa viene codificata utilizzando una stringa vuota per *language*.

`decode_params`(*params*)
Decodifica la lista di parametri in accordo con la RFC 2231. *params* è una sequenza di tuple di 2 elementi contenenti gli elementi nella forma `(content-type, string-value)`.

Le seguenti funzioni sono considerate deprecate:

`dump_address_pair`(*pair*)
Deprecato dalla versione 2.2.2. Usare invece `formataddr()`.

`decode`(*s*)
Deprecato dalla versione 2.2.2. Usare invece `Header.decode_header()`.

`encode`(*s*[, *charset*[, *encoding*]])
Deprecato dalla versione 2.2.2. Usare invece `Header.encode()`.

12.2.10 Iteratori

Iterare su un albero di oggetti message è abbastanza semplice con il metodo `Message.walk()`. Il modulo `email.Iterators` fornisce alcune iterazioni di alto livello utili per gli alberi di oggetti message.

`body_line_iterator`(*msg*[, *decode*])
Questo itera su tutto il carico utile in tutte le sotto parti di *msg*, restituendo la stringa di carico utile riga per riga. Salta tutte le intestazioni delle sotto parti e salta tutte le sotto parti che non sono stringhe Python. Questo è qualcosa di equivalente che leggere la rappresentazione testuale del messaggio da un file, utilizzando `readline()`, saltando tutte le intestazioni.

Il parametro facoltativo *decode* viene passato tramite `Message.get_payload()`.

`typed_subpart_iterator`(*msg*[, *maintype*[, *subtype*]])
Questo itera su tutte le sotto parti di *msg*, restituendo solo quelle sotto parti che corrispondono al tipo MIME specificato da *maintype* e *subtype*.

Notare che *subtype* è facoltativo; se omesso, la corrispondenza del tipo MIME viene fatta solo considerando il tipo principale. *maintype* è anch'esso facoltativo; il valore predefinito è text.

Perciò, il comportamento predefinito di `typed_subpart_iterator()` è quello di restituire tutte le parti che sono di tipo MIME text/*.

Le seguenti funzioni sono state aggiunte come un utile strumento di debug. *Non* devono essere considerate parte dell'interfaccia pubblica supportata dal package.

`_structure`(*msg*[, *fp*[, *level*]])
Stampa una rappresentazione indentata dei tipi contenuti nella struttura dell'oggetto messaggio. Ad esempio:

```
>>> msg = email.message_from_file(somefile)
>>> _structure(msg)
multipart/mixed
    text/plain
    text/plain
    multipart/digest
        message/rfc822
            text/plain
        message/rfc822
            text/plain
        message/rfc822
            text/plain
        message/rfc822
            text/plain
        message/rfc822
            text/plain
    text/plain
```

Il parametro facoltativo *fp* è un oggetto simile a file dove stampare l'output. Deve soddisfare le istruzioni estese di stampa di Python. *level* viene usato internamente.

12.2.11 Differenze tra il modulo `email` v1 (fino a Python 2.2.1)

La versione 1 del package `email` era parte integrante delle distribuzioni Python fino a Python 2.2.1. La versione 2 fu sviluppata per la distribuzione Python 2.3 e resa compatibile per Python 2.2.2. Era disponibile anche come package separato, basato su distutils. La versione 2 di `email` è quasi per intero compatibile con la precedente versione 1, con le seguenti differenze:

- Sono stati aggiunti i moduli `email.Header` e `email.Charset`.
- È stato cambiato il formato di serializzazione per le istanze `Message`. Poiché tale formato non era mai stato (e non lo è ancora) formalmente definito, questa non viene considerata un'incompatibilità all'indietro. Ad ogni modo se l' applicazione in uso serializza e deserializza istanze di `Message`, fare attenzione che nella versione 2 di `email` le istanze di `Message` dispongono adesso delle variabili private *_charset* e *_default_type*.
- Parecchi metodi nella classe `Message` sono stati deprecati o le loro sigle cambiate. Inoltre sono stati aggiunti parecchi nuovi metodi. Vedere la documentazione della classe `Message` per i dettagli. I cambiamenti dovrebbero essere interamente compatibili all'indietro.
- È stata cambiata la struttura dell'oggetto per quanto riguarda i tipi di contenuto message/rfc822. Nella versione 1 di `email` un tipo di questo genere sarebbe stato rappresentato da un carico utile scalare, per esempio il contenitore di messaggi `is_multipart()` restituiva falso, `get_payload()` non era un oggetto lista, ma un'istanza singola di `Message`.

  Questa struttura era estranea al resto del package, così la rappresentazione oggetto per i tipi di contenuto message/rfc822 è stata cambiata. Nella versione 2 di email il contenitore restituisce vero da `is_multipart()` e `get_payload()` restituisce una lista che contiene un singolo elemento `Message`.

  Si può notare che in questo caso non si poteva ottenere una completa compatibilità all'indietro. Comunque se si sta già testando il tipo restituito da `get_payload()` non si dovrebbero avere problemi. Si avrà solamente bisogno di essere sicuri che il codice usato non esegua un `set_payload()` con un'istanza di `Message` o con un contenitore di tipi di contenuto message/rfc822.
- È stato introdotto lo specifico argomento del costruttore `Parser` ed i suoi metodi `parse()` e `parsestr()` sono diventati semplicemente argomenti *headersonly*. Inoltre è stata aggiunta la specifica opzione *strict* alle funzioni `email.message_from_file()` e `email.message_from_string()`.
- `Generator.__call__()` è deprecato; usate invece `Generator.flatten()`. La classe `Generator` ha inoltre sviluppato il metodo `clone()`.

- È stata aggiunta la classe `DecodedGenerator` al modulo `email.Generator`.
- Sono state aggiunte le classi base intermedie `MIMENonMultipart`, `MIMEMultipart` ed inserite nella gerarchia della classe per la maggior parte delle altre classi derivate che usano MIME.
- Deprecato l'argomento *_encoder* per il costruttore `MIMEText`. Adesso la codifica si basa implicitamente sull'argomento *_charset*.
- Vengono deprecate le seguenti funzioni nel modulo `email.Utils`: `dump_address_pairs()`, `decode()` ed `encode()`. Aggiunte al modulo le seguenti funzioni: `make_msgid()`, `decode_rfc2231()`, `encode_rfc2231()` e `decode_params()`.
- Aggiunta la funzione non pubblica `email.Iterators._structure()`.

12.2.12 Differenze rispetto a `mimelib`

Il package `email` fu in origine prototipato come libreria separata chiamata `mimelib`. Le modifiche sono state effettuate in modo che i nomi dei metodi fossero più efficaci ed alcuni metodi e moduli sono stati aggiunti o eliminati. Di alcuni metodi è stata cambiata la semantica. Per la maggior parte, tutte le funzionalità disponibili in `mimelib` sono tuttora disponibili nel package `email`, sebbene spesso in maniera diversa. La compatibilità all'indietro tra i package `mimelib` ed `email` non era una priorità.

Ecco una breve descrizione delle differenze tra i package `mimelib` ed `email` insieme ad alcuni suggerimenti sul porting delle vostre applicazioni.

Ovviamente la differenza più evidente tra i due package è il cambio del nome del package stesso in `email`. In aggiunta abbiamo le seguenti differenze nella parte più alta del package:

- `messageFromString()` è stato rinominato in `message_from_string()`.
- `messageFromFile()` è stato rinominato in `message_from_file()`.

La classe `Message` presenta le seguenti differenze:

- Il metodo `asString()` è stato rinominato in `as_string()`.
- Il metodo `ismultipart()` è stato rinominato in `is_multipart()`.
- Il metodo `get_payload()` ha acquistato un argomento *decode* facoltativo.
- Il metodo `getall()` è stato rinominato in `get_all()`.
- Il metodo `addheader()` è stato rinominato in `add_header()`.
- il metodo `gettype()` è stato rinominato in `get_type()`.
- il metodo `getmaintype()` è stato rinominato in `get_main_type()`.
- il metodo `getsubtype()` è stato rinominato in `get_subtype()`.
- Il metodo `getparams()` è stato rinominato in `get_params()`. Inoltre laddove `getparams()` restituiva una lista di stringhe, `get_params()` restituisce una lista di 2 tuple, le coppie chiave/valore dei parametri senza il segno di '='.
- Il metodo `getparam()` è stato rinominato in `get_param()`.
- Il metodo `getcharsets()` è stato rinominato in `get_charsets()`.
- Il metodo `getfilename()` è stato rinominato in `get_filename()`.
- Il metodo `getboundary()` è stato rinominato in `get_boundary()`.
- Il metodo `setboundary()` è stato rinominato in `set_boundary()`.

- Il metodo `getdecodedpayload()` è stato eliminato. Per ottenere una funzionalità simile occorre passare il valore `1` all'opzione *decode* del metodo `get_payload()`.
- Il metodo `getpayloadastext()` è stato eliminato. Una simile funzionalità viene fornita dalla classe `DecodedGenerator` nel modulo `email`.
- Il metodo `getbodyastext()` è stato rimosso. Si può ottenere una funzionalità simile creando un iteratore con `typed_subpart_iterator()`, presente nel modulo `email.Iterators`.

La classe `Parser` non ha differenze nell'interfaccia pubblica. Sono state aggiunte alcune peculiarità per riconoscere lo stato di consegna del messaggio, che viene rappresentato come un'istanza di `Message` che contiene sotto parti separate di `Message` per ognuno dei blocchi di intestazione nella notifica dello stato di consegna[3].

La classe `Generator` non ha differenze nell'interfaccia pubblica. Abbiamo però una nuova classe nel modulo `email.Generator`, chiamata `DecodedGenerator` che fornisce la maggior parte delle funzionalità precedentemente disponibili nel metodo `Message.getpayloadastext()`.

Sono stati modificati i seguenti moduli e classi:

- La classe `MIMEBase` Gli argomenti del costruttore di classe `MIMEBase`, *_major* sono stati rispettivamente modificati in *_maintype* e *_subtype*.
- La classe/modulo `Image` è stata/o rinominata/o `MIMEImage`. L'argomento *_minor* è stato rinominato *_subtype*.
- La classe/modulo `Text` è stata/o rinominata/o `MIMEText`. L'argomento *_minor* è stato rinominato *_subtype*.
- La classe/modulo `MessageRFC822` è stata/o rinominata/o `MIMEMessage`. Notare che una precedente versione di `mimelib` chiamava questa classe/modulo `RFC822`, ma questa/o, in presenza di alcuni file system non sensibili alle differenze tra maiuscole e minuscole, interferiva con il modulo `rfc822` della libreria standard di Python.

  Inoltre la classe `MIMEMessage` adesso rappresenta tutti i tipi di messaggi MIME di tipo message. È necessario l'argomento facoltativo *_subtype* che si usa per impostare il sotto tipo MIME. Il valore *_subtype* è predefinito per rfc822.

`mimelib` forniva alcune utili funzioni nei moduli `address` e `date`. Tutte queste funzioni sono state spostate nel modulo `email.Utils`.

La classe/modulo `MsgReader` è stata/o eliminata/o. La funzionalità più simile viene supportata dalla funzione `body_line_iterator()` nel modulo `email.Iterators`.

12.2.13 Esempi

Ecco alcuni esempi su come usare il package `email` per leggere, scrivere ed inviare semplici messaggi di posta elettronica, così come complessi messaggi MIME.

Iniziamo col vedere come creare e inviare un semplice messaggio di testo:

```
# importa smtplib per l'attuale funzione di invio
import smtplib

# importa i moduli email necessari
from email.MIMEText import MIMEText

#  Apre un file di testo per la lettura. Per questo esempio supponiamo
#+ che il file di testo contenga solo caratteri ASCII.
fp = open(textfile, 'rb')
# Crea un messaggio in puro testo
```

[3] Le "Delivery Status Notifications," (DSN) (Notificazione dello Stato del Trasporto) vengono definite nella RFC 1894.

```
msg = MIMEText(fp.read())
fp.close()

# me == indirizzo di chi invia l'email
# you == indirizzo di chi riceve l'email
msg['Subject'] = 'The contents of %s' % textfile
msg['From'] = me
msg['To'] = you

#  Invia il messaggio tramite il nostro server SMTP ma non include
#+ l'intestazione della busta.
s = smtplib.SMTP()
s.connect()
s.sendmail(me, [you], msg.as_string())
s.close()
```

Ecco un esempio di come inviare un messaggio MIME contenente un gruppo di foto di famiglia che possono essere recuperate in una directory:

```
# importa smtplib per l'attuale funzione di invio
import smtplib

# Ecco i moduli del package email di cui abbiamo bisogno
from email.MIMEImage import MIMEImage
from email.MIMEMultipart import MIMEMultipart

COMMASPACE = ', '

# Crea il messaggio email contenitore (outer).
msg = MIMEMultipart()
msg['Subject'] = 'Our family reunion'
# me == indirizzo email di chi invia
# family = la lista di tutti gli indirizzi email destinatari
msg['From'] = me
msg['To'] = COMMASPACE.join(family)
msg.preamble = 'Our family reunion'
# Fa in modo che il messaggio termini con un fine riga
msg.epilogue = ''

# Si presume che i file di immagine siano tutti in formato PNG
for file in pngfiles:
    #  Apre i file in modo binario.  Lascia che la classe MIMEImage
    #+ automaticamente individui il tipo di immagine specifico.
    fp = open(file, 'rb')
    img = MIMEImage(fp.read())
    fp.close()
    msg.attach(img)

# Invia l'email tramite il nostro server SMTP.
s = smtplib.SMTP()
s.connect()
s.sendmail(me, family, msg.as_string())
s.close()
```

Ecco un esempio di come inviare l'intero contenuto di una directory come messaggio email: [4]

```
#!/usr/bin/env python

"""Invia il contenuto di una directory come messaggio MIME.

Uso: dirmail [opzioni] from to [to ...]*
```

[4] Un ringraziamento a Matthew Dixon Cowles per l'originale ispirazione e gli esempi.

```
Opzioni:
    -h / --help
        Stampa questo messaggio ed esce.

    -d directory
    --directory=directory
        Invia il contenuto della directory specificata altrimenti usa
        la directory corrente. Vengono inviati solo i file regolari
        della directory e non vengono coinvolte le sotto directory.

'from' è l'indirizzo email di chi invia il messaggio.

'to' è l'indirizzo email del destinatario del messaggio. Possono essere
passati più destinatari.

L'email viene inviata tramite inoltro al vostro server SMTP locale che
quindi esegue il normale processo di consegna. La vostra macchina locale
deve avere in esecuzione un server SMTP.
"""

import sys
import os
import getopt
import smtplib
# Per l'individuazione del tipo MIME in base a nome e estensione del file
import mimetypes

from email import Encoders
from email.Message import Message
from email.MIMEAudio import MIMEAudio
from email.MIMEBase import MIMEBase
from email.MIMEMultipart import MIMEMultipart
from email.MIMEImage import MIMEImage
from email.MIMEText import MIMEText

COMMASPACE = ', '


def usage(code, msg=''):
    print >> sys.stderr, __doc__
    if msg:
        print >> sys.stderr, msg
    sys.exit(code)


def main():
    try:
        opts, args = getopt.getopt(sys.argv[1:], 'hd:', ['help', 'directory='])
    except getopt.error, msg:
        usage(1, msg)

    dir = os.curdir
    for opt, arg in opts:
        if opt in ('-h', '--help'):
            usage(0)
        elif opt in ('-d', '--directory'):
            dir = arg

    if len(args) < 2:
        usage(1)

    sender = args[0]
    recips = args[1:]
```

```
    # Crea il messaggio contenitore (outer)
    outer = MIMEMultipart()
    outer['Subject'] = 'Contents of directory %s' % os.path.abspath(dir)
    outer['To'] = COMMASPACE.join(recips)
    outer['From'] = sender
    outer.preamble = 'You will not see this in a MIME-aware mail reader.\n'
    # Fa in modo che il messaggio termini con un fine riga
    outer.epilogue = ''

    for filename in os.listdir(dir):
        path = os.path.join(dir, filename)
        if not os.path.isfile(path):
            continue
        #  Individua il tipo del contenuto in base all'estensione del
        #+ file. La codifica verrà ignorata, anche se dovremmo effettuare
        #+ un controllo su cosette come file compressi e gzip.
        ctype, encoding = mimetypes.guess_type(path)
        if ctype is None or encoding is not None:
            #  Non si capisce il tipo o il file è codificato (compresso)
            #+ quindi si usa un generico tipo bag-of-bits.
            ctype = 'application/octet-stream'
        maintype, subtype = ctype.split('/', 1)
        if maintype == 'text':
            fp = open(path)
            # Notate: dovrebbe gestire la tipologia del carattere usato
            msg = MIMEText(fp.read(), _subtype=subtype)
            fp.close()
        elif maintype == 'image':
            fp = open(path, 'rb')
            msg = MIMEImage(fp.read(), _subtype=subtype)
            fp.close()
        elif maintype == 'audio':
            fp = open(path, 'rb')
            msg = MIMEAudio(fp.read(), _subtype=subtype)
            fp.close()
        else:
            fp = open(path, 'rb')
            msg = MIMEBase(maintype, subtype)
            msg.set_payload(fp.read())
            fp.close()
            # Codifica il carico utile usando Base64
            Encoders.encode_base64(msg)
        # Imposta il parametro del nome del file
        msg.add_header('Content-Disposition', 'attachment', filename=filename)
        outer.attach(msg)

    # Adesso invia il messaggio
    s = smtplib.SMTP()
    s.connect()
    s.sendmail(sender, recips, outer.as_string())
    s.close()


if __name__ == '__main__':
    main()
```

Infine, ecco un esempio di come scompattare un messaggio MIME come il precedente, in una directory di file:

```
#!/usr/bin/env python

"""Scompatta un messaggio MIME in una directory di file.

Uso: unpackmail [opzioni] msgfile
```

```
Opzioni:
    -h / --help
        stampa questo messaggio ed esce.

    -d directory
    --directory=directory
        Scompatta il messaggio MIME nella directory indicata che verrà
        creata se non già esistente.

msgfile è il percorso del file che contiene il messaggio MIME.
"""

import sys
import os
import getopt
import errno
import mimetypes
import email


def usage(code, msg=''):
    print >> sys.stderr, __doc__
    if msg:
        print >> sys.stderr, msg
    sys.exit(code)


def main():
    try:
        opts, args = getopt.getopt(sys.argv[1:], 'hd:', ['help', 'directory='])
    except getopt.error, msg:
        usage(1, msg)

    dir = os.curdir
    for opt, arg in opts:
        if opt in ('-h', '--help'):
            usage(0)
        elif opt in ('-d', '--directory'):
            dir = arg

    try:
        msgfile = args[0]
    except IndexError:
        usage(1)

    try:
        os.mkdir(dir)
    except OSError, e:
        # Ignora l'errore che evidenzia l'esistenza della directory
        if e.errno <> errno.EEXIST: raise

    fp = open(msgfile)
    msg = email.message_from_file(fp)
    fp.close()

    counter = 1
    for part in msg.walk():
        # multipart/* sono semplici contenitori
        if part.get_content_maintype() == 'multipart':
            continue
        #  Le applicazioni dovrebbero veramente controllare a fondo il
        #+ nome del file fornito, in modo che il messaggio di posta
```

```
            #+ elettronica non possa essere usato per sovrascrivere file
            #+ importanti
            filename = part.get_filename()
            if not filename:
                ext = mimetypes.guess_extension(part.get_type())
                if not ext:
                    # Use a generic bag-of-bits extension
                    ext = '.bin'
                filename = 'part-%03d%s' % (counter, ext)
            counter += 1
            fp = open(os.path.join(dir, filename), 'wb')
            fp.write(part.get_payload(decode=1))
            fp.close()


    if __name__ == '__main__':
        main()
```

12.3 `mailcap` — Gestione di file Mailcap

I file mailcap vengono usati per configurare applicazioni in modo MIME, cosicché lettori di posta e browser Web reagiscano a file con differenti tipi MIME. Il nome "mailcap" deriva dalla frase "mail capability" (NdT: potenzialità della posta elettronica). Ad esempio un file mailcap potrebbe contenere una riga come '`video/mpeg; xmpeg %s`'. A questo punto se l'utente incontra un messaggio di posta elettronica o un documento Web con il tipo MIME video/mpeg, '`%s`' verrà sostituito con un file con nome (in genere un file temporaneo) ed il programma **xmpeg** verrà avviato automaticamente per eseguire il file.

Il formato mailcap è documentato nella RFC 1524, "A User Agent Configuration Mechanism For Multimedia Mail Format Information," ma non è uno standard internet. Comunque i file mailcap vengono supportati dalla maggior parte dei sistemi UNIX.

`findmatch`(*caps, MIMEtype*[, *key*[, *filename*[, *plist*]]])

Restituisce una doppia tupla; il primo elemento è una stringa che contiene la riga di comando da eseguire (che può essere passata ad `os.system()`), il secondo elemento è il dato mailcap per un tipo MIME passato. Se non viene trovato un tipo MIME corrispondente viene restituito `(None, None)`.

key è il nome del campo desiderato, che rappresenta il tipo di attività da eseguire; il valore predefinito è 'view', visto che nei casi più comuni si vuole semplicemente vedere il corpo dei dati di tipo MIME. Altri possibili valori potrebbero essere 'compose' ed 'edit', se si volesse creare un nuovo corpo del tipo MIME dato o si volesse modificarlo. Vedere la RFC 1524 per una lista completa di tali campi.

filename è il file con nome che sostituirà '`%s`' sulla riga di comando; il valore predefinito è `'/dev/null'` che quasi certamente non è ciò che vogliamo, quindi in genere verrà sovrascritto specificando un nome di file.

plist può essere una lista contenente parametri con nome; il valore predefinito è semplicemente una lista vuota. Ogni inserimento nella lista deve essere una stringa contenente il nome del parametro, un segno di uguale ('=') ed il valore del parametro. Le voci relative ai mailcap possono contenere parametri con nome quali `%{foo}`, che verrà sostituito dal valore del parametro di nome 'foo'. Ad esempio se la riga di comando '`showpartial %{id} %{number} %{total}`' si trovasse in un file mailcap e *plist* fosse impostata a `['id=1', 'number=2', 'total=3']`, la riga di comando risultante sarebbe `'showpartial 1 2 3'`.

In un file mailcap il campo "test" può essere facoltativamente specificato per testare alcune condizioni esterne (quali l'architettura della macchina o l'ambiente grafico in uso) per determinare se applicare o meno la riga del mailcap. `findmatch()` controllerà automaticamente tali condizioni e salterà l'istruzione se il controllo fallisce.

`getcaps`()

Restituisce un dizionario che mappa i tipi MIME in una lista di voci per file mailcap. Il dizionario può essere passato alla funzione `findmatch()`. Un'istruzione viene immagazzinata come una lista di dizionari ma non dovrebbe essere necessario conoscere i dettagli di tale rappresentazione.

L'informazione è derivata da tutti i file mailcap trovati sul sistema. Le impostazioni del file mailcap utente '$HOME/.mailcap' sovrascriveranno le impostazioni dei file mailcap di sistema '/etc/mailcap', '/usr/etc/mailcap' e '/usr/local/etc/mailcap'.

Un esempio d'uso:

```
>>> import mailcap
>>> d=mailcap.getcaps()
>>> mailcap.findmatch(d, 'video/mpeg', filename='/tmp/tmp1223')
('xmpeg /tmp/tmp1223', {'view': 'xmpeg %s'})
```

12.4 `mailbox` — Gestione dei vari tipi di mailbox

Questo modulo definisce un certo numero di classi che permettono un facile ed uniforme accesso ai messaggi di posta in una mailbox (di tipo UNIX).

class `UnixMailbox`(*fp*[, *factory*])

Accede ad una classica mailbox in stile UNIX, dove tutti i messaggi sono contenuti in un singolo file e separati dalle linee che iniziano con '`From ` ' (leggere '`From_`'). L'oggetto file *fp* punta al file della mailbox. Il parametro facoltativo *factory* è un oggetto invocabile che dovrebbe creare nuovi oggetti messaggio. *factory* viene chiamato con un argomento, ottenuto dalla chiamata del metodo `next()` su *fp*. Il tipo predefinito per l'oggetto messaggio è `rfc822.Message` (vedere il modulo `rfc822` – e le relative note).

Note: Per ragioni legate all'implementazione di questo modulo, si dovrà probabilmente aprire l'oggetto *fp* in modalità binaria. Questo è particolarmente importante su Windows.

Per una maggiore portabilità, i messaggi in una mailbox in stile UNIX vengono separati da una riga che inizia esattamente con la stringa `'From '` (notare lo spazio finale) se preceduta da esattamente due fine riga. A causa però delle numerose variazioni esistenti rispetto a questa organizzazione, non si dovrebbe considerare nient'altro che la riga `From_`. Comunque, l'implementazione corrente non si preoccupa della mancanza dei due fine riga. Il che va bene per molte applicazioni.

La classe `UnixMailbox` si comporta in modo più rigido riguardo alla separazione dei vari messaggi, in quanto usa un'espressione regolare che di solito riconosce correttamente i delimitatori `From_`. Essa considera che le separazioni siano in base a righe del tipo '`From` *name time*'. Per una maggiore portabilità è meglio però usare la classe `PortableUnixMailbox`. È identica a `UnixMailbox`, tranne per il fatto che i singoli messaggi vengono considerati separati da righe '`From ` ' e non '`From` *name time*'.

Per altre informazioni, vedere UNIX: *Configuring Netscape Mail on UNIX: Why the Content-Length Format is Bad*.

class `PortableUnixMailbox`(*fp*[, *factory*])

Una versione più leggera di `UnixMailbox`, che considera solo il '`From ` ' all'inizio delle righe che separano i messaggi. La parte "*name time*" della riga viene ignorata, per poter essere compatibile con le variazioni che esistono su questo formato. Questa classe lavora sulle righe del messaggio che iniziano con `'From '`, che vengono evidenziate dai software di gestione della posta al momento del trasporto.

class `MmdfMailbox`(*fp*[, *factory*])

Accede ad una mailbox stile MMDF, in cui tutti i messaggi vengono racchiusi in un singolo file e sono separati da 4 caratteri "Ctrl-A". L'oggetto file *fp* punta al file della mailbox. Il parametro facoltativo *factory* ha le stesse funzioni di quello della classe `UnixMailbox`.

class `MHMailbox`(*dirname*[, *factory*])

Accede ad una mailbox di tipo MH, dove ogni messaggio è in un singolo file con un nome numerico e tutti questi file sono in una directory. Il nome della directory viene passato in *dirname*. Per il parametro facoltativo *factory* vedere la classe `UnixMailbox`.

class `Maildir`(*dirname*[, *factory*])

Accede ad una directory di messaggi di tipo Qmail. Tutti i messaggi nuovi presenti all'interno della directory passata come parametro *dirname* diventano accessibili. Per il parametro facoltativo *factory* vedere la classe `UnixMailbox`.

class **`BabylMailbox`**(*fp*[, *factory*])

Accede ad una mailbox Babyl, che è simile a quella di tipo MMDF. Nel formato Babyl, ogni messaggio ha due tipi di intestazioni, le intestazioni *originali*, e quelle *visibili*. Le intestazioni originali appaiono prima di una riga contenente solo `EOOH` (End-Of-Original-Headers), mentre le intestazioni visibili si trovano dopo. I gestori di mailbox compatibili con Babyl mostreranno solamente le intestazioni visibili, e gli oggetti `BabylMailbox` restituiranno i messaggi con solo questo tipo di intestazioni. Per poter avere anche le intestazioni originali è necessario analizzare la mailbox "da soli". I messaggi iniziano con la linea EOOH e terminano con una riga contenente i caratteri `'\037\014'`. Il parametro *factory* è analogo a quello della classe `UnixMailbox`.

Dato che il modulo `rfc822` è deprecato, si raccomanda l'uso del modulo `email` per creare gli oggetti messaggio da una mailbox (il modulo `rfc822` non può essere eliminato per questioni di compatibilità). Un modo molto sicuro per fare ciò è il seguente:

```
import email
import email.Errors
import mailbox

def msgfactory(fp):
    try:
        return email.message_from_file(fp)
    except email.Errors.MessageParseError:
        # Non fatevi restituire None, perché
        # si fermerebbe l'iteratore della mailbox
return ''

mbox = mailbox.UnixMailbox(fp, msgfactory)
```

Il codice precedente è difensivo rispetto alla struttura della mailbox, in quanto si è preparati a ricevere un'eventuale stringa vuota dal metodo `next()` della mailbox. Comunque, se si è sicuri che la mailbox è perfettamente standard, è possibile semplificare il codice:

```
import email
import mailbox

mbox = mailbox.UnixMailbox(fp, email.message_from_file)
```

Vedete anche:

mbox - file contenente i messaggi di posta
(http://www.qmail.org/man/man5/mbox.html)

Descrizione del formato tradizionale della mail box "mbox".

maildir - directory per i nuovi messaggi ricevuti
(http://www.qmail.org/man/man5/maildir.html)

Descrizione del formato "maildir" di una mailbox.

Configurare la posta in Netscape su UNIX*: perché il formato Content-Length è pessimo*
(http://home.netscape.com/eng/mozilla/2.0/relnotes/demo/content-length.html)

Una descrizione dei problemi correlati alle intestazioni Content-Length: per i messaggi memorizzati in file mailbox.

12.4.1 Oggetti Mailbox

Tutte le implementazioni degli oggetti mailbox sono oggetti iterabili, ed hanno un metodo visibile all'esterno. Questo metodo viene usato dagli iteratori creati dagli oggetti mailbox, ma può anche essere usato direttamente.

`next``()`

Restituisce il messaggio successivo nella mailbox, creato con il parametro facoltativo *factory* passato al

costruttore dell'oggetto mailbox. L'oggetto predefinito è del tipo `rfc822.Message` (vedere il modulo `rfc822`). In base all'implementazione della mailbox, l'attributo *fp* di questo oggetto può essere un vero oggetto file o solo l'istanza di una classe che simula un oggetto file, sul quale però si possono ottenere informazioni riguardo i limiti del messaggio, se in un file ci sono più messaggi, etc. Se non ci sono altri messaggi, questo metodo restituisce `None`.

12.5 `mhlib` — Accesso alle mailbox MH

Il modulo `mhlib` offre un'interfaccia Python alle cartelle MH ed al loro contenuto.

Il modulo contiene tre classi fondamentali: `MH`, che rappresenta una particolare collezione di cartelle; `Folder`, che rappresenta una singola cartella, e `Message`, che rappresenta un singolo messaggio.

class `MH`([*path*[, *profile*]])
`MH` rappresenta una collezione di cartelle MH.

class `Folder`(*mh, name*)
La classe `Folder` rappresenta una singola cartella ed i suoi messaggi.

class `Message`(*folder, number*[, *name*])
Gli oggetti `Message` rappresentano singoli messaggi in una cartella. La classe Message eredita `mimetools.Message`.

12.5.1 Oggetti MH

Le istanze di `MH` hanno i seguenti metodi:

`error`(*format*[, ...])
Stampa un messaggio di errore – può essere sovrascritto.

`getprofile`(*key*)
Restituisce un record nel profilo (`None` se non viene impostato).

`getpath`()
Restituisce il percorso della mailbox.

`getcontext`()
Restituisce il nome della cartella corrente.

`setcontext`(*name*)
Imposta il nome della cartella corrente.

`listfolders`()
Restituisce una lista delle cartelle superiori.

`listallfolders`()
Restituisce una lista di tutte le cartelle.

`listsubfolders`(*name*)
Restituisce una lista delle sotto cartelle successive alla cartella data.

`listallsubfolders`(*name*)
Restituisce una lista di tutte le sotto cartelle della cartella data.

`makefolder`(*name*)
Crea una nuova cartella.

`deletefolder`(*name*)
Cancella una cartella – che non deve avere sotto cartelle.

`openfolder`(*name*)
Restituisce un nuovo oggetto cartella aperto.

12.5.2 Oggetti Folder

Le istanze di `Folder` rappresentano cartelle aperte ed hanno i seguenti metodi:

`error`(*format*[, ...])
Stampa un messaggio di errore – può essere sovrascritto.

`getfullname`()
Restituisce il percorso completo della cartella.

`getsequencesfilename`()
Restituisce il percorso completo delle sequenze di file della cartella.

`getmessagefilename`(*n*)
Restituisce il percorso completo dell'*n*-esimo messaggio della cartella.

`listmessages`()
Restituisce una lista dei messaggi nella cartella (in numeri).

`getcurrent`()
Restituisce il numero del messaggio corrente.

`setcurrent`(*n*)
Imposta il numero del messaggio corrente ad *n*.

`parsesequence`(*seq*)
Analizza la sintassi dei messaggi nella lista.

`getlast`()
Restituisce l'ultimo messaggio, o `0` se non ci sono messaggi nella cartella.

`setlast`(*n*)
Imposta l'ultimo messaggio (usato solo internamente).

`getsequences`()
Restituisce il dizionario delle sequenze nella cartella. I nomi delle sequenze vengono usate come chiavi, e i valori sono le liste dei numeri dei messaggi nelle sequenze stesse.

`putsequences`(*dict*)
Restituisce il dizionario *nome:lista* delle sequenze nella cartella.

`removemessages`(*list*)
Rimuove i messaggi nella lista dalla cartella.

`refilemessages`(*list, tofolder*)
Sposta i messaggi nella lista in un'altra cartella.

`movemessage`(*n, tofolder, ton*)
Sposta un solo messaggio alla data destinazione in un'altra cartella.

`copymessage`(*n, tofolder, ton*)
Copia un solo messaggio alla data destinazione in un'altra cartella.

12.5.3 Oggetti Message

La classe `Message` aggiunge un metodo a quelli di `mimetools.Message`:

`openmessage`(*n*)
Restituisce un nuovo oggetto messaggio aperto (equivalente ad un descrittore di file).

12.6 `mimetools` — Strumenti per analizzare messaggi MIME

Deprecato dalla versione 2.3. Dovrebbe essere usato il modulo `email` al posto di questo. È ancora presente solo per questioni di compatibilità.

Questo modulo definisce una sotto classe del modulo `rfc822` della classe `Message` ed un certo numero di funzioni di utilità per la manipolazione di messaggi MIME codificati o a più parti.

Questo definisce i seguenti elementi:

class `Message`(*fp*[, *seekable*])
Restituisce una nuova istanza della classe `Message`. Questa è una sotto classe di `rfc822.Message`, con metodi aggiuntivi (vedere oltre). L'argomento *seekable* ha lo stesso significato che ha nella classe `rfc822.Message`.

`choose_boundary`()
Restituisce una stringa univoca che ha la più alta probabilità di essere utilizzabile come interruzione (NdT tra i messaggi). La stringa ha la forma `'`*hostipaddr*`.`*uid*`.`*pid*`.`*timestamp*`.`*random*`'`.

`decode`(*input*, *output*, *encoding*)
Legge i dati codificati usando la giusta codifica, *encoding*, MIME dal file oggetto *input* e scrive i dati decodificati nel file *output*. Valori validi per *encoding* comprendono `'base64'`, `'quoted-printable'`, `'uuencode'`, `'x-uuencode'`, `'uue'`, `'x-uue'`, `'7bit'` e `'8bit'`. La decodifica dei messaggi in `'7bit'` o `'8bit'` non ha effetto. L'input è semplicemente copiato nell'output.

`encode`(*input*, *output*, *encoding*)
Legge i dati dal file *input* e li scrive codificati nel file *output* usando la giusta *encoding* MIME per il file *output*. I valori validi per *encoding* sono gli stessi di `decode()`.

`copyliteral`(*input*, *output*)
Copia integralmente il file *input* nel file *output* finché non incontra un EOF.

`copybinary`(*input*, *output*)
Legge, fino ad incontrare un EOF, blocchi dal file *input* e li scrive nel file *output*. La grandezza di un blocco è fissata a `8192`.

Vedete anche:

Modulo `email` (sezione 12.2):
Modulo esteso per la gestione delle email; è più avanzato del modulo `mimetools`.

Modulo `rfc822` (sezione 12.11):
Fornisce la classe base `mimetools.Message`.

Modulo `multifile` (sezione 12.10):
Permette di leggere file che contengono varie parti distinte, come dati MIME.

http://www.cs.uu.nl/wais/html/na-dir/mail/mime-faq/.html
Le FAQ, le domande poste più frequentemente, su MIME. Per una panoramica, vedete la risposta alla domanda 1.1 nella Parte 1 di questo documento.

12.6.1 Ulteriori metodi degli oggetti Message

La classe `Message` definisce i seguenti metodi in aggiunta a quelli del metodo `rfc822.Message`:

`getplist`()
Restituisce la lista dei parametri relativi all'intestazione Content-Type:. Questa è una lista di stringhe. Per i parametri nella forma '*chiave*=*valore*', la *chiave* viene convertita tutta in minuscolo, ma non *valore*. Per esempio, se il messaggio contiene '`Content-type: text/html; spam=1; Spam=2; Spam`' allora il metodo `getplist()` restituisce la lista `['spam=1', 'spam=2', 'Spam']`.

`getparam`(*nome*)
Restituisce il valore del primo parametro (di quelli restituiti da `getplist()`) nella forma '*nome*=*valore*' per il *nome* dato. Se il valore viene racchiuso da caratteri del tipo '<...>' o '"...', vengono rimossi.

`getencoding`()
Restituisce la codifica specificata nell'intestazione del messaggio Content-Transfer-Encoding:. Se questa voce non è presente viene restituito `'7bit'`. La codifica viene convertita in minuscolo.

`gettype`()

Restituisce il tipo di messaggio (nella forma '*tipo/sotto tipo*') come specificato in Content-Type:. Se questa voce non è presente viene restituito `'text/plain'`. Il tipo viene convertito in minuscolo.

`getmaintype``()`
Restituisce solo il tipo (non il sotto tipo) come specificato nell'intestazione Content-Type:. Se questa voce non esiste viene restituito `'text'`. Il tipo restituito viene convertito in minuscolo.

`getsubtype``()`
Restituisce il sotto tipo come specificato in Content-Type:. Se questa voce non esiste viene restituito `'plain'`. Il sotto tipo restituito viene convertito in minuscolo.

12.7 `mimetypes` — Mappa i nomi dei file ai tipi MIME

Il modulo `mimetypes` opera la conversione fra un file o una URL ed il tipo MIME associato all'estensione del file. Le conversioni avvengono dal file al tipo MIME e dal tipo MIME all'estensione del file; le codifiche non sono supportate per le ultime conversioni.

Il modulo fornisce una classe ed alcune funzioni pratiche – che sono la sua normale interfaccia, ma con qualche applicazione coinvolta anche nella classe.

Qui di seguito verranno descritte le funzioni che gli offrono l'interfaccia primaria. Se il modulo non è stato inizializzato, queste chiameranno `init()`, qualora si riferiscano alle informazioni che `init()` imposta.

`guess_type``(`*filename*`[`, *strict* `])`
Risale al tipo di un file basandosi sul suo nome o sull'URL fornita dal *filename*. Il risultato è una tupla `(`*tipo*`,` *codifica*`)`, la cui parte tipo è `None`, se non può essere determinata (suffisso mancante o sconosciuto), oppure è una stringa formata da `'`*tipo/sotto tipo*`'`, utilizzabile per un MIME content-type: intestazione .

In assenza di *codifica* la voce è `None`, altrimenti è il nome del programma impiegato (ad esempio, **compress** o **gzip**). La codifica funge da intestazione Content-Encoding: e *non* da intestazione Content-Transfer-Encoding:. Le mappature rispecchiano una tabella, i suffissi di codifica sono sensibili alle differenze tra maiuscole e minuscole, mentre i suffissi dei tipi vengono considerati prima come sensibili e poi come insensibili a queste differenze.

Il facoltativo *strict* è un'opzione che specifica se la lista dei tipi MIME noti sia limitata o meno ai soli registrati con IANA. Quando *strict* assume il valore vero (predefinito), vengono supportati solo i tipi IANA, quando viene impostato a falso, verranno riconosciuti anche alcuni tipi MIME non standard, ma comunemente usati.

`guess_all_extensions``(`*type*`[`, *strict* `])`
Risale all'estensione di un file basandosi sul suo tipo MIME risultante da *type*. Il risultato è una lista di stringhe riportanti tutte le possibili estensioni di file, punto iniziale ('`.`') compreso. Non è detto che siano state associate con una particolare fonte di dati, ma verrebbero riferite al tipo MIME *type* dalla funzione `guess_type()`.

L'opzione *strict* ha lo stesso significato della funzione `guess_type()`.

`guess_extension``(`*type*`[`, *strict* `])`
Risale all'estensione di un file basandosi sul suo tipo MIME, risultante da *type*. Il risultato è una stringa riportante un'estensione di file, punto iniziale ('`.`') compreso. Non è detto che sia stata associata con una particolare fonte di dati, ma verrebbero riferite al tipo MIME *type* dalla funzione `guess_type()`.

L'opzione *strict* ha lo stesso significato della funzione `guess_type()`.

Alcune funzioni aggiuntive ed elementi di dati sono disponibili per il controllo del comportamento del modulo.

`init``([`*files*`])`
Inizializza la struttura interna dei dati. Se presente, *files* deve essere una sequenza di nomi di file che dovrebbero essere usati per aumentare la mappa dei tipi predefinita. Se omesso, i nomi dei file da usare vengono presi da `knownfiles`. Ogni file nominato in files o `knownfiles` prende la precedenza su quelli chiamati precedentemente. È permessa la chiamata di `init()` ripetutamente.

`read_mime_types``(`*filename*`)`
Carica la mappa dei tipi dei dati in *filename*, se esiste. La mappa dei tipi viene restituita come un dizionario

che mappa le estensioni di *filename*, incluso il punto iniziale (‘.’), per stringhe nella forma `'type/subtype'`. Se il file *filename* non esiste o non può essere letto, viene restituito `None`.

`add_type`(*type, ext*[, *strict*])
Aggiunge una mappa dal tipo *type* mimetype alla estensione *ext*. Quando l’estensione viene sempre conosciuta, il nuovo type sostituirà il vecchio. Quando il tipo viene sempre riconosciuto l’estensione verrà aggiunta alla lista delle estensioni conosciute.

Quando *strict* è la traccia, verrà aggiunta MIME types ufficiale, altrimenti a quelli non standard.

`inited`
L’opzione indica se la struttura dei dati è stata inizializzata oppure no. Questa è impostata su vero da `init()`.

`knownfiles`
La lista dei nomi dei file di tipo map comunemente installati. Questi file vengono comunemente chiamati ‘mime.types’ e vengono installati in differenti posizioni dai diversi pacchetti.

`suffix_map`
Dizionario che tiene traccia dei suffissi per i suffissi. Questo è usato per permettere il riconoscimento dei file codificati, per i quali la codifica ed il tipo vengono indicati dalla stessa estensione. Per esempio l’estensione ‘.tgz’ viene mappata per ‘.tar.gz’ per consentire alla codifica ed al tipo di essere riconosciuti separatamente.

`encodings_map`
Dizionario che tiene traccia delle estensioni dei nomi dei file per la codifica dei tipi.

`types_map`
Il dizionario che tiene traccia le estensioni dei filename per i tipi MIME.

`common_types`
Il dizionario che tiene traccia delle estensioni dei nomi dei file non standard, ma comunemente riscontrabili nei tipi MIME.

La classe `MimeTypes` può essere utile per applicazioni che possono volere più di un database MIME-type:

class `MimeTypes`([*filenames*])
Questa classe rappresenta il database dei MIME-types. Fornisce l’accesso in modo predefinito allo stesso database come il resto di questo modulo. Il database iniziale è una copia di quello fornito dal modulo e può essere esteso caricando file addizionali in stile ‘mime.types’ nel database usando i metodi `read()` o `readfp()`. La traccia dei dizionari può essere anche azzerata prima di caricare dati aggiuntivi, se i dati predefiniti non sono quelli desiderati.

Il parametro facoltativo *filenames* può essere usato per effettuare il caricamento di file aggiuntivi “on top” da database predefinito.

Nuovo nella versione 2.2.

12.7.1 Oggetti MimeTypes

Le istanze `MimeTypes` forniscono un’interfaccia molto simile a quella del modulo `mimetypes`.

`suffix_map`
Il dizionario che tiene traccia di suffissi per suffissi. Viene usato per permettere la ricerca dei file codificati per i quali la codifica ed il tipo vengono indicati dalla stessa estensione. Per esempio l’estensione ‘.tgz’ viene mappata per ‘.tar.gz’ per permettere alla codifica ed al tipo di essere riconosciuti separatamente. Questo è inizialmente una copia del `suffix_map` globale definito nel modulo.

`encodings_map`
Dizionario che tiene traccia delle estensioni dei nomi dei file per i tipi di codifica. Questo è inizialmente una copia dell’`encodings_map` definito nel modulo.

`types_map`
Dizionario che tiene traccia delle estensioni dei nomi dei file per i tipi MIME. Questo è inizialmente una copia del `types_map` globale definito nel modulo.

`common_types`

Dizionario che tiene traccia delle estensioni dei nomi dei file non standard, ma comunemente riconosciuti come tipi MIME. Questa è inizialmente una copia della variabile globale `common_types` definita nel modulo.

`guess_extension`(*type*[, *strict*])
Simile alla funzione `guess_extension()`, usando le tabelle immagazzinate come parte dell'oggetto.

`guess_type`(*url*[, *strict*])
Simile alla funzione `guess_type()`, usando le tabelle immagazzinate come parte dell'oggetto.

`read`(*path*)
Carica, da un file di nome *path*, informazioni MIME. Viene usata `readfp()` per analizzare il file.

`readfp`(*file*)
Carica da un file aperto informazioni di tipo MIME. Il file deve avere il formato standard dei file 'mime.types'.

12.8 `MimeWriter` — scrittore generico di file MIME

Deprecato dalla versione 2.3. Dovrebbe essere usato preferibilmente il package `email` invece del modulo `MimeWriter`. Questo modulo è presente soltanto per mantenere la compatibilità con precedenti versioni.

Questo modulo definisce la classe `MimeWriter`. Questa classe implementa un formattatore base per creare file MIME formati da più parti. Non esegue operazioni di posizionamento internamente al file di output, né fa uso di grandi quantità di spazio del buffer. Si devono scrivere le varie parti del messaggio nell'ordine in cui dovranno apparire nel file finale. `MimeWriter` inserisce in un buffer le intestazioni aggiunte di volta in volta, permettendo di riarrangiarne l'ordine.

class `MimeWriter`(*fp*)
Restituisce una nuova istanza della classe `MimeWriter`. L'unico argomento richiesto, *fp*, è un oggetto file da usare per la scrittura. Notare che potrebbe essere usato anche un oggetto `StringIO`.

12.8.1 Oggetti MimeWriter

Le istanze `MimeWriter` hanno i seguenti metodi:

`addheader`(*key*, *value*[, *prefix*])
Aggiunge una riga d'intestazione al messaggio MIME. Il parametro *key* è il nome dell'intestazione, laddove *value* fornisce ovviamente il valore di tale intestazione. L'argomento facoltativo *prefix* determina dove verrà inserita l'intestazione; '`0`' significa aggiunta alla fine, '`1`' inserita all'inizio. Il valore predefinito è aggiungere in coda.

`flushheaders`()
Fa sì che tutte le intestazioni accumulate finora siano scritte (e poi scartate). Questo è utile se non si ha affatto bisogno di una parte del corpo del messaggio, ad esempio per una sezione di tipo message/rfc822 che è (mal) usata per racchiudere alcune informazioni simili alle intestazioni.

`startbody`(*ctype*[, *plist*[, *prefix*]])
Restituisce un oggetto simile ad un file, che può essere usato per scrivere il testo del messaggio. L'attributo content-type viene impostato al valore fornito in *ctype* ed il parametro facoltativo *plist* fornisce gli ulteriori parametri per la dichiarazione content-type. L'argomento *prefix* funziona come in `addheader()` con l'unica differenza che il suo valore predefinito è l'inserimento all'inizio.

`startmultipartbody`(*subtype*[, *boundary*[, *plist*[, *prefix*]]])
Restituisce un oggetto simile a file, che può essere usato per scrivere il testo del messaggio. In aggiunta, questo metodo inizializza il codice multi parte, dove *subtype* fornisce il sotto tipo della sezione multi parte, *boundary* fornisce una specifica definita dall'utente per delimitare la sezione e *plist* fornisce parametri facoltativi per il sotto tipo. L'argomento *prefix* funziona come in `startbody()`. Ulteriori sotto parti dovranno essere create usando la funzione `nextpart()`.

`nextpart`()

Restituisce una nuova istanza di `MimeWriter` che rappresenta una singola sezione di un messaggio ripartito in più parti. Può essere usato per scrivere una singola parte, così come per creare ricorsivamente complessi messaggi multi parte. Il messaggio deve essere dapprima inizializzato con `startmultipartbody()` prima di usare `nextpart()`.

`lastpart``()`
Questo metodo viene usato per designare l'ultima parte di un messaggio costituito da più parti, e dovrà essere *sempre* usato nella scrittura di tali messaggi multi parte.

12.9 `mimify` — elaboratore MIME di messaggi di posta

Deprecato dalla versione 2.3. Dovrebbe essere usato preferibilmente il package `email` invece del modulo `mimify`. Questo modulo è presente soltanto per mantenere la compatibilità con precedenti versioni.

Il modulo `mimify` definisce due funzioni per convertire messaggi di posta in e dal formato MIME. Il messaggio di posta può essere sia un messaggio semplice che un messaggio cosiddetto multi parte. Ogni parte viene trattata separatamente. Mimificare (una parte di) un messaggio richiede la codifica del messaggio come quoted-printable se contiene un qualsiasi carattere che non può essere rappresentato utilizzando la codifica ASCIIa 7 bit. Demimificare (una parte di) un messaggio richiede l'eliminazione della codifica quoted-printable. Mimificazione e Demimificazione sono particolarmente utili quando un messaggio deve essere modificato prima di essere spedito. Un utilizzo tipico potrebbe essere:

```
demimifica il messaggio
modifica il messaggio
mimifica il messaggio
invia il messaggio
```

Il modulo definisce le seguenti funzioni richiamabili dall'utente e le seguenti variabili assegnabili dall'utente stesso:

`mimify``(`*infile, outfile*`)`
Copia il messaggio da *infile* a *outfile*, convertendo le parti in quoted-printable e aggiungendo le intestazioni di posta MIME quando necessario. *infile* ed *outfile* possono essere oggetti file (in effetti, ogni oggetto che possiede un metodo `readline()` (per *infile*) o un metodo `write()` (per *outfile*)) o stringhe che indicano file. Se *infile* ed *outfile* sono entrambi stringhe, possono avere lo stesso valore.

`unmimify``(`*infile, outfile*`[`*, decode_base64* `])`
Copia il messaggio da *infile* ad *outfile*, decodificando tutte le parti in quoted-printable. *infile* ed *outfile* possono essere oggetti file (in effetti, ogni oggetto che possiede un metodo `readline()` (per *infile*) o un metodo `write()` (per *outfile*)) o stringhe che indicano file. Se *infile* ed *outfile* sono entrambi stringhe, possono avere lo stesso valore. Se viene fornito l'argomento *decode_base64* e viene valutato vero, ognuna delle parti codificate in base64 verrà decodificata a sua volta.

`mime_decode_header``(`*line*`)`
Restituisce una versione decodificata della riga d'intestazione codificata in *line*. Supporta solamente l'insieme di caratteri ISO 8859-1 (Latin-1).

`mime_encode_header``(`*line*`)`
Restituisce una versione codificata MIME della riga d'intestazione in *line*.

`MAXLEN`
In modo predefinito, una parte verrà codificata come quoted-printable quando contiene un qualsiasi carattere non ASCII (caratteri con l'ottavo bit impostato ad 1), o se ci sia qualsiasi riga più lunga di `MAXLEN` caratteri (il valore predefinito è 200).

`CHARSET`
Quando non specificato nelle intestazioni del messaggio di posta, deve essere inserito un insieme di caratteri. La stringa utilizzata viene memorizzata in `CHARSET` ed il valore predefinito è ISO-8859-1 (anche noto come Latin1 (latin-uno)).

Questo modulo può anche essere utilizzato da riga di comando. L'uso è il seguente:

```
mimify.py -e [-l length] [infile [outfile]]
mimify.py -d [-b] [infile [outfile]]
```

rispettivamente per codificare (mimify) e decodificare (unmimify). *infile* ha come valore predefinito lo standard input, mentre *outfile* ha come valore predefinito lo standard output. Lo stesso file può essere specificato sia come input che come output.

Se in fase di codifica viene specificata l'opzione **-l**, ogni riga di lunghezza maggiore del valore *length* specificato, verrà codificata solamente per la parte pari a tale valore.

Se in fase di decodifica viene specificata l'opzione **-b**, verrà decodificata anche ogni parte in base64.

Vedete anche:

Modulo `quopri` (sezione 12.15):
Codifica e decodifica i file MIME quoted-printable.

12.10 `multifile` — Supporto per file contenenti parti distinte

L'oggetto `MultiFile` vi permette di gestire sezioni di un file di testo in maniera similare ad oggetti file di input, con `readline()` che restituisce `''` quando incontra una determinata corrispondenza di delimitazione. I valori predefiniti di questa classe sono progettati per renderla utile per l'analisi di messaggi MIME multiparte, ma effettuandone il subclassamento o sovrascrivendone i metodi, può essere facilmente adattata ad un utilizzo più generale.

class `MultiFile`(*fp*[, *seekable*])
Crea un multifile. Si deve istanziare questa classe utilizzando come argomento un oggetto di input, come ad esempio un oggetto file restituito da `open()`, dal quale l'istanza `MultiFile` recupererà le righe.

`MultiFile` considera sempre e solamente i metodi `readline()`, `seek()` e `tell()` dell'oggetto di input, gli ultimi due sono necessari solamente se desiderate un accesso casuale alle parti MIME individuali. Per utilizzare `MultiFile` su un oggetto flusso non-seekable, impostare a falso l'argomento facoltativo *seekable*; ciò inibirà l'utilizzo dei metodi `seek()` e `tell()` dell'oggetto in input.

Sarà utile sapere che nella visione del mondo `MultiFile`, il testo è composto da tre tipi di righe: dati, divisori di sezione e marcatori di fine. Multifile è progettata per supportare l'analisi di messaggi che potrebbero avere parti di messaggio annidate multiple, ognuna con la propria corrispondenza per il divisore di sezione ed il marcatore di fine.

Vedete anche:

Modulo `email` (sezione 12.2):
Package completo di gestione della posta elettronica; prevarica il modulo `multifile`.

12.10.1 Oggetti MultiFile

Un'istanza `MultiFile` possiede i seguenti metodi:

`readline`(*str*)
Legge una riga. Se la riga è composta da dati (non un separatore di sezione o un marcatore di fine o un vero EOF) la restituisce. Se la riga corrisponde al delimitatore impilato più recentemente, restituisce `''` ed imposta `self.last` a `1` o a `0`, a seconda che la corrispondenza sia o non sia un marcatore di fine. Se la riga corrisponde ad un qualsiasi altro delimitatore impilato, solleva un'eccezione `Error`. Al raggiungimento della fine del file sul sottostante oggetto flusso, il metodo genera un errore, a meno che non siano stati estratti dalla pila tutti i delimitatori.

`readlines`(*str*)
Restituisce tutte le righe rimanenti in questa parte come lista di stringhe.

read()
: Legge tutte le righe, fino alla sezione successiva. Le restituisce come singola stringa (multiriga). Notare che questo metodo non accetta un argomento legato alla dimensione!

seek(*pos*[, *whence*])
: Ricerca. Gli indici di ricerca sono relativi all'inizio della sezione corrente. Gli argomenti *pos* e *whence* vengono interpretati come nella ricerca su file.

tell()
: Restituisce la posizione nel file relativa all'inizio della sezione corrente.

next()
: Salta le righe fino alla sezione successiva (cioè, legge le righe finché non viene utilizzato un divisore di sezione o un marcatore di fine). Restituisce vero se tale sezione esiste, falso se viene trovato un marcatore di fine. Riabilita il delimitatore impilato più di recente.

is_data(*str*)
: Restituisce vero se *str* è costituita da dati e falso in caso risulti un limite della sezione. Cosi come scritto, controlla la presenza di un prefisso diverso da `'--'` all'inizio della riga (cosa che tutti i delimitatori MIME possiedono) ma viene dichiarato cosicché possa essere sovrascritto nelle classi derivate.

  Notare che questo test viene utilizzato come avanguardia per i test effettivi sui delimitatori; se restituisce sempre falso rallenterà semplicemente la velocità d'elaborazione, ma non provocherà il fallimento dei test stessi.

push(*str*)
: Inserisce nella pila una stringa di delimitazione. Quando viene trovata come riga di input una versione debitamente corredata di delimitatore, verrà interpretata come divisore di sezione o marcatore. Tutte le operazioni di lettura seguenti restituiranno la stringa vuota per indicare la fine del file, finché una chiamata al metodo `pop()` non rimuoverà il delimitatore o una chiamata al metodo `next()` lo riabiliterà.

  È possibile inserire nella pila più di un delimitatore. Incontrare il delimitatore inserito più recentemente restituirà EOF; incontrare ogni altro delimitatore solleverà un'eccezione `Error`.

pop()
: Estrae dalla pila un delimitatore di sezione. Questo delimitatore non verrà più interpretato come EOF.

section_divider(*str*)
: Trasforma un delimitatore in una riga che dividerà la sezione. Come comportamento predefinito, questo metodo antepone `'--'` (inserito in ogni delimitatore di sezione MIME) ma viene dichiarato in maniera tale che possa essere sovrascritto nelle classi derivate. Questo metodo non ha bisogno di aggiungere in coda LF o CR-LF, dato che il confronto con il risultato ignora gli spazi vuoti iniziali.

end_marker(*str*)
: Trasforma un delimitatore in una riga marcatore di fine. Come comportamento predefinito, questo metodo antepone `'--'` ed aggiunge in coda `'--'` (come un marcatore di fine messaggio MIME-multiparte) ma viene dichiarato in maniera tale che possa essere sovrascritto nelle classi derivate. Questo metodo non ha bisogno di aggiungere in coda LF o CR-LF, dato che il confronto con il risultato ignora gli spazi vuoti iniziali.

Infine, le istanze `MultiFile` hanno due variabili d'istanza pubbliche:

level
: Profondità di annidamento della parte corrente.

last
: Vero se l'ultima condizione di end-of-file era relativa ad un marcatore di fine messaggio.

12.10.2 `MultiFile` Example

```
import mimetools
import multifile
import StringIO

def extract_mime_part_matching(stream, mimetype):
    """Restituisce il primo elemento di un messaggio MIME multiparte
    con mimetype corrispondente al mimetype sul flusso stream."""

    msg = mimetools.Message(stream)
    msgtype = msg.gettype()
    params = msg.getplist()

    data = StringIO.StringIO()
    if msgtype[:10] == "multipart/":

        file = multifile.MultiFile(stream)
        file.push(msg.getparam("boundary"))
        while file.next():
            submsg = mimetools.Message(file)
            try:
                data = StringIO.StringIO()
                mimetools.decode(file, data, submsg.getencoding())
            except ValueError:
                continue
            if submsg.gettype() == mimetype:
                break
        file.pop()
    return data.getvalue()
```

12.11 `rfc822` — Analizza le intestazioni di posta RFC 2822

Deprecato dalla versione 2.3. Si dovrebbe utilizzare, invece del modulo `rfc822`, il package `email`. Il modulo `rfc822` è presente solamente per compatibilità con le versioni precedenti.

Questo modulo definisce una classe, `Message`, che rappresenta un "messaggio di posta elettronica," così come definito dallo standard Internet RFC 2822.[5] Tali messaggi si compongono di una collezione di intestazioni di messaggio e di un corpo del messaggio. Questo modulo definisce inoltre una classe helper, `AddressList`, per l'analisi di indirizzi RFC 2822. Far riferimento alla RFC per informazioni sulla sintassi specifica dei messaggi RFC 2822.

Il modulo `mailbox` fornisce classi per leggere le caselle di posta prodotte da vari programmi di gestione di posta elettronica.

class `Message`(*file*[, *seekable*])

Un'istanza di `Message` viene istanziata con un oggetto di input come parametro. Il messaggio fa affidamento sulla presenza nell'oggetto di input di un metodo `readline()`: nello specifico, i comuni oggetti file soddisfano tale requisito. L'istanziazione legge le intestazioni dall'oggetto di input fino ad una riga di delimitazione (normalmente una riga vuota) e li memorizza nell'istanza. Il corpo del messaggio, consecutivo all'intestazione, non viene utilizzato.

Questa classe può trattare ogni oggetto di input che supporti un metodo `readline()`. Se l'oggetto di input possiede le funzionalità seek e tell, sarà attivo anche il metodo `rewindbody()`; altrimenti, righe non conformi verranno respinte indietro nel flusso di input. Se l'oggetto di input non possiede la funzionalità seek ma possiede un metodo `unread()` che può respingere indietro una riga di input, `Message`

[5] Questo modulo era originariamente conforme a RFC 822, da cui il nome. Successivamente, RFC 2822 è stato rilasciato come un aggiornamento di RFC 822. Questo modulo dovrebbe essere considerato conforme a RFC 2822, specialmente nei casi in cui è stata cambiata la sintassi o la semantica rispetto a RFC 822.

lo utilizzerà per respingere indietro le righe non conformi. Quindi questa classe può essere utilizzata per analizzare messaggi provenienti da un flusso bufferizzato.

L'argomento facoltativo *seekable* viene fornito come rimedio per certe librerie stdio nelle quali `tell()` elimina i dati bufferizzati prima di scoprire che la chiamata di sistema `lseek()` non funziona. Per la massima portabilità, dovreste impostare l'argomento seekable a zero, per evitare la chiamata iniziale a `tell()` quando viene passato in input un oggetto unseekable come ad esempio un oggetto file creato da un oggetto socket.

Le righe di input, così come lette dal file, possono essere sia terminate dalla coppia CR-LF che da un singolo linefeed; una terminazione CR-LF viene sostituita da un singolo linefeed prima di memorizzare la riga.

Il controllo di corrispondenza dell'intestazione viene svolto in modalità non sensibile a maiuscole o minuscole; ad esempio *m*`['From']`, *m*`['from']` e *m*`['FROM']` conducono tutti allo stesso risultato.

class `AddressList`(*field*)
Potete istanziare la classe helper `AddressList` utilizzando un singolo parametro stringa o anche una lista separata da virgole di indirizzi RFC 2822 da analizzare. Il parametro `None` produce una lista vuota.

`quote`(*str*)
Restituisce una nuova stringa con i backslash nella stringa *str* sostituiti da due backslash e le doppie virgolette sostituite da backslash-doppie virgolette.

`unquote`(*str*)
Restituisce una nuova stringa che è la versione *senza virgolette* della stringa *str*. Se *str* inizia e finisce con le doppie virgolette, queste vengono eliminate. Allo stesso modo se *str* inizia e finisce con le parentesi acute, queste vengono eliminate.

`parseaddr`(*address*)
Analizza il parametro *address*, che dovrebbe essere costituito dal valore di un campo contenente un indirizzo quale To: oppure Cc:, e lo divide nelle parti componenti "realname" ed "email address" (NdT: nome reale ed indirizzo di posta). Restituisce una doppia tupla con le precedenti informazioni, a meno che l'analisi fallisca, nel qual caso viene restituita la tupla `(None, None)`.

`dump_address_pair`(*pair*)
È l'inverso di `parseaddr()`; riceve in ingresso una doppia tupla della forma (*realname*, *email_address*) e restituisce il valore stringa adatto per un header To: o Cc:. Se il primo elemento della coppia, *pair*, è falso, allora il secondo elemento viene restituito invariato.

`parsedate`(*date*)
Tenta di analizzare una data in base alle regole incluse nella RFC 2822. Comunque, alcuni programmi di posta non seguono il formato cosi come specificato ed allora `parsedate()` cerca di gestire il formato in maniera corretta. *date* è una stringa contenente una data in formato RFC 2822, del tipo `'Mon, 20 Nov 1995 19:12:08 -0500'`. Se l'analisi ha successo, `parsedate()` restituisce una tupla di nove elementi che può essere passata direttamente a `time.mktime()`; altrimenti verrà restituito `None`. Notate che i campi 6, 7 e 8 della tupla risultante non sono utilizzabili.

`parsedate_tz`(*date*)
Svolge la stessa funzione di `parsedate()`, ma restituisce o `None` o una tupla di dieci elementi; i primi nove elementi costituiscono una tupla che può essere passata direttamente a `time.mktime()`, ed il decimo è lo scostamento di fuso orario della data UTC (che è la dicitura ufficiale per il Greenwich Mean Time). Notare che il segno che indica lo scostamento di fuso orario è l'opposto del segno della variabile `time.timezone` per lo stesso fuso orario; quest'ultima variabile segue lo standard POSIX mentre questo modulo segue l'RFC 2822. Se la stringa in input non specifica il fuso orario, l'ultimo elemento della tupla restituita avrà il valore `None`. Notare che i campi 6, 7 e 8 della tupla risultante non sono utilizzabili.

`mktime_tz`(*tuple*)
Trasforma una tupla di dieci elementi, così come restituita `parsedate_tz()` in un orario UTC. Se l'elemento fuso orario nella tupla vale `None`, si assume l'ora locale. Mancanza minore: questa funzione comincia con l'interpretare i primi 8 elementi come ora locale e quindi compensa tenendo conto della differenza di fuso orario; questo potrebbe portare ad un minimo errore sulle date di passaggio all'ora legale, comunque non tale da preoccupare per i comuni utilizzi.

Vedete anche:

Modulo `email` (sezione 12.2):
: Pacchetto completo di gestione della posta elettronica; prevarica il modulo `rfc822`.

Modulo `mailbox` (sezione 12.4):
: Classi per leggere vari formati delle caselle di posta prodotte da programmi client per la posta.

Modulo `mimetools` (sezione 12.6):
: Sottoclasse di `rfc822.Message` per la gestione dei messaggi codificati MIME.

12.11.1 Oggetti Message

Un'istanza `Message` ha i seguenti metodi:

`rewindbody``()`
: Posiziona all'inizio del corpo del messaggio. Questo funziona solamente se l'oggetto file consente la ricerca.

`isheader``(`*line*`)`
: Restituisce la riga corrispondente al campo del nome in forma canonica (la chiave di dizionario che verrà utilizzata per indicizzarlo) se la riga è un'intestazione conforme a RFC 2822; altrimenti restituisce `None` (questo implica che l'analisi dovrebbe fermarsi qui e la riga dovrebbe essere respinta indietro nel flusso in input). Talvolta è utile sovrascrivere questo metodo in una sotto classe.

`islast``(`*line*`)`
: Restituisce vero se la riga fornita è un delimitatore sul quale `Message` dovrebbe fermarsi. La riga di delimitazione viene eliminata ed il punto di lettura dell'oggetto file posizionato immediatamente dopo la riga stessa. Come comportamento predefinito questo metodo controlla semplicemente che la riga sia vuota, ma potete sovrascrivere tale comportamento in una sotto classe.

`iscomment``(`*line*`)`
: Restituisce `True` se la riga fornita dovrebbe essere ignorata interamente, semplicemente saltata. Come comportamento predefinito restituisce sempre `False`, ma si può sovrascrivere tale comportamento in una sotto classe.

`getallmatchingheaders``(`*name*`)`
: Restituisce una lista di righe composte da tutte le intestazioni che corrispondono al parametro *name*, se ve ne sono. Ogni riga fisica, indipendentemente dal fatto di essere una riga di continuazione oppure no, è un elemento separato della lista. Se nessuna intestazione corrisponde a *name* restituisce la lista vuota.

`getfirstmatchingheader``(`*name*`)`
: Restituisce una lista di righe comprendente la prima intestazione corrispondente a *name*, e le sue righe di continuazione, nel caso ve ne siano. Restituisce `None` se non ci sono intestazioni corrispondenti a *name*.

`getrawheader``(`*name*`)`
: Restituisce una singola stringa contenente il testo dopo i due punti nella prima intestazione corrispondente a *name*. Questo include gli spazi ad inizio stringa, il linefeed a fine stringa, i linefeed interni e gli spazi delle eventuali righe di continuazione. Restituisce `None` se non ci sono intestazioni corrispondenti a *name*.

`getheader``(`*name*`[,` *default* `])`
: Come `getrawheader(`*name*`)`, ma elimina gli spazi in testa ed in coda. Gli spazi interni non vengono eliminati. L'argomento facoltativo *default* può essere utilizzato per specificare un differente valore predefinito come valore restituito, nel caso non ci siano intestazioni corrispondenti a *name*.

`get``(`*name*`[,` *default* `])`
: Un sinonimo per `getheader()`, per rendere l'interfaccia più compatibile con i dizionari regolari.

`getaddr``(`*name*`)`
: Restituisce una coppia `(`*nome completo*`,` *indirizzo email*`)` risultante dall'analisi della stringa restituita da `getheader(`*name*`)`. Se non esiste alcuna intestazione corrispondente a *name*, restituisce `(None, None)`; altrimenti sia il nome completo che l'indirizzo sono stringhe (eventualmente vuote).

  Esempio: se la prima intestazione From: di *m* contiene la stringa `'jack@cwi.nl (Jack Jansen)'`, allora `m.getaddr('From')` restituirà la coppia `('Jack Jansen', 'jack@cwi.nl')`. Se l'intestazione avesse invece contenuto `'Jack Jansen <jack@cwi.nl>'`, sarebbe stato restituito esattamente lo stesso risultato.

getaddrlist(*name*)
: È simile al metodo `getaddr(`*list*`)`, ma analizza un'intestazione che contiene una lista di indirizzi email (ad esempio un'intestazione To:) e restituisce una lista di coppie (*nome completo*, *indirizzo email*) (anche se l'intestazione contiene un solo indirizzo). Nel caso non ci sia nessuna intestazione corrispondente a *name*, restituisce una lista vuota.

  Se ci fossero più intestazioni corrispondenti all'intestazione in analisi (ad esempio se ci fossero più intestazioni Cc:), verranno tutti analizzati per la ricerca degli indirizzi. Verranno analizzate anche tutte le eventuali righe di continuazione.

getdate(*name*)
: Recupera un'intestazione utilizzando `getheader()` e la analizza suddividendola in una tupla di nove elementi compatibile con `time.mktime()`; notare che i campi 6, 7 e 8 non sono utilizzabili. Se non ci fosse nessuna intestazione corrispondente a *name*, o non fosse analizzabile, restituisce `None`.

  L'analisi delle date sembra essere un'arte oscura, e non tutti i server di posta aderiscono agli standard. Sebbene sia stata provata e riconosciuta corretta su un gran numero di messaggi email provenienti da fonti diverse, è tuttavia possibile che questa funzione possa occasionalmente produrre risultati non corretti.

getdate_tz(*name*)
: Recupera un'intestazione utilizzando `getheader()` e la analizza suddividendola in una tupla di dieci elementi; i primi 9 elementi formeranno una tupla compatibile con `time.mktime()`, ed il decimo sarà un numero che indica la differenza di fuso tra la zona della data e UTC. Notare che i campi 6, 7 e 8 non sono utilizzabili. In modo simile a `getdate()`, se non ci fosse nessuna intestazione corrispondente a *name* o fosse non analizzabile, restituisce `None`.

Le istanze `Message` supportano inoltre una interfaccia limitata di mappatura. In particolare: *m*`[name]` è simile a *m*`.getheader(name)` ma solleva l'eccezione `KeyError` se non ci sono intestazioni che corrispondono a *name*; e `len(`*m*`)`, *m*`.get(`*name*`[,` *default*`])`, *m*`.has_key(`*name*`)`, *m*`.keys()`, *m*`.values()` *m*`.items()`, e *m*`.setdefault(`*name*`[,` *default*`])` si comportano come previsto, con la sola differenza che `setdefault()` utilizza una stringa vuota come valore predefinito. Le istanze di `Message` inoltre supportano l'interfaccia di mappatura scrivibile *m*`[name] = value` e `del` *m*`[name]`. Gli oggetti `Message` non supportano i metodi `clear()`, `copy()`, `popitem()` o `update()` dell'interfaccia di mappatura. Il supporto per `get()` e `setdefault()` è stato aggiunto solamente in Python 2.2.

Infine, le istanze `Message` hanno alcune variabili d'istanza pubbliche:

headers
: Una lista che contiene l'intero insieme delle righe di intestazione, nell'ordine in cui sono state lette (con l'eccezione che le chiamate a setitem potrebbero influire su quest'ordine). Ogni linea viene terminata da un fine riga. La riga vuota che chiude le intestazioni non è contenuta nella lista.

fp
: L'oggetto file o simile a file passato al momento dell'istanziazione. Questo può venire usato per leggere il contenuto del messaggio.

unixfrom
: La riga UNIX corrispondente a '`From `', se il messaggio l'aveva, altrimenti una stringa vuota. Questo è necessario per rigenerare il messaggio in alcune circostanze, come per i file di mailbox in stile `mbox`.

12.11.2 Oggetti AddressList

Un'istanza di `AddressList` ha i seguenti metodi:

__len__()
: Restituisce il numero di indirizzi nella lista di indirizzi.

__str__()
: Restituisce una rappresentazione in forma di stringa della lista di indirizzi. Gli indirizzi sono resi nella forma nome <host@dominio>, separati da virgole.

__add__(*alist*)
: Restituisce una nuova istanza di `AddressList` che contiene tutti gli indirizzi di entrambi gli operandi di tipo `AddressList`, con i duplicati rimossi (corrisponde ad un'unione).

__iadd__(*alist*)
Versione simmetrica di `__add__()`; modifica questa istanza di `AddressList` nell'unione di sé stessa e di *alist*.

__sub__(*alist*)
Restituisce una nuova istanza di `AddressList` che contiene tutti gli indirizzi dell'operando di sinistra meno quelli presenti nell'operando di destra (corrisponde ad una differenza).

__isub__(*alist*)
Versione simmetrica di `__sub__()`, rimuovendo gli indirizzi in questa lista che sono anche in *alist*.

Infine, le istanze di `AddressList` hanno una variabile di istanza pubblica:

addresslist
Una lista di tuple, una per indirizzo, di due stringhe. In ogni membro, la prima stringa è il nome vero e proprio, la seconda l'indirizzo attuale (cioè sono coppie <nomeutente, nodo.dominio>, che sarebbero separate da '@').

12.12 base64 — RFC 3548: codifiche di dati Base16, Base32, Base64

Questo modulo fornisce la codifica e la decodifica dei dati specificati dalla RFC 3548. Questo standard definisce gli algoritmi Base16, Base32 e Base64 per la codifica e la decodifica di stringhe binarie arbitrarie in stringhe di testo che possono essere tranquillamente spedite tramite email, usate come parti di URL o incluse come parti di richieste HTTP POST. L'algoritmo di codifica non è il medesimo usato dal programma **uuencode**.

Il modulo fornisce due diverse interfacce. L'interfaccia moderna supporta la codifica e la decodifica di oggetti stringa usando tre alfabeti. La vecchia interfaccia ereditata fornisce la codifica e la decodifica per e da oggetti simile a file come fossero stringhe, ma usa solamente l'alfabeto Base64 standard.

L'interfaccia moderna fornisce:

b64encode(*s*[, *altchars*])
Codifica in stringa usando Base64.

s è la stringa da codificare. Il facoltativo *altchars* deve essere una stringa di lunghezza minima 2 (ulteriori caratteri vengono ignorati) che specifichi un alfabeto alternativo per i caratteri + e /. Questo consente ad applicazioni, per esempio, di generare delle URL o filesystem sicuramente in stringhe Base64. Il valore predefinito è `None`, per cui viene usato l'alfabeto Base64 standard.

La stringa codificata viene restituita.

b64decode(*s*[, *altchars*])
Decodifica la stringa codificata in Base64.

s è la stringa da decodificare. Il facoltativo *altchars* deve essere una stringa di lunghezza minima 2 (ulteriori caratteri vengono ignorati) che specifichi un alfabeto alternativo da usare al posto dei caratteri + e /.

La stringa decodificata viene restituita. Viene sollevata un'eccezione `TypeError` se *s* non viene riempita correttamente o se nella stringa vi sono contenuti caratteri non alfabetici.

standard_b64encode(*s*)
Codifica la stringa *s* usando l'alfabeto Base64 standard.

standard_b64decode(*s*)
Decodifica la stringa *s* usando l'alfabeto Base64 standard.

urlsafe_b64encode(*s*)
Codifica la stringa *s* usando un alfabeto sicuro per le URL, che sostituisca + con - e / con _ nell'alfabeto Base64 standard.

urlsafe_b64decode(*s*)
Decodifica la stringa *s* usando un alfabeto sicuro per le URL, che sostituisca + con - e / con _ nell'alfabeto Base64 standard.

b32encode(*s*)
Codifica una stringa usando Base32. *s* è la stringa da codificare. La stringa codificata viene restituita.

b32decode(*s*[, *casefold*[, *map01*]])
Decodifica una stringa codificata in base32.

s è la stringa da decodificare. Il facoltativo *casefold* è un'opzione che specifica se un alfabeto in minuscolo sia accettabile in input. Per questioni di sicurezza, il valore predefinito viene impostato a `False`.

La RFC 3548 permette una facoltativa mappatura della cifra 0 (zero) con la lettera O (oh) e la sempre facoltativa mappatura della cifra 1 (uno) sia alla lettera I (eye) o la lettera (el). L'argomento facoltativo *map01*, quando non `None`, specifica quale lettera della cifra 1 deve essere mappata (quando *map01* non è `None`, la cifra 0 è sempre mappata alla lettera O). Per questioni di sicurezza il valore predefinito è `None`, così 0 ed 1 non vengono permessi in input.

La stringa decodificata viene restituita. Viene sollevata un'eccezione `TypeError` se *s* non viene riempita correttamente o se nella stringa vi sono contenuti caratteri non alfabetici.

b16encode(*s*)
Codifica la stringa usando Base16.

s è la stringa da codificare. La stringa codificata viene restituita.

b16decode(*s*[, *casefold*])
Decodifica la stringa codificata in Base16.

s è la stringa da decodificare. Il facoltativo *casefold* è un'opzione che specifica se un alfabeto in minuscolo sia accettabile in input. Per questioni di sicurezza, il valore predefinito viene impostato a `False`.

La stringa decodificata viene restituita. Viene sollevata un'eccezione `TypeError` se *s* non viene riempita correttamente o se nella stringa vi sono contenuti caratteri non alfabetici.

La vecchia interfaccia ereditata:

decode(*input*, *output*)
Decodifica il contenuto del file *input* e scrive i dati binari risultanti sul file *output*. *input* ed *output* devono essere degli oggetti file, o qualsiasi altro oggetto che possiede almeno la stessa interfaccia di un oggetto file. *input* verrà letto finché *input*`.read()` non restituisce una stringa vuota.

decodestring(*s*)
Decodifica la stringa *s*, che deve contenere una o più linee di dati codificati in base64, e restituisce la stringa contenente i dati binari risultanti.

encode(*input*, *output*)
Codifica il contenuto del file *input* e scrive la stringa risultante nel file *output*. *input* ed *output* devono essere degli oggetti file, o qualsiasi altro oggetto che possiede almeno la stessa interfaccia di un oggetto file. *input* verrà letto finché *input*`.read()` non restituirà una stringa vuota. `encode()` restituisce i dati codificati insieme ad un carattere di fine riga (`'\n'`).

encodestring(*s*)
Codifica la stringa *s* di dati binari qualsiasi e la converte in una stringa codificata in base64. `encodestring()` restituisce una stringa con un carattere di fine riga (`'\n'`).

Vedete anche:

Modulo `binascii` (sezione 12.13):
Modulo di utilità che permette di convertire da ASCII a binario e viceversa.

RFC 1521, "*MIME (Multipurpose Internet Mail Extensions) Parte prima: Meccanismi per specificare e descrivere il corpo del mes*
Sezione 5.2, "Codifica-Trasferimento-Contenuto Base64", fornisce la definizione della codifica base64.

12.13 `binascii` — Conversione tra binario e ASCII

Il modulo `binascii` contiene un certo numero di metodi per convertire tra binario e vari tipi di rappresentazioni di sequenze binarie in ASCII. Normalmente, non si devono usare queste funzioni direttamente; è meglio usare altri moduli che invece le incapsulano (estendendole), come `uu` o `binhex`. Questo modulo esiste solo per permettere una gestione più veloce di grosse quantità di dati, che solitamente è piuttosto lenta in Python.

Il modulo `binascii` definisce le seguenti funzioni:

`a2b_uu`(*string*)
: Converte una singola riga di dati uuencoded in binario e restituisce tali dati. Le righe di solito contengono 45 byte (binari), tranne l'ultima. Alla riga di dati possono seguire degli spazi vuoti.

`b2a_uu`(*data*)
: Converte i dati binari in una riga di ASCII caratteri ed il valore restituito viene convertito in una riga, incluso un codice di controllo di fine riga. La lunghezza di *data* dovrebbe essere al massimo di 45 byte.

`a2b_base64`(*string*)
: Converte un blocco di dati base64 in binario e lo restituisce. È possibile passare più di una riga per volta.

`b2a_base64`(*data*)
: Converte i dati binari in una riga di ASCII caratteri codificata in base64. Il valore restituito viene convertito in una riga, incluso un codice di controllo di fine riga. La lunghezza di *data* dovrebbe essere al più di 57 byte attenendosi allo standard per base64.

`a2b_qp`(*string*[, *header*])
: Converte un blocco di dati caratteri quoted-printable in binario, restituendo i dati binari. Si può passare più di una riga per volta. Se è presente l'argomento facoltativo *header* ed ha valore vero, i caratteri di trattino basso verranno decodificati come spazi.

`b2a_qp`(*data*[, *quotetabs, istext, header*])
: Converte dati binari in una o più righe di caratteri ASCII codificati come quoted-printable. Il valore restituito sono le righe convertite. Se l'argomento facoltativo *quotetabs* è presente e vale true (NdT: vero), verranno codificati tutti gli spazi e tutte le tabulazioni. Se è presente l'argomento facoltativo *header* e vale true, gli spazi verranno codificati come i caratteri di trattino basso come da RFC1522. Se invece l'argomento facoltativo *header* vale false (NdT: falso), i caratteri di fine riga verranno codificati come tali, per prevenire corruzioni dei dati in caso di conversione di flussi di dati binari.

`a2b_hqx`(*string*)
: Converte i dati ASCII formattati con binhex4 in binario, senza eseguire una decompressione RLE. La stringa dovrebbe contenere un numero completo di byte binari, o (nel caso dell'ultima parte di dati binhex4) dovrebbe mantenere tutti i bit a zero.

`rledecode_hqx`(*data*)
: Esegue una decompressione RLE sui dati, come dallo standard binhex4. L'algoritmo usa `0x90` come indicatore di ripetizione dopo un byte, seguito da un numero che rappresenta il conteggio delle occorrenze. Il conteggio di `0` specifica il valore del byte di `0x90`. Questa funzione restituisce i dati decompressi, a meno che i dati di input non finiscano con un indicatore di ripetizione orfano, nel qual caso verrà sollevata l'eccezione `Incomplete`.

`rlecode_hqx`(*data*)
: Esegue una compressione RLE di tipo binhex4 sui dati, *data*, in input e restituisce i dati compressi.

`b2a_hqx`(*data*)
: Esegue una traduzione hexbin4 da binario ad ASCII e restituisce la stringa risultante. L'argomento dovrebbe essere già codificato con RLE, e dovrebbe avere una lunghezza divisibile per 3 (possibilmente con l'eccezione per l'ultimo frammento).

`crc_hqx`(*data, crc*)
: Calcola il valore crc in binhex4 dei dati *data*, a partire da un *crc* iniziale e restituendo il risultato.

`crc32`(*data*[, *crc*])
: Calcola il CRC-32, il checksum a 32 bit dei dati, a partire da un eventuale crc iniziale. È consistente con il checksum ZIP file. Dato che questo algoritmo è stato progettato come algoritmo di checksum, non è adatto per essere usato come un algoritmo di hash. Il giusto utilizzo è il seguente:

```
print binascii.crc32("ciao mondo")
# Oppure, in due pezzi:
crc = binascii.crc32("ciao")
crc = binascii.crc32(" mondo", crc)
print crc
```

b2a_hex(*data*)
hexlify(*data*)
Restituisce la rappresentazione esadecimale dei dati, *data*, binari. Ogni byte dei dati viene convertito in una rappresentazione esadecimale a 2 cifre. La stringa risultante ha quindi una lunghezza doppia rispetto ai dati binari *data*.

a2b_hex(*hexstr*)
unhexlify(*hexstr*)
Restituisce i dati binari rappresentati dalla stringa esadecimale *hexstr*. Questa funzione è l'inverso di `b2a_hex()`. *hexstr* deve contenere un numero pari di cifre decimali (che possono essere sia maiuscole che minuscole), altrimenti viene sollevata l'eccezione `TypeError`.

exception Error
Eccezione sollevata in caso di errori, per lo più di programmazione.

exception Incomplete
Eccezione sollevata in caso di dati incompleti. Questi non sono di solito errori di programmazione, ma possono essere evitati leggendo un po' più di dati e riprovando ad elaborarli.

Vedete anche:

Modulo `base64` (sezione 12.12):
Supporto per la codifca base64 usata in messaggi di posta elettronica MIME.

Modulo `binhex` (sezione 12.14):
Supporto per il formato binhex usato su Macintosh.

Modulo `uu` (sezione 12.16):
Supporto per la codifica UU usata su UNIX.

Modulo `quopri` (sezione 12.15):
Supporto per la codifica quoted-printable usata in messaggi di posta elettronica MIME.

12.14 `binhex` — Codifica e decodifica per file binhex4

Questo modulo codifica e decodifica file in formato binhex4, che permette di rappresentare i file Macintosh in ASCII. Su Macintosh, vengono codificati sia il contenuto del file che le informazioni per il finder, mentre sulle altre piattaforme viene gestito solamente il contenuto del file.

Il modulo `binhex` definisce le seguenti funzioni:

binhex(*input, output*)
Converte un file binario di nome *input* nel file in formato binhex *output*. Il parametro *output* può essere sia un vero file che un oggetto con la stessa interfaccia degli oggetti file (che supporti almeno i metodi `write()` e `close()`.

hexbin(*input*[, *output*])
Decodifica il file binhex di *input*. *input* può essere un vero file oppure un oggetto con un'interfaccia simile a quella degli oggetti file, che supporti almeno i metodi `read()` e `close()`. Il file risultante viene scritto sull'*output*, a meno che il secondo parametro non venga omesso; in tal caso il nome del file di *output* viene letto dal file di binhex in *input*.

Viene definita la seguente eccezione:

exception Error
Eccezione sollevata quando qualcosa non può essere codificato usando il formato binhex (per esempio, un nome di file è troppo lungo per rientrare nei parametri dei nomi di file), o quando l'input non è ben codificato rispetto al formato binhex.

Vedete anche:

Modulo `binascii` (sezione 12.13):
Modulo che permette conversioni da ASCII a binario e viceversa.

12.14.1 Note

Esiste un'interfaccia alternativa e più potente per il codificatore e decodificatore; vedere i sorgenti di questo modulo per i dettagli.

Se volete codificare i file di testo su piattaforme non Macintosh, verrà usata la convenzione Macintosh per i fine riga (ritorno-carrello come fine riga).

In questo momento, `hexbin()` non sembra lavorare bene in tutti i casi.

12.15 `quopri` — Codifica e decodifica di dati MIME quoted-printable

Questo modulo esegue una codifca o decodifica per trasporto in quoted-printable, come definito nella RFC 1521: "MIME (Multipurpose Internet Mail Extensions) Parte prima: Meccanismi per specificare e descrivere il Formato del Corpo dei Messaggi Internet". La codifica con caratteri quotati stampabili è progettata per dati dove ci sono relativamente pochi caratteri non stampabili: lo schema di codifica base64, disponibile attraverso il modulo `base64`, è più compatta se sono presenti molti di quei caratteri, come quando si invia un file grafico.

`decode`(*input, output*[,*header*])
: Decodifica il contenuto del file *input* e scrive risultante dato decodificato in binario nel file *output*. *input* ed *output* devono essere entrambi oggetti file o oggetti che rispecchiano l'interfaccia dell'oggetto file. *input* viene letto fino a che *input*`.readline()` non restituisce una stringa vuota. Se l'argomento facoltativo *header* è presente e vero, il carattere di trattino basso verrà decodificato come spazio. Questo viene usato per decodificare le intestazioni codificate "Q" come descritto nell'RFC 1522: "MIME (Multipurpose Internet Mail Extension) Parte seconda: Estensioni per intestazioni di messaggi per testo non Ascii".

`encode`(*input, output, quotetabs*)
: Codifica il contenuto del file *input* e scrive i risultanti dati, quotati e stampabili, nel file *output*. *input* ed *output* devono essere entrambi file oggetto o oggetti che rispecchiano l'interfaccia dell'oggetto file. *input* viene letto fino a che *input*`.readline()` non restituisce una stringa vuota. *quotetabs* è un'opzione che controlla come codificare gli spazi vuoti, e quando falso li lascia non codificati. Notate che gli spazi e i tabulatori che appaiono alla fine delle righe vengono sempre codificati, come indicati da RFC 1521.

`decodestring`(*s*[,*header*])
: Come `decode()`, tranne per il fatto che accetta una stringa sorgente e restituisce la corrispondente stringa decodificata.

`encodestring`(*s*[, *quotetabs*])
: Come `encode()`, tranne per il fatto che accetta una stringa sorgente e restituisce la corrispondete stringa decodificata. *quotetabs* è facoltativo (il suo valore predefinito è `0`), e viene passata direttamente ad `encode()`.

Vedete anche:

Modulo `mimify` (sezione 12.9):
: Utilità generica per gestire i messaggi in formato MIME.

Modulo `base64` (sezione 12.12):
: Codifica e decodifica dati in formato MIME Base64.

12.16 uu — Codifica e decodifica file in formato uuencode

Questo modulo codifica e decodifica file in formato uuencode, permettendo che dati binari arbitrari possano essere trasferiti su connessioni solo ASCII. Dovunque viene atteso un argomento composto da un oggetto file, i relativi metodi accettano un oggetto simile a file. Per compatibilità con il passato, viene accettata anche una stringa contenente un nome ed un percorso, ed il corrispondente file verrà aperto in lettura e scrittura; `'-'` indicato come nome con percorso viene interpretato come lo standard input o output. Comunque questa interfaccia è obsoleta; è preferibile per il chiamante aprire il file, ed essere sicuro che, quando richiesto, il modo sia `'rb'` or `'wb'` su Windows.

Questo codice è un contributo di Lance Ellinghouse, e modificato da Jack Jansen.

Il modulo `uu` definisce le seguenti funzioni:

`encode`(*in_file, out_file*[, *name*[, *mode*]])
Codifica il file uuencode *in_file* nel file *out_file*. Il file uuencode il cui l'intestazione specifica nome e modo per consentire la successiva decodifica del file. Il valore predefinito viene preso rispettivamente da *in_file* o `'-'` e `0666`.

`decode`(*in_file*[, *out_file*[, *mode*]])
Questa chiamata decodifica il file uuencoded *in_file*, inserendo il risultato nel file *out_file*. Se *out_file* è un nome con percorso, *mode* viene usato per impostare i bit dei permessi per quando il file viene creato. Il valore predefinito per *out_file* e *mode* vengono presi dalle intestazioni della codifica uuencode. Comunque, se il file specificato nell'intestazione esiste già, viene sollevata un'eccezione `uu.Error`.

eccezione `Error`()
Sottoclasse di `Exception`, questa può essere sollevata da `uu.decode()` in varie circostanze, come descritto sopra, ma anche includendo un'intestazione mal formata, o un file di input incompleto.

Vedete anche:

Modulo `binascii` (sezione 12.13):
Modulo di supporto contenente le conversioni da ASCII-a-binario e da binario-a-ASCII.

12.17 `xdrlib` — Codifica e decodifica dati XDR

Il modulo `xdrlib` supporta lo standard 'External Data Representation' (NdT: Rappresentazione di Dati Esterni), come descritto nell'RFC 1014, scritta da Sun Microsystems inc, nel giugno del 1987. Supporta molti dei tipi di dati descritti nell'RFC.

Il modulo `xdrlib` definisce due classi, una per impacchettare le variabili in rappresentazioni XDR ed un altro per scompattare dati dalla rappresentazione XDR. Esistono anche due classi di eccezioni.

class `Packer`()
`Packer` è la classe che impacchetta dei dati secondo la rappresentazione XDR. La classe `Packer` viene istanziata senza argomenti.

class `Unpacker`(*data*)
`Unpacker` è la classe complementare che estrae dei valori di dati XDR da un buffer di stringa. Il buffer di input viene passato con *data*.

Vedete anche:

RFC 1014, "*XDR: External Data Representation Standard*"
Questa RFC definisce la codifica dei dati secondo XDR al tempo in cui questo modulo fu originariamente scritto. È stata apparentemente resa obsoleta dalla RFC 1832.

RFC 1832, "*XDR: External Data Representation Standard*"
RFC più recente che fornisce una definizione rivisitata di XDR.

12.17.1 Oggetti Packer

Le istanze di `Packer` hanno i seguenti metodi:

`get_buffer`()
Restituisce il buffer corrente compattato sotto forma di stringa.

`reset`()
Azzera il buffer compattato in una stringa vuota.

In generale è possibile impacchettare tutti i più comuni tipi di dati XDR chiamando il metodo `pack_`*type*`()` appropriato. Ogni metodo accetta un singolo argomento, il valore da impacchettare. Vengono supportati i seguenti metodi di impacchettamento di tipi di dati semplici: `pack_uint()`, `pack_int()`, `pack_enum()`, `pack_bool()`, `pack_uhyper()` e `pack_hyper()`.

pack_float(*value*)
Impacchetta i valori numerici in virgola mobile presenti in *value* in precisione singola.

pack_double(*value*)
Impacchetta i valori numerici in virgola mobile presenti in *value* in precisione doppia.

I seguenti metodi supportano l'impacchettamento di stringhe, bytes e dati poco chiari:

pack_fstring(*n, s*)
Impacchetta una stringa *s* di lunghezza fissa. *n* è la lunghezza della stringa ma *non* viene impacchettata nel buffer dei dati. Se è necessario garantire l'allineamento a 4 byte, la stringa viene riempita da byte null.

pack_fopaque(*n, data*)
Impacchetta un flusso di dati poco chiari di lunghezza fissa, similmente a `pack_fstring()`.

pack_string(*s*)
Impacchetta una stringa *s* di lunghezza variabile. La lunghezza della stringa viene prima impacchettata come intero senza segno, in seguito i dati della stringa vengono impacchettati con `pack_fstring()`.

pack_opaque(*data*)
Impacchetta una stringa di dati poco chiari di lunghezza variabile, similmente a `pack_string()`.

pack_bytes(*bytes*)
Impacchetta un flusso di byte di lunghezza variabile, similmente a `pack_string()`.

I seguenti metodi supportano l'impacchettamento di array e liste:

pack_list(*list, pack_item*)
Impacchetta una lista, *list*, di elementi omogenei. Questo metodo risulta utile per liste di dimensioni indeterminate. Ad esempio la dimensione non è disponibile finché la lista non viene elaborata. Per ogni elemento della lista viene impacchettato prima un intero `1` senza segno, seguito dal valore dei dati presi dalla lista. *pack_item* è la funzione chiamata per impacchettare ogni elemento. Al termine della lista viene impacchettato un intero `0` senza segno.

Ad esempio per impacchettare una lista di interi il codice potrebbe apparire così:

```
import xdrlib
p = xdrlib.Packer()
p.pack_list([1, 2, 3], p.pack_int)
```

pack_farray(*n, array, pack_item*)
Impacchetta una lista di lunghezza fissa (*array*) di elementi omogenei. *n* è la lunghezza della lista; *non* viene impacchettata nel buffer ma se `len(`*array*`)` non è uguale a *n* viene sollevata un'eccezione `ValueError`. Come precedentemente descritto, *pack_item* è la funzione chiamata per impacchettare ogni elemento.

pack_array(*list, pack_item*)
Impacchetta una lista *list* di lunghezza variabile di elementi omogenei. Prima la lunghezza della lista viene impacchettata come intero senza segno, quindi ogni elemento viene impacchettato come nel metodo `pack_farray()` già visto.

12.17.2 Oggetti Unpacker

La classe `Unpacker` offre i seguenti metodi:

reset(*data*)
Resetta il buffer stringa con i dati *data* forniti.

get_position()
Restituisce la posizione di spacchettamento corrente nel buffer di dati.

set_position(*position*)
Imposta la posizione *position* di spacchettamento nel buffer di dati. Fare attenzione nell'usare `get_position()` e `set_position()`.

get_buffer()
Restituisce sotto forma di stringa il buffer di dati spacchettato corrente.

done()
Indica il completamento dello spacchettamento. Solleva un'eccezione `Error` se non sono stati estratti tutti i dati.

In aggiunta, qualunque tipo di dati che può essere immagazzinato con `Packer`, può essere estratto con un `Unpacker`. I metodi di estrazione seguono il modello `unpack_`*type*`()` senza argomenti. Restituiscono l'oggetto estratto.

unpack_float()
Spacchetta un numero in virgola mobile a precisione singola.

unpack_double()
Spacchetta un numero in virgola mobile a precisione doppia, in maniera simile a `unpack_float()`.

Inoltre i seguenti metodi estraggono stringhe, byte e dati opachi:

unpack_fstring(*n*)
Spacchetta e restituisce una stringa di lunghezza fissa. *n* è il numero previsto di caratteri. Per garantire l'allineamento a 4 byte si aggiungono una serie di byte null.

unpack_fopaque(*n*)
Spacchetta e restituisce un flusso di dati opachi di lunghezza fissa, in maniera simile ad `unpack_fstring()`.

unpack_string()
Spacchetta e restituisce una stringa di lunghezza variabile. La lunghezza della stringa viene prima estratta come intero senza segno quindi la stringa di dati viene estratta con `unpack_fstring()`.

unpack_opaque()
Spacchetta e restituisce una stringa di dati opachi di lunghezza variabile, in maniera simile a `unpack_string()`.

unpack_bytes()
Spacchetta e restituisce un flusso di byte di lunghezza variabile, in maniera simile ad `unpack_string()`.

I seguenti metodi supportano lo spacchettamento di array e liste:

unpack_list(*unpack_item*)
Spacchetta e restituisce una lista di elementi omogenei. La lista viene estratta un elemento per volta, estraendo prima un'opzione intera senza segno. Se l'opzione è `1` allora l'elemento viene estratto ed aggiunto alla lista. Un'opzione impostata a `0` indica la fine della lista. *unpack_item* è la funzione chiamata per estrarre gli elementi.

unpack_farray(*n, unpack_item*)
Estrae e restituisce (in forma di lista) un array di lunghezza fissa di elementi omogenei. *n* è il numero degli elementi della lista attesi nel buffer. Come già visto *unpack_item* è la funzione usata per estrarre ogni elemento.

unpack_array(*unpack_item*)
Spacchetta e restituisce una lista di lunghezza variabile di elementi omogenei. Per prima cosa la lunghezza della lista viene estratta come intero senza segno quindi viene estratto ogni elemento come in `unpack_farray()`.

12.17.3 Eccezioni

In questo modulo le eccezioni vengono codificate come istanze di classe:

exception Error
La classe base exception. `Error` possiede un solo dato membro pubblico `msg` che contiene la descrizione dell'errore.

exception ConversionError
Classe derivata da `Error`. Non contiene variabili d'istanza aggiuntive.

Ecco un esempio di come catturare una di queste eccezioni:

```
import xdrlib
p = xdrlib.Packer()
try:
    p.pack_double(8.01)
except xdrlib.ConversionError, instance:
    print 'packing the double failed:', instance.msg
```

12.18 netrc — Elaborazione di file netrc

Nuovo nella versione 1.5.2.

La classe `netrc` analizza ed incapsula il formato di file netrc usato dal programma **ftp** di UNIX e da altri client FTP.

class `netrc`([*file*])
Un'istanza o un'istanza della sotto classe `netrc` incapsula dati da un file netrc. L'argomento di inizializzazione, se presente, specifica il file da analizzare. Se non viene passato un argomento verrà letto il file '.netrc' presente nella directory home dell'utente. Gli errori di analisi solleveranno l'eccezione `NetrcParseError` con informazioni diagnostiche comprendenti il nome del file, il numero della riga ed il token che la termina.

exception `NetrcParseError`
Eccezione sollevata dalla classe `netrc` quando vengono incontrati errori di sintassi riscontrati nel testo sorgente. Le istanze di questa eccezione forniscono tre interessanti attributi: `msg` è una spiegazione testuale dell'errore, `filename` è il nome del file sorgente e `lineno` riporta il numero della riga in cui è stato trovato l'errore.

12.18.1 Oggetti netrc

Un'istanza `netrc` ha i seguenti metodi:

`authenticators`(*host*)
Restituisce una tupla di tre elementi (*login*, *account*, *password*) che servono ad effettuare l'autenticazione presso l'*host*. Se il file netrc non conteneva un valore per l'host, dato restituisce la tupla associata con il valore predefinito. Se non sono disponibili né l'host corrispondente né il valore predefinito restituisce `None`.

`__repr__`()
Scarica i dati della classe come se fosse una stringa nel formato del file netrc. Questo elimina i commenti e può riarrangiare le voci presenti.

Le istanze di `netrc` hanno variabili pubbliche d'istanza:

`hosts`
Dizionario che mappa i nomi di host con le tuple (*login*, *account*, *password*). La voce con il valore predefinito, se esiste, viene rappresentato come uno pseudo host con quel nome.

`macros`
Dizionario che mappa i nomi delle macro alle liste di stringhe.

Note: Le password vengono limitate ad un sotto insieme dell'insieme dei caratteri ASCII. Le versioni di questo modulo precedenti alla 2.3 erano estremamente limitate. A partire dalla 2.3 nelle password è consentita tutta la punteggiatura ASCII. Notare comunque che nelle password non sono consentiti lo spazio ed i caratteri non stampabili. Questa limitazione è dovuta al modo in cui viene analizzato il file .netrc e potrebbe essere rimossa in futuro.

12.19 `robotparser` — Parser per robots.txt

Questo modulo fornisce un'unica classe, `RobotFileParser`, che risponde ad interrogazioni sulla possibilità o meno che un particolare user agent possa prelevare una URL sul sito Web che pubblica il file 'robots.txt'. Per maggiori dettagli sulla struttura dei file 'robots.txt' vedere http://www.robotstxt.org/wc/norobots.html.

class `RobotFileParser`()

Questa classe fornisce un insieme di metodi per leggere, analizzare e rispondere ad interrogazioni su un singolo file 'robots.txt'.

`set_url`(*url*)
Imposta l'URL che fa riferimento ad un file 'robots.txt'.

`read`()
Legge l'URL di 'robots.txt' e lo passa al parser.

`parse`(*lines*)
Analizza l'argomento *lines*.

`can_fetch`(*useragent, url*)
Ritorna `True` se *useragent* è abilitato a recuperare l'url in base alle regole contenute nel file analizzato 'robots.txt'.

`mtime`()
Restituisce l'orario in cui il file `robots.txt` è stato recuperato l'ultima volta. Questa informazione è utile per programmi web spider che rimangono in esecuzione per molto tempo e che richiedono di verificare periodicamente eventuali nuovi file `robots.txt`.

`modified`()
Assegna l'orario in cui il file `robots.txt` è stato recuperato per l'ultima volta all'ora attuale.

L'esempio seguente dimostra l'uso della classe `RobotFileParser`.

```
>>> import robotparser
>>> rp = robotparser.RobotFileParser()
>>> rp.set_url("http://www.musi-cal.com/robots.txt")
>>> rp.read()
>>> rp.can_fetch("*", "http://www.musi-cal.com/cgi-bin/search?city=San+Francisco")
False
>>> rp.can_fetch("*", "http://www.musi-cal.com/")
True
```

12.20 `csv` — Lettura e scrittura di file CSV

Nuovo nella versione 2.3.

Il cosiddetto formato CSV (NdT: acronimo inglese di "valori separati da virgole") è il più comune formato per l'importazione e l'esportazione in fogli elettronici e database. Non esiste uno "standard CSV" esplicito, di conseguenza il formato viene definito in modo implicito dalle molte applicazioni che lo leggono e lo scrivono. La mancanza di uno standard comporta l'esistenza di sottili differenze tra i dati prodotti da un'applicazione e gli stessi dati letti da un'altra. Queste differenze possono rendere l'analisi dei file CSV generati da diverse applicazioni un po' difficoltosa. Inoltre, mentre i separatori ed i caratteri per quotare il testo possono variare, il formato complessivo è sufficientemente simile e quindi ha reso possibile scrivere un singolo modulo che gestisca in modo efficiente questi aspetti, nascondendo al programmatore i dettagli relativi alla lettura ed alla scrittura.

Il modulo `csv` implementa delle classi per leggere e scrivere dati da forma tabulare in file CSV. Il modulo permette al programmatore di dire: "Scrivi questi dati nel formato preferito da Excel," oppure "leggi questi dati che sono stati generati da Excel," senza la necessità di conoscere i dettagli relativi al formato CSV utilizzato da Excel. I programmatori possono anche descrivere il formato CSV utilizzato dalle loro applicazioni o definire degli opportuni formati CSV per loro scopi particolari.

Gli oggetti `reader` e `writer` del modulo `csv` leggono e scrivono sequenze. I programmatori possono anche leggere e scrivere dati sotto forma di dizionari, utilizzando le classi `DictReader` e `DictWriter`.

Note: Questa versione del modulo `csv` non supporta input in formato unicode. Inoltre, ci sono attualmente alcune problematiche relative ai caratteri ASCII NUL. Di conseguenza i dati in input dovrebbero essere caratteri ASCII stampabili per non creare problemi. Queste restrizioni verranno rimosse in futuro.

Vedete anche:

PEP 305, "*CSV File API*"
La proposta di miglioramento di Python che ha suggerito questa aggiunta a Python.

12.20.1 Contenuti del modulo

Il modulo `csv` definisce le seguenti funzioni:

`reader`(*csvfile*[, *dialect='excel'*[, *fmtparam*]])
Restituisce un oggetto reader che effettuerà un'iterazione sulle righe di un dato *csvfile*. *csvfile* può essere qualsiasi oggetto che supporti il protocollo `iterator` e che restituisca una stringa ogni volta che il suo metodo `next` viene chiamato. Se *csvfile* è un oggetto di tipo file deve essere stato aperto con l'opzione 'b' in tutte le piattaforme dove questo può fare differenza. Un parametro facoltativo *dialect* può essere utilizzato per definire una serie di parametri specifici di un particolare dialetto CSV. Tale parametro può essere un'istanza di una sotto classe di una classe `Dialect` oppure una delle stringhe restituite dalla funzione `list_dialects`. L'ulteriore parametro facoltativo *fmtparam* può essere utilizzato per sostituire alcuni parametri di formattazione previsti dal dialetto scelto, vedete la sezione 12.20.2, "Dialetti e parametri di formattazione" per i dettagli relativi a questi parametri.

Tutti i dati letti vengono restituiti sotto forma di stringhe. Non viene effettuata alcuna conversione automatica del tipo.

`writer`(*csvfile*[, *dialect='excel'*[, *fmtparam*]])
Restituisce un oggetto writer responsabile per la conversione dei dati dell'utente da stringhe limitate ad un oggetto simile a file. *csvfile* può essere qualsiasi oggetto con un metodo `write`. Se *csvfile* è un oggetto di tipo file deve essere stato aperto con il flag 'b' in tutte le piattaforme dove questo può fare differenza. Un parametro facoltativo *dialect* può essere utilizzato per definire una serie di parametri specifici di un particolare dialetto CSV. Tale parametro può essere un'istanza di una sotto classe di una classe `Dialect`, oppure una delle stringhe restituite dalla funzione `list_dialects`. L'ulteriore parametro facoltativo *fmtparam* può essere utilizzato per sostituire alcuni parametri di formattazione previsti dal dialetto scelto, vedete la sezione 12.20.2, "Dialetti e parametri di formattazione" per i dettagli relativi a questi parametri. Per rendere più semplice possibile l'utilizzo di questo modulo insieme ad altri moduli che implementano la DB API, il valore `None` viene scritto come una stringa vuota. Sebbene questa sia una trasformazione irreversibile, tuttavia rende più facile la scrittura di valori SQL NULL all'interno di file CSV senza la necessità di elaborare i dati restituiti dalla chiamata `cursor.fetch*()`. Tutti i dati che non sono di tipo stringa vengono convertiti a tale tipo mediante `str()` prima di essere scritti.

`register_dialect`(*name, dialect*)
Associate *dialect* con *name*. *dialect* deve essere una sotto classe della classe `csv.Dialect`. *name* deve essere una stringa o un oggetto Unicode.

`unregister_dialect`(*name*)
Cancella dall'elenco dei dialetti registrati dal dialetto associato al parametro *name*. Viene sollevata un'eccezione `Error` se *name* non è un dialetto registrato.

`get_dialect`(*name*)
Restituisce il dialetto associato al parametro *name*. Viene sollevata un'eccezione `Error` se *name* non è un dialetto registrato.

`list_dialects`()
Restituisce i nomi di tutti i dialetti registrati.

Il modulo `csv` definisce le seguenti classi:

class `DictReader`(*csvfile*[, *fieldnames=None*,[, *restkey=None*[, *restval=None*[, *dialect='excel'*[, **args, **kwds*]]]]])
Crea un oggetto che opera come un normale `reader` ma che mappa l'informazione letta all'interno di un dizionario le cui chiavi vengono passate dal parametro facoltativo opzionale *fieldnames*. Se il parametro

fieldnames è assente vengono utilizzate come chiavi i valori presenti nella prima riga di *csvfile*. Se una riga letta ha meno valori della sequenza *fieldnames*, il valore *restval* viene usato come valore predefinito. Se una riga letta ha più valori della sequenza *fieldnames*, i valori rimanenti vengono aggiunti come ad una sequenza, la cui chiave associata è *restkey*. Se una riga letta ha meno valori della sequenza *fieldnames*, le chiavi rimanenti acquisiscono il loro valore dal parametro facoltativo *restval*. Qualsiasi altro argomento facoltativo o a parola chiave viene passato alla sottostante istanza `reader`.

class `DictWriter`(*csvfile*, *fieldnames*[, *restval*=''[, *extrasaction*='raise'[, *dialect*='excel'[, **args*, ***kwds*]]]])

Crea un oggetto che opera come un normale `writer` ma che mappa dizionari in righe di output. Il parametro *fieldnames* specifica l'ordine con il quale i valori del dizionario passato al metodo `writerow()` vengono scritti in *csvfile*. Il parametro facoltativo *restval* specifica il valore da scrivere se il dizionario non contiene una particolare chiave. Se il dizionario passato al metodo `writerow()` contiene una chiave non presente in *fieldnames*, il parametro facoltativo *extrasaction* indica l'azione da eseguire. Se viene impostato a `'raise'` viene sollevata un'eccezione `ValueError`. Se viene impostato a `'ignore'`, i valori eccedenti vengono ignorati. Qualsiasi altro argomento facoltativo o a parola chiave viene passato alla sottostante istanza `writer`.

Notare che, diversamente da quanto accade per la classe `DictReader`, il parametro *fieldnames* di `DictWriter` non è facoltativo. Dato che i dizionari, `dict`, in Python non sono ordinati, risulta mancante l'informazione circa l'ordine con il quale la riga dovrebbe essere scritta in *csvfile*.

class `Dialect`

La classe `Dialect` è un contenitore importante principalmente per i sui attributi, che vengono utilizzati come parametri per una particolare istanza `reader` o `writer`.

class `Sniffer`()

La classe `Sniffer` viene usata per dedurre il formato di un file CSV.

La classe `Sniffer` fornisce un solo metodo:

`sniff`(*sample*[,*delimiters=None*])

Analizza un dato *sample* e restituisce una sotto classe di `Dialect` che rispecchia i parametri trovati. Se il parametro facoltativo *delimiters* è presente, viene interpretato come stringa contenente i possibili caratteri separatori validi.

`has_header`(*sample*)

Analizza un semplice testo, *sample*, (presupponendo che sia in formato CSV) e restituisce `True` se la prima riga sembra essere una serie di colonne di intestazioni.

Il modulo `csv` definisce le seguenti costanti:

`QUOTE_ALL`

Ordina agli oggetti `writer` di quotare tutti i campi.

`QUOTE_MINIMAL`

Ordina agli oggetti `writer` di quotare solo i campi che contengono il delimitatore, *delimiter*, corrente o quelli iniziano con *quotechar*, ovvero il carattere utilizzato per quotare il testo.

`QUOTE_NONNUMERIC`

Ordina agli oggetti `writer` di quotare tutti i campi che non contengono valori numerici.

`QUOTE_NONE`

Ordina agli oggetti `writer` di non quotare i campi. Quando il carattere *delimiter* corrente compare nei dati di output viene preceduto da *escapechar*, ovvero il carattere di protezione corrente. Quando è attiva `QUOTE_NONE` è un errore non definire un carattere singolo di protezione anche se nessun dato da scrivere contiene il carattere *delimiter* corrente.

Il modulo `csv` definisce la seguente eccezione:

exception `Error`

Viene sollevata quando viene rilevato un errore in una delle funzioni del modulo

12.20.2 Dialetti e parametri di formattazione

Per semplificare la descrizione del formato di input ed output dei record, i vari parametri di formattazione vengono raggruppati in dialetti. Un dialetto è una sotto classe della classe `Dialect` che possiede una serie di metodi specifici ed un solo metodo `validate()`. Durante la creazione di oggetti `reader` o `writer`, il programmatore può specificare una stringa o una sotto classe della classe `Dialect` come parametro indicante un dialetto. In aggiunta a tale parametro indicante il dialetto o in sostituzione di questo, il programmatore può specificare particolari parametri di formattazione, aventi gli stessi nomi degli attributi di seguito definiti per la classe `Dialect`.

I dialetti possiedono i seguenti attributi:

`delimiter`
Stringa che ha un solo carattere che viene utilizzata per separare i vari campi. Il suo valore predefinito è `','`.

`doublequote`
Indica come occorrenze del carattere usato per quotare, *quotechar*, debbano essere a loro volta quotate. Se `True`, il carattere viene duplicato. Se `False`, il carattere di protezione, *escapechar*, deve essere una stringa composta da un solo carattere che viene usata come prefisso al carattere di quotatura *quotechar*. Il suo valore predefinito è `True`.

`escapechar`
Stringa avente un solo carattere che viene utilizzata per proteggere il delimitatore, *delimiter*, se *quoting* viene impostato a `QUOTE_NONE`. Il suo valore predefinito è `None`.

`lineterminator`
Stringa che indica la fine di una riga in un file CSV. Il suo valore predefinito è `'\r\n'`.

`quotechar`
Stringa avente un solo carattere che viene utilizzata per quotare elementi che contengano il delimitatore *delimiter* o che inizino con il carattere utilizzato per quotare *quotechar*. Il suo valore predefinito è `''`.

`quoting`
Indica in che modo il `writer` quota i campi. Può assumere uno qualsiasi dei valori delle costanti `QUOTE_*` (vedere la sezione 12.20.1) ed il suo valore predefinito è `QUOTE_MINIMAL`.

`skipinitialspace`
Se `True`, lo spazio immediatamente di seguito al delimitatore *delimiter* viene ignorato. Il suo valore predefinito è `False`.

12.20.3 Oggetti Reader

Gli oggetti `reader` (istanze `DictReader` e oggetti restituiti dalla funzione `reader()`) possiedono i seguenti metodi pubblici:

`next``()`
Restituisce la riga successiva di un oggetto iterabile reader sotto forma di lista, analizzata in base al dialetto corrente.

12.20.4 Oggetti Writer

Gli oggetti `Writer` (istanze `DictWriter` ed oggetti restituiti dalla funzione `writer()`) hanno i seguenti metodi pubblici. *row* deve essere una sequenza di stringhe o un numero per oggetti `Writer` e dizionari che mappano campi di nomi con stringhe o numeri (tramite il loro passaggio prima attraverso `str()`) per oggetti `DictWriter`. Notare che i numeri complessi vengono scritti delimitati da parentesi. Questo può causare qualche problema per altri programmi che leggono file CSV (assumendo che anche questi supportino i numeri complessi).

`writerow``(`*row*`)`
Scrive il parametro *row* nell'oggetto file del writer, formattato in base al dialetto corrente.

`writerows``(`*rows*`)`

Scrive tutte le righe del parametro *rows* (una lista di oggetti *row* come descritto in precedenza) nell'oggetto file del writer, formattate in base al dialetto corrente.

12.20.5 Esempi

Il programma "Hello, world" per la lettura csv è:

```
import csv
reader = csv.reader(file("some.csv", "rb"))
for row in reader:
    print row
```

Il corrispondente piu' semplice esempio di scrittura csv è

```
import csv
writer = csv.writer(file("some.csv", "wb"))
for row in someiterable:
    writer.writerow(row)
```

CAPITOLO

TREDICI

Strumenti per l'analisi dei linguaggi di markup strutturati

Python supporta una grande varietà di moduli per lavorare con varie forme di linguaggi di markup strutturati. Questi includono moduli per lavorare con lo Standard Generalized Markup Language (SGML) e l'HyperText Markup Language (HTML), e diverse interfacce per utilizzare l'eXtensible Markup Language (XML).

È importante notare che i moduli nel package `xml` richiedono la presenza di almeno un parser (NdT: analizzatore) XML SAX-compatibile. A partire da Python 2.3, il parser Expat è incluso in Python, in modo che il modulo `xml.parsers.expat` sia sempre disponibile. Si potrebbe anche volere il package aggiuntivo PyXML add-on package; questo package fornisce un set esteso di librerie XML per Python.

La documentazione per i package `xml.dom` e `xml.sax` è la definizione dei collegamenti per le interfacce DOM e SAX di Python.

`HTMLParser`	Un semplice parser per gestire HTML e XHTML.
`sgmllib`	Niente più che un analizzatore SGML necessario per analizzare il formato HTML.
`htmllib`	Un parser per i documenti HTML
`htmlentitydefs`	Definizioni generali di entità HTML.
`xml.parsers.expat`	Un'interfaccia Python al parser Expat non validante XML.
`xml.dom`	API del Modello di Oggetto Documento per Python – DOM.
`xml.dom.minidom`	Implementazione minimale del Document Object Model (DOM).
`xml.dom.pulldom`	Supporto per la costruzione di alberi DOM parziali da eventi SAX.
`xml.sax`	Package che contiene classi base ed utili funzioni per SAX2.
`xml.sax.handler`	Classe di base per gestori di eventi SAX.
`xml.sax.saxutils`	Funzioni convenienti e classi da usarsi con SAX.
`xml.sax.xmlreader`	Interfaccia per implementazioni di parser XML SAX compatibili.
`xmllib`	Un parser per documenti XML.

Vedete anche:

Python/XML Libraries
(http://pyxml.sourceforge.net/)
: Home page del package PyXML, contenente una estensione del package `xml` per Python.

13.1 `HTMLParser` — Semplice parser per HTML e XHTML

Questo modulo definisce una classe `HTMLParser` che rappresenta la base per analizzare file di testo formattati in HTML (HyperText Markup Languages) e XHTML. Diversamente dal parser in `htmllib`, questo non è basato sul parser SGML `sgmllib`.

class `HTMLParser`()
: La classe `HTMLParser` viene istanziata senza argomenti.

  Un'istanza di HTMLParser scorre i dati in HTML e richiama funzioni di gestione quando i tag vengono aperti e chiusi. La classe `HTMLParser` è stata ideata per essere sovrascritta dall'utente per ottenere i comportamenti desiderati.

Diversamente dal parser in `htmllib`, questo non controlla che il tag di chiusura coincida con quello di apertura e non richiama il gestore dei tag di chiusura per gli elementi che vengono implicitamente chiusi dalla chiusura di altri elementi.

Le istanze di `HTMLParser` hanno i seguenti metodi:

`reset``()`
Reimposta l'istanza. Tutti i dati non analizzati andranno persi. Questo metodo viene invocato implicitamente al momento dell'istanziazione.

`feed``(`*data*`)`
Invia dati al parser. Il testo a questo punto viene analizzato come se fosse un elemento completo; i dati incompleti vengono bufferizzati finché non vengono inviati altri dati o viene invocato il metodo `close()`.

`close``()`
Forza l'analisi di tutti i dati bufferizzati come se fossero seguiti dal loro tag di chiusura. Questo metodo può essere ridefinito in una classe derivata per definire ulteriori analisi alla fine dell'input, ma la versione ridefinita dovrebbe sempre invocare il metodo `close()` sulla classe base `HTMLParser`.

`getpos``()`
Restituisce il numero di riga corrente e l'offset.

`get_starttag_text``()`
Restituisce il testo dell'ultimo tag di apertura analizzato. Normalmente questo non dovrebbe essere necessario per una analisi, ma potrebbe essere utile da distribuire con HTML "così come sviluppato" o per rigenerare l'input con cambiamenti minimali (spazi tra gli attributi preservati, ecc. etc.).

`handle_starttag``(`*tag, attrs*`)`
Questo metodo viene invocato per gestire l'apertura di un tag. È pensato per essere sovrascritto in una classe derivata; l'implementazione nella classe base non fa niente.

L'argomento *tag* è il nome del tag scritto in caratteri minuscoli. L'argomento *attrs* è una lista di coppie `(`*nome*`, `*valore*`)` che contengono gli attributi contenuti tra le parentesi angolari `<>` del tag. Il nome *name* viene convertito in caratteri minuscoli e le virgolette doppie ed i backslash nel valore *value* dell'attributo, vengono interpretati. Ad esempio, per il tag `<A HREF=http://www.cwi.nl/>`, il metodo sarà '`handle_starttag('a', [('href', 'http://www.cwi.nl/')])`'.

`handle_startendtag``(`*tag, attrs*`)`
Simile a `handle_starttag()`, ma viene chiamato quando il parser incontra un tag vuoto in stile XHTML (`<a .../>`). Il metodo può essere sovrascritto da una sotto classe che richiede questa particolare informazione lessicale; l'implementazione predefinita invoca semplicemente `handle_starttag()` e `handle_endtag()`.

`handle_endtag``(`*tag*`)`
Questo metodo è invocato per gestire il tag di chiusura di un elemento. È progettato per essere sovrascritto da una classe derivata; l'implementazione della classe base non fa nulla. L'argomento *tag* è il nome del tag in caratteri minuscoli.

`handle_data``(`*data*`)`
Questo metodo viene invocato per analizzare dati arbitrari. È progettato per essere sovrascritto da una classe derivata; l'implementazione della classe base non fa nulla.

`handle_charref``(`*name*`)`
Questo metodo viene invocato per analizzare il riferimento ad un carattere della forma '`&#`*ref*`;`'. È progettato per essere sovrascritto da una classe derivata; l'implementazione della classe base non fa nulla.

`handle_entityref``(`*name*`)`
Questo metodo viene invocato per analizzare un generico riferimento ad un'entità dalla forma '`&`*name*`;`' dove *name* è un generico riferimento ad un'entità. È progettato per essere sovrascritto da una classe derivata; l'implementazione della classe base non fa nulla.

`handle_comment``(`*data*`)`
Questo metodo viene invocato quando viene incontrato un commento. L'argomento *comment* è una stringa che contiene del testo racchiuso tra i delimitatori '`-`' e '`-`', ma non i delimitatori stessi. Per esempio, per il commento '`<!-text->`', il metodo verrà richiamato con l'argomento `'text'`. È progettato per essere sovrascritto da una classe derivata; l'implementazione della classe base non fa nulla.

handle_decl(*decl*)
Questo metodo viene invocato quando il parser legge una dichiarazione dichiarazione SGML. Il parametro *decl* dovrà essere interamente contenuto nella dichiarazione markup `<!...>`. È progettato per essere sovrascritto da una classe derivata; l'implementazione della classe base non fa nulla.

handle_pi(*data*)
Questo metodo viene invocato quando il parser incontra una istruzione che comporta un'analisi. Il parametro *data* contiene l'intera istruzione di analisi. Per esempio, per l'istruzione `<?proc color='red'>`, il metodo verrà chiamato in questo modo: `handle_pi(proc color='red')`. È progettato per essere sovrascritto da una classe derivata; l'implementazione della classe base non fa nulla.

Note: La classe `HTMLParser` usa le regole sintattiche dell'SGML per analizzare le istruzioni. Se una istruzione di analisi XHTML usa il carattere'`?`', questo '`?`' verrà incluso nel parametro *data*.

13.1.1 Esempio di applicazione di un parser HTML

Come esempio base, sotto è presentato un Parser HTML che utilizza la classe `HTMLParser` per stampare a video i tag quando vengono incontrati:

```
from HTMLParser import HTMLParser

class MyHTMLParser(HTMLParser):

    def handle_starttag(self, tag, attrs):
        print "Incontrata l'apertura del tag %s tag" % tag

    def handle_endtag(self, tag):
        print "Incontrata la chiusura del tag %s tag" % tag
```

13.2 `sgmllib` — Simple SGML parser

Questo modulo definisce la classe `SGMLParser` che serve da base per analizzare file di testo formattati in SGML (Standard Generalized Mark-up Language). Infatti non fornisce un completo parser SGML – è solamente un parser SGML fino a che viene usato da HTML, ed il modulo esiste solamente come base per il modulo `htmllib`. Un altro parser HTML che supporta XHTML ed offre un'interfaccia leggermente differente è disponibile nel modulo `HTMLParser`.

class SGMLParser()
La classe `SGMLParser` si istanzia senza argomenti. Il parser è progettato per riconoscere i seguenti costrutti:

- Tag di apertura e chiusura, rispettivamente nella forma '`<`*tag attr=value* `...>`' e '`</`*tag*`>`', rispettivamente.
- Riferimenti a codici di controllo numerici nella forma '`&#`*name*`;`'.
- Riferimenti ad entità nella forma '`&`*name*`;`'.
- Commenti SGML nella forma '`<!-`*text*`-->`'. Notare che codici di controllo relativi a spazi, tabulazioni e fine riga sono permessi se inseriti tra '`>`' ed immediatamente preceduti da '`--`'.

Le istanze di `SGMLParser` hanno la seguente interfaccia a metodi:

reset()
Reinizializza l'istanza. Tutti i dati non analizzati andranno persi. Questo metodo viene chiamato implicitamente al momento dell'istanziazione.

setnomoretags()
Ferma l'analisi dei tag. Tratta tutti i dati seguenti come input letterali (CDATA). Questo metodo è fornito solo perché così può essere implementato il tag HTML `<PLAINTEXT>`.

setliteral()
Entra in modalità stringa costante (modalità CDATA).

feed(*data*)
Invia dei dati al parser. Viene analizzato finché consiste di elementi completi; i dati incompleti vengono bufferizzati fino a che non saranno inviati altri dati o non sia chiamato il metodo `close()`.

close()
Forza l'analisi di tutti i dati bufferizzati come se fossero seguiti da un carattere di marcatura fine del file. Questo metodo può essere ridefinito tramite una classe derivata, per definire ulteriori analisi alla fine dell'input, ma la versione ridefinita dovrebbe sempre chiamare il metodo `close()`.

get_starttag_text()
Restituisce il testo dell'ultimo tag di apertura analizzato. Normalmente questo non dovrebbe essere necessario per una analisi, ma potrebbe essere utile da distribuire con HTML "così come sviluppato" o per rigenerare l'input con cambiamenti minimali (spazi tra gli attributi preservati, etc. etc.).

handle_starttag(*tag, method, attributes*)
Questo metodo viene invocato per gestire l'apertura di un tag per il quale sia stato definito il metodo `start_`*tag*`()` o il `do_`*tag*`()`. L'argomento *tag* è il nome del tag convertito in caratteri minuscoli, e l'argomento *method* è il metodo associato che dovrebbe essere usato per supportare l'interpretazione semantica del tag di apertura. L'argomento *attributes* è una lista di coppie (*nome*, *valore*) che contengono gli attributi trovati tra le parentesi angolari <> del tag. Il nome viene convertito in caratteri minuscoli mentre le virgolette doppie ed i backslash vengono interpretati. Per esempio per il tag `<A HREF=http://www.cwi.nl/>`, questo metodo verrà chiamato con '`handle_starttag('a', [('href', 'http://www.cwi.nl/')])`'. L'implementazione base semplicemente chiama *method* con *attributes* come unico argomento.

handle_endtag(*tag, method*)
Questo metodo viene invocato per gestire la chiusura di un tag per il quale un metodo `end_`*tag*`()` sia stato definito. L'argomento *tag* è il nome del tag in caratteri minuscoli e l'argomento *method* è il metodo associato che dovrebbe essere utilizzato per supportare l'interpretazione semantica della chiusura del tag. Se non è stato definito alcun metodo `end_`*tag*`()` per l'elemento che è stato chiuso, questo gestore non viene invocato. L'implementazione base chiama semplicemente *method*.

handle_data(*data*)
Questo metodo viene invocato per analizzare dati arbitrari. È progettato per essere sovrascritto da una classe derivata; l'implementazione nella classe base non fa nulla.

handle_charref(*ref*)
Questo metodo viene chiamato per analizzare un riferimento a caratteri nella forma '`&#`*ref*`;`'. Nell'implementazione base, *ref* deve essere un numero decimale compreso tra 0 e 255. Il metodo converte il carattere in ASCII e chiama il metodo `handle_data()` con il carattere come argomento. Se *ref* non è valido o è fuori dai valori consentiti, viene chiamato il metodo `unknown_charref(`*ref*`)` che gestisce l'errore. Una sotto classe deve sovrascrivere questo metodo per fornire il supporto per le entità a caratteri viste sopra.

handle_entityref(*ref*)
Questo metodo viene invocato per analizzare un riferimento generico ad entità nella forma '`&`*ref*`;`' dove *ref* è un riferimento generico ad entità. Cerca *ref* nella variabile d'istanza (o classe) `entitydefs` che dovrebbe essere una mappatura dai nomi di entità alle corrispondenti traduzioni. Se viene trovata una traduzione, il metodo invoca `handle_data()` con la traduzione; altrimenti, chiama il metodo `unknown_entityref(`*ref*`)`. La variabile predefinita `entitydefs` definisce le traduzioni per `&`, `&apos;`, `>`, `<` e `&quot;`.

handle_comment(*comment*)
Questo metodo viene chiamato quando si incontra un commento. L'argomento *comment* è una stringa che contiene il testo compreso tra '`<!-`' e '`->`', delimitatori esclusi. Per esempio il commento '`<!-text->`' fa in modo che il metodo venga chiamato con '`'text'`' come argomento. Il metodo predefinito non fa niente.

handle_decl(*data*)
Questo metodo viene chiamato quando il parser legge una dichiarazione SGML. In pratica, la dichiarazione `DOCTYPE` è l'unica che viene osservata in documenti HTML, ma il parser non discrimina dichiarazioni dif-

ferenti (o errate). Sottosezioni all'interno di dichiarazioni `DOCTYPE` non vengono supportate. Il parametro *data* consiste nell'intero contenuto della dichiarazione nel marcatore `<!...>`. L'implementazione predefinita non fa niente.

`report_unbalanced`(*tag*)
Questo metodo viene chiamato incontrando un tag di chiusura a cui manca il corrispettivo tag di apertura.

`unknown_starttag`(*tag, attributes*)
Questo metodo viene chiamato incontrando un tag di apertura sconosciuto. È stato progettato per essere sovrascritto da una classe derivata; l'implementazione della classe base non fa niente.

`unknown_endtag`(*tag*)
Questo metodo viene chiamato incontrando un tag di chiusura sconosciuto. È stato progettato per essere sovrascritto da una classe derivata; l'implementazione della classe base non fa niente.

`unknown_charref`(*ref*)
Questo metodo viene invocato per elaborare riferimenti irrisolvibili a caratteri numerici. Vedere `handle_charref()` per determinare cosa viene gestito in modo predefinito. È stato progettato per essere sovrascritto da una classe derivata; l'implementazione della classe base non fa niente.

`unknown_entityref`(*ref*)
Questo metodo viene chiamato per processare un riferimento ad un'entità sconosciuta. È stato progettato per essere sovrascritto da una classe derivata; l'implementazione della classe base non fa niente.

Oltre a poter estendere o sovrascrivere i metodi elencati sopra, le classi derivate possono anche definire metodi della seguente forma, per definire come comportarsi con tag specifici. In nomi dei tag nel flusso in input non sono sensibili alle differenze tra maiuscole e minuscole; i *tag* che si trovano nei nomi dei metodi invece devono essere scritti minuscoli:

`start_`*tag*(*attributes*)
Questo metodo viene chiamato per processare un tag di apertura chiamato *tag*. Ha la preferenza rispetto al metodo `do_`*tag*`()`. L'argomento *attributes* ha lo stesso significato dell'omonimo argomento descritto sopra per `handle_starttag()`.

`do_`*tag*(*attributes*)
Questo metodo viene chiamato per processare un tag di apertura chiamato *tag* che non presenta un tag di chiusura. L'argomento *attributes* ha lo stesso significato dell'omonimo argomento descritto sopra per `handle_starttag()`.

`end_`*tag*()
Questo metodo è richiamato per processare un tag di chiusura chiamato *tag*.

Notare che il parser mantiene uno stack di elementi aperti per i quali non è ancora stato trovato un tag di chiusura. Solo i tag elaborati da `start_`*tag*`()` vengono messi in questo stack, la definizione del metodo `end_`*tag*`()` è facoltativa per questi tag. Per i tag elaborati dal metodo `do_`*tag*`()` o da `unknown_tag()`, non deve essere definito nessun metodo `end_`*tag*`()`, se fosse definito non verrebbe comunque usato. Se entrambi i metodi `start_`*tag*`()` e `do_`*tag*`()` vengono definiti per uno stesso tag, il metodo `start_`*tag*`()` ha la precedenza.

13.3 `htmllib` — A parser for HTML documents

Questo modulo definisce una classe che può essere utilizzata come base per analizzare file di testo formattati in HyperText Mark-up Language (HTML). La classe non riguarda direttamente l'I/O — gli deve essere inviato un input in forma di stringa attraverso un metodo, poi fa le chiamate ai metodi di un oggetto "formatter" in modo da produrre un output. La classe `HTMLParser` viene progettata per essere usata da classe base per altre classi con lo scopo di aggiungere funzionalità, infatti molti dei suoi metodi possono essere estesi o sovrascritti. Da parte sua, questa classe deriva dalla classe `SGMLParser`, definita in `sgmllib` e la estende. L'implementazione di `HTMLParser` supporta il linguaggio HTML 2.0 come descritto nell'RFC 1866. Il modulo `formatter` fornisce due implementazioni dell'oggetto formatter; ci si riferisca alla documentazione per quel modulo per informazioni sull'interfaccia formatter.

Il seguente è un sommario delle interfacce definite da `sgmllib.SGMLParser`:

- L'interfaccia per aggiungere dati ad un'istanza attraverso il metodo `feed()`, che prende come argomento una stringa. Questo metodo può essere chiamato con tanto o poco testo alla volta, come si desidera; '`p.feed(a); p.feed(b)`' hanno lo stesso effetto di '`p.feed(a+b)`'. Quando i dati contengono tag HTML completi, vengono elaborati immediatamente; gli elementi incompleti vengono memorizzati in un buffer. Per forzare l'elaborazione di tutti i dati non elaborati, chiamare il metodo `close()`.

  Per esempio, per analizzare l'intero contenuto di un file, usare:

  ```
  parser.feed(open('myfile.html').read())
  parser.close()
  ```

- L'interfaccia per definire le semantiche dei tag HTML sono molto semplici: deriva una classe e definisce i metodi chiamati `start_`*tag*`()`, `end_`*tag*`()` o `do_`*tag*`()`. Il parser li chiamerà al momento opportuno: `start_`*tag* o `do_`*tag*`()` vengono chiamati quando troverà l'apertura del tag di forma <*tag* ...>; `end_`*tag*`()` viene chiamato quando troverà la chiusura del tag nella forma <*tag*>. Quando l'apertura di un tag richiede una corrispondente chiusura, come `<H1>` ... `</H1>`, la classe dovrebbe definire il metodo `start_`*tag*`()`; se il tag richiede un tag che non necessita di chiusura, come `<P>`, la classe dovrebbe definire il metodo `do_`*tag*`()`.

Il modulo definisce una singola classe:

class `HTMLParser`(*formatter***)**
Questa è la classe di base della formattazione HTML. Questa supporta tutte le entità richieste per le specifiche XHTML 1.0 (http://www.w3.org/TR/xhtml1). Definisce anche i gestori per tutti gli elementi HTML 2.0 e molti degli elementi HTML 3.0 e HTML 3.2.

Vedete anche:

Modulo `formatter` (sezione 12.1):
Definizione di interfaccia per trasformare un flusso astratto di eventi di formattazione in specifici eventi di uscita su oggetti scrivibili.

Modulo `HTMLParser` (sezione 13.1):
Un parser HTML alternativo che offre una vista un po' più a basso livello dell'input, ma disegnato per lavorare con XHTML, non implementa alcune delle sintassi SGML usate in "HTML as deployed", che è illegale per XHTML.

Modulo `htmlentitydefs` (sezione 13.4):
Definizione del testo sostitutivo per le entità XHTML 1.0.

Modulo `sgmllib` (sezione 13.2):
Classe base per `HTMLParser`.

13.3.1 Oggetti HTMLParser

In aggiunta ai metodi tag, la classe `HTMLParser` fornisce alcuni metodi e variabili istanza da usare senza i metodi tag.

`formatter`
Questa è l'istanza di formattazione associata con il parser.

`nofill`
Opzione booleana che dovrebbe essere vera quando gli spazi bianchi non dovrebbero essere collassati, o falso in caso contrario. In generale questo dovrebbe essere vero solo quando dati caratteri dovrebbero essere trattati come testo "preformattato", come senza l'elemento `<PRE>`. Il valore predefinito è falso. Questo influenza le operazioni `handle_data()` e `save_end()`.

`anchor_bgn`(*href, name, type***)**
Questo metodo viene chiamato all'inizio di una regione ancorata. Gli argomenti corrispondono agli attributi del tag `<A>` con gli stessi nomi. L'implementazione predefinita mantiene una lista degli iperlink (definiti dagli attributi `HREF` per i tag `<A>`) dentro il documento. La lista degli iperlink è disponibile come attributo dei dati `anchorlist`.

anchor_end()
Questo metodo viene chiamato alla fine di una regione ancorata. L'implementazione predefinita aggiunge una marcatore testuale a pié di pagina usando un indice nella lista degli iperlink creati da `anchor_bgn()`.

handle_image(*source, alt*[, *ismap*[, *align*[, *width*[, *height*]]]])
Questo metodo viene chiamato dalle immagini manipolate. L'implementazione predefinita passa semplicemente il valore *alt* al metodo `handle_data()`.

save_bgn()
Inizia memorizzando i dati carattere in un buffer al posto di spedirli all'oggetto formatter. Ricerca i dati conservati attraverso `save_end()`. L'uso della coppia `save_bgn()` / `save_end()` potrebbe non essere annidato.

save_end()
Finisce la bufferizzazione dei dati carattere e restituisce tutti i dati salvati dalla precedente chiamata a `save_bgn()`. Se l'opzione `nofill` è falsa, lo spazio vuoto viene collassato in singoli spazi. Una chiamata a questo metodo senza una precedente chiamata a `save_bgn()` solleverà un'eccezione `TypeError`.

13.4 `htmlentitydefs` — Definizioni generali di entità HTML

Questo modulo definisce tre dizionari, `name2codepoint`, `codepoint2name` e `entitydefs`. `entitydefs` viene usato dal modulo `htmllib` per fornire il membro `entitydefs` della classe `HTMLParser`. La definizione qui fornita contiene tutte le entità definite da XHTML 1.0 che può essere gestita usando una semplice sostituzione di testo nell'insieme dei caratteri Latin-1 (ISO-8859-1).

entitydefs
Un dizionario che mappa le entità delle definizioni XHTML per la loro sostituzione in testo ISO Latin-1.

name2codepoint
Un dizionario che mappa i nomi delle entità HTML per i codepoints Unicode. Nuovo nella versione 2.3.

codepoint2name
Un dizionario che mappa i codepoints Unicode per i nomi di entità HTML. Nuovo nella versione 2.3.

13.5 `xml.parsers.expat` — Analizzare velocemente XML usando Expat

Nuovo nella versione 2.0.

Il modulo `xml.parsers.expat` è un'interfaccia Python all'analizzatore Expat non validante XML. Il modulo fornisce un tipo singolo di estensione, `xmlparser`, che rappresenta lo stato corrente di un parser XML. Dopo che un oggetto `xmlparser` viene creato, vari attributi dell'oggetto possono essere impostati alle funzioni del gestore. Quando un documento XML viene quindi passato al parser, vengono chiamate le funzioni del gestore per i dati relativi ai caratteri ed il margine nel documento XML.

Questo modulo usa il modulo `pyexpat` per fornire l'accesso al parser Expat. L'uso diretto del modulo `pyexpat` è deprecato.

Questo modulo fornisce un'eccezione ed un tipo oggetto:

exception ExpatError
L'eccezione sollevata quando Expat riporta un errore. Vedere la sezione 13.5.2, "Eccezioni ExpatError," per maggiori informazioni sull'interpretazione degli errori Expat.

exception error
un alias per `ExpatError`.

XMLParserType
Il tipo del valore restituito dalla funzione `ParserCreate()`.

Il modulo `xml.parsers.expat` contiene due funzioni.:

`ErrorString`(*errno*)
Restituisce una stringa esplicativa per l'errore dato numero *errno*.

`ParserCreate`([*encoding*[, *namespace_separator*]])
Crea e restituisce un nuovo oggetto `xmlparser`. *encoding*, se specificato, deve essere una stringa che chiama la codifica usata dai dati XML. Expat non supporta nessun'altra codifica come fa Python ed il suo repertorio di codifiche non può essere esteso; supporta UTF-8, UTF-16, ISO-8859-1(Latin1) e ASCII. Se *encoding* è presente, sovrascriverà l'implicita o esplicita codifica del documento.

Expat può, facoltativamente, fare l'elaborazione dello spazio dei nomi XML per voi, abilitarlo per fornire un valore per lo *namespace_separator*. Il valore deve essere una stringa di un carattere; verrà sollevata l'eccezione un `ValueError` se la stringa ha una lunghezza illegale (`None` viene considerato alla stregua di un'omissione). Quando l'elaborazione dello spazio dei nomi viene abilitata, i nomi degli elementi tipo che appartengono allo spazio dei nomi verranno espansi. L'elemento con nome, passato agli elementi di gestione `StartElementHandler` e `EndElementHandler` rappresenteranno la concatenazione dello spazio dei nomi URI, il carattere separatore dello spazio dei nomi e la parte locale del nome. Se il separatore dello spazio dei nomi viene impostato a zero byte (`chr(0)`) allora lo spazio dei nomi URI e la parte locale verranno concatenate senza alcun separatore.

Per esempio, se *namespace_separator* viene impostato sul carattere spazio (' ') e viene analizzato il seguente documento:

```
<?xml version="1.0"?>
<root xmlns    = "http://default-namespace.org/"
      xmlns:py = "http://www.python.org/ns/">
  <py:elem1 />
  <elem2 xmlns="" />
</root>
```

`StartElementHandler` riceverà la seguente stringa per ogni elemento:

```
http://default-namespace.org/ root
http://www.python.org/ns/ elem1
elem2
```

Vedete anche:

Il parser XML Expat
(http://www.libexpat.org/)
Home page del progetto Expat.

13.5.1 Oggetti XMLParser

Gli oggetti `xmlparser` hanno i seguenti metodi:

`Parse`(*data*[, *isfinal*])
Analizza i contenuti dei dati di una stringa, chiamando le appropriate funzioni di gestione per elaborare i dati, *data*, analizzati. *isfinal* deve essere vero nella chiamata finale a questo metodo. *data* può essere la stringa vuota in qualsiasi momento.

`ParseFile`(*file*)
Analizza i dati XML leggendo da un *file* oggetto. *file* ha solamente bisogno di fornire il metodo `read(`*nbytes*`)`, restituendo la stringa vuota dove non ci sono più dati.

`SetBase`(*base*)
Imposta la base che deve essere usata per risolvere le relative URI negli identificatori di sistema nelle dichiarazioni. La risoluzione dei relativi identificatori viene lasciata all'applicazione: questo valore verrà passato come argomento *base* per le funzioni `ExternalEntityRefHandler`, `NotationDeclHandler` e `UnparsedEntityDeclHandler`.

GetBase()
Restituisce una stringa contenente le impostazioni base da una precedente chiamata a SetBase() o None se SetBase() non è stato chiamato.

GetInputContext()
Restituisce i dati di input che vengono generati dall'evento corrente come una stringa. I dati sono nella codifica dell'entità che ha generato il testo. Quando chiamata mentre un evento di gestione non è attivo, il valore restituito è None. Nuovo nella versione 2.1.

ExternalEntityParserCreate(*context*[, *encoding*])
Crea un analizzatore "figlio" che può essere usato per analizzare un'entità esterna analizzata, con riferimento al contenuto analizzato dal parser genitore. Il parametro *context* dovrebbe essere la stringa passata alla funzione di gestione ExternalEntityRefHandler(), descritta sotto. Il parser figlio viene creato con ordered_attributes, returns_unicode e specified_attributes impostati al valore di questo parser.

Gli oggetti xmlparser hanno i seguenti attributi:

buffer_size
La dimensione del buffer usata quando buffer_text è vero. Questi valori, a questo punto, non possono essere modificati. Nuovo nella versione 2.3.

buffer_text
Impostare buffer_text a vero comporta che l'oggetto xmlparser, rispetto al buffer del contenuto del testo restituito da Expat, eviti molteplici chiamate a CharacterDataHandler() ogni qualvolta sia possibile. Questo può sostanzialmente migliorare le prestazioni da quando Expat normalmente spezza dati costituiti da caratteri in prossimità della fine della riga. Il valore predefinito per questo attributo è falso e può essere modificato in ogni momento. Nuovo nella versione 2.3.

buffer_used
Se buffer_text è abilitato, rappresenta il numero dei byte immagazzinati in un buffer. Questi byte rappresentano la codifica di testo UTF-8. Questo attributo non ha rappresentazioni significative quando buffer_text viene impostato a falso. Nuovo nella versione 2.3.

ordered_attributes
Impostare questo attributo ad un intero diverso da zero comporta il riconoscimento degli attributi come lista piuttosto che come dizionario. Gli attributi vengono rappresentati nello stesso ordine trovato nel documento di testo. Per ogni attributo sono presenti due voci di lista: il nome dell'attributo ed il valore dell'attributo. Anche la versione più vecchia di questo modulo usa lo stesso formato. Per definizione questo attributo è falso; esso può essere cambiato in ogni momento. Nuovo nella versione 2.1.

returns_unicode
Se questo attributo è impostato ad un intero differente da zero, le funzioni di gestione verranno passate come stringhe Unicode. Se returns_unicode è 0, stringhe a 8 bit contenenti dati codificati UTF-8 verranno passate agli appropriati gestori. Modificato nella versione 1.6: Può essere cambiata in ogni momento per influenzare il tipo risultante..

specified_attributes
Se impostato ad un intero diverso da zero, il parser riporterà solo quegli attributi specificati nel documento istanza e non quelli derivati dalle dichiarazioni degli attributi. Le applicazioni così impostate necessitano di particolare attenzione per usare quelle informazioni addizionali rintracciabili nelle dichiarazioni, che devono essere necessariamente compatibili con gli standard per lo sviluppo dei processori XML. Per definizione, questo attributo è falso; può essere cambiato in ogni momento. Nuovo nella versione 2.1.

I seguenti attributi contengono valori relativi ai più recenti errori incontrati dall'oggetto xmlparser, e avranno solo valori corretti quando un'eccezione xml.parsers.expat.ExpatError verrà sollevata da una chiamata a Parse() o ParseFile().

ErrorByteIndex
Indice del byte dove viene riscontrato l'errore.

ErrorCode
Codice numerico che specifica il problema. Questo valore può essere passato alla funzione ErrorString(), o confrontato con una delle costanti definite nell'oggetto errors.

ErrorColumnNumber
Numero di colonna dove viene riscontrato l'errore.

ErrorLineNumber
Il numero della riga dove viene riscontrato l'errore.

Qui c'è una lista dei gestori che possono essere impostati. Per impostare un gestore su un oggetto `xmlparser` *o*, usare *o*`.`*handlername* `=` *func*. *handlername* deve essere preso dalla lista seguente, e *func* deve essere un oggetto chiamabile che accetta il corretto numero di argomenti. Gli argomenti sono tutte stringhe, salvo indicazioni contrarie.

XmlDeclHandler(*versione, codifica, standalone*)
Chiamato quando una dichiarazione XML viene analizzata. La dichiarazione XML è la dichiarazione (facoltativa) dell'applicabilità della versione XML raccomandata, la codifica del documento di testo ed una facoltativa dichiarazione "standalone". *versione* e *codifica* dovrebbero essere stringhe del tipo dettato dall'attributo `returns_unicode` e *standalone* verrà impostato ad `1` se il documento viene dichiarato standalone, `0` se non viene dichiarato standalone o `-1` se la clausola standalone viene omessa. Questo è disponibile solamente con Expat versione 1.95.0 o più recenti. Nuovo nella versione 2.1.

StartDoctypeDeclHandler(*doctypeName, systemId, publicId, has_internal_subset*)
Chiamato quando Expat inizia ad analizzare la dichiarazione del tipo del documento (`<!DOCTYPE ...`). Il *doctypeName* viene fornito esattamente come presentato. I parametri *systemId* e *publicId* forniscono gli identificatori del sistema locale e pubblico se specificati, o `None` se omessi. *has_internal_subset* sarà vero se il documento contiene internamente un altro documento ed è presente un dichiarazione del suddetto sotto insieme. Questo richiede la versione di Expat 1.2 o più recente.

EndDoctypeDeclHandler()
Chiamato quando Expat sta analizzando la dichiarazione del tipo del documento. Richiede la versione di Expat 1.2 o più recente.

ElementDeclHandler(*name, model*)
Chiamato una volta per ogni dichiarazione del tipo di elemento. *name* è il nome del tipo di elemento e *model* è la rappresentazione del contenuto del modello.

AttlistDeclHandler(*elname, attname, tipo, default, required*)
Chiamato per ogni dichiarazione di attributo per ogni tipo di elemento. Se la dichiarazione, in forma di lista, degli attributi dichiara tre attributi, questo gestore viene chiamato tre volte, una volta aperto, ogni attributo. *elname* è il nome dell'elemento al quale la dichiarazione si applica e *attname* è il nome dell'attributo dichiarato. Il tipo di attributo è una stringa passata come tipo *type*; i possibili valori sono `'CDATA'`, `'ID'`, `'IDREF'`, ... *default* fornisce il valore predefinito per l'attributo usato, quando l'attributo non viene specificato dall'istanza del documento, o `None` se non c'è un valore predefinito (`#IMPLIED` valore). Se l'attributo viene richiesto per essere fornito all'istanza del documento, *required* dovrebbe essere vero. Questo richiede Expat versione 1.95.0 o più recente.

StartElementHandler(*name, attributes*)
Chiamato per l'avvio di ogni elemento. *name* è una stringa contenente il nome dell'elemento, e *attributes* è il dizionario che tiene traccia dei nomi degli attributi attraverso i loro valori.

EndElementHandler(*name*)
Chiamato per la fine di ogni elemento.

ProcessingInstructionHandler(*target, data*)
Chiamato per ogni istruzione eleborata.

CharacterDataHandler(*data*)
Chiamato per i dati carattere. Questo verrà chiamato per i dati carattere normali, contenuti marcati CDATA, e spazi vuoti ignorabili. Le applicazioni che devono distinguere questi casi possono usare le chiamate `StartCdataSectionHandler`, `EndCdataSectionHandler` e `ElementDeclHandler` per raccogliere le informazioni richieste.

UnparsedEntityDeclHandler(*entityName, base, systemId, publicId, notationName*)
Chiamata per dichiarazioni di entità non analizzate (NDATA). Questo è presente solo nella versione 1.2 della libreria Expat; per versioni più recenti, usare invece `EntityDeclHandler`. La funzione sottostante nella libreria di Expat è stata dichiarata obsoleta.

EntityDeclHandler(*entityName, is_parameter_entity, value, base, systemId, publicId, notationName*)
Chiamato da tutte le dichiarazioni di entità. Per parametri ed entità interne, *value* sarà una stringa che fornirà il contenuto dichiarato dall'entità; questo varrà `None` per le entità esterne. Il parametro *notationName* avrà il valore `None` per le entità analizzate, ed il nome della notazione per le entità non analizzate. *is_parameter_entity* sarà vero se l'entità è un parametro o falso per entità generiche (la maggior parte delle applicazioni ha bisogno soltanto di essere interessata ad entità generiche). Questo comportamento è disponibile solamente a partire dalla versione 1.95.0 della libreria Expat. Nuovo nella versione 2.1.

NotationDeclHandler(*notationName, base, systemId, publicId*)
Chiamato per notazioni di dichiarazioni. *notationName*, *base*, *systemId* e *publicId* sono stringhe, se presenti. Se l'identificatore pubblico viene omesso, *publicId* varrà `None`.

StartNamespaceDeclHandler(*prefix, uri*)
Chiamato quando un elemento contiene una dichiarazione sullo spazio dei nomi. Le dichiarazioni sullo spazio dei nomi vengono elaborate prima che `StartElementHandler` venga chiamato per l'elemento sul quale le dichiarazioni vengono disposte.

EndNamespaceDeclHandler(*prefix*)
Chiamato quando il tag di chiusura viene richiesto per un elemento che contiene una dichiarazione sullo spazio dei nomi. Questo viene chiamato una volta per ogni dichiarazione concernente lo spazio dei nomi, sull'elemento, nell'ordine inverso per il quale `StartNamespaceDeclHandler` veniva chiamato per indicare l'inizio di ogni ambito di visibilità dello spazio dei nomi. Le chiamate a questo gestore vengono fatte dopo il corrispondente `EndElementHandler` per la fine dell'elemento.

CommentHandler(*data*)
Chiamato per i commenti. *data* è il testo del commento, escluso i caratteri di inizio '`<!--`' e fine '`-->`'.

StartCdataSectionHandler()
Chiamato all'inizio di una sezione CDATA. Questo e `StartCdataSectionHandler` vengono richiesti per essere in grado di identificare l'inizio e la fine sintattiche per le sezioni CDATA.

EndCdataSectionHandler()
Chiamato alla fine di una sezione CDATA.

DefaultHandler(*data*)
Chiamato da ogni carattere nel documento XML per il quale non è stato specificato nessun gestore applicabile. Questo significa che i caratteri fanno parte di un costrutto che potrebbe essere riportato, ma per il quale non è stato fornito alcun gestore.

DefaultHandlerExpand(*data*)
Questo è la stessa cosa del `DefaultHandler`, ma non inibisce l'espansione delle entità interne. Il riferimento all'entità non verrà passato al gestore predefinito.

NotStandaloneHandler()
Called if the Chiamato nel documento XML che non è stato dichiarato come inizio di un documento autonomo. Questo succede quando c'è un sotto insieme esterno o un riferimento ad un parametro entità, ma la dichiarazione XML non imposta in autonomia a `sì` una dichiarazione XML. Se il gestore restituisce `0`, allora il parser provocherà un errore `XML_ERROR_NOT_STANDALONE`. Se questo gestore non è impostato, non viene sollevata alcuna eccezione per questa condizione del parser.

ExternalEntityRefHandler(*context, base, systemId, publicId*)
Chiamato per riferimento ad entità esterne. *base* è la base corrente, è stata impostata da una chiamata precedente a `SetBase()`. Gli identificatori *systemId* e *publicId*, se passati, sono stringhe; se l'identificatore pubblico non viene passato, *publicId* varrà `None`. Il valore di *context* è poco chiaro, dovrebbe essere usato solo come descritto qui sotto.

Per analizzare entità esterne da analizzare, deve essere implementato questo gestore. È responsabile della creazione di un sotto parser usando `ExternalEntityParserCreate(`*context*`)`, inizializzandolo con le appropriate callbacks ed analizzando l'entità. Questo gestore dovrebbe restituire un intero; se restituisce `0`, il parser provocherà un errore `XML_ERROR_EXTERNAL_ENTITY_HANDLING`, altrimenti l'analisi continuerà.

Se il gestore non viene fornito, le entità esterne vengono riportate dal `DefaultHandler`, sempre che questo sia fornito.

13.5.2 ExpatError Exceptions

Le eccezioni `ExpatError` hanno un numero interessante di attributi:

`code`
Il numero di errore interno di Expat per l'errore specifico. Questo ricercherà una delle costanti definite nell'oggetto `errors` da questo modulo. Nuovo nella versione 2.1.

`lineno`
Numero di riga al quale l'errore è stato rilevato. La prima riga è numerata con la cifra `1`. Nuovo nella versione 2.1.

`offset`
Carattere offset nella riga dove è stato rilevato l'errore. La prima colonna viene rappresentata dalla cifra `0`. Nuovo nella versione 2.1.

13.5.3 Esempio

Il seguente programma definisce tre gestori che stampano soltanto i loro argomenti.

```
import xml.parsers.expat

# 3 funzioni di gestori
def start_element(name, attrs):
    print 'Start element:', name, attrs
def end_element(name):
    print 'End element:', name
def char_data(data):
    print 'Character data:', repr(data)

p = xml.parsers.expat.ParserCreate()

p.StartElementHandler = start_element
p.EndElementHandler = end_element
p.CharacterDataHandler = char_data

p.Parse("""<?xml version="1.0"?>
<parent id="top"><child1 name="paul">Text goes here</child1>
<child2 name="fred">More text</child2>
</parent>""", 1)
```

L'output da questo programma è:

```
Start element: parent {'id': 'top'}
Start element: child1 {'name': 'paul'}
Character data: 'Text goes here'
End element: child1
Character data: '\n'
Start element: child2 {'name': 'fred'}
Character data: 'More text'
End element: child2
Character data: '\n'
End element: parent
```

13.5.4 Descrizione del modello di contenuto

I modelli di contenuto vengono descritti usando tuple annidate. Ogni tupla contiene quattro valori: il tipo, il quantificatore, il nome ed una tupla di figli. I figli sono semplici descrizioni del modello di contenuto.

Il valore dei primi due campi sono costanti definite nell'oggetto `model` del modulo `xml.parsers.expat`. Queste costanti possono essere collezionate in due gruppi: il gruppo del tipo di modello ed il gruppo quantificatore.

Le costanti nel gruppo del tipo di modello sono:

XML_CTYPE_ANY
L'elemento chiamato dal nome del modello era stato dichiarato per avere il modello di contenuto di `ANY`.

XML_CTYPE_CHOICE
L'elemento chiamato permette la scelta da un numero di opzioni; questo viene usato per i modelli di contenuto come `(A | B | C)`.

XML_CTYPE_EMPTY
Gli elementi che vengono dichiarati per essere `EMPTY` hanno questo tipo di modello.

XML_CTYPE_MIXED

XML_CTYPE_NAME

XML_CTYPE_SEQ
I modelli indicati con questo tipo di modello rappresentano una serie di modelli che seguono uno dopo l'altro e vengono indicati con questo tipo di modello. Questo viene usato per modelli come `(A, B, C)`.

Le costanti nel gruppo quantificatore sono:

XML_CQUANT_NONE
Non viene passato alcun modificatore, così può apparire esattamente una volta, come per `A`.

XML_CQUANT_OPT
Il modello è facoltativo, può apparire una volta sola o nessuna, come per `A?`.

XML_CQUANT_PLUS
Il modello deve essere presente una o più volte (come `A+`).

XML_CQUANT_REP
Il modello deve essere presente zero o più volte, come per `A*`.

13.5.5 Costanti relative ad errori in Expat

Le seguenti costanti sono presenti nell'oggetto `errors` del modulo `xml.parsers.expat`. Queste costanti sono utili nell'interpretazione di alcuni degli attributi dell'oggetto di eccezione `ExpatError`, sollevato quando avviene un errore.

L'oggetto `errors` ha i seguenti attributi:

XML_ERROR_ASYNC_ENTITY

XML_ERROR_ATTRIBUTE_EXTERNAL_ENTITY_REF
Un riferimento ad un'entità in un valore di attributo riferito ad un'entità invece che ad un'entità interna.

XML_ERROR_BAD_CHAR_REF
Un riferimento ad un carattere per un carattere che è illegale in XML (per esempio, caratteri `0`, o '`&#0;`').

XML_ERROR_BINARY_ENTITY_REF
Un riferimento ad un'entità riferito ad un'entità che è stata dichiarata con una notazione, per cui non può essere analizzata.

XML_ERROR_DUPLICATE_ATTRIBUTE
Un attributo che è stato usato più di una volta in un tag iniziale.

XML_ERROR_INCORRECT_ENCODING

XML_ERROR_INVALID_TOKEN

Eccezione sollevata quando un byte in ingresso non può essere correttamente assegnato ad un carattere; per esempio, un byte NULL (valore `0`) in un flusso di ingresso UTF-8.

`XML_ERROR_JUNK_AFTER_DOC_ELEMENT`
Qualcosa di diverso di uno spazio vuoto è arrivato dopo l'elemento del documento.

`XML_ERROR_MISPLACED_XML_PI`
Una dichiarazione XML che viene trovata da qualche altra parte, invece che all'inizio dei dati in ingresso.

`XML_ERROR_NO_ELEMENTS`
Il documento non contiene alcun elemento (XML richiede che tutti i documenti contengano esattamente un elemento di livello principale).

`XML_ERROR_NO_MEMORY`
Expat non è stato in grado di allocara memoria al suo interno.

`XML_ERROR_PARAM_ENTITY_REF`
Un riferimento ad un parametro di entità che è stato trovato dove non era consentito.

`XML_ERROR_PARTIAL_CHAR`

`XML_ERROR_RECURSIVE_ENTITY_REF`
Un riferimento ad entità contiene un'altro riferimento alla medesima entità; per mezzo di un nome differente e possibilmente in modo indiretto.

`XML_ERROR_SYNTAX`
Alcuni non meglio specificati errori di sintassi sono stati riscontrati.

`XML_ERROR_TAG_MISMATCH`
Un tag di chiusura che non corrisponde al relativo tag di apertura.

`XML_ERROR_UNCLOSED_TOKEN`
Alcuni token (come un tag di apertura) non sono stati chiusi prima della fine del flusso, o è stato trovato un ulteriore token.

`XML_ERROR_UNDEFINED_ENTITY`
È stato creato un riferimento ad un'entità che non era stata definita.

`XML_ERROR_UNKNOWN_ENCODING`
La codifica del documento non è supportata da Expat.

13.6 `xml.dom` — API del Modello di Oggetto Documento – DOM

Nuovo nella versione 2.0.

Il modello di oggetto di documento, o "DOM," è una API disponibile per molti linguaggi del W3C (consorzio mondiale per il web) per l'accesso e la modifica di documenti XML. Un'implementazione DOM presenta un documento XML come una struttura ad albero, o permette al codice del cliente di costruire queste strutture da zero. Fornisce l'accesso alla struttura attraverso un insieme di oggetti che forniscono interfacce ben conosciute.

Il DOM è estremamente utile per un'applicazione con accesso casuale. Solo SAX permette di vedere anche un solo bit per volta del documento. Se si sta guardando un elemento SAX, non si ha accesso ad un altro elemento. Se si sta guardando verso un nodo di testo, non si ha accesso ai nodi che lo contengono. Quando si scrive una applicazione SAX, si deve tener presente nel proprio codice della posizione del programma all'interno del documento. SAX non fa questo al posto dello sviluppatore. Se si ha la necessità di dare uno sguardo dietro al documento XML, si può dire che siete sfortunati.

Alcune applicazione sono semplicemente impossibili in un modello di gestione ad eventi senza l'accesso a un albero. Di solito ci si può costruire una qualche sorta di albero degli eventi SAX, ma DOM permette di evitare la scrittura di questo codice. DOM è una rappresentazione ad albero standard per i dati XML.

Il modello di oggetto di documento (DOM) è stato definito dal W3C in strati, o "livelli" nella lora terminologia comune. Il modo che ha Python di mappare le API è sostanzialmente basato sulle raccomandazioni di livello 2 del W3C per DOM. L'adeguamento alle specifiche di livello 3, è disponibile attualmente solo in un formato di bozza, ed è in sviluppo da parte del gruppo di interesse specifico XML di Python come parte del package

PyXMLPyXML. Fare riferimento alla documentazione allegata con questo package per informazioni sullo stato corrente del supporto al livello 3 di DOM.

Le applicazioni DOM si avviano tipicamente analizzando un po' di codice XML in un DOM. Come questo viene svolto non è coperto in nessun modo dal DOM di livello 1, ed il livello 2 fornisce solo alcune parziali indicazioni: c'e' una classe oggetto `DOMImplementation` che fornisce accesso ai metodi di creazione `Document`, ma nessun modo di accedere ad un XML lettore/parser/Document con un metodo indipendente di implementazione. Non è stato neanche ben definito come accedere a questi metodi senza un oggetto `Document` preesistente. In Python, ogni implementazione DOM fornisce una funzione `getDOMImplementation()`. Il DOM di livello 3 aggiunge una specifica di caricamento/memorizzazione (Load/Store), che definisce una interfaccia al lettore, ma questa specifica non è ancora disponibile nella libreria standard di Python.

Dal momento che si dispone di un oggetto documento DOM, si può accedere a parti del documento XML attraverso le sue proprietà ed i suoi metodi. Queste proprietà vengono definite nelle specifiche DOM; questa porzione del manuale di riferimento descrive l'interpretazione delle specifiche riportate in Python.

Le specifiche fornite dal W3C definiscono le API DOM per Java, ECMAScript ed OMG IDL. Lo schema Python definito qui viene basato in larga parte sulla versione IDL delle specifiche, ma una stretta conformità non viene richiesta (le implementazioni sono libere comunque di supportare una stretta aderenza all'IDL). Vedere la sezione 13.6.3, "Conformità," per una dettagliata discussione dei requisiti di conformità.

Vedete anche:

Specifiche di Livello 2 del Modello di Oggetto Documento (DOM)
(http://www.w3.org/TR/DOM-Level-2-Core/)
: Le raccomandazioni del W3C su cui sono basate le API DOM di Python.

Specifiche di Livello 1 del Modello di Oggetto Documento (DOM)
(http://www.w3.org/TR/REC-DOM-Level-1/)
: Le raccomandazioni W3C per il DOM supportato da `xml.dom.minidom`.

PyXML
(http://pyxml.sourceforge.net)
: Gli utenti che richiedono un'implementazione completa del DOM dovrebbero usare il package PyXML.

Scripting CORBA con Python
(http://cgi.omg.org/cgi-bin/doc?orbos/99-08-02.pdf)
: Questo specifica le corrispondenze tra OMG IDL e Python.

13.6.1 Contenuto del modulo

Il modulo `xml.dom` contiene le seguenti funzioni:

`registerDOMImplementation``(`*name, factory*`)`
: Registra la funzione *factory* con il nome *name*. La funzione di base deve restituire un oggetto che implementa l'interfaccia `DOMImplementation`. La funzione *factory* può restituire lo stesso oggetto ogni volta, o uno nuovo per ogni chiamata, come meglio appropriato per la specifica implementazione (per esempio se l'implementazione supporta alcune personalizzazioni).

`getDOMImplementation``([`*name*`[`*, features*`]])`
: Restituisce un'implementazione DOM utilizzabile. Il parametro *name* può essere sia un ben conosciuto nome di modulo di una implementazione DOM che `None`. Se non viene impostato a `None`, importa il modulo corrispondente e restituisce un oggetto `DOMImplementation` se l'importazione avviene correttamente. Se nessun nome, *name*, viene indicato, e la variabile d'ambiente PYTHON_DOM è impostata, viene usata questa variabile per trovare l'implementazione.

  Se il nome non viene indicato, questa esamina le implementazioni disponibili per trovarne una con le opzioni richieste impostate. Se nessuna implementazione può essere trovata, solleva un'eccezione `ImportError`. La lista *features* deve essere una sequenza di coppie `(`*feature*`, `*version*`)` che verranno passate al metodo `hasFeature()` sugli oggetti `DOMImplementation` disponibili.

Vengono indicate alcune utili costanti:

`EMPTY_NAMESPACE`

Il valore usato per indicare che nessuno spazio dei nomi è associato con un nodo nel DOM. Questo è tipicamente identificato come `namespaceURI` (NdT: URI dello spazio di nomi) di un nodo, o usato come parametro *namespaceURI* per un metodo di uno spazio dei nomi specifico. Nuovo nella versione 2.2.

`XML_NAMESPACE`
L'URI dello spazio dei nomi associato con il prefisso `xml` riservato, come definito in *Lo spazio dei nomi in XML* (sezione 4). Nuovo nella versione 2.2.

`XMLNS_NAMESPACE`
L'URI dello spazio dei nomi per la dichiarazione dello spazio di nomi, come definito in *Modello di Oggetto di Documento (DOM) di livello 2 – specifiche fondamentali* (sezione 1.1.8). Nuovo nella versione 2.2.

`XHTML_NAMESPACE`
L'URI dello spazio dei nomi XHTML come definito in *XHTML 1.0: Linguaggio estensibile di Marcatura Ipertestuale* (sexione 3.1.1). Nuovo nella versione 2.2.

In aggiunta, `xml.dom` contiene una classe `Node` di base e classi di eccezione DOM. La classe `Node` fornita da questo modulo non implementa nessuno dei metodi o degli attributi definiti dalle specifiche DOM; implementazioni DOM concrete devono fornirle autonomamente. La classe `Node` fornita come parte di questo modulo fornisce anche le costanti utilizzate per gli attributi `nodeType` sull'oggetto concreto `Node`; queste sono presenti all'interno della classe, diversamente che a livello di modulo, per conformarsi alle specifiche DOM.

13.6.2 Oggetti nel DOM

La documentazione definitiva per il DOM consiste nelle specifiche DOM indicate dal W3C.

Si tenga presente che gli attributi DOM possono essere anche manipolati come nodi anziché come semplici stringhe. È abbastanza raro che si debba fare in questo modo, quindi, questo metodo non è ancora documentato.

Interfaccia	Sezione	Scopo
`DOMImplementation`	13.6.2	Interfaccia per le implementazioni sottostanti.
`Node`	13.6.2	Interfaccia di base per la maggior parte degli oggetti in un documento.
`NodeList`	13.6.2	Interfaccia per una sequenza di nodi.
`DocumentType`	13.6.2	Informazioni circa le dichiarazioni necessarie per elaborare un documento.
`Document`	13.6.2	Oggetto che rappresenta l'intero documento.
`Element`	13.6.2	Nodi di elementi nella gerarchia del documento.
`Attr`	13.6.2	Nodi di valore di attributo nei nodi di elemento.
`Comment`	13.6.2	Rappresentazione di commenti nel sorgente del documento.
`Text`	13.6.2	Nodi contenenti contenuti testuali all'interno del documento.
`ProcessingInstruction`	13.6.2	Rappresentazione delle istruzioni per il ciclo di elaborazione.

Un'ulteriore sezione descrive le eccezioni definite per lavorare con il DOM in Python.

Oggetti DOMImplementation

L'interfaccia `DOMImplementation` fornisce un modo per consentire alle applicazioni di determinare la disponibilità di particolari opzioni nel DOM di cui tali applicazioni hanno bisogno. Il DOM di livello 2 aggiunge la particolare facoltà di creare nuovi oggetti `Document` e `DocumentType` usando direttamente `DOMImplementation`.

`hasFeature`(*feature, version*)

Oggetti Node

Tutti i componenti di un documento XML sono sotto classi di `Node`.

`nodeType`
Un intero che rappresenta il tipo di nodo. Vengono definite alcune costanti simboliche per i tipo nell'oggetto `Node`: `ELEMENT_NODE`, `ATTRIBUTE_NODE`, `TEXT_NODE`, `CDATA_SECTION_NODE`,

ENTITY_NODE, PROCESSING_INSTRUCTION_NODE, COMMENT_NODE, DOCUMENT_NODE, DOCUMENT_TYPE_NODE, NOTATION_NODE. Questi sono attributi in sola lettura.

parentNode
Il genitore del nodo corrente, o None per il nodo rappresentante il documento. Il valore è sempre un oggetto Node o None. Per nodi Element, questo sarà un elemento genitore, eccetto per l'elemento root, nel cui caso sarà l'oggetto Document. Per il nodi Attr, questo è sempre None. Questo è un attributo in sola lettura.

attributes
Un NamedNodeMap di oggetti di attributo. Solo gli elementi hanno attualmente valore per questo; altri, per questo attributo, indicano None. Questo è un attributo in sola lettura.

previousSibling
Il nodo che precede immediatamente il corrente, con lo stesso genitore. Per istanza gli elementi con un indicatore di fine tag che arrivano subito prima del proprio, *self*, elemento di inizio tag. Di solito, i documenti XML sono composti da più componenti che semplici elementi, per cui il precedente sibling (NdT: rivale) può essere testo, un commento, o qualche altra cosa. Se questo nodo è il primo figlio del genitore, questo attributo assume il valore None. Questo è un attributo in sola lettura.

nextSibling
Il nodo che immediatamente precede questo, con lo stesso genitore. Vedere anche anche previousSibling. Se questo è l'ultimo dei figli del genitore, questo attributo assume il valore None. Questo è un attributo in sola lettura.

childNodes
Una lista di nodi contenuti all'interno di questo nodo. Questo è un attributo in sola lettura.

firstChild
Il primo figlio all'interno del nodo, se non ce ne sono allora varrà None. Questo è un attributo in sola lettura.

lastChild
L'ultimo figlio all'interno del nodo, se non ce ne sono allora varrà None. Questo è un attributo in sola lettura.

localName
La parte di tagName successiva ai due punti, se ne è presente una, altrimenti l'intero tagName. Il valore è una stringa.

prefix
La parte di tagName che precede i due punti, se ce n'è una, altrimenti una stringa vuota. Il valore è una stringa o None.

namespaceURI
Lo spazio dei nomi associato con al nome dell'elemento. Il valore sarà una stringa o None. Questo è un attributo in sola lettura.

nodeName
Questo ha un significato differente per ogni tipo di nodo; vedere le specifiche DOM per ulteriori dettagli. Si possono sempre ottenere le informazioni volute da questo attributo, ma anche da altre proprietà, come la proprietà tagName per gli elementi o la proprietà name per gli attributi. Per tutti i tipi di nodi, il valore di questo attributo è sia una stringa che None. Questo è un attributo in sola lettura.

nodeValue
Questo assume differente significato per ogni tipo di nodo; per i dettagli vedere le specifiche DOM. La situazione è simile a nodeName. Il valore è una stringa o None.

hasAttributes()
Restituisce vero se il nodo possiede qualche attributo.

hasChildNodes()
Restituisce vero se il nodo possiede quache nodo figlio.

isSameNode(*other*)
Restituisce vero se altri, *other*, si riferiscono allo stesso nodo come questo nodo. Questo è particolarmente

utile per le implementazioni DOM che usano alcune forme di architettura proxy (per cui più di un oggetto può riferirsi allo stesso nodo).

Note: Questo è basato sun una proposta per le API DOM di livello 3 che si trova ancora nello stadio di "bozza in lavorazione", ma questa particolare interfaccia appare incontrovertibile. Eventuali modifiche indicate dal W3C non necessariamente influenzeranno questo metodo nell'interfaccia DOM di Python (compreso il fatto che ogni nuova API W3C verrà supportata).

`appendChild`(*newChild*)
Aggiunge un nuovo nodo figlio per questo nodo alla fine della lista dei figli, restituendo *newChild*.

`insertBefore`(*newChild, refChild*)
Inserisce un nuovo nodo figlio prima di un figlio esistente. È sottinteso che *refChild* deve essere un figlio di questo nodo; altrimenti viene sollevata l'eccezione `ValueError`. Viene restituito *newChild*.

`removeChild`(*oldChild*)
Rimuove un nodo figlio. *oldChild* deve essere un figlio di questo nodo; altrimenti, viene sollevata l'eccezione `ValueError`. *oldChild* viene restituito se si ha successo. Se *oldChild* non verrà usato successivamente, dovrebbe essere chiamato il suo metodo `unlink()`.

`replaceChild`(*newChild, oldChild*)
Sostituisce un nodo esistente con un nuovo nodo. Ovviamente *oldChild* è un figlio di questo nodo; altrimenti viene sollevata l'eccezione `ValueError`.

`normalize`()
Unisce nodi di testo adiacenti in modo che tutti gli elementi di testo vengano uniti in una singola istanza `Text`. Questo semplifica l'elaborazione del testo in un albero DOM per molte applicazioni. Nuovo nella versione 2.1.

`cloneNode`(*deep*)
Clona questo nodo. Impostare *deep* significa la clonazione anche di tutti i nodi figli. Questo restituirà il clone.

Oggetti NodeList

Un `NodeList` rappresenta una sequenza di nodi. Nelle raccomandazioni di base del DOM, questi oggetti vengono usati in due modi: gli oggetti `Element` ne forniscono una come lista di nodi figli, ed i metodi di `Node` (`getElementsByTagName()` e `getElementsByTagNameNS()`) restituiscono oggetti con questa interfaccia per rappresentare i risultati dell'interrogazione.

Le raccomandazioni del DOM di livello 2 definiscono un metodo ed un attributo per questi oggetti:

`item`(*i*)
Restituisce l'ennesimo elemento *i* della sequenza, se ce n'è uno, altrimenti `None`. L'indice *i* non può essere minore di zero o più grande o uguale alla lunghezza della sequenza.

`length`
Il numero di nodi nella sequenza.

In aggiunta, l'interfaccia DOM di Python richiede che alcuni ulteriori elementi vengano forniti per consentire agli oggetti `NodeList` di essere usati come sequenze Python. Tutte le implementazioni `NodeList` devono includere il supporto per `__len__()` e `__getitem__()`; questo consente iterazioni sulla `NodeList`, durante istruzioni `for` ed il corretto supporto per la funzione built-in `len()`.

Se un'implementazione DOM supporta modifiche del documento, l'implementazione `NodeList` deve anche supportare i metodi `__setitem__()` e `__delitem__()`.

Oggetti DocumentType

Informazioni circa le notazioni e le entità dichiarate da un documento (inclusi gli elementi esterni se un analizzatore ne fa uso e fornisce le corrette informazioni) sono disponibili nell'oggetto `DocumentType`. Il `DocumentType` per un documento è disponibile per l'attributo `doctype` dell'oggetto `Document`; se non

c'è la dichiarazione DOCTYPE per il documento, l'attributo doctype del documento viene impostato a None invece che con un'istanza di questa interfaccia.

DocumentType è una specializzazione di Node, ed aggiunge i seguenti attributi:

publicId
L'identificare pubblico per il sotto insieme esterno della definizione di tipo documento. Questo è una stringa o None.

systemId
L'identificare di sistema per il sotto insieme esterno della definizione del tipo documento. Questo è una URI in forma di stringa, o None.

internalSubset
Una stringa che indica il completo sotto insieme interno del documento. Questo non include le parentesi che includono il sotto insieme. Se il documento non possiede un suo sotto insieme, assumerà come valore None.

name
Il nome dell'elemento root, principale, come indicato nella dichiarazione DOCTYPE, se presente.

entities
Questo è un NamedNodeMap, indicato nella definizione delle entità esterne. Per i nomi di entità definite più di una volta, viene indicata solamente la prima definizione (le altre vengono ignorate, come indicato nelle raccomandazioni XML). Questo deve essere None se le informazioni non vengono fornite dal parser, o se non vengono definite entità.

notations
Questo è un NamedNodeMap che indica la definizione di notazione. Per i nomi delle notazioni definite più di una volta, viene fornita solamente la prima definizione (le altre vengono ignorate, come richiesto dalle raccomandazioni XML). Questo deve essere None se le informazioni non sono fornite dal perser, o se nessuna notazione viene definita.

Oggetti Document

Un Document rappresenta un intero documento XML, inclusi i suoi elementi costitutivi, gli attributi, le istruzioni di elaborazione, commento, etc.. Ricordare che eredita le proprietà da Node.

documentElement
Il solo e unico elemento principale del documento.

createElement(*tagName*)
Crea e restituisce un nuovo elemento nodo. L'elemento non viene inserito nel documento quando viene creato. Si deve inserirlo esplicitamente con uno degli altri metodi disponibili come insertBefore() o appendChild().

createElementNS(*namespaceURI, tagName*)
Crea e restituisce un nuovo elemento con uno spazio dei nomi. *tagName* può avere un prefisso. L'elemento non viene inserito nel documento quando viene creato. Si deve esplicitamente inserirlo usando uno dei metodi disponibili come insertBefore() o appendChild().

createTextNode(*data*)
Crea e restituisce un nodo di testo contenente i dati passati come parametro. Come per gli altri metodi di creazione, questo nodo non viene inserito automaticamente nell'albero.

createComment(*data*)
Crea e restituisce un nodo di commento contenente i dati passati come parametro. Come per gli altri metodi di creazione, anche questo non viene inserito automaticamente nell'albero.

createProcessingInstruction(*target, data*)
Crea e restituisce un nodo di istruzioni di elaborazione contenente *target* e *data* passati come parametri. Come con gli altri metodi di creazione, questo non viene automaticamente inserito all'interno dell'albero.

createAttribute(*name*)
Crea e restituisce un nodo di attributi. Questo metodo non associa il nodo di attributi con nessun partico-

lare elemento. Si deve usare `setAttributeNode()` sull'oggetto `Element` appropriato per utilizzare l'istanza di attributo appena creata.

`createAttributeNS`(*namespaceURI, qualifiedName*)
Crea e restituisce un nodo di attributi con uno spazio di nomi. *tagName* può avere un prefisso. Questo metodo non associa il nodo di attributi con un particolare elemento. Si deve usare `setAttributeNode()` sull'oggetto `Element` appropriato per utilizzare l'istanza di attributo appena creata.

`getElementsByTagName`(*tagName*)
Cerca in tutti i discendenti (figli diretti, figli di figli etc. etc.) con un particolare nome di tipo di elemento.

`getElementsByTagNameNS`(*namespaceURI, localName*)
Cerca in tutti i discendenti (figli diretti, figli di figli, etc. etc.) con un particolare spazio dei nomi URI e nome locale. Il nome locale, o localname, è la parte dello spazio dei nomi dopo il prefisso.

Oggetti Element

`Element` è una sotto classe di `Node`, che eredita tutti gli attributi di questa classe.

`tagName`
Il nome del tipo di elemento. In uno spazio di nomi usato dal documento ci possono essere dei due punti all'interno. Il valore è una stringa.

`getElementsByTagName`(*tagName*)
Come l'equivalente metodo nella classe `Document`.

`getElementsByTagNameNS`(*tagName*)
Come l'equivalente metodo nella classe `Document`.

`getAttribute`(*attname*)
Restituisce un valore di attributo sotto forma di stringa.

`getAttributeNode`(*attrname*)
Restituisce il nodo `Attr` per l'attributo indicato come *attrname*.

`getAttributeNS`(*namespaceURI, localName*)
Restituisce un valore di attributo come una stringa, indicando un *namespaceURI* ed il *localName*.

`getAttributeNodeNS`(*namespaceURI, localName*)
Restituisce un valore di attributo come un nodo, indicando un *namespaceURI* ed il *localName*.

`removeAttribute`(*attname*)
Rimuove un attributo per nome. Nessuna eccezione viene sollevata se non ci sono attributi corrispondenti.

`removeAttributeNode`(*oldAttr*)
Rimuove e restituisce *oldAttr* dalla lista degli attributi, se presente. Se *oldAttr* non è presente, viene sollevata l'eccezione `NotFoundErr`.

`removeAttributeNS`(*namespaceURI, localName*)
Rimuove un attributo per nome. Notare che viene usato *localName*, non un *qname*. Nessuna eccezione viene sollevata se non ci sono attributi corrispondenti.

`setAttribute`(*attname, value*)
Imposta un valore di attributo da una stringa.

`setAttributeNode`(*newAttr*)
Aggiunge un nuovo nodo di attributo all'elemento, rimpiazzando un attributo esistente quando necessario se il nome, `name`, dell'attributo corrisponde. Se avviene una sostituzione, verrà restituito il vecchio nodo dell'attributo. Se *newAttr* è già in uso, verrà sollevata l'eccezione `InuseAttributeErr`.

`setAttributeNodeNS`(*newAttr*)
Aggiunge un nuovo nodo di attributo all'elemento, se necessario sostituendo un attributo esistente se `namespaceURI` e `localName` corrispondono. Se avviene una sostituzione, verrà restituito il vecchio nodo di attributo. Se *newAttr* è già in uso, verrà sollevata l'eccezione `InuseAttributeErr`.

`setAttributeNS`(*namespaceURI, qname, value*)

Imposta un valore di attributo da una stringa, indicando un *namespaceURI* ed un *qname*. Notare che *qname* è il nome completo dell'attributo. Questo è differente dalle precedenti indicazioni.

Oggetti Attr

`Attr` eredita da `Node`, quindi eredita tutti i suoi attributi.

`name`
Il nome dell'attributo. Nello spazio dei nomi in uso nel documento vi possono essere dei due punti al suo interno.

`localName`
La parte del nome seguito dal punto, se ce ne è uno, oppure l'intero nome. Questo è un attributo in sola lettura.

`prefix`
La parte del nome che precede il punto, se ce n'è uno, altrimento la stringa vuota.

Oggetti NamedNodeMap

`NamedNodeMap` *non* eredita nulla da `Node`.

`length`
La lunghezza della lista degli attributi.

`item`(*index*)
Restituisce un attributo con un particolare indice *index*. L'ordine con cui si ottiene l'attributo è arbitrario, ma resterà consistente nella vita del DOM. Ogni elemento è un nodo di attributo. Prende il suo valore dal valore `value` attribuito.

Ci sono anche metodi sperimentali che forniscono a questa classe maggiori elementi di individuazione. Si possono utilizzare questi metodi o si può utilizzare la famiglia standard `getAttribute*()` del metodo sull'oggetto `Element`.

Oggetti Comment

`Comment` rappresenta un commento all'interno del documento XML. È una sotto classe di `Node`, ma non può avere nodi figli.

`data`
Il contenuto del commento come stringa. L'attributo contiene tutti i caratteri tra la testa `<!--` e la chiusura `-->`, ma non li include.

Oggetti Text e CDATASection

L'interfaccia `Text` rappresenta il testo all'interno del documento XML. Se il parser e l'implementazione DOM supportano le estensioni XML DOM, porzioni del testo incluso nelle sezioni marcate CDATA vengono memorizzate in oggetti `CDATASection`. Queste due interfacce sono identiche, ma forniscono valori differenti per gli attributi `nodeType`.

Queste interfacce estendono l'interfaccia `Node`. Non possono avere nodi figli.

`data`
Il contenuto del nodo di testo sotto forma di stringa.

Note: L'uso del nodo `CDATASection` non indica che il nodo rappresenti una completa sezione marcata CDATA. Una singola sezione CDATA può essere rappresentata da più di un nodo nell'albero del documento. Non c'è modo di determinare se due adiacenti nodi `CDATASection` rappresentano differenti sezioni marcate CDATA.

Oggetti ProcessingInstruction

Rappresenta un'istruzione di elaborazione nel documento XML; questa eredita dall'interfaccia `Node` e non può avere nodi figli.

`target`
Il contenuto dell'istruzione di elaborazione fino al primo carattere vuoto. Questo è un attributo in sola lettura.

`data`
Il contenuto di un'istruzione di elaborazione segue il primo carattere vuoto.

Eccezioni

Nuovo nella versione 2.1.

Le raccomandazioni DOM di livello 2 definiscono una singola eccezione, DOMException ed un certo numero di costanti che consentono alle applicazioni di determinare quale errore è intervenuto. Le istanze `DOMException` comunicano un attributo `code` che fornisce il valore appropriato per la specifica eccezione.

L'interfaccia DOM del Python fornisce le costanti, ma estende anche l'insieme delle eccezioni così da avere ogni specifica eccezione per ognuno dei codici di eccezione definiti dal DOM. L'implementazione deve sollevare l'eccezione di specifica appropriata, ognuna delle quali trasporta il valore appropriato per l'attributo `code`.

exception `DOMException`
Classe di eccezione di base usata per tutte le eccezioni DOM specificate. Questa classe di eccezioni non può essere istanziata direttamente.

exception `DomstringSizeErr`
Viene sollevata quando uno specifico intervallo di testo non entra in una stringa. Non si sa se sia usato nelle implementazioni DOM di Python, ma potrebbe essere ricevuto da implementazioni DOM non scritte in Python.

exception `HierarchyRequestErr`
Viene sollevata quando viene fatto un tentativo di inserire un nodo laddove il tipo di nodo non è consentito.

exception `IndexSizeErr`
Viene sollevata quando un indice o un parametro di dimensione verso un metodo è negativo o supera i valori ammessi.

exception `InuseAttributeErr`
Viene sollevata quando viene fatto un tentativo di inserire un nodo `Attr` che è già presente in un'altra parte del documento.

exception `InvalidAccessErr`
Viene sollevata se un parametro o una operazione non è supportata dall'oggetto sottostante.

exception `InvalidCharacterErr`
Questa eccezione viene sollevata quando un parametro di stringa che contiene caratteri che non sono permessi in quel particolare contesto, viene usata laddove sono in uso le raccomandazioni XML 1.0. Per esempio, tentare di creare un nodo `Element` con uno spazio nel nome del tipo di elemento causa che questo tipo di errore venga sollevato.

exception `InvalidModificationErr`
Viene sollevata quando viene effettuato un tentativo di modificare il tipo di un nodo.

exception `InvalidStateErr`
Viene sollevata quando viene fatto un tentativo di usare un oggetto che non è più usabile.

exception `NamespaceErr`
Se viene fatto un tentativo di cambiare un oggetto in un nodo che non è permesso dalle raccomandazioni per *lo spazio dei nomi XML*, viene sollevata questa eccezione.

exception `NotFoundErr`
Eccezione sollevata quando un nodo non esiste nel contesto a cui si riferisce. Per esempio,

`NamedNodeMap.removeNamedItem()` solleverà questa eccezione se il nodo passato non esiste nella mappa.

exception `NotSupportedErr`
Viene sollevata quando l'implementazione non supporta l'oggetto del tipo richiesto o una particolare operazione.

exception `NoDataAllowedErr`
Questa eccezione viene sollevata se i dati sono specifici per un nodo che non supporta i dati stessi.

exception `NoModificationAllowedErr`
Viene sollevata nel tentativo di modificare un oggetto dove le modifiche non sono consentite (come per i nodi in sola lettura).

exception `SyntaxErr`
Viene sollevata quando viene specificata una stringa non valida o illegale.

exception `WrongDocumentErr`
Viene sollevata quando un nodo è inserito un un documento diverso da quello in uso, e l'implementazione non supporta la migrazione del nodo da un documento ad un altro.

I codici di eccezione definiti nella mappa delle raccomandazioni DOM sono di seguito descritti con le eccezioni collegate:

Costante	Eccezione
`DOMSTRING_SIZE_ERR`	`DomstringSizeErr`
`HIERARCHY_REQUEST_ERR`	`HierarchyRequestErr`
`INDEX_SIZE_ERR`	`IndexSizeErr`
`INUSE_ATTRIBUTE_ERR`	`InuseAttributeErr`
`INVALID_ACCESS_ERR`	`InvalidAccessErr`
`INVALID_CHARACTER_ERR`	`InvalidCharacterErr`
`INVALID_MODIFICATION_ERR`	`InvalidModificationErr`
`INVALID_STATE_ERR`	`InvalidStateErr`
`NAMESPACE_ERR`	`NamespaceErr`
`NOT_FOUND_ERR`	`NotFoundErr`
`NOT_SUPPORTED_ERR`	`NotSupportedErr`
`NO_DATA_ALLOWED_ERR`	`NoDataAllowedErr`
`NO_MODIFICATION_ALLOWED_ERR`	`NoModificationAllowedErr`
`SYNTAX_ERR`	`SyntaxErr`
`WRONG_DOCUMENT_ERR`	`WrongDocumentErr`

13.6.3 Conformità

La sezione descrive i requisiti di conformità e le relazioni tra le API DOM di Python, le raccomandazioni DOM del W3C, e la mappatura OMG IDL per Python.

Mappatura di Type

I tipi IDL primitivi usati nella specifica DOM sono mappate in Python in conformità alla seguente tavola:

Tipo IDL	Tipo Python
`boolean`	`IntegerType` (con valore `0` o `1`)
`int`	`IntegerType`
`long int`	`IntegerType`
`unsigned int`	`IntegerType`

Inoltre, il `DOMString` definito nelle raccomandazioni viene mappato alla stringa Python o alla stringa Unicode. Le applicazioni devono essere in grado di gestire Unicode laddove viene restituita una stringa dal DOM.

Il valore IDL `null` viene mappato a `None`, che può essere accettato o fornito dall'implementazione laddove `null` è permesso dalle API.

Metodi accessori

Le corrispondenze tra IDL OMG e Python definiscono le funzioni accessorie per le dichiarazioni di attributi IDL, `attribute`, in un modo quasi identico a come lo fa java. Mappando le dichiarazioni IDL si ottengono tre funzioni accessorie:

```
readonly attribute string someValue;
         attribute string anotherValue;
```

Un metodo "get" per `someValue` (`_get_someValue()`) ed un metodo "set" per `_set_anotherValue()`). L'associazione, in particolare, non richiede che gli attributi IDL siano accessibili come normali attributi Python: *non* è necessario che *object*`.someValue` lavori, e può sollevare l'eccezione `AttributeError`.

Le API DOM di Python, comunque, *richiedono* che il normale accesso all'attributo funzioni. Questo significa che il tipico surrogato generato dai compilatori IDL di Python non lavori correttamente, e gli oggetti wrapper possono essere necessari nel client se l'accesso all'oggetto DOM avviene attraverso CORBA. Se questo richiede alcune considerazioni sui client DOM CORBA, il programmatore con esperienza nell'uso di DOM e CORBA in Python non considera questo un problema. Gli attributi che vengono dichiarati in sola lettura, `readonly`, possono non limitare l'accesso in scrittura in tutte le implementazioni DOM.

In aggiunta, la funzione di accesso non viene richiesta. Se fornita, deve prendere la form definita dall'associazione IDL di Python, ma questi metodi sono considerati non necessari finché l'attributo è accessibile direttamente dal Python. Gli accessori "Set" non devono mai essere forniti di attributi di sola lettura, `readonly`.

13.7 `xml.dom.minidom` — Implementazione minimale di DOM

Nuovo nella versione 2.0.

`xml.dom.minidom` è un'implementazione minimale dell'interfaccia del modello di oggetto documento. È chiaro che si tratta di una implementazione più semplice del DOM completo ed è anche significativamente più piccola.

Le applicazioni DOM, tipicamente iniziano analizzando del codice XML in un DOM. Con `xml.dom.minidom`, questo avviene attraverso la funzione di analisi:

```
from xml.dom.minidom import parse, parseString

dom1 = parse('c:\\temp\\mydata.xml') # analizza un file XML per nome

datasource = open('c:\\temp\\mydata.xml')
dom2 = parse(datasource)   # analizza un file aperto

dom3 = parseString('<myxml>Alcuni dati<empty/> ancora altri dati</myxml>')
```

La funzione `parse()` può prendere sia un nome di file che un oggetto file aperto.

`parse`(*filename_or_file*, *parser*)
: Restituisce un `Document` dall'input indicato. *filename_or_file* può essere sia un nome file o un oggetto assimilabile all'oggetto file. *parser*, se indicato, deve essere un oggetto analizzatore SAX2. Questa funzione cambia il gestore del documento del parser e attiva il supporto allo spazio dei nomi; altre configurazioni del parser (come l'impostare un risolutore di entità) devono essere impostate prima di inizializzarlo.

Se si ha a disposizione del codice XML in una stringa, si può utilizzare la funzione `parseString()`:

`parseString`(*string*[, *parser*])
: Restituisce un `Document` che rappresenta la stringa. Questo metodo crea un oggetto `StringIO` per la stringa e passa il tutto a `parse()`.

Entrambe le funzioni restituiscono un oggetto `Document` che rappresenta il contenuto del documento.

Quello che le funzioni `parse()` e `parseString()` fanno è connettere un analizzatore XML con un "costruttore DOM" che può accettare eventi di analisi da ogni parser SAX e li converte in un albero DOM. Il nome delle funzioni è spesso causa di confusione, ma sono facili da capire mentre si studia l'interfaccia. L'analisi del documento viene completata prima che queste funzioni terminino; semplicemente queste funzioni non forniscono un'implementazione di un parser loro stesse.

Si può anche creare un `Document` chiamando un metodo su di un oggetto di "implementazione DOM". Si può ottenere questo oggetto sia chiamando la funzione `getDOMImplementation()` nel package `xml.dom` o il modulo `xml.dom.minidom`. Usando l'implementazione del modulo `xml.dom.minidom` si otterrà sempre la restituzione di una istanza `Document` dall'implementazione minidom, quando la versione di `xml.dom` può fornire una implementazione alternativa (questo è facile se si ha il package PyXML installato). Quando si ha un oggetto `Document`, gli si possono aggiungere nodi figli per popolare il DOM:

```
from xml.dom.minidom import getDOMImplementation

impl = getDOMImplementation()

newdoc = impl.createDocument(None, "alcuni_tag", None)
top_element = newdoc.documentElement
text = newdoc.createTextNode('Alcuni contenuti di testo.')
top_element.appendChild(text)
```

Una volta che si possiede un oggetto di documento DOM, si può accedere a parti del documento XML attraverso le sue proprietà ed i suoi metodi. Queste proprietà sono definite nelle specifiche DOM. La proprietà principale dell'oggetto documento è la `documentElement`. Fornisce il principale elemento nel documento XML. Il primo che contiene tutti gli altri. Qui un esempio di programma:

```
dom3 = parseString("<myxml>Some data</myxml>")
assert dom3.documentElement.tagName == "myxml"
```

Quando si finisce con un DOM, lo si deve cancellare. Questo è necessario perché alcune versioni di Python non supportano il meccanismo della garbage collection di oggetti che fanno riferimento ad altri all'interno del ciclo. Finché questa restrizione non verrà rimossa da tutte le versioni di Python, è cosa buona e giusta scrivere il proprio codice come se il ciclo non dovesse essere ripulito.

Il modo per ripulire un DOM è di chiamare il suo metodo `unlink()`:

```
dom1.unlink()
dom2.unlink()
dom3.unlink()
```

`unlink()` è una estensione specifica di `xml.dom.minidom` alle API DOM. Prima di chiamare `unlink()` su di un nodo, il nodo ed i suoi discendenti devono essere sostanzialmente inutili.

Vedete anche:

Modello di Oggetto Documento (DOM) specifiche di livello 1
(http://www.w3.org/TR/REC-DOM-Level-1/)
: Raccomandazione W3C per il DOM supportata da `xml.dom.minidom`.

13.7.1 Oggetti DOM

La definizione delle API DOM per Python viene fornita in parte dalla documentazione del modulo `xml.dom`. Questa sezione elenca le differenze tra API e `xml.dom.minidom`.

`unlink``()`
: Termina i riferimenti interni al DOM in modo che questi possa essere passato alla garbage collection anche in versioni di Python che non abbiano il meccanismo della GC. Solo quando la GC è disponibile, questa funzione consente il recupero di discreti quantitativi di memoria in modo molto rapido, pertanto chiamare

questo metodo sugli oggetti DOM non appena essi non sono più necessari è una buona pratica. Questo metodo necessita di essere chiamato su un oggetto `Document`, ma potrebbe essere chiamato anche sui nodi figli per scaricare i figli di quel nodo.

`writexml`(*writer*)
Scrive XML sull'oggetto *writer*. *writer* dovrebbe avere un metodo `write()` che corrisponde a quello dell'interfaccia dell'oggetto file.

Modificato nella versione 2.1: Per ottenere un output più grazioso, sono state aggiunte nuove parole chiave, *indent*, *addindent* e *newl*.

Modificato nella versione 2.3: Per il nodo `Document`, un ulteriore parola chiave, *encoding*, può essere usata per specificare il campo di codifica dell'intestazione XML.

`toxml`([*encoding*])
Restituisce l'XML che il DOM rappresenta sotto forma di stringa.

Senza nessun argomento, l'intestazione XML non specifica alcuna codifica ed il risultato è una stringa Unicode quando la codifica predefinita non riesce a rappresentare tutti i caratteri del documento. Codificare questa stringa con una codifica diversa da UTF-8 è spesso non corretto, in quanto UTF-8 è la codifica predefinita di XML.

Con un argomento esplicito di codifica, *encoding*, il risultato è una stringa di byte nella specificata codifica. Si raccomanda che questo argomento sia sempre specificato. Per evitare eccezioni UnicodeError nel caso di dati testuali non rappresentabili, l'argomento di codifica dovrebbe essere specificato come utf-8.

Modificato nella versione 2.3: è stato introdotto l'argomento di codifica *encoding*.

`toprettyxml`([*indent*[, *newl*]])
Restituisce una versione da stampare più graziosa del documento. *indent* specifica la stringa di indentazione e il valore predefinito come un tabulatore; *newl* specifica la stringa emessa alla fine di ogni riga ed il valore predefinito è
n.

Nuovo nella versione 2.1.

Modificato nella versione 2.3: per codificare gli argomenti; vedere `toxml`.

I seguenti metodi standard DOM sono tenuti in speciale considerazione all'interno di `xml.dom.minidom`:

`cloneNode`(*deep*)
Nonostante questo metodo fosse presente nella versione di `xml.dom.minidom` in Python 2.0, in realtà non era funzionante. Questo è stato corretto nelle versioni successive.

13.7.2 Esempi DOM

Questo programma di esempio è abbastanza realistico di come potrebbe essere un semplice programma. In questo particolare caso, non otteniamo particolare vantaggio dalla flessibilità del DOM.

```
import xml.dom.minidom

document = """\
<slideshow>
<title>Demo slideshow</title>
<slide><title>Titolo della slide</title>
<point>Questa è una demo</point>
<point>Di un programma per elaborare slide</point>
</slide>

<slide><title>Altra demo slide</title>
<point>E' importante</point>
<point>Avere più di</point>
<point>una slide</point>
</slide>
</slideshow>
"""
```

```
dom = xml.dom.minidom.parseString(document)

def getText(nodelist):
    rc = ""
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE:
            rc = rc + node.data
    return rc

def handleSlideshow(slideshow):
    print "<html>"
    handleSlideshowTitle(slideshow.getElementsByTagName("title")[0])
    slides = slideshow.getElementsByTagName("slide")
    handleToc(slides)
    handleSlides(slides)
    print "</html>"

def handleSlides(slides):
    for slide in slides:
       handleSlide(slide)

def handleSlide(slide):
    handleSlideTitle(slide.getElementsByTagName("title")[0])
    handlePoints(slide.getElementsByTagName("point"))

def handleSlideshowTitle(title):
    print "<title>%s</title>" % getText(title.childNodes)

def handleSlideTitle(title):
    print "<h2>%s</h2>" % getText(title.childNodes)

def handlePoints(points):
    print "<ul>"
    for point in points:
        handlePoint(point)
    print "</ul>"

def handlePoint(point):
    print "<li>%s</li>" % getText(point.childNodes)

def handleToc(slides):
    for slide in slides:
        title = slide.getElementsByTagName("title")[0]
        print "<p>%s</p>" % getText(title.childNodes)

handleSlideshow(dom)
```

13.7.3 Il minidom e lo standard DOM

Il modulo `xml.dom.minidom` è essenzialmente un'implementazione compatibile con il DOM 1.0 con alcune estensioni DOM 2 (in primo luogo le estensioni sullo spazio dei nomi).

L'uso dell'interfaccia DOM all'interno di Python è diretta. Si applicano le seguenti regole:

- Le interfacce sono accessibili attraverso istanze di oggetti. Le applicazioni non devono istanziare le classi; devono usare le funzioni di creazione disponibili nell'oggetto `Document`. Le interfacce derivate supportano tutte le operazioni (e gli attributi) dell'interfaccia di base, più alcune nuove operazioni.
- Le operazioni vengono usate come metodi. Finché il DOM usa solo parametri `in` (NdT: in ingresso), gli argomenti vengono passati nel normale ordine (da sinistra a destra). Non ci sono argomenti facoltativi. Operazioni nulle (NdT: `void`) restituiscono `None`.

- Gli attributi IDL si mappano con gli attributi di istanza. Per compatibilità con lo schema di mappatura del linguaggio OMG IDL per Python, un attributo `foo` diviene accessibile solo attraverso il metodo accessorio `_get_foo()` e `_set_foo()`. Gli attributi in sola lettura non devono essere cambiati; questo non è permesso durante l'esecuzione.
- Il tipo `intero corto`, `intero senza segno`, `long senza segno` e `booleano`, mappano tutti ad oggetti interi di Python.
- Il tipo `DOMString` viene mappato alle stringhe Python. `xml.dom.minidom` supporta sia byte che stringhe Unicode, ma normalmente produce stringhe Unicode. I valori del tipo `DOMString` possono avere come valore anche `None`, dove è permesso che IDL abbia valore `null` dalle specifiche DOM del W3C.
- dichiarazioni di costanti `const` corrispondono a variabili nei loro rispettivi spazi (per esempio `xml.dom.minidom.Node.PROCESSING_INSTRUCTION_NODE`); essi non devono essere cambiati.
- Le eccezioni `DOMException` non vengono attualmente supportate in `xml.dom.minidom`. Al loro posto, `xml.dom.minidom` utilizza le eccezioni standard di Python `TypeError` e `AttributeError`.
- Gli oggetti `NodeList` vengono implementati utilizzando il tipo lista built-in di Python. A partire dalla versione 2.3 di Python, questi oggetti forniscono l'interfaccia definita nella specifica DOM, ma con le versioni più vecchie di Python non supportano le API ufficiali. Sono, comunque, molto più "Pythoniche" che l'interfaccia definita nelle raccomandazioni W3C.

Le seguenti interfacce non possiedono un'implementazione in `xml.dom.minidom`:

- `DOMTimeStamp`
- `DocumentType` (aggiunto in Python 2.1)
- `DOMImplementation` (aggiunto in Python 2.1)
- `CharacterData`
- `CDATASection`
- `Notation`
- `Entity`
- `EntityReference`
- `DocumentFragment`

La maggior parte di queste riflettono le informazioni nel documento XML che non sono generalmente utili alla maggior parte degli utenti di DOM.

13.8 `xml.dom.pulldom` — Supporto per la costruzione di alberi DOM parziali

Nuovo nella versione 2.0.

`xml.dom.pulldom` permette la costruzione solo di porzioni selezionate di una rappresentazione DOM di un documento a partire da eventi SAX.

class `PullDOM`([*documentFactory*])
`xml.sax.handler.ContentHandler` implementazione che ...

class `DOMEventStream`(*stream, parser, bufsize*)
...

class `SAX2DOM`([*documentFactory*])
`xml.sax.handler.ContentHandler` implementazione che ...

parse(*stream_or_string*[, *parser*[, *bufsize*]])
...

parseString(*string*[, *parser*])
...

default_bufsize
Il valore predefinito per il parametro *bufsize* per `parse()`. Modificato nella versione 2.1: Il valore di questa variabile può essere modificato prima di chiamare `parse()` ed il nuovo valore sarà subito operativo..

13.8.1 Oggetti DOMEventStream

getEvent()
...

expandNode(*node*)
...

reset()
...

13.9 `xml.sax` — Supporto per parser SAX2

Nuovo nella versione 2.0.

Il package `xml.sax` fornisce un numero di moduli che implementano la API di base per l'interfaccia XML (SAX) di Python. Il package stesso fornisce le eccezioni SAX ed utili funzioni che saranno le più usate dagli utenti delle API SAX.

Le funzioni utili sono:

make_parser([*parser_list*])
Crea e restituisce un oggetto SAX `XMLReader`. Il primo parser trovato verrà usato. Se *parser_list* viene fornito, dovrà essere composto da una sequenza di stringhe che chiameranno moduli che hanno una funzione chiamata `create_parser()`. I moduli elencati in *parser_list* verranno usati prima dei moduli predefiniti dei parser.

parse(*filename_or_stream, handler*[, *error_handler*])
Crea un parser SAX e lo utilizza per analizzare un documento. Il documento, passato come *filename_or_stream*, può essere un file con nome o un oggetto file. Il parametro *handler* deve necessariamente essere un'istanza SAX `ContentHandler`. Se viene passato *error_handler*, deve essere una istanza SAX `ErrorHandler`; se omesso, verrà sollevata un'eccezione `SAXParseException` su tutti gli errori. Questo non restituirà valori;

`SAXParseException` sollevata per tutti gli errori. Non vi è restituzione di alcun valore; tutte le operazioni devono essere fatte attraverso il gestore, *handler* passato.

parseString(*string, handler*[, *error_handler*])
Simile a `parse()`, ma effettua l'analisi da un buffer stringa, *string*, passato come parametro.

Una tipica applicazione SAX usa tre tipi di oggetti: lettore, gestore e sorgente in ingresso (reader, handler, input source). Il "lettore" o "READER" in questo contesto non è altro che un altro sinonimo per un parser, per esempio alcuni pezzi di codice che leggono i byte o i caratteri dalla sorgente in ingresso e producono una sequenza di eventi. Gli eventi vengono poi distribuiti agli oggetti gestori, per esempio il reader invoca un metodo sul gestore. Un'applicazione SAX deve prima ottenere un oggetto reader, creare o aprire una sorgente di ingresso, creare i gestori e connettere questi oggetti tutti assieme. Come passaggio finale della preparazione, il reader viene chiamato per analizzare l'input. Durante l'analisi, i metodi sull'oggetto gestore vengono chiamati basandosi su una struttura e sintassi di eventi passati dal sorgente in ingresso.

Per questi oggetti, è rilevante solo l'interfaccia; essi normalmente non vengono istanziati dalle applicazioni. Fino a che Python non ha una esplicita conoscenza dell'interfaccia, questi vengono formalmente introdotti come

classi, ma le applicazioni possono usare implementazioni che non ereditano dalle classi indicate. Le interfacce `InputSource`, `Locator`, `Attributes`, `AttributesNS` e `XMLReader` vengono definite nel modulo `xml.sax.xmlreader`. Le interfacce del gestore vengono definite in `xml.sax.handler`. Per convenienza, `InputSource` (che viene spesso istanziata direttamente) e la classe handler sono anche disponibili da `xml.sax`. Queste interfacce vengono descritte di seguito.

In aggiunta a queste classi, `xml.sax` fornisce le seguenti classi di eccezione.

eccezione `SAXException`(*msg*[, *exception*])
: Incapsula un errore o un avviso XML. Questa classe può contenere errori di base o informazioni di avviso sia del parser XML che dell'applicazione: può essere derivata per supportare ulteriori funzionalità o per aggiungere la localizzazione. Notare che, diversamente dai gestori definiti nell'interfaccia `ErrorHandler`, che ricevono le istanze di questa eccezione, non viene richiesto che venga attualmente sollevata una eccezione — è anche utile come contenitore per informazioni.

  Quando istanziata, *msg* deve essere una descrizione dell'errore chiaro e leggibile. Il parametro facoltativo *exception*, se indicato, deve essere o `None` o un'eccezione che viene rilevata dal codice in analisi e viene passata come un'informazione.

  Questa è la classe di base per le altre classi di eccezione SAX.

eccezione `SAXParseException`(*msg*, *exception*, *locator*)
: Classi derivate da `SAXException`, sollevate durante l'analisi degli errori. Istanze di questa classe vengono passate ai metodi dell'interfaccia SAX `ErrorHandler` per fornire informazioni circa l'errore dell'analisi. Questa classe supporta l'interfaccia SAX `Locator` tanto quanto l'interfaccia `SAXException`.

eccezione `SAXNotRecognizedException`(*msg*[, *exception*])
: Classi derivate da `SAXException`, sollevate quando un SAX `XMLReader` viene confrontato con una caratteristica non riconosciuta o una proprietà. Le applicazioni SAX e le estensioni possono usare questa classe per scopi simili.

eccezione `SAXNotSupportedException`(*msg*[, *exception*])
: Classi derivate di `SAXException`, sollevate quando ad un SAX `XMLReader` viene richiesto di abilitare una caratteristica che non è supportata, o impostare una proprietà con un valore che l'implementazione non supporta. Le applicazioni SAX e le estensioni possono usare questa classe per scopi simili.

Vedete anche:

SAX: The Simple API for XML
(http://www.saxproject.org/)
: Questa piattaforma è il punto focale per la definizione delle API SAX. Fornisce un'implementazione Java e documentazione in linea. Sono anche disponibili collegamenti ad implementazioni e informazioni storiche.

Modulo `xml.sax.handler` (sezione 13.10):
: Definizione dell'interfaccia per gli oggetti forniti dalle implementazioni.

Modulo `xml.sax.saxutils` (sezione 13.11):
: Funzioni utili da usarsi nelle applicazioni SAX.

Modulo `xml.sax.xmlreader` (sezione 13.12):
: Definizione dell'interfaccia per oggetti forniti da parser.

13.9.1 Oggetti SAXException

La classe di eccezione `SAXException` supporta i seguenti metodi:

`getMessage`()
: Restituisce un messaggio umanamente comprensibile che descrive la condizione di errore.

`getException`()
: Restituisce un oggetto eccezione incapsulato o `None`.

13.10 `xml.sax.handler` — Classe di base per gestori SAX

Nuovo nella versione 2.0.

L'API SAX definisce quattro tipi di gestori: gestori del contenuto, gestori DTD, gestori di errori, e risolutori di entità. Le applicazioni necessitano soltanto di implementare queste interfacce per gli eventi per i quali vengono interessate; queste possono implementare le interfacce in un oggetto singolo o in oggetti multipli. Le implementazioni dei gestori devono ereditare dalla classe di base fornita nel modulo `xml.sax`, cosicché tutti i metodi ricevano la stessa implementazione.

class `ContentHandler`
Questa è l'interfaccia principale di callback per SAX e una delle più importanti nelle applicazioni. L'ordine di eventi in questa interfaccia ricalca l'ordine delle informazioni nel documento.

class `DTDHandler`
Gestisce eventi DTD.

Questa interfaccia specifica solo quegli eventi DTD richiesti per l'analisi di base (entità non analizzate ed attributi).

class `EntityResolver`
Interfaccia di base per le entità in risoluzione. Se si crea un oggetto implementando questa interfaccia, si registra quindi l'oggetto con il proprio Parser, che chiamerà il metodo nel proprio oggetto per risolvere tutte le entità esterne.

class `ErrorHandler`
Interfaccia usata dal parser per presentare l'errore ed i messaggi di avviso alle applicazioni. I metodi di questo oggetto controllano che gli errori vengano convertiti immediatamente in eccezioni o se debbano essere gestiti in qualche altro modo.

In aggiunta a queste classi, `xml.sax.handler` fornisce costanti simboliche per le caratteristiche avanzate e nomi di proprietà.

`feature_namespaces`
Valore: `http://xml.org/sax/features/namespaces`
vero: Processa lo spazio dei nomi.
falso: Facoltativamente non processa lo spazio dei nomi (richiede il prefisso dello spazio dei nomi; predefinito).
accesso: (analisi) sola lettura; (non analisi) lettura/scrittura.

`feature_namespace_prefixes`
Valore: `http://xml.org/sax/features/namespace-prefixes`
vero: Riporta il nome del prefisso originale e gli attributi usati dalla dichiarazione dello spazio dei nomi.
falso: Non riporta gli attributi usati dalla dichiarazione dello spazio dei nomi e, facoltativamente non riporta il prefisso originale (predefinito).
accesso: (analisi) sola lettura; (non analisi) lettura/scrittura.

`feature_string_interning`
Valore: `http://xml.org/sax/features/string-interning` vero: Tutti i nomi degli elementi, prefissi, nomi di attributo, URI di spazi dei nomi e nomi locali vengono acquisiti usando la funzione built-in intern.
falso: I nomi non vengono necessariamente acquisiti tramite intern, oppure lo possono essere (predefinito).
accesso: (analisi) sola lettura; (non analisi) lettura/scrittura.

`feature_validation`
Valore: `http://xml.org/sax/features/validation`
vero: Riporta tutti gli errori di validazione (incluse le external-general-entitites e le external-parameter-entities).
falso: Non riporta gli errori di validazione.
accesso: (analisi) sola lettura; (non analisi) lettura/scrittura.

`feature_external_ges`
Valore: `http://xml.org/sax/features/external-general-entities`
vero: Include tutte le entità (testo) generali esterne.
falso: Non include le entità generali esterne.
accesso: (analisi) sola lettura; (non analisi) lettura/scrittura.

feature_external_pes
Valore: `http://xml.org/sax/features/external-parameter-entities`
vero: Include tutte le entità di parametri esterni, inclusi i sotto insiemi dei DTD esterni.
falso: Non include le entità generali esterne tranne i sottoinsiemi di DTD esterni.
accesso: (analisi) sola lettura; (non analisi) lettura/scrittura.

all_features
Elenco delle funzionalità.

property_lexical_handler
Valore: `http://xml.org/sax/properties/lexical-handler`
tipo di dato: xml.sax.sax2lib.LexicalHandler (non supportato in Python 2)
descrizione: Una facoltativa estensione per gestire eventi lessicali come fossero commenti.
accesso: lettura/scrittura

property_declaration_handler
Valore: `http://xml.org/sax/properties/declaration-handler`
tipo di dato: xml.sax.sax2lib.DeclHandler (non supportato in Python 2)
descrizione: un'estensione gestore facoltativa, per eventi correlati a DTD oltreché notazioni ed entità non analizzate.
Accesso: lettura/scrittura

property_dom_node
Valore: `http://xml.org/sax/properties/dom-node`
tipo di dato: org.w3c.dom.Node (non supportato in Python 2) descrizione: quando viene analizzato, il corrente nodo DOM viene visitato se è un iteratore DOM; quando non analizza per iterazione la directory root del nodo DOM.
Accesso: (analisi) in sola lettura, (nessuna analisi) lettura/scrittura

property_xml_string
Valore: `http://xml.org/sax/properties/xml-string`
tipo di dato: Stringa
descrizione: la stringa costante dei caratteri che erano il sorgente per l'evento corrente.
Accesso: in sola lettura.

all_properties
Elenco di tutti i nomi delle proprietà.

13.10.1 Oggetti ContentHandler

Gli utenti si aspettano che la sotto classe `ContentHandler` supporti le loro applicazioni. I seguenti metodi vengono chiamati dal parser sugli appropriati eventi nei documenti in input:

setDocumentLocator(*locator*)
Chiamato dal parser per dare all'applicazione un indicatore di posizione per localizzare l'origine degli eventi del documento.

I parser SAX vengono fortemente richiesti (anche se non assolutamente richiesti) per sostituire un indicatore di posizione. Se un parser si comporta così, allora deve fornire un indicatore di posizione invocando questo metodo prima di invocare altri metodi nell'interfaccia del DocumentHandler.

L'indicatore di posizione permette all'applicazione di determinare la posizione finale di ogni evento relativo ad un documento, se il parser non riporta un errore. Tipicamente, le applicazioni useranno questa informazione per riportare i propri errori (come i caratteri che non corrispondono con le regole di applicazioni business). L'informazione restituita dall'indicatore di posizione probabilmente non è sufficiente per essere usata con un motore di ricerca.

Notare che l'indicatore di posizione restituirà le informazioni soltanto durante l'invocazione di eventi su questa interfaccia. L'applicazione non tenterà di usarlo un'altra volta.

startDocument()
Riceve la notifica dell'inizio di un documento.

Il parser SAX invocherà questo metodo una volta sola, prima di ogni altro metodo in questa interfaccia o nel DTDHandler (con un'eccezione rappresentata da `setDocumentLocator()`).

`endDocument``()`
Riceve la notifica della fine di un documento.

Il parser SAX invocherà questo metodo una volta sola e sarà l'ultimo metodo invocato durante l'analisi. Il parser non invocherà questo metodo prima di avere finito di analizzare (per un errore irrecuperabile) o per aver raggiunto la fine dell'input.

`startPrefixMapping``(`*prefix, uri*`)`
Inizia il campo d'azione di un prefisso URI della mappatura dello spazio dei nomi.

L'informazione da questo evento non è necessaria per la normale elaborazione dello spazio dei nomi: il lettore SAX XML rimpiazzerà automaticamente i prefissi per i nomi degli elementi e degli attributi quando la caratteristica `feature_namespaces` viene abilitata (predefinito).

Ci sono casi, comunque, quando le applicazioni hanno la necessità dell'uso di prefissi in dati carattere o nel valore degli attributi, dove non possono essere espansi automaticamente in modo sicuro; gli eventi `startPrefixMapping()` e `endPrefixMapping()` forniscono le informazioni per l'applicazione per espandere i prefissi in quegli stessi contesti, se necessario.

Notare che gli eventi `startPrefixMapping()` e `endPrefixMapping()` non vengono garantiti perché siano annidati in relazione agli altri: tutti gli eventi `startPrefixMapping()` e tutti gli eventi `endPrefixMapping()` avverranno dopo l'evento corrispondente `endElement()`, ma il loro ordine non è garantito.

`endPrefixMapping``(`*prefix*`)`
Termina il campo d'azione della mappatura del prefisso URI.

Vedere `startPrefixMapping()` per i dettagli. Questo evento avviene sempre dopo l'evento corrispondente `endElement()`, ma l'ordine degli eventi `endPrefixMapping()` non viene altrettanto garantito.

`startElement``(`*name, attrs*`)`
Segnali che indicano l'avvio di un elemento in una modalità che non tiene conto dello spazio dei nomi.

Il parametro *name* contiene il nome del tipo di elemento grezzo XML 1.0 come fosse una stringa e il parametro *attrs* mantiene un oggetto dell'interfaccia `Attributes` contenente gli attributi dell'elemento. L'oggetto passato come *attrs* può essere riusato dal parser; mantenere un riferimento ad esso non è un modo certo per mantenere una copia degli attributi. Per mantenere una copia degli attributi, usare il metodo `copy()` dell'oggetto *attrs*.

`endElement``(`*name*`)`
Segnali che indicano l'avvio di un elemento in una modalità che non tiene conto dello spazio dei nomi.

Il parametro *name* contiene il nome del tipo dell'elemento, come con l'evento `startElement()`.

`startElementNS``(`*name, qname, attrs*`)`
Segnali che indicano l'avvio di un elemento in una modalità che non tiene conto dello spazio dei nomi.

Il parametro *name* contiene il nome del tipo dell'elemento come se fosse una tupla `(`*uri*`,` *nomelocale*`)`, il parametro *qname* contiene il nome grezzo XML 1.0 usato nel sorgente del documento ed il parametro *attrs* mantiene un'istanza dell'interfaccia `AttributesNS` che contiene gli attributi dell'elemento. Se nessuno spazio dei nomi viene associato con l'elemento, il componente uri del nome avrà il valore `None`. L'oggetto passato come *attrs* può essere riusato dal parser; mantenere un riferimento ad esso non è un modo certo per mantenere una copia degli attributi. Per mantenere una copia degli attributi, usate il metodo `copy()` dell'oggetto *attrs*.

I parser possono impostare il parametro *qname* a `None`, a meno che la funzionalità `feature_namespace_prefixes` sia attivata.

`endElementNS``(`*name, qname*`)`
Segnali che indicano l'avvio di un elemento in una modalità che non tiene conto dello spazio dei nomi.

Il parametro *name* contiene il nome del tipo dell'elemento, come con il metodo `startElementNS()`, alla stregua del parametro *qname*.

`characters``(`*content*`)`
Riceve una notifica di dati carattere.

Il parser chiamerà questo metodo per riportare ogni spezzone di dati carattere. Il parser SAX può restituire tutti i dati carattere contigui in un singolo spezzone, o possono essere suddivisi in più pezzi; tuttavia, tutti i caratteri in ogni singolo evento devono provenire dalla medesima entità esterna, in questo modo il Locator fornisce informazioni utili.

content può essere una stringa Unicode o una stringa composta da byte; il modulo lettore `expat` produce sempre stringhe Unicode.

Note: La prima interfaccia SAX 1 fornita dal Python XML Special Interest Group usava, per questo metodo, molte interfacce simili a quelle di Java. Poiché la maggior parte dei parser usati da Python non traggono vantaggi dalla vecchia interfaccia, è stata scelta una forma più semplice per sostituirla. Per convertire il vecchio codice alla nuova interfaccia, usare *content* invece di affettare il contenuto con i vecchi parametri *offset* e *length*.

`ignorableWhitespace`(*whitespace*)
Riceve una notifica di uno spazio vuoto, ignorabile nell'elemento content.

I parser convalidanti devono usare questo metodo per riportare ogni spezzone di spazi vuoti ignorabili (vedere la raccomandazione W3C XML 1.0 sezione 2.10): anche i parser non convalidanti possono usare questo metodo se sono capaci di analizzare ed usano i modelli di contenuto.

Il parser SAX può restituire tutti i dati carattere contigui in un singolo spezzone, o possono essere suddivisi in più pezzi; tuttavia, tutti i caratteri in ogni singolo evento devono provenire dalla medesima entità esterna, in questo modo il Locator fornisce informazioni utili.

`processingInstruction`(*target, data*)
Riceve notifica di un'istruzione elaborata.

Il parser invocherà questo metodo una volta per tutte le istruzioni trovate: notare che queste istruzioni di elaborazione possono trovarsi prima o dopo l'elemento del documento principale.

Un parser SAX non dovrebbe mai riportare una dichiarazione XML (XML 1.0, sezione 2.8) o una dichiarazione di testo (XML 1.0, sezione 4.3.1) usando questo metodo.

`skippedEntity`(*name*)
Riceve notifica di un'entità saltata.

Il Parser chiamerà questo metodo per ogni entità saltata. Processori che non validano possono saltare entità se non hanno visto le dichiarazioni (perché, per esempio, l'entità è stata dichiarata in un sotto insieme DTD esterno). Tutti i processori possono saltare delle entità in base al valore delle proprietà `feature_external_ges` e `feature_external_pes`.

13.10.2 Oggetti DTDHandler

Le istanze `DTDHandler` forniscono i metodi seguenti:

`notationDecl`(*name, publicId, systemId*)
Gestisce un evento dichiarazione di una notazione.

`unparsedEntityDecl`(*name, publicId, systemId, ndata*)
Gestisce un evento dichiarazione di una entità non analizzata.

13.10.3 Oggetti EntityResolver

`resolveEntity`(*publicId, systemId*)
Risolve l'identificatore di sistema di una entità e restituisce o l'identificatore di sistema da leggere da una stringa, oppure una *InputSource* da dove leggere. L'implementazione predefinita restituisce *systemId*.

13.10.4 Oggetti ErrorHandler

Gli oggetti con questa interfaccia vengono utilizzati per ricevere informazioni di errore o avvertimento da `XMLReader`. Se si crea un oggetto che implementa questa interfaccia e lo si registra con `XMLReader`, il parser

chiamerà i metodi di questo oggetto per segnalare tutti gli errori e gli avvertimenti. Sono disponibili tre livelli di errore: avvertimenti, errori (potenzialmente) recuperabili, ed errori irrecuperabili. Tutti i metodi ricevono un oggetto `SAXParseException` come unico parametro. Errori ed avvertimenti possono essere convertiti in un'eccezione sollevando a loro volta l'eccezione passata in ingresso come oggetto eccezione.

`error`(*exception*)
Chiamato quando il parser incontra un errore recuperabile. Se questo metodo non solleva un'eccezione, l'analisi può continuare, ma l'applicazione non si deve aspettare l'informazione contenuta nel documento successiva all'errore. Il fatto di permettere al parser di continuare l'analisi può risultare utile per l'identificazione di ulteriori errori all'interno del documento.

`fatalError`(*exception*)
Chiamato quando il parser incontra un errore che non è in grado di recuperare; ci si aspetta che l'analisi termini una volta terminata l'esecuzione del metodo.

`warning`(*exception*)
Chiamato quando il parser segnala degli avvertimenti all'applicazione. L'analisi ci si aspetta continui una volta terminata l'esecuzione di questo metodo, e l'informazione contenuta nel documento successiva alla segnalazione continuerà ad essere restituita all'applicazione. Sollevare un'eccezione in questo metodo causerà la fine dell'analisi.

13.11 `xml.sax.saxutils` — Utilità SAX

Nuovo nella versione 2.0.

Il modulo `xml.sax.saxutils` contiene una serie di classi e funzioni che di solito risultano utili durante la creazione di applicazioni SAX, sia mediante il loro uso diretto che come classi base.

`escape`(*data*[, *entities*])
Codifica '`&`', '`<`' e '`>`' in una stringa di dati.

È possibile codificare altre stringhe passando un dizionario come parametro facoltativo *entities*. Sia le chiavi che i valori devono essere stringhe; ogni chiave verrà sostituita con il proprio corrispondente valore.

`unescape`(*data*[, *entities*])
Decodifica '`&`', '`<`', e '`>`' in una stringa di dati.

È possibile decodificare altre stringhe passando un dizionario come parametro facoltativo *entities*. Sia le chiavi che i valori devono essere stringhe; ogni chiave verrà sostituita con il proprio corrispondente valore.

Nuovo nella versione 2.3.

`quoteattr`(*data*[, *entities*])
Simile a `escape()`, ma inoltre prepara *data* per essere utilizzato come un valore per un attributo. Il valore restituito è una versione quotata di *data* con tutte le sostituzioni richieste. `quoteattr()` utilizzerà il carattere per quotare in base al contenuto di *data*, cercando di evitare di codificare i caratteri utilizzati per quotare all'interno della stringa. Se sia il carattere singolo apice che il carattere doppio apice sono presenti in *data*, allora il carattere doppio apice verrà codificato e *data* verrà tramutato in doppio apice. La stringa risultante può essere utilizzata direttamente come un valore di attributo:

```
>>> print "<element attr=%s>" % quoteattr("ab ' cd \" ef")
<element attr="ab ' cd &quot; ef">
```

Questa funzione è utile per generare valori attributo per HTML o SGML usando il riferimento della sintassi concreta. Nuovo nella versione 2.2.

class `XMLGenerator`([*out*[, *encoding*]])
Questa classe implementa l'interfaccia `ContentHandler` per scrivere eventi Sax facendo fare un passo indietro al documento XML. In altre parole, usando un `XMLGenerator` come contenuto delle intestazioni, riprodurrà l'originale documento di cui si è iniziata l'analisi. *out* dovrebbe essere un oggetto simile a file che per definizione stamperà su *sys.stdout* il suo output. *encoding* è la codifica del flusso in uscita che avrà come impostazione predefinita per la codifica `'iso-8859-1'`.

class `XMLFilterBase`(*base*)
Questa classe è stata progettata per essere una via di mezzo tra `XMLReader` ed un gestore di eventi di un'applicazione client. Di norma, non fa niente ma passa richieste al lettore ed eventi ai gestori senza apportare modifiche, ma sotto classi possono sovrascrivere specifici metodi per modificare flussi di eventi o la configurazione richiesta così come gli viene passata.

`prepare_input_source`(*sorgente*[, *base*])
Questa funzione prende un sorgente ed una URL facoltativa e restituisce un oggetto `InputSource` pronto per essere letto. Il sorgente può essere fornito sotto forma di stringa, un oggetto simile a file, o un oggetto `InputSource`; i parser usano questa funzione per implementare una poliformica *sorgente* al metodo `parse()`.

13.12 `xml.sax.xmlreader` — Interfaccia per parser XML

Nuovo nella versione 2.0.

I parser SAX implementano l'interfaccia `XMLReader`. Vengono implementati in un modulo Python, che deve fornire una funzione `create_parser()`. Questa funzione viene invocata da `xml.sax.make_parser()` senza argomenti per creare un nuovo oggetto parser.

class `XMLReader`()
Classe base che può essere ereditata dal parser SAX.

class `IncrementalParser`()
In alcuni casi è desiderabile non analizzare un sorgente in input subito, ma introdurre buona parte del documento così come questo è disponibile. Notare che il lettore di solito non legge l'intero file, ma, più comodamente, ne legge un pezzo; anche `parse()` non può restituire i suoi risultati finché l'intero documento non viene elaborato. Queste interfacce possono essere usate se il comportamento bloccante di `parse()` non è desiderabile.

Quando il parser viene istanziato, viene letto per iniziare ad accettare dati dal metodo di alimentazione immediatamente. Successivamente l'analisi viene terminata con una chiamata a `close()`, il metodo `reset()` può essere chiamato per preparare il parser ad accettare nuovi dati, per alimentare il parser o per usare il metodo `parse()`.

Notare che questi metodi *non* devono essere chiamati durante l'analisi, ma dopo che l'analisi è stata invocata e prima che essa restituisca i suoi risultati.

Di norma, la classe implementa il metodo parse dell'interfaccia XMLReader usando metodi feed, close e reset dell'interfaccia IncrementalParser come un conveniente driver per la scrittura in SAX 2.0.

class `Locator`()
Interfaccia per associare un evento SAX ad un determinato documento. Un oggetto locator restituisce risultati validi solamente durante chiamate con metodi DocumentHandler; in ogni altro momento i risultati saranno imprevedibili. Se non sono disponibili informazioni, i metodi dovrebbero restituire `None`.

class `InputSource`([*systemId*])
L'incapsulamento di informazioni necessita dell'`XMLReader` per leggere le entità.

Questa clase dovrebbe includere informazioni circa l'identificatore pubblico e l'identificatore del sistema, il flusso dei byte (possibilmente con informazioni sulla codifica dei caratteri) e o con il flussi di caratteri da un'entità.

Applicazioni creeranno oggetti di questa classe per usarla nel metodo `XMLReader.parse()` o per ritornare da EntityResolver.resolveEntity.

Un `InputSource` appartiene ad un'applicazione. L'`XMLReader` non consente di modificare oggetti `InputSource` passati a esso da un'applicazione. Questo nonostante il suo comportamento dovrebbe essere quello di realizzare una copia e modificarla.

class `AttributesImpl`(*attrs*)
Questa è un implementazione degli `attributi` dell'interfaccia (vedere la sezione 13.12.5). Questo è un oggetto simil dizionario che rappresenta un attributo nella chiamata `startElement()`. In aggiunta alle più utili operazioni del tipo dizionario, supporta una varietà di altri metodi come descritto per le interfacce.

Gli oggetti di questa classe devono essere istanziati attraverso un lettore; *attrs* deve esssere un oggetto simil dizionario contenente una mappa dei nomi degli attributi ed il loro valore.

class **`AttributesNSImpl`**(*attrs*, *qnames*)
Spazio dei nomi consapevoli, variante di `AttributesImpl`, che verrà passata a `startElementNS()`. Questo deriva da `AttributesImpl`, ma non capisce i nomi degli attributi come le due tuple di *namespaceURI* e *localname*. In aggiunta esso fornisce un numero di metodi eccetto i nomi qualificati come essi appaiono nel documento originale. Questa classe implementa l'interfaccia `AttributesNS` (vedere la sezione 13.12.6).

13.12.1 Oggetti XMLReader

L'interfaccia `XMLReader` supporta i seguenti metodi:

`parse`(*source*)
Processa un codice in ingresso, producendo eventi SAX. L'oggetto *source* può essere un identificatore di sistema (una stringa identificante il codice sorgente – tipicamente un nome di un file o di una URL), un oggetto simile a file, o un oggetto `InputSource`. Quando `parse()` ritorna, l'input è stato completamente processato e l'oggetto parser può essere abbandonato o resettato. Come limitazione la corrente implementazione accetta flussi di byte. I futuri studi vanno nella direzione dei flussi di caratteri.

`getContentHandler`()
Restituisce il corrente `ContentHandler`.

`setContentHandler`(*handler*)
Imposta il corrente `ContentHandler`. Se nessun `ContentHandler` viene impostato, gli eventi contenuti vengono abbandonati.

`getDTDHandler`()
Restituisce il corrente `DTDHandler`.

`setDTDHandler`(*handler*)
Imposta il corrente `DTDHandler`. Se nessun `DTDHandler` viene assegnato, gli eventi DTD vengono abbandonati.

`getEntityResolver`()
Restituisce il corrente `EntityResolver`.

`setEntityResolver`(*handler*)
Imposta il corrente `EntityResolver`. Se nessun `EntityResolver` viene assegnato, tenta di risolvere un'entità esterna che risulterà nell'apertura di un identificatore di sistema per l'entità e fallisce se non è disponibile.

`getErrorHandler`()
Restituisce il corrente `ErrorHandler`.

`setErrorHandler`(*handler*)
Imposta il corrente gestore di errori. Se `ErrorHandler` non viene impostata, l'errore solleverà un'eccezione e verranno stampati gli avvertimenti.

`setLocale`(*locale*)
Consente ad un'applicazione di impostare l'ambiente per errori ed avvisi.

I parser SAX non vengono richiesti per fornire la localizzazione se non supportano l'ambiente richiesto. Comunque dovranno lanciare un'eccezione SAX. Applicazioni possono richiedere cambiamenti dell'ambiente locale nel mezzo di un'analisi.

`getFeature`(*featurename*)
Restituisce la corrente impostazione per la funzionalità *featurename*. Se la funzionalità non viene riconosciuta verrà sollevata un'eccezione `SAXNotRecognizedException`. Le ben conosciute funzionalità sui nomi vengono elencate nel modulo `xml.sax.handler`.

`setFeature`(*featurename*, *value*)
Imposta *featurename* a *value*. Se una funzionalità non viene riconosciuta, viene sollevata un'eccezione

`SAXNotRecognizedException`. Se la funzionalità o la sua impostazione non è supportata dal parser, viene sollevata un'eccezione `SAXNotRecognizedException`.

`getProperty`(*propertyname*)
Restituisce la corrente impostazione della proprietà *propertyname*. Se la proprietà non viene riconosciuta, verrà sollevata un'eccezione `SAXNotRecognizedException`. Le ben conosciute *propertyname* vengono elencate nel modulo `xml.sax.handler`.

`setProperty`(*propertyname, value*)
Imposta *propertyname* a *value*. Se la proprietà non viene riconosciuta, viene sollevata l'eccezione `SAXNotRecognizedException`. Se la proprietà o la sua impostazione non sono supportate dal parser viene sollevata l'eccezione *SAXNotSupportedException*.

13.12.2 Oggetti IncrementalParser

Le istanze di `IncrementalParser` offrono i seguenti ulteriori metodi:

`feed`(*dati*)
Processa un blocco di *dati*.

`close`()
Stabilisce la fine del documento. Verificherà se il documento è ben formato dal punto di vista XML, condizione verificabile solo alla fine, invocherà i gestori e potrà liberare le risorse allocate durante l'analisi.

`reset`()
Questo metodo viene chiamato dopo la chiamata a close per resettare il parser, in modo che sia pronto per analizzare nuovi documenti. Nel caso non venisse chiamato reset dopo close, i risultati delle chiamate a parse o feed sarebbero indefinite.

13.12.3 Oggetti Locator

Le istanze di `Locator` forniscono i seguenti metodi:

`getColumnNumber`()
Restituisce il numero della colonna in cui termina l'evento corrente.

`getLineNumber`()
Restituisce il numero della linea in cui termina l'evento corrente.

`getPublicId`()
Restituisce l'identificatore pubblico per l'evento corrente.

`getSystemId`()
Restituisce l'identificatore di sistema per l'evento corrente.

13.12.4 Oggetti InputSource

`setPublicId`(*id*)
Imposta l'identificatore pubblico di questo `InputSource`.

`getPublicId`()
Restituisce l'identificatore pubblico di questo `InputSource`.

`setSystemId`(*id*)
Imposta l'identificatore di sistema di questo `InputSource`.

`getSystemId`()
Restituisce l'identificatore di sistema di questo `InputSource`.

`setEncoding`(*encoding*)
Imposta la codifica dei caratteri di questo `InputSource`.

La codifica deve essere una stringa valida per una dichiarazione di codifica XML (vedere la sezione 4.3.3 delle raccomandazioni XML).

L'attributo *encoding* di `InputSource` viene ignorata se `InputSource` contiene anche un flusso di caratteri.

`getEncoding``()`
Acquisisce la codifica dei caratteri di questo `InputSource`.

`setByteStream``(`*bytefile*`)`
Imposta il flusso dei byte (un oggetto simile a file Python che non effettua la conversione da byte a carattere) per questa sorgente in input.

Il parser SAX ignorerà il flusso dei byte se vi è già specificato un flusso di caratteri ma lo userà preferibilmente per l'apertura di una connessione con una URI.

Se l'applicazione conosce la codifica dei caratteri del flusso dei byte la imposterà tramite il metodo setEncoding.

`getByteStream``()`
Acquisisce il flusso dei byte da questa sorgente di input.

Il metodo getEncoding restituirà la codifica dei caratteri per questo flusso di byte, o `None` se risultasse sconosciuto.

`setCharacterStream``(`*charfile*`)`
Imposta il flusso dei caratteri per questo sorgente in input (il flusso deve essere un simile a file del tipo Python 1.6 Unicode-wrapped che effettui la conversione in stringhe Unicode).

Se viene specificato un flusso di caratteri il parser SAX ignorerà qualsiasi flusso di byte e non cercherà di aprire una connessione URI verso l'identificatore di sistema.

`getCharacterStream``()`
Ottiene il flusso dei caratteri per questo sorgente in input.

13.12.5 L'interfaccia `Attributes`

Gli oggetti `Attributes` implementano una porzione del protocollo di mappatura che comprende i metodi `copy()`, `get()`, `has_key()`, `items()`, `keys()` e `values()`. Vengono forniti inoltre i seguenti metodi;

`getLength``()`
Restituisce il numero degli attributi.

`getNames``()`
Restituisce il nome degli attributi.

`getType``(`*name*`)`
Restituisce il tipo del nome dell'attributo che di norma è `'CDATA'`.

`getValue``(`*name*`)`
Restituisce il valore del nome, *name* dell'attributo.

13.12.6 L'interfaccia `AttributesNS`

Questa interfaccia è un sotto tipo dell'interfaccia `Attributes` (vedere la sezione 13.12.5). Tutti i metodi supportati da tale interfaccia sono disponibili anche per gli oggetti `AttributesNS`.

Sono inoltre disponibili i seguenti metodi:

`getValueByQName``(`*name*`)`
Restituisce il valore di un nome qualificato.

`getNameByQName``(`*name*`)`
Restituisce la coppia `(`*namespace*`, `*localname*`)` di un nome, *name*, qualificato.

`getQNameByName``(`*name*`)`
Restituisce il nome qualificato della coppia `(`*namespace*`, `*localname*`)`.

getQNames()
Restituisce i nomi qualificati di tutti gli attributi.

13.13 xmllib — Un parser per documenti XML

Deprecato dalla versione 2.0. Usare invece xml.sax. Il più recente package XML fornisce pieno supporto per XML 1.0.

Modificato nella versione 1.5.2: Aggiunto il supporto per lo spazio dei nomi.

Questo modulo definisce una classe XMLParser che serve da base per l'analisi di file di testo formattati in XML (Extensible Markup Language).

class XMLParser()
La classe XMLParser deve essere istanziata senza argomenti.[1]

Questa classe fornisce i seguenti metodi di interfaccia e variabili d'istanza:

attributes
Una mappatura dei nomi di elementi in un dizionario. L'ultima mappatura mappa i nomi degli attributi validi per l'elemento al valore predefinito valido per l'attributo o, se non vi è valore predefinito, a None. Il valore predefinito è un dizionario vuoto. Questa variabile si intende sovrascritta, non viene estesa poiché il valore predefinito viene condiviso da tutte le istanze di XMLParser.

elements
Una mappatura di nomi di elementi in tuple. Le tuple contengono una funzione per la gestione rispettivamente del tag di inizio e di fine dell'elemento, oppure None se viene chiamato il metodo unknown_starttag() o unknown_endtag(). Il valore predefinito è il dizionario vuoto. Questa variabile si intende sovrascritta, non viene estesa poiché il valore predefinito viene condiviso da tutte le istanze di XMLParser.

entitydefs
Un mappatura di nomi di entità nei propri valori. Il valore predefinito contiene definizioni per 'lt', 'gt', 'amp', 'quot', e 'apos'.

reset()
Resetta l'istanza. Si perdono tutti i dati non processati. Viene chiamata implicitamente al momento dell'istanziazione.

setnomoretags()
Interrompe l'elaborazione dei tag. Tutto l'input successivo viene trattato come input literal (CDATA).

setliteral()
Entra in modalità costante (modalità CDATA). Esce automaticamente da questa modalità quando incontra un tag close che chiude l'ultimo tag rimasto aperto.

feed(*data*)
Fornisce del testo al parser. Tale testo viene processato fintanto che si riscontrano tag completi; dati incompleti vengono bufferizzati fino a quando non se ne forniscono altri o fino alla chiamata close().

close()
Forza l'elaborazione di tutti i dati bufferizzati come se fossero seguiti da un segno di fine file. Questo metodo può essere ridefinito da una classe derivata per impostare ulteriori elaborazioni alla fine dell'input, comunque la versione ridefinita deve sempre chiamare close().

translate_references(*data*)
Converte tutte le entità ed i riferimenti ai carattere in dati, *data*, e restituisce la stringa convertita.

[1] Attualmente, un certo numero di particolari argomenti possono influenzare l'interprete ad accettare costrutti non standard. Segue una lista di questi argomenti. Per tutti il valore predefinito è 0 (false) eccetto che per l'ultimo, per il quale è 1 (vero). *accept_unquoted_attributes* (accetta certi valori senza le virgolette), *accept_missing_endtag_name* (accetta tag finali simili a </>), *map_case* (considera le maiuscole come minuscole nei tag e negli attributi), *accept_utf8* (permette caratteri UTF-8 in input; questo viene richiesto dallo standard XML, ma Python non si trova bene con questi caratteri, così non è l'impostazione predefinita), *translate_attribute_references* (non tenta di tradurre i riferimenti a caratteri ed entità in valori attributo).

getnamespace()
Restituisce una mappatura di abbreviazioni di spazi dei nomi in spazi dei nomi URI esistenti al momento.

handle_xml(*encoding, standalone*)
Questo metodo viene chiamato quando viene elaborato il tag '<?xml ...?>'. Gli argomenti sono i valori della codifica e attributi isolati del tag. Sia la codifica che gli attributi isolati sono facoltativi. I valori predefiniti passati a handle_xml() sono rispettivamente None e la stringa 'no'.

handle_doctype(*tag, pubid, syslit, data*)
Questo metodo viene chiamato quando si processa la dichiarazione '<!DOCTYPE...>'. Gli argomenti sono il nome del tag dell'elemento radice, il Formal Public Identifier (o None se non specificato), l'identificatore di sistema ed i contenuti non interpretati del sotto insieme interno DTD sotto forma di stringa (o None se non presenti).

handle_starttag(*tag, method, attributes*)
Questo metodo viene chiamato per gestire i tag di inizio per i quali viene definita un'istanza variabile di elements. L'argomento *tag* è il nome del tag e l'argomento *method* è la funzione (metodo) che si dovrebbe usare per supportare l'interpretazione semantica del tag di inizio. L'argomento *attributes* è un dizionario di attributi, dove la chiave è il nome *name* ed il valore *value* è il valore dell'attributo trovato tra i delimitatori <> del tag. I riferimenti di entità e caratteri in *value* vengono interpretati. Ad esempio per il tag di inizio <A HREF=http://www.cwi.nl/», questo metodo verrebbe chiamato come handle_starttag('A', self.elements['A'][0], {'HREF': 'http://www.cwi.nl/'}). L'implementazione base chiama semplicemente il metodo *method* con l'attributo *attributes* come unico argomento.

handle_endtag(*tag, method*)
Questo metodo viene chiamato per gestire i tag di fine per i quali viene definito nelle istanze variabili di elements. L'argomento *tag* è il nome del tag e l'argomento *method* è la funzione (metodo) che si dovrebbe usare per supportare l'interpretazione semantica del tag di fine. Ad esempio per il tag di fine </A> questo metodo verrebbe chiamato come handle_endtag('A', self.elements['A'][1]). L'implementazione base chiama semplicemente *method*.

handle_data(*data*)
Questo metodo viene chiamato per elaborare dati arbitrari. Su di esso viene effettuata una sovrascrittura mediante una classe derivata. L'implementazione della classe base non fa nulla.

handle_charref(*ref*)
Questo metodo viene chiamato per processare un riferimento a caratteri del tipo '&#*ref*;'. *ref* può essere sia un numero decimale che un esadecimale quando preceduto da una 'x'. Nell'implementazione base, *ref* deve essere un numero compreso nell'intervallo 0-255. Converte il carattere in ASCII e chiama il metodo handle_data() con il carattere come argomento. Se *ref* non è valido o eccede l'intervallo, viene chiamato il metodo unknown_charref(*ref*) per gestire l'errore. Una sotto classe deve effettuare la sovrascrittura di questo metodo per fornire un supporto per quei riferimenti a caratteri non compresi nell'intervallo ASCII.

handle_comment(*comment*)
Questo metodo viene chiamato quando si incontra un commento. L'argomento *comment* è una stringa che contiene il testo compreso tra i delimitatori '<!-' e '->' ma non i delimitatori stessi. Ad esempio il commento '<!-text->' causerà la chiamata di questo metodo con 'text' come argomento. Il metodo predefinito non fa nulla.

handle_cdata(*data*)
Questo metodo viene chiamato quando si incontra un elemento CDATA. L'argomento *data* è una stringa che contiene il testo compreso tra i delimitatori '<![CDATA[' e ']]>' ma non i delimitatori stessi. Ad esempio l'entità '<![CDATA[text]]>' causerà la chiamata di questo metodo con text come argomento. Il metodo predefinito non fa nulla ed è inteso come da sovrascrivere.

handle_proc(*name, data*)
Questo metodo viene chiamato quando si incontra un'istruzione che indica un'elaborazione (PI (NdT: Processing Instruction)). *name* è l'obiettivo della PI e l'argomento *data* è una stringa che contiene il testo compreso tra il target della PI ed il delimitatore di chiusura ma non il delimitatore stesso. Ad esempio l'istruzione '<?XML text?>' causerà la chiamata di questo metodo con 'XML' e 'text' come argomenti. Il metodo predefinito non fa nulla. Notare che se un documento inizia con '<?xml ..?>', per gestirlo viene chiamato handle_xml().

handle_special(*data*)
Questo metodo viene chiamato quando viene incontrata una dichiarazione. L'argomento *data* è una stringa che contiene il testo compreso tra i delimitatori '<!' e '>' ma non i delimitatori stessi. Ad esempio la dichiarazione di entità '<!ENTITY text>' causerà la chiamata di questo metodo con 'ENTITY text' come argomento. Il metodo predefinito non fa nulla. Notare che '<!DOCTYPE ...>' viene gestito separatamente se si trova all'inizio del documento.

syntax_error(*message*)
Questo metodo viene chiamato quando viene incontrato un errore di sintassi. *message* è una descrizione di ciò che non va. Il metodo predefinito solleva un'eccezione RuntimeError. Se di questo metodo è stata effettuata una sovrascrittura è possibile la restituzione di un valore. Questo metodo viene chiamato solo quando l'errore è recuperabile. Gli errori irrecuperabili sollevano un'eccezione RuntimeError senza prima chiamare syntax_error().

unknown_starttag(*tag, attributes*)
Questo metodo viene chiamato per processare un tag di inizio sconosciuto. Su di esso si effettua una sovrascrittura da una classe derivata; l'implementazione della classe base non fa nulla.

unknown_endtag(*tag*)
Questo metodo viene chiamato per processare un tag di fine sconosciuto. Su di esso si effettua una sovrascrittura da una classe derivata; l'implementazione della classe base non fa nulla.

unknown_charref(*ref*)
Questo metodo viene chiamato per processare un riferimento a caratteri numerici irrisolvibili. Su di esso si effettua una sovrascrittura da una classe derivata; l'implementazione della classe base non fa nulla.

unknown_entityref(*ref*)
Questo metodo viene chiamato per processare un riferimento ad entità sconosciuta. Su di esso si effettua una sovrascrittura da una classe derivata; l'implementazione della classe base chiama syntax_error() per segnalare un errore.

Vedete anche:

Extensible Markup Language (XML) 1.0
(http://www.w3.org/TR/REC-xml)
La specifica XML pubblicata dal World Wide Web Consortium (W3C), definisce la sintassi e i requisiti del processore per XML. Riferimenti a materiale aggiuntivo su XML, comprese le traduzioni della specifica, sono disponibili presso http://www.w3.org/XML/.

Python and XML Processing
(http://www.python.org/topics/xml/)
La Python XML Topic Guide fornisce un gran numero di informazioni sull'uso di XML in Python e di link ad altre fonti di informazioni su XML.

SIG for XML Processing in Python
(http://www.python.org/sigs/xml-sig/)
Il Python XML Special Interest Group sta sviluppando un sostanziale supporto per l'elaborazione di XML in Python.

13.13.1 Spazio dei nomi XML

Questo modulo fornisce il supporto allo spazio dei nomi XML come stabilito nelle raccomandazioni proposte per XML.

I tag e i nomi degli attributi che vengono definiti in uno spazio dei nomi XML vengono gestiti come se il nome del tag o dell'elemento fosse composto dallo spazio dei nomi (l'URL che definisce lo spazio dei nomi) seguito da uno spazio e dal nome del tag o dell'attributo. Ad esempio il tag <html xmlns='http://www.w3.org/TR/REC-html40'> viene trattato come se il nome del tag fosse 'http://www.w3.org/TR/REC-html40 html' ed il tag <html:a href='http://frob.com'> all'interno dell'elemento summenzionato viene trattato come se il nome del tag fosse 'http://www.w3.org/TR/REC-html40 a' ed il nome dell'attributo come se fosse 'http://www.w3.org/TR/REC-html40 href'.

Viene anche riconosiuta una vecchia stesura dello spazio dei nomi di XML, ma viene generato un avvertimento.

Vedete anche:

Namespaces in XML
(http://www.w3.org/TR/REC-xml-names/)
: Questa raccomandazione del World Wide Web Consortium descrive la corretta sintassi ed i requisiti di elaborazione per gli spazi dei nomi in XML.

CAPITOLO

QUATTORDICI

Servizi Multimediali

I moduli descritti in questo capitolo implementano vari algoritmi o interfacce che sono principalmente usate per applicazioni multimediali. Sono disponibili alla discrezione di chi installa. A seguito una panoramica:

audioop	Manipolare i dati audio grezzi.
imageop	Manipolare dati di immagini grezzi.
aifc	Lettura e scrittura di file audio nel formato AIFF o AIFC.
sunau	Fornisce un'interfaccia al formato audio AU di Sun.
wave	Fornisce un'interfaccia al formato audio WAV.
chunk	Modulo per leggere spezzoni di dati IFF.
colorsys	Conversioni di funzioni tra colori RGB e gli altri di sistema.
rgbimg	Leggere e scrivere file di immagini nel formato "SGI RGB" (il modulo *non* è comunque specifico per SGI'!).
imghdr	Determina il tipo di immagine contenuta in un file o in un flusso di dati.
sndhdr	Determina il tipo del file sonoro.
ossaudiodev	Accesso ai dispositivi audio OSS compatili.

14.1 audioop — Manipolare i dati audio grezzi

Il modulo `audioop` contiene alcune utili operazioni sulla frammentazione del suono. Esso opera su frammenti di suono consistenti di 8, 16 o 32 bit interi firmati, immagazzinati in stringhe Python. Questo è lo stesso formato usato dai moduli `al` e `sunaudiodev`. Tutti gli elementi scalari sono interi, a meno che non altrimenti specificato.

Questo modulo fornisce supporto per la codifica u-LAW e Intel/DVI ADPCM.

Alcune delle operazioni più complicate prelevano soltanto i campioni a 16 bit, altrimenti la dimensione del campione (in byte) è sempre un parametro dell'operazione.

Il modulo definisce le seguenti variabili e funzioni:

exception `error`
Questa eccezione viene sollevata per tutti gli errori, come un numero sconosciuto di byte per campione, etc. etc..

`add`(*fragment1, fragment2, width*)
Restituisce un frammento che è la somma dei due campioni passati come parametro. *width* è la lunghezza del campione passata in byte che può essere `1` e `2` o `4`. Entrambi i frammenti dovrebbero avere la stessa lunghezza.

`adpcm2lin`(*adpcmfragment, width, state*)
Decodifica un frammento codificato con Intel/DVI ADPCM in un frammento lineare. Vedere la descrizione di `lin2adpcm()` per i dettagli sulla codifica ADPCM. Restituisce una tupla `(`*sample*`,` *newstate*`)` dove *sample* ha la larghezza specificata in *width*.

`adpcm32lin`(*adpcmfragment, width, state*)
Decodifica un codice a 3-bit ADPCM alternativo. Vedere `lin2adpcm3()` per i dettagli.

`avg`(*fragment, width*)
Restituisce la media su tutti i campioni nei frammenti.

avgpp(*fragment, width*)
Restituisce il valore medio del peak-peak su tutti i campioni del frammento. Non viene applicato alcun filtraggio, così l'utilità di questa routine rimane discutibile.

bias(*fragment, width, bias*)
Restituisce un frammento che è il frammento originale con un bias aggiunto ad ogni campione.

cross(*fragment, width*)
Restituisce il numero di zero attraversati nel frammento passati come un argomento.

findfactor(*fragment, reference*)
Restituisce un fattore *F* come quello `rms(add(`*fragment*`, mul(`*reference*`, -`*F*`)))`, è minimale, per campione restituisce il fattore con il quale si dovrebbe moltiplicare il riferimento *reference* per farlo confrontare il meglio possibile con il frammento *fragment*. I frammenti dovrebbero contenere campioni a 2-byte.

Il tempo preso da questa routine è proporzionale a `len(`*fragment*`)`.

findfit(*fragment, reference*)
Cerca la corrispondenza tra *reference* e una porzione di *fragment* (che dovrebbe essere il frammento più lungo). Questo è (concettualmente) fatto per estrarre porzioni del frammento, usando `findfactor()` per calcolare la migliore corrispondenza e minimizzare il risultato. I frammenti dovrebbero contenere entrambi campioni a 2-byte. Restituisce una tupla `(`*offset*`, `*factor*`)` dove *offset* è l'offset (intero) in *fragment*, da dove inizia la corrispondenza ottimale e *factor* è il fattore (in virgola mobile) come per `findfactor()`.

findmax(*fragment, length*)
Cerca *fragment* per una fetta della lunghezza del campione *length* (non in byte!) con la massima energia, per campione restituisce *i* per cui `rms(fragment[i*2:(i+length)*2])` è massimale. I frammenti dovrebbero contenere entrambi campioni ad 2-byte.

La routine prende il tempo proporzionalmente a `len(`*fragment*`)`.

getsample(*fragment, width, index*)
Restituisce il valore del campione *index* dal frammento.

lin2lin(*fragment, width, newwidth*)
Converte campioni tra formati a 1-, 2- e 4-byte.

lin2adpcm(*fragment, width, state*)
Converte campioni a 4 bit codificati Intel/DVI ADPCM. La codifica ADPCM è uno schema di codifica adattabile, per cui ogni numero a 4 bit è la differenza tra un campione ed il prossimo, diviso da un passo (variabile). L'algoritmo Intel/DVI ADPCM è stato selezionato per essere usato dalla IMA, così avrà buone possibilità di divenire lo standard.

state è una tupla contenente lo stato del codificatore. Il codificatore restituisce una tupla `(`*adpcmfrag*`, `*newstate*`)`, e *newstate* dovrebbe essere passato per la prossima chiamata di `lin2adpcm()`. Nella chiamata iniziale, `None` può essere passato come *state*. *adpcmfrag* è il frammento codificato ADPCM pacchettizzato con valori 2 4-bit per byte.

lin2adpcm3(*fragment, width, state*)
Questo è un codificatore alternativo ADPCM che usa solo 3 bit per campione. Non è compatibile con il codificatore Intel/DVI ADPCM e il suo output non è pacchettizzato (dovuto alla pigrezza dell'autore). Il suo uso è scoraggiato.

lin2ulaw(*fragment, width*)
Converte campioni in frammentazione audio, nella codifica u-LAW, e li restituisce come una stringa Python. u-LAW è un formato di codifica audio con il quale si può prendere un intervallo dinamico di circa 14 bit usando solo campioni ad 8 bit. Viene usato dall'hardware audio della Sun, tra gli altri.

minmax(*fragment, width*)
Restituisce una tupla consistente nei minimi a massimi valori di tutti i campioni nel frammento di suono.

max(*fragment, width*)
Restituisce il massimo del valore *assoluto* di tutti i campioni in un frammento.

maxpp(*fragment, width*)
Restituisce il massimo valore di peak-peak nel frammento di suono.

mul(*fragment, width, factor*)
Restituisce un frammento che ha tutti i campioni nel frammento originale, moltiplicati per il valore in virgola mobile di *factor*. L'overflow viene silenziosamente ignorato.

ratecv(*fragment, width, nchannels, inrate, outrate, state*[, *weightA*[, *weightB*]])
Converte il frame rate del frammento di input.

state è la tupla contenente lo stato del convertitore. Il convertitore restituisce una tupla (*newfragment*, *newstate*), e *newstate*) dovrebbe essere passato per la prossima chiamata di `ratecv()`. La chiamata iniziale dovrebbe passare `None` per *state*.

Gli argomenti *weightA* e *weightB* sono parametri per un semplice filtro digitale ed il loro valore predefinito è rispettivamente `1` e `0`.

reverse(*fragment, width*)
Inverte l'esempio in un frammento e restituisce il frammento modificato.

rms(*fragment, width*)
Restituisce la root-mean-square di un frammento, per es.

$$\sqrt{\frac{\sum {S_i}^2}{n}}$$

Questa è la misura della potenza in un segnale audio.

tomono(*fragment, width, lfactor, rfactor*)
Converte un frammento stereo in un frammento mono. Il canale di sinistra viene moltiplicato per *lfactor* ed il canale di destra per *rfactor*, prima di aggiungere i due canali per fornire un segnale mono.

tostereo(*fragment, width, lfactor, rfactor*)
Genera un frammento stereo da un frammento mono. Ogni coppia di campionature nel frammento stereo viene computata dal campione mono, laddove il canale sinistro viene moltiplicato per *lfactor* ed il canale destro per *rfactor*.

ulaw2lin(*fragment, width*)
Converte frammenti di suono. La codifica u-LAW usa sempre campioni a 8 bit, così la dimensione *width* fa riferimento solo alla dimensione del campione del frammento di output presente.

Notare che operazioni come `mul()` o `max()` non fanno distinzione tra frammenti mono e stereo, per esempio tutti i campioni sono trattati allo stesso modo. Se questo è un problema, il frammento stereo dovrebbe essere prima diviso in due frammenti mono, e dopo ricombinato. Qui c'è un esempio di come fare questo:

```
def mul_stereo(sample, width, lfactor, rfactor):
    lsample = audioop.tomono(sample, width, 1, 0)
    rsample = audioop.tomono(sample, width, 0, 1)
    lsample = audioop.mul(sample, width, lfactor)
    rsample = audioop.mul(sample, width, rfactor)
    lsample = audioop.tostereo(lsample, width, 1, 0)
    rsample = audioop.tostereo(rsample, width, 0, 1)
    return audioop.add(lsample, rsample, width)
```

Se si usa il codificatore ADPCM per costruire pacchetti di rete e volete che il vostro protocollo sia indipendente (per esempio per tollerare la perdita di pacchetti), non dovreste solamente trasmettere i dati ma anche lo stato dei dati. Notare che si potrebbe inviare lo stato iniziale, *initial*, (quello passato a `lin2adpcm()`) attraverso il decodificatore e non lo stato finale (come restituito dal codificatore). Se si vuole usare `struct.struct()` per conservare lo stato in forma binaria, si può codificare il primo elemento (il valore previsto) a 16 bit e il secondo (l'indice delta) in 8.

I codificatori ADPCM non sono mai stati confrontati con altri codificatori ADPCM, ma solamente con sé stessi. Potrebbe benissimo essere che abbia interpretato male lo standard, in questo caso non saranno interoperabili con i rispettivi standard.

Le routine `find*()` possono sembrare un po' buffe a prima vista. Sono significanti per fare da echo alle can-

cellazioni. Un modo ragionevolmente veloce per fare questo è prendere il pezzo più energetico del campione di output, posizionarlo nel campione di input e sottrarre l'intero campione di output dal campione di input:

```
def echocancel(outputdata, inputdata):
    pos = audioop.findmax(outputdata, 800)    # one tenth second
    out_test = outputdata[pos*2:]
    in_test = inputdata[pos*2:]
    ipos, factor = audioop.findfit(in_test, out_test)
    # Optional (for better cancellation):
    # factor = audioop.findfactor(in_test[ipos*2:ipos*2+len(out_test)],
    #              out_test)
    prefill = '\0'*(pos+ipos)*2
    postfill = '\0'*(len(inputdata)-len(prefill)-len(outputdata))
    outputdata = prefill + audioop.mul(outputdata,2,-factor) + postfill
    return audioop.add(inputdata, outputdata, 2)
```

14.2 `imageop` — Manipolare dati di immagini grezzi

Il modulo `imageop` contiene alcune utili operazioni sulle immagini. Questo opera su immagini costituite da 8 o 32 pixels conservate in stringhe Python. Questo è il solito formato che viene usato da `gl.lrectwrite()` e dal modulo `imgfile`.

Il modulo definisce le seguenti variabili e funzioni:

exception `error`
Questa eccezione viene rilasciata per tutti gli errori, come un numero sconosciuto di bit per pixel, etc. etc..

`crop`(*image, psize, width, height, x0, y0, x1, y1*)
Restituisce la parte selezionata dell'immagine *image*, che dovrebbe avere le dimensioni *width* per *height* (NdT: larghezza per altezza) ed essere formata dai pixel dei byte di *psize*. *x0*, *y0*, *x1* e *y1* sono come i parametri di `gl.lrectread()`, per esempio, la cornice viene inclusa nella nuova immagine. La nuova cornice ha la necessità di essere inserita dentro l'immagine. I pixel che rimangono fuori dalla vecchia immagine avranno i loro valori impostati a zero. Se *x0* è più grande di *x1*, la nuova immagine è speculare. Lo stesso vale per le coordinate y.

`scale`(*image, psize, width, height, newwidth, newheight*)
Restituisce l'immagine *image* scalata per la dimensione *newwidth* per *newheight*. Non viene fatta nessuna interpolazione, la scala è fatta per una semplice duplicazione dei pixel o loro rimozione. Di conseguenza, le immagini generate da computer o mosse, non saranno belle dopo essere state scalate.

`tovideo`(*image, psize, width, height*)
Esegue un filtro verticale passa-basso sopra una immagine. Si comporta così per computare ogni pixel di destinazione come la media delle due sorgenti verticali di pixel. Il principale uso di questa routine è anticipare l'eccessivo sfarfallio se l'immagine viene mostrata su un video che usa l'interlacciamento, da qui il nome.

`grey2mono`(*image, width, height, threshold*)
Converte un'immagine con profondità in scala di grigi a 8-bit in un'immagine con profondità a 1-bit per la soglia di tutti i pixel. L'immagine risultante è molto compressa ed è probabilmente usabile solo come argomento per `mono2grey()`.

`dither2mono`(*image, width, height*)
Converte un'immagine a 8-bit in scala di grigi in una immagine monocromatica ad 1-bit usando l'algoritmo per il dithering (NdT: amalgama di colori, sintesi di punti in colori diversi per la realizzazione di un'area a colore intermedio).

`mono2grey`(*image, width, height, p0, p1*)
Converte un'immagine monocromatica a 1-bit in una immagine a 8-bit in scala di grigi. Tutti i pixel che hanno valore zero in input prendono il valore *p0* in output e tutti i pixel che hanno valore uno in input

prendono il valore *p1* in output. Per convertire un'immagine in bianco e nero monocromatica in un scala di grigi passare rispettivamente i valori `0` e `255`.

`grey2grey4`(*image, width, height*)
Converte un'immagine a 8-bit in scala di grigi in un'immagine a 4-bit in scala di grigi senza il dithering.

`grey2grey2`(*image, width, height*)
Converte una immagine a 8-bit in scala di grigi in un'immagine a 2-bit in scala di grigi senza il dithering.

`dither2grey2`(*image, width, height*)
Converte un'immagine a 8-bit in scala di grigi in un'immagine a 2-bit in scala di grigi con il dithering. Come per `dither2mono()`, l'algoritmo di dithering è molto semplice.

`grey42grey`(*image, width, height*)
Converte un'immagine a 4-bit in scala di grigi in un'immagine a 8-bit in scala di grigi.

`grey22grey`(*image, width, height*)
Converte un'immagine a 2-bit in scala di grigi in un'immagine a 8-bit in scala di grigi.

`backward_compatible`
Se viene impostata a 0, le funzioni in questo modulo diventano non compatibili, su sistemi little-endian, per la rappresentazione di pixel multi-byte. La SGI su cui questo modulo è stato originariamente scritto era un sistema big-endian, così impostando questa variabile non vi saranno effetti. Comunque, in origine il codice fu scritto per girare su entrambi i sistemi, così furono fatte considerazioni circa il byte order su ciò che sia universale o meno. Impostando questa variabile a 0 si avrà un byte order contrario al sistema little-endian, in questo modo si otterrà un funzionamento come quello del sistema big-endian. +

14.3 `aifc` — Lettura e scrittura di file AIFF e AIFC

Questo modulo fornisce il supporto per la lettura e la scrittura dei file AIFF e AIFF-C. AIFF è l'Audio Interchange File Format, un formato per salvare campioni di audio digitale in un file. AIFFC-C è la nuova versione del formato, che include la capacità di comprimere l'audio.

Avvertenze: Alcune operazioni possono funzionare soltanto sotto IRIX; questo solleverà un'eccezione `ImportError` quando si tenterà di importare un modulo `cl`, che è disponibile solo su IRIX.

I file audio hanno un numero di parametri che descrivono i dati audio. Il tasso di campionamento o il tasso di frammentazione è il numero di volte per secondo in cui il suono viene campionato. Il numero di canali indica se l'audio è mono, stereo o quadro. Ogni frammento consiste di un campione per canale. La misura del campione è la misura in byte di ogni campione. Così un frammento consiste di *ncanali*dimensionecampione* byte, ed il secondo valore audio consiste di *ncanali*dimensionecampione*framerate* byte.

Per esempio, i CD hanno una qualità audio con una misura di campionamento di 2 byte (16 bit), usano due canali (stereo) ed hanno un frame rate di 44.100 frames/per secondo. Questo dà una dimensione del frame di 4 byte (2*2) ed il valore dei secondi occupa 2*2*44100 byte (176.400 byte).

Il modulo `aifc` definisce la seguente funzione:

`open`(*file*[, *mode*])
Apre un file AIFF o AIFF-C e restituisce un oggetto istanza con i metodi che sono descritti qui sotto. L'argomento *file* è sia un file stringa che un file oggetto. *mode* deve essere `'r'` o `'rb'` quando il file deve essere aperto per la lettura, o `'w'`, `'wb'` quando il file deve essere aperto per la scrittura. Se omesso, viene usato *file*`.mode` se esiste, altrimenti viene usato `'rb'`. Quando usato per la scrittura, il file oggetto dovrebbe essere disponibile per le ricerche, a meno che non si conosca quanti campioni si sta andando a scrivere in totale ed usare `writeframesraw()` e `setnframes()`.

Gli oggetti restituiti da `open()` quando un file viene aperto per la lettura hanno i seguenti metodi:

`getnchannels`()
Restituisce il numero dei canali audio (1 per mono, 2 per stereo).

`getsampwidth`()
Restituisce la misura in byte del campione individuale.

`getframerate``()`
Restituisce la frequenza di campionatura (il numero dei frammenti audio per secondo).

`getnframes``()`
Restituisce il numero dei frammenti audio nel file.

`getcomptype``()`
Restituisce una stringa di quattro caratteri che descrive il tipo di compressione usata nel file audio. Per i file AIFF, il valore restituito è `'NONE'`.

`getcompname``()`
Restituisce una descrizione comprensibile del tipo di compressione usata nel file audio. Per i file AIFF, il valore restituito è `'not compressed'`.

`getparams``()`
Restituisce una tupla che consiste in tutti i suddetti valori nello stesso ordine.

`getmarkers``()`
Restituisce una lista di marcatori del file audio. Un marcatore è composto da una tupla di tre elementi. Il primo viene indicato come ID (un intero), il secondo è la posizione indicata nel frammento dall'inizio dei dati (un intero), il terzo è il nome dell'indicatore (stringa).

`getmark``(`*id*`)`
Restituisce una tupla come descritto in `getmarkers()` per il marcatore con l'*id* passato.

`readframes``(`*nframes*`)`
Legge il prossimo frammento *nframes* restituito dal file audio. Il dato restituito è una stringa contenente i campioni non compressi di tutti i canali.

`rewind``()`
Torna all'inizio del punto letto. Il prossimo `readframes()` partirà dall'inizio.

`setpos``(`*pos*`)`
Cerca il frammento con il numero specificato.

`tell``()`
Restituisce il numero del frammento corrente

`close``()`
Chiude il file AIFF. Dopo avere chiamato questo metodo, l'oggetto non può essere più usato.

Gli oggetti restituiti da `open()` quando un file viene aperto in scrittura, hanno tutti gli stessi metodi, con l'eccezione di `readframes()` e `setpos()`. In aggiunta esistono altri metodi. I metodi `get*()` possono essere chiamati solo dopo che i metodi corrispondenti `set*()` vengono chiamati. Prima del primo `writeframes()` o `writeframesraw()`, tutti i parametri, eccetto il numero dei frammenti, devono essere riempiti.

`aiff``()`
Crea un file AIFF. In modo predefinito viene creato un file AIFF-C, anche se il nome del file non termina con l'estensione `'.aiff'`, viene comunque creato un file AIFF.

`aifc``()`
Crea un file AIFF-C. In modo predefinito viene creato un file AIFF-C, anche se il nome del file non termina con l'estensione `'.aiff'`, viene comunque creato un file AIFF.

`setnchannels``(`*nchannels*`)`
Specifica il numero dei canali nel file audio.

`setsampwidth``(`*width*`)`
Specifica la misura in byte del campione audio.

`setframerate``(`*rate*`)`
Specifica la frequenza di campionamento in frammenti per secondo.

`setnframes``(`*nframes*`)`
Specifica il numero di frammenti che devono essere scritti nel file audio. Se questo parametro non viene impostato, o viene impostato non correttamente, il file ha bisogno del supporto per la ricerca.

`setcomptype``(`*type, name*`)`

Specifica il tipo di compressione. Se non specificato, i dati audio non saranno compressi. Nei file AIFF, la compressione non è possibile. Il parametro *name* dovrebbe essere una descrizione comprensibile del tipo di compressione, il parametro *type* dovrebbe essere una stringa composta da quattro caratteri. Correntemente vengono supportate le seguenti tipologie di compressione: NONE, ULAW, ALAW, G722.

`setparams`(*nchannels, sampwidth, framerate, comptype, compname*)
Imposta tutti i parametri in una volta. L'argomento è una tupla consistente nei vari parametri. Questo significa che è possibile usare il risultato di `getparams()` chiamato come argomento di `setparams()`.

`setmark`(*id, pos, name*)
Aggiunge un marcatore con l'id passato (più grande di 0), il nome *name* passato e la posizione *pos* passata. Questo metodo può essere chiamato in qualsiasi momento prima di `close()`.

`tell`()
Restituisce la corrente posizione di scrittura nel file di output. Utile in combinazione con `setmark()`.

`writeframes`(*data*)
Scrive i dati nel file di output. Questo metodo può essere chiamato solo dopo che i parametri del file audio siano stati impostati.

`writeframesraw`(*data*)
Come `writeframes()`, eccetto che l'header del file audio non è aggiornato.

`close`()
Chiude il file AIFF. L'intestazione del file viene aggiornata e riflette la dimensione attuale dei dati audio. Dopo aver chiamato questo metodo, l'oggetto non può più essere usato.

14.4 sunau — Lettura e scrittura di file AU

Il modulo `sunau` fornisce un'interfaccia utile per il formato AU di Sun. Notare che questo modulo è compatibile per interfacciarsi con i moduli `aifc` e `wave`.

Un file audio consiste di un'intestazione seguita da dati. I campi dell'intestazione sono:

Campo	Contenuto
magic word	I 4 byte '`.snd`'.
header size	La dimensione dell'intestazione, incluse le informazioni, in byte.
data size	Dimensione fisica dei dati, in byte.
encoding	Indica come i campioni audio vengono codificati.
sample rate	Il tasso di campionatura.
# of channels	Il numero di canali nel campione.
info	Stringa ASCII che descrive il file audio (riempita con byte nulli).

Separata dall'informazione sul campo, tutti i campi intestazione hanno 4 byte di dimensione. Sono tutti interi a 32-bit non firmati codificati in ordine big-endian.

Il modulo `sunau` definisce le seguenti funzioni:

`open`(*file, mode*)
Se il file *file* è una stringa, apre il file con quel nome, altrimenti lo tratta come un oggetto ricercabile simile a file. *mode* può essere uno tra

`'r'` Modalità in sola lettura.

`'w'` Modalità in sola scrittura.

Notare che non sono permessi file in lettura/scrittura.

Una modalità `'r'` restituisce un oggetto `AU_read`, mentre una modalità `'w'` o `'wb'` restituisce un oggetto `AU_write`.

`openfp`(*file, mode*)
Un sinonimo per `open`, mantenuto per retrocompatibilità.

Il modulo `sunau` definisce le seguenti eccezioni:

exception `Error`
Un errore rilasciato quando qualcosa è impossibile perché il modulo Sun AU è deficitario delle specifiche o dell'implementazione.

Il modulo `sunau` definisce i seguenti elementi dato:

`AUDIO_FILE_MAGIC`
Un intero per ciascun file Sun AU valido, inizia con l'immagazzinamento nella forma big-endian. Questa è la stringa '`.snd`' interpretata come un intero.

`AUDIO_FILE_ENCODING_MULAW_8`
`AUDIO_FILE_ENCODING_LINEAR_8`
`AUDIO_FILE_ENCODING_LINEAR_16`
`AUDIO_FILE_ENCODING_LINEAR_24`
`AUDIO_FILE_ENCODING_LINEAR_32`
`AUDIO_FILE_ENCODING_ALAW_8`
Valori del campo codifica dell'intestazione AU supportati da questo modulo.

`AUDIO_FILE_ENCODING_FLOAT`
`AUDIO_FILE_ENCODING_DOUBLE`
`AUDIO_FILE_ENCODING_ADPCM_G721`
`AUDIO_FILE_ENCODING_ADPCM_G722`
`AUDIO_FILE_ENCODING_ADPCM_G723_3`
`AUDIO_FILE_ENCODING_ADPCM_G723_5`
Valori aggiuntivi conosciuti del campo codifica dell'intestazione AU, ma non supportati da questo modulo.

14.4.1 Oggetti AU_read

Gli oggetti AU_read vengono restituiti da `open()` come sopra, hanno i seguenti metodi:

`close``()`
Chiude il flusso audio e rende l'istanza inutilizzabile. Fatto automaticamente sulla cancellazione.

`getnchannels``()`
Restituisce il numero dei canali audio (1 per mono, 2 per stereo)

`getsampwidth``()`
Restituisce l'ampiezza del campione in byte.

`getframerate``()`
Restituisce la frequenza di campionamento.

`getnframes``()`
Restituisce il numero dei frammenti audio.

`getcomptype``()`
Restituisce il tipo di compressione. Le compressioni supportate sono `'ULAW'`, `'ALAW'` e `'NONE'`.

`getcompname``()`
Versione umanamente comprensibile di `getcomptype()`. I tipi supportati hanno i rispettivi nomi `'CCITT G.711 u-law'`, `'CCITT G.711 A-law'` e `'not compressed'`.

`getparams``()`
Restituisce una tupla (*nchannels*, *sampwidth*, *framerate*, *nframes*, *comptype*, *compname*), equivalente all'output del metodo `get*()`.

`readframes``(`*n*`)`
Legge e restituisce al massimo *n* frame di audio, nella forma di una stringa di byte. I dati verranno restituiti come un formato lineare. Se i dati originali erano nel formato u-LAW, verranno riconvertiti.

`rewind``()`
Riporta il puntatore del file all'inizio del flusso audio.

I seguenti due metodi definiscono un termine “posizione (NdT: pos)” che è compatibile tra loro, ed è altrimenti dipendente dall’implementazione.

setpos(*pos*)
Imposta il puntatore del file alla posizione specificata. Solo i valori restituiti da `tell()` dovrebbero essere usati per *pos*.

tell()
Restituisce la posizione corrente del puntatore del file. Notare che il valore restituito non ha niente a che fare con l’attuale posizione nel file.

Le seguenti due funzioni vengono definite per compatibilità con `aifc` e non fanno niente di interessante.

getmarkers()
Restituisce `None`.

getmark(*id*)
Solleva un’eccezione per un errore.

14.4.2 Oggetti AU_write

Gli oggetti AU_write, vengono restituiti per `open()`, come descritto sopra, hanno i seguenti metodi:

setnchannels(*n*)
Imposta il numero dei canali.

setsampwidth(*n*)
Imposta la grandezza dei campioni (in byte).

setframerate(*n*)
Imposta il frame rate.

setnframes(*n*)
Imposta il numero dei frame. Questo può essere cambiato successivamente, quando e se nuovi frame vengono aggiunti.

setcomptype(*type*, *name*)
Imposta il tipo di compressione e la descrizione. In output vengono supportati solamente `'NONE'` e `'ULAW'`.

setparams(*tuple*)
La *tupla* dovrebbe essere (*nchannels*, *sampwidth*, *framerate*, *nframes*, *comptype*, *compname*), con valori validi per i metodi `set*()`. Imposta tutti i parametri.

tell()
Restituisce la posizione corrente nel file, con le stesse avvertenze dei metodi `AU_read.tell()` e `AU_read.setpos()`.

writeframesraw(*data*)
Scrive frammenti audio, senza correggere *nframes*.

writeframes(*data*)
Scrive frammenti audio e si assicura che *nframes* sia corretto.

close()
Si assicura che *nframes* sia corretto e chiude il file.

Questo metodo viene chiamato subito dopo la cancellazione.

Notare che non è valido impostare qualsiasi parametro dopo avere effettuato le chiamate `writeframes()` o `writeframesraw()`.

14.5 `wave` — Leggere e scrivere file WAV

Il modulo `wave` fornisce un'interfaccia utile per il formato audio WAV. Non supporta la compressione/decompressione, ma supporta il mono/stereo.

Il modulo `wave` definisce le seguenti funzioni ed eccezioni:

`open`(*file*[, *mode*])
Se *file* è una stringa, apre il file con quel nome, un'altro trattamento è un oggetto ricercabile di tipo simile a file. Le modalità *mode* possono essere una fra queste elencate:

`'r'`, `'rb'` Modalità in sola lettura.

`'w'`, `'wb'` Modalità in sola scrittura.

Notare che non permette la lettura/scrittura di file WAV.

Un *mode* `'r'` o `'rb'` restituisce un oggetto `Wave_read`, quando un *mode* `'w'` o `'wb'` restituisce un oggetto `Wave_write`. Se *mode* viene omesso ed un oggetto simile a file viene passato come *file*, *file*`.mode` viene usato come valore predefinito per *mode* (l'opzione '`b`' viene aggiunta se necessario).

`openfp`(*file, mode*)
Un sinonimo per `open()`, mantenuto per retrocompatibilità.

exception `Error`
Un'eccezione sollevata quando è impossibile fare qualcosa perché viola la specifiche WAV o riguarda una mancanza di implementazione.

14.5.1 Oggetti Wave_read

Gli oggetti Wave_read, vengono restituiti da `open()`, hanno i seguenti metodi:

`close`()
Chiude il flusso, e rende l'istanza inutilizzabile. Questo metodo viene chiamato in automatico sull'istanza collection.

`getnchannels`()
Restituisce il numero dei canali audio (`1` per mono, `2` per stereo).

`getsampwidth`()
Restituisce la lunghezza del campione in byte.

`getframerate`()
Restituisce la frequenza di campionamento.

`getnframes`()
Restituisce il numero dei frammenti audio.

`getcomptype`()
Restituisce il tipo di compressione (`'NONE'` è il solo tipo supportato).

`getcompname`()
Versione umanamente comprensibile di `getcomptype()`. Di solito `'not compressed'` è parallelo a `'NONE'`.

`getparams`()
Restituisce una tupla (*nchannels*, *sampwidth*, *framerate*, *nframes*, *comptype*, *compname*), equivalente all'output dei metodi `get*()`.

`readframes`(*n*)
Legge e restituisce al massimo *n* frammenti di audio, come una stringa di byte.

`rewind`()
Riporta indietro il puntatore del file all'inizio del flusso audio.

I due metodi seguenti vengono definiti per compatibilità con il modulo `aifc`, e non fanno niente di interessante.

`getmarkers`()
Restituisce `None`.

getmark(*id*)
Genera un errore.

I due metodi seguenti definiscono il concetto “position” (NdT: posizione) che è compatibile tra essi, ed è al contrario dipendente dall’implementazione.

setpos(*pos*)
Imposta il puntatore del file alla posizione specificata.

tell()
Restituisce la posizione corrente del puntatore del file.

14.5.2 Oggetti Wave_write

Gli oggetti Wave_write, come restituiti da `open()`, hanno i metodi seguenti:

close()
Si assicura che *nframes* sia corretto e chiude il file. Questo metodo viene chiamato dopo la cancellazione.

setnchannels(*n*)
Imposta il numero dei canali.

setsampwidth(*n*)
Imposta la dimensione del campione a *n* byte.

setframerate(*n*)
Imposta il frame rate a *n*.

setnframes(*n*)
Imposta il numero dei frame a *n*. Se nuovi frammenti audio venissero aggiunti, allora *n* verrebbe modificato.

setcomptype(*type, name*)
Imposta il tipo di compressione e la descrizione.

setparams(*tuple*)
La tupla, *tuple*, dovrebbe essere (*nchannels*, *sampwidth*, *framerate*, *nframes*, *comptype*, *compname*), con valori validi per i metodi `set*()`. Imposta tutti i parametri.

tell()
Restituisce la posizione corrente nel file, con le stesse avvertenze per i metodi `Wave_read.tell()` e `Wave_read.setpos()`.

writeframesraw(*data*)
Scrive frammenti audio senza la correzione *nframes*.

writeframes(*data*)
Scrive frammenti audio e si assicura che *nframes* sia corretto.

Notare che non è corretta l’impostazione di qualsiasi parametro dopo avere chiamato `writeframes()` o `writeframesraw()`, ed ogni tentativo solleverà un’eccezione `wave.Error`.

14.6 chunk — Leggere spezzoni di dati IFF

Questo modulo fornisce un’interfaccia per leggere i file che usano spezzoni.[1] Questo formato viene usato nei file con formato Audio Interchange File Format (AIFF/AIFF-C) e nel formato Real Media File Format (RMFF). Il formato audio WAVE è strettamente correlato e si può leggere usando questo modulo.

Uno spezzone ha la seguente struttura:

[1] “EA IFF 85” Standard for Interchange Format Files, di Jerry Morrison, Electronic Arts, January 1985.

Offset	Lunghezza	Contenuto
0	4	Chunk ID
4	4	Dimensione dello spezzone nell'ordine big-endian, non include l'header
8	*n*	Dati in bytes, dove *n* è la dimensione fornita nel precedente campo
8 + *n*	0 o 1	Blocco di byte necessario se *n* è dispari e viene usato l'allineamento degli spezzoni

L'ID è una stringa di 4 byte che identifica il tipo di spezzone.

La dimensione del campo (un valore a 32 bit, codificato usando big-endian) fornisce la dimensione dei dati dello spezzone, esclusi gli 8 byte dell'intestazione.

Di solito un file di tipo IFF si compone di uno o più chunk. L'uso proposto della classe `Chunk` qui definita è istanziare un'istanza all'avvio di ogni spezzone e leggere dall'istanza prima che raggiunga la fine, fatto questo, può essere creata una nuova istanza. Giunti alla fine del file, la creazione di una nuova istanza fallirà, sollevando l'eccezione `EOFError`.

class Chunk(*file*[, *align, bigendian, inclheader*])

Classe che rappresenta uno spezzone. L'argomento *file* viene atteso come un oggetto simile a file. Un'istanza di questa classe è specificatamente permessa. Il solo metodo richiesto è `read()`. Se i metodi `seek()` e `tell()` sono presenti e non sollevano eccezioni, allora significa che si aspettano di non dovere alterare l'oggetto. Se l'argomento facoltativo *align* è vero, si presuppone che gli spezzoni siano allineati sul limite dei 2 byte. Se *align* è falso, non si presuppone alcun allineamento. Il valore predefinito è vero. Se l'argomento facoltativo *bigendian* è falso, si assume che la dimensione dello spezzone sia ordinata in little-endian. Questo è necessario per i file audio WAVE. Il valore predefinito è vero. Se l'argomento facoltativo *inclheader* è vero, la dimensione fornita nell'intestazione dello spezzone include la misura dell'intestazione. Il valore predefinito è falso.

Un oggetto `Chunk` supporta i seguenti metodi:

getname()

Restituisce il nome (ID) dello spezzone. Questo è il primo dei 4 byte dello spezzone.

getsize()

Restituisce la dimensione dello spezzone.

close()

Chiude e passa alla fine dello spezzone. Questo non chiude i file sottostanti.

I restanti metodi sollevano un'eccezione `IOError` se vengono chiamati dopo aver chiamato il metodo `close()`.

isatty()

Restituisce `False`.

seek(*pos*[, *whence*])

Imposta la posizione corrente dello spezzone. L'argomento *whence* è facoltativo ed ha come valore predefinito lo `0` (posizionamento assoluto nel file); altri valori sono `1` (cerca relativamente alla posizione corrente) e `2` (cerca relativamente alla fine del file). Non viene restituito alcun valore. Se il file sottostante non permette la ricerca con seek, è permessa la sola ricerca di dati successivi al posizionamento nel file.

tell()

Restituisce la corrente posizione nello spezzone.

read([*size*])

Legge la maggiore parte della dimensione, *size*, dello spezzone (meno se la lettura raggiunge la fine dello spezzone prima di ottenerne la dimensione *size* in byte). Se l'argomento *size* è negativo o viene omesso, legge tutti i dati prima della fine dello spezzone. I byte vengono restituiti come un oggetto stringa. Viene restituita una stringa vuota quando viene raggiunta la fine dello spezzone.

skip()

Salta alla fine dello spezzone. Tutte le chiamate a `read()` per lo spezzone restituiranno `''`. Se non si è interessati al contenuto dello spezzone, questo metodo dovrebbe essere chiamato in modo da far puntare il file all'inizio del prossimo spezzone.

14.7 `colorsys` — Conversioni tra colori di sistema

Il modulo `colorsys` definisce le conversioni bidirezionali di valori di colore tra quelli espressi in RGB (Red, Green, Blue) usati nei monitor dei computer e tre altre coordinate di sistema: YIQ, HLS (Hue Lightness Saturation - Tonalità di luce) e HSV (Hue Saturation Value - Saturazione di luce). Le coordinate in questi spazi di colore sono valori numerici in virgola mobile. Nello spazio YIQ, la coordinata Y è tra 0 e 1, ma le coordinate I e Q possono essere positive o negative. In tutti gli altri spazi, le coordinate sono tutte tra 0 e 1.

Maggiori informazioni sugli spazi colore possono essere trovate presso http://www.poynton.com/ColorFAQ.html.

Il modulo `colorsys` definisce le seguenti funzioni:

`rgb_to_yiq`(*r, g, b*)
Converte il colore dalle coordinate RGB a quelle YIQ.

`yiq_to_rgb`(*y, i, q*)
Converte il colore dalle coordinate YIQ a quelle RGB.

`rgb_to_hls`(*r, g, b*)
Converte i colori dalle coordinate RGB alle coordinate HLS.

`hls_to_rgb`(*h, l, s*)
Converte il colore dalle coordinate HLS alle coordinate RGB.

`rgb_to_hsv`(*r, g, b*)
Converte i colori dalle coordinate RGB alle coordinate HSV

`hsv_to_rgb`(*h, s, v*)
Converte i colori dalle coordinate HSV alle coordinate RGB.

Esempio:

```
>>> import colorsys
>>> colorsys.rgb_to_hsv(.3, .4, .2)
(0.25, 0.5, 0.4)
>>> colorsys.hsv_to_rgb(0.25, 0.5, 0.4)
(0.3, 0.4, 0.2)
```

14.8 `rgbimg` — Leggere e scrivere file “SGI RGB”

Il modulo `rgbimg` permette ai programmi Python l’accesso a file di immagini SGI imglib (conosciuti anche come file ‘.rgb’). Il modulo è lontano dall’essere completo, ma fornisce ad ogni modo funzionalità spesso sufficienti in alcuni casi. I file colormap non sono al momento supportati.

Note: Questo modulo è costruito solamente per piattaforme a 32-bit; non ci si aspetta che funzioni correttamente su altri sistemi.

Il modulo definisce le seguenti variabili e funzioni:

exception `error`
Questa eccezione viene sollevata su tutti gli errori, come file di tipo non supportati, etc. etc..

`sizeofimage`(*file*)
Questa funzione restituisce una tupla `(`*x*`, `*y*`)` dove *x* e *y* sono la misura dell’immagine in pixel. Solo pixel RGBA a 4 byte, pixel RGB a 3 byte e scale di grigio ad 1 byte vengono correntemente supportate.

`longimagedata`(*file*)
Questa funzione legge e decodifica l’immagine del file specificato, e lo restituisce come una stringa Python. La stringa ha pixel RGBA a 4 byte. Il pixel nella parte inferiore di sinistra nella stringa è il primo. Questo formato è adatto per essere passato a `gl.lrectwrite()`, per istanza.

`longstoimage`(*data, x, y, z, file*)

Questa funzione scrive i dati RGBA *data* in un file, *file*, di un'immagine. *x* ed *y* forniscono la dimensione dell'immagine. *z* vale 1 se l'immagine salvata dovrebbe essere in scala di grigi ad 1 byte, 3 se l'immagine salvata dovrebbe essere un dato in RGB a 3 byte, o 4 se l'immagine salvata dovrebbe essere un dato in RGBA a 4 byte. I dati di input devono sempre contenere 4 byte per pixel. Questi sono i formati restituiti da `gl.lrectread()`.

`ttob`(*flag*)
Questa funzione imposta un flag globale che definisce quando le righe di ricerca dell'immagine vengono lette o scritte dal basso verso l'alto (l'opzione è zero, compatibile con SGI GL) o dall'alto verso il basso (opzione uno, compatibile con X). Per definizione è zero.

14.9 `imghdr` — Determina il tipo di immagine

Il modulo `imghdr` determina il tipo di immagine contenuto in un file o in un flusso di dati.

Il modulo `imghdr` definisce le seguenti funzioni:

`what`(*filename*[, *h*])
Verifica i dati di un'immagine contenuta nel file chiamato *filename* e restituisce una stringa che descrive il tipo di immagine. Se viene fornita l'opzione *h*, *filename* viene ignorato e si assume che *h* contenga il flusso di byte da verificare.

I seguenti tipi di immagini vengono riconosciuti, come i valori elencati qui sotto con il valore restituito da `what()`:

Valore	Formato dell'immagine
`'rgb'`	FIle SGI ImgLib
`'gif'`	FIle GIF 87a e 89a
`'pbm'`	File Bitmap Portabile
`'pgm'`	File Graymap Portabile
`'ppm'`	File Pixmap Portabile
`'tiff'`	File TIFF
`'rast'`	File Raster Sun
`'xbm'`	File X Bitmap
`'jpeg'`	Dati JPEG nel formato JFIF
`'bmp'`	File BMP
`'png'`	Portable Network Graphics

Si può estendere la lista dei file che il tipo `imghdr` può riconoscere per aggiungerli a questa variabile:

`tests`
Una lista di funzioni che effettua dei test individuali. Ogni funzione prende due argomenti: il flusso di byte e un oggetto simile a file aperto. Quando `what()` viene chiamata con un flusso di byte, l'oggetto simile a file avrà il valore `None`.

La funzione test dovrebbe restituire una stringa che descrive il tipo di immagine se il test ha successo, o `None` se fallisce.

Esempio:

```
>>> import imghdr
>>> imghdr.what('/tmp/bass.gif')
'gif'
```

14.10 `sndhdr` — Determina il tipo del file sonoro

Il modulo `sndhdr` fornisce funzioni di utilità che tentano di determinare il tipo dei dati sonori che sono in un file. Quando queste funzioni sono capaci di determinare quale tipo di dati del suono sono immagazzinati in un file, restituiscono una tupla (*type*, *frequenza_di_campionamento*, *canali*, *frames*, *bits_per_campione*). Il valore di *type* indica il tipo di dati che sarà una di queste stringhe `'aifc'`, `'aiff'`, `'au'`, `'hcom'`, `'sndr'`, `'sndt'`, `'voc'`, `'wav'`, `'8svx'`, `'sb'`, `'ub'` o `'ul'`. *sampling_rate* sarà uno dei valori attuali o `0` se sconosciuto o difficile da decodificare. In modo simile, *canali* sarà uno dei numeri dei canali o `0` se non potrà essere determinato o se il valore è difficilmente decodificabile. Il valore per *frames* sarà il numero dei frames o `-1`. L'ultimo elemento nella tupla, *bits_per_sample* potrebbe essere la misura del campione in bit o `'A'` per A-LAW o `'U'` per u-LAW.

`what`(*filename*)
: Determina il tipo di dati audio immagazzinati nel file *filename* usato in `whathdr()`. Se questo riesce, restituisce una tupla come descritto sopra, altrimenti viene restituito `None`.

`whathdr`(*filename*)
: Determina il tipo di dati audio immagazzinati in un file basato sul file d'intestazione. Il nome del file viene assegnato da *filename*. Questa funzione restituisce una tupla come descritto sopra, se ha successo, oppure `None`.

14.11 `ossaudiodev` — Accesso ai dispositivi audio OSS-compatibili

Nuovo nella versione 2.3.

Questo modulo permette di accedere all'interfaccia OSS (Open Sound System). OSS è disponibile per una vasta gamma di prodotti open source e commerciali derivati da UNIX e l'interfaccia audio standard per Linux e le versioni recenti di FreeBSD.

Vedete anche:

Open Sound System Programmer's Guide
(http://www.opensound.com/pguide/oss.pdf)
: la documentazione ufficiale per le API C OSS

Il modulo definisce un largo numero di costanti supportate del driver dispositivo di periferica OSS, vedere `<sys/soundcard.h>` su ogni Linux o FreeBSD per un elenco.

`ossaudiodev` definisce le seguenti variabili e funzioni:

exception `OSSAudioError`
: Questa eccezione viene sollevata per errori certi. L'argomento è una stringa che descrive ciò che è andato storto.

  Se `ossaudiodev` riceve un errore da una chiamata di sistema come `open()`, `write()`, o `ioctl()`, solleva un'eccezione `IOError`. Gli errori rilevati direttamente da `ossaudiodev` risultano in `OSSAudioError`.

  Per retrocompatibilitè , la classe dell'eccezione è anche disponibile come `ossaudiodev.error`.

`open`([*device,*]*mode*)
: Apre una periferica audio e restituisce un oggetto dispositivo OSS audio. Questo oggetto supporta molti metodi simile a file, come `read()`, `write()` e `fileno()` (anche se ci sono differenze sottili fra la semantica nella lettura/scrittura convenzionale di UNIXe quelle dei dispositivi audio OSS). Supporta anche un numero di metodi specifici per l'audio; vedere sotto per la lista completa dei metodi.

  device è il nome della periferica audio da usare. Se il nome del file non viene specificato, questo modulo cerca prima nella variabile di sviluppo AUDIODEV per il dispositivo da usare. Se non la trova, ritorna a '/dev/dsp'.

  mode indica `'r'` per l'accesso in sola lettura (record), `'w'` per l'accesso in sola scrittura (riproduzione) ed `'rw'` per entrambi. Da quando molte schede audio permettono ad un solo processo di avere il registratore o il riproduttore aperti per volta, è una buona idea aprire il dispositivo solo per l'attività richiesta. Ulteriormente alcune schede audio sono mezze duplex: possono essere aperte in lettura e scrittura ma non insieme.

  Notare la sintassi della chiamata inusuale: il *primo* argomento è facoltativo ed il secondo viene richiesto.

Questo è uno storico artefatto per conservare la compatibilità con il vecchio modulo `linuxaudiodev` che `ossaudiodev` sostituisce.

`openmixer`([*device*])
Apre una periferica mixer e restituisce un dispositivo oggetto OSS. *device* è il nome del dispositivo mixer da usare. Se non viene specificata, questo modulo cerca prima nella variabile di sviluppo MIXERDEV per la periferica da usare. Se non la trova, ritorna a '/dev/mixer'.

14.11.1 Oggetti dispositivi audio

Prima che si possa scrivere o leggere da un dispositivo audio, si devono chiamare tre metodi nel corretto ordine:

1. `setfmt()` per impostare il formato di uscita
2. `channels()` per impostare il numero dei canali
3. `speed()` per impostare la frequenza di campionamento

Alternativamente, si può usare il metodo `setparameters()` per impostare tutti e tre i parametri in una volta. Questo è più conveniente, ma potrebbe non essere così flessibile in tutti i casi.

L'oggetto dispositivo audio restituito da `open()` definisce i seguenti metodi:

`close`()
Chiude esplicitamente il dispositivo audio. Quando si sta scrivendo sopra o leggendo da una periferica audio, si dovrebbe esplicitamente chiuderla. Una periferica chiusa non può essere nuovamente usata.

`fileno`()
Restituisce il descrittore di file associato al dispositivo.

`read`(*size*)
Legge la dimensione, *size*, in byte dall'input audio e lo restituisce come una stringa Python. Diversamente da molti driver di dispositivi UNIX, i dispositivi audio OSS in modalità bloccante (predefinita) bloccheranno `read()` finché l'intero ammontare di dati richiesto sia disponibile.

`write`(*data*)
Scrive la stringa Python *data* nel dispositivo audio e restituisce il numero dei byte scritti. Se la periferica audio è in modalità bloccante (predefinità), l'intera stringa viene sempre scritta (ancora, questo è differente dalla solita semantica dei dispositivi UNIX). Se il dispositivo è in modalità non bloccante, alcuni dati potrebbero non essere scritti (vedere `writeall()`).

`writeall`(*data*)
Scrive l'intera stringa Python *data* nel dispositivo audio: aspetta finché la periferica audio non sia in grado di accettare dati e ripete finché i dati *data* non verranno interamente scritti. Se il dispositivo è in modalità bloccante (predefinita), questo ha lo stesso effetto di `write()`; `writeall()` è utile soltanto in modalità non bloccante. Non vengono restituiti valori poiché la quantità di dati scritti è sempre uguale a quella dei dati letti.

I seguenti metodi mappano ognuno una chiamata di sistema `ioctl()`. La corrispondenza è evidente: per esempio, `setfmt()` corrisponde al `SNDCTL_DSP_SETFMT` ioctl e `sync()` per `SNDCTL_DSP_SYNC` (questo può essere utile quando si consulta la documentazione OSS). Se la sottostante `ioctl()` fallisce, verranno sollevate tutte eccezioni `IOError`.

`nonblock`()
Pone la periferica in modalità non bloccante. Una volta in modo non bloccante, non c'è modo di farla tornare in modo bloccante.

`getfmts`()
Restituisce una maschera dei bit dei formati audio supportati dalla scheda sonora. Su un tpico sistema Linux, questi formati sono:

Formato	Descrizione
AFMT_MU_LAW	un logaritmo di codifica (usato dai file .au Sun e /dev/audio)
AFMT_A_LAW	un logaritmo di codifica
AFMT_IMA_ADPCM	Un formato di compressione 4:1 definito da Interactive Multimedia Association
AFMT_U8	Non firmato, audio ad 8-bit
AFMT_S16_LE	Non firmato, audio a 16-bit, con ordine little-endian (come usato dai processori Intel)
AFMT_S16_BE	Non firmato, audio a 16-bit audio, con ordine big-endian (come usato dai processori 68k, PowerPC,
AFMT_S8	Non firmato, audio ad 8 bit
AFMT_U16_LE	Non firmato, audio little-endian a 16-bit
AFMT_U16_BE	Non firmato, audio big-endian a 16-bit

Molti sistemi supportano solamente un sotto insieme di questi formati. Numerosi dispositivi supportano solamente AFMT_U8; il formato oggi più comunemente usato è il AFMT_S16_LE.

setfmt(*format*)
Tenta di impostare il corrente formato audio a *format* (vedere getfmts() per una lista). Restituisce il formato audio per il quale il dispositivo era stata impostata, che potrebbe non essere il formato richiesto. Potrebbe anche essere usato per restituire il corrente formato audio—si otterrà questo passando un "audio format" di AFMT_QUERY.

channels(*nchannels*)
Imposta il numero dei canali di output a *nchannels*. Un valore di 1 indica suono mono, 2 stereo. Alcuni dispositivi possono avere più di due canali ed alcune periferiche hig-end potrebbero non supportare il mono. Restituisce il numero dei canali del dispositivo che erano stati impostati.

speed(*samplerate*)
Tenta di impostare il campionamento audio alla frequenza di campionamento *samplerate*, di campioni per secondo. Restituisce la frequenza attualmente impostata. La maggior parte dei dispositivi audio non supporta campionamenti di frequenza arbitrari. I più comuni campionamenti sono:

Frequenza	Descrizione
8000	frequenza predefinita per /dev/audio
11025	registrazione vocale
22050	
44100	Qualità audio CD (a 16 bits/campione e 2 canali)
96000	Qualità audio DVD (a 24 bits/campione)

sync()
Attende finché il dispositivo audio abbia riprodotto ogni byte nel suo buffer. Questo succede implicitamente quando la periferica è chiusa. La documentazione OSS raccomanda la chiusura e la riapertura del dispositivo piuttosto che l'uso di sync().

reset()
Interrompe immediatamente la riproduzione e la registrazione, e riporta la periferica ad uno stato che può accettare comandi. La documentazione OSS raccomanda la chiusura e la riapertura del dispositivo dopo avere chiamato reset().

post()
Dice al driver che c'è qualcosa di simile ad una pausa nell'output, rendendo possibile per il dispositivo la gestione della pausa in modo più intelligente. E' possibile perciò usarla dopo avere riprodotto un effetto sonoro, prima di aspettare l'input dell'utente o prima di accedere all'I/O del disco.

I seguenti metodi sono convenienti e combinano vari ioctls, o un ioctl ed alcuni semplici calcoli.

setparameters(*format, nchannels, samplerate* [, *strict=False*])
Imposta i parametri della chiave audio di campionatura: il formato dei campioni, il numero dei canali e la frequenza di campionamento in una chiamata. *format*, *nchannels* e *samplerate* dovrebbero essere specificati nei metodi setfmt(), channels() e speed(). Se *strict* è vero, setparameters() cerca di vedere se ogni parametro era attualmente impostato al valore richiesto, e solleva un'eccezione OSSAudioError se così non è. Restituisce una tupla (*format*, *nchannels*, *samplerate*) che indica i parametri dei valori che venivano attualmente impostati dal driver del dispositivo (ad esempio, la stessa cosa dei valori restituiti da setfmt(), channels() e speed()).

Per esempio,

```
(fmt, channels, rate) = dsp.setparameters(fmt, channels, rate)
```

è equivalente a

```
fmt = dsp.setfmt(fmt)
channels = dsp.channels(channels)
rate = dsp.rate(channels)
```

bufsize()
Restituisce la grandezza del buffer hardware, in campioni.

obufcount()
Restituisce il numero di campioni che sono nel buffer hardware che sono ancora da riprodurre.

obuffree()
Restituisce il numero di campioni presenti nel buffer hardware e che sono ancora da riprodurre.

14.11.2 Oggetti dispositivo Mixer

L'oggetto mixer fornisce due metodi simile a file:

close()
Questo metodo chiude il file del dispositivo mixer aperto. Ogni altro tentativo di usare il mixer dopo che questo file è stato chiuso solleverà un'eccezione `IOError`.

fileno()
Restituisce il numero della gestione del file del dispositivo mixer aperto.

I restanti metodi sono specifici per il mixing audio:

controls()
Questo metodo restituisce una maschera di bit che specifica i controlli mixer disponibili ("Control" diventa uno specifico "channel" mixabile, come `SOUND_MIXER_PCM` o `SOUND_MIXER_SYNTH`). Questa maschera di bit indica un sotto insieme di tutti i controlli mixer disponibili—le costanti `SOUND_MIXER_*` definite a livello di modulo. Per determinarle, per esempio, l'oggetto mixer corrente che supporta un mixer PCM, userà il seguente codice Python:

```
mixer=ossaudiodev.openmixer()
if mixer.controls() & (1 << ossaudiodev.SOUND_MIXER_PCM):
    # PCM viene supportato
    ... code ...
```

Per molti scopi, i canali `SOUND_MIXER_VOLUME` (volume master) ed i controlli `SOUND_MIXER_PCM` dovrebbero bastare; ma il codice che usa il mixer dovrebbe essere flessibile quando inizia a scegliere i controlli del mixer. Sulla Gravis Ultrasound per esempio `SOUND_MIXER_VOLUME` non esiste.

stereocontrols()
Restituisce una maschera di bit che rappresenta i controlli stereo del mixer. Se un bit viene impostato ad `1`, il corrispondente controllo è stereo; se non viene impostato significa che è un controllo mono o non viene supportato dal mixer (lo si usi in combinazione con `controls()` per determinare qual'è il valore impostato).

Vedere il codice di esempio della funzione `controls()` per un esempio di lettura da una maschera di bit.

reccontrols()
Restituisce una maschera di bit che specifica i controlli del mixer che potrebbero essere usati per registrare. Vedere il codice di esempio per `controls()` per un esempio di lettura da una maschera di bit.

get(*control*)
Restituisce il volume del controllo mixer fornito. Il volume restituito è una tupla di due elementi `(left_volume,right_volume)`. I volumi vengono specificati come numeri da `0` (silenzioso) a `100`

(volume pieno). Se il controllo è mono, viene sempre restituita una tupla di due elementi, ma entrambi i volumi sono uguali.

Solleva un'eccezione `OSSAudioError` se un controllo non valido viene specificato, o `IOError` se viene specificato un controllo non supportato.

`set`(*control, (left, right)*)
Imposta il volume per i controlli del mixer forniti a `(left,right)`. `left` e `right` devono essere interi e tra `0` (silenzioso) e `100` (volume pieno). Se l'operazione ha successo, il nuovo volume viene restituito come una tupla di due elementi. Notare che questo potrebbe non essere esattamente come il volume specificato, per colpa della limitata risoluzione di alcuni mixer di schede audio.

Solleva l'eccezione `OSSAudioError` se viene specificato un controllo mixer non valido, o se i volumi specificati sono fuori dall'intervallo.

`get_recsrc`()
Questo metodo restituisce una maschera di bit che rappresenta quale controllo/i vengono correntemente inizializzati per essere usati come fonte di registrazione.

`set_recsrc`(*bitmask*)
Viene chiamata questa funzione per specificare una fonte di registrazione. Restituisce una maschera di bit che rappresenta la nuova fonte (o fonti) di registrazione se l'operazione ha successo; solleva un'eccezione `IOError` se viene specificata una sorgente non valida. Per impostare la corrente fonte di registrazione all'input del microfono:

```
mixer.setrecsrc (1 << ossaudiodev.SOUND_MIXER_MIC)
```

CAPITOLO

QUINDICI

Servizi di crittografia

I moduli descritti in questo capitolo implementano diversi algoritmi di natura crittografica. Questi sono disponibili a seconda delle installazioni. Eccone una panoramica:

hmac	Implementazione Keyed-Hashing per l'autenticazione dei messaggi (HMAC) per Python.
md5	Algoritmo di messaggi digest RSA MD5.
sha	NIST, algoritmo di hash sicuro, SHA.
mpz	Interfaccia alla libreria GNU MP per precisione aritmetica arbitraria.
rotor	Crittazione e decrittazione di tipo Enigma.

I cifratori incalliti probabilmente troveranno i moduli crittografici scritti da A.M. Kuchling di maggior interesse; questo package aggiunge moduli built-in per la crittazione DES e IDEA, fornisce un modulo Python che legge e descritta i file PGP ed altre funzioni. Questi moduli non sono distribuiti con Python ma sono disponibili separatamente. Vedere all'URL http://www.amk.ca/python/code/crypto.html per ulteriori informazioni.

15.1 hmac — Keyed-Hashing per l'autenticazione dei messaggi

Nuovo nella versione 2.2.

Questo modulo implementa l'algoritmo HMAC come descritto nella RFC 2104.

new(*key*[, *msg*[, *digestmod*]])
Restituisce un nuovo oggetto hmac. Se *msg* è presente, viene eseguita la chiamata al metodo `update(`*msg*`)`. *digestmod* è il modulo digest che l'oggetto HMAC userà. Per definizione è impostato al modulo `md5`.

Un oggetto HMAC possiede i seguenti metodi:

update(*msg*)
Aggiorna l'oggetto hmac con la stringa *msg*. Chiamate ripetute equivalgono ad una singola chiamata i cui argomenti sono concatenati: `m.update(a); m.update(b)` equivale a `m.update(a + b)`.

digest()
Restituisce il digest delle stringhe passate fino a quel momento al metodo `update()`. Questo è una stringa da 16 byte (per `md5`) o una stringa da 20 byte (per `sha`) che può contenere caratteri non ASCII, compresi i bytes NUL.

hexdigest()
Equivalente a `digest()` tranne il fatto che il digest viene restituito come una stringa di lunghezza 32 per `md5` (40 per `sha`), contenente solamente cifre esadecimali. Può essere usato nelle email o in altri ambienti non binari per modificarne il valore senza rischi.

copy()
Restituisce una copia ("clone") dell'oggetto hmac. Può essere usato per calcolare efficientemente i digest di quelle stringhe che condividono una medesima sottostringa iniziale.

15.2 `md5` — Algoritmo di messaggi digest MD5

Questo modulo implementa l'interfaccia per l'algoritmo del messaggio digest RSA MD5 (si veda anche Internet RFC 1321). Il suo uso è piuttosto diretto: si usa `new()` per creare un oggetto md5. Quindi si passano a questo oggetto stringhe diverse usando il metodo `update()` potendo, in qualsiasi momento, ottenerne il *digest* (un potente tipo di checksum a 128 bit, noto anche come "fingerprint") della concatenazione delle stringhe fino a quel momento passate, grazie al metodo `digest()`.

Per esempio, per ottenere il digest della stringa `'Nobody inspects the spammish repetition'`:

```
>>> import md5
>>> m = md5.new()
>>> m.update("Nobody inspects")
>>> m.update(" the spammish repetition")
>>> m.digest()
'\xbbd\x9c\x83\xdd\x1e\xa5\xc9\xd9\xde\xc9\xa1\x8d\xf0\xff\xe9'
```

In maniera più concisa:

```
>>> md5.new("Nobody inspects the spammish repetition").digest()
'\xbbd\x9c\x83\xdd\x1e\xa5\xc9\xd9\xde\xc9\xa1\x8d\xf0\xff\xe9'
```

I seguenti valori sono forniti come costanti nel modulo e come attributi degli oggetti md5 restituiti da `new()`:

`digest_size`
La dimensione in bytes del digest risultante. È sempre `16`.

Gli oggetti md5 supportano i seguenti metodi:

`new``([`*arg*`])`
Restituisce un nuovo oggetto md5. Se viene passato *arg*, viene eseguita la chiamata al metodo `update(`*arg*`)`.

`md5``([`*arg*`])`
Per ragioni di compatibilità con il passato, questo è un nome alternativo per la funzione `new()`.

Un oggetto md5 possiede i seguenti metodi:

`update``(`*arg*`)`
Aggiorna l'oggetto md5 con la stringa *arg*. Chiamate ripetute equivalgono ad una singola chiamata i cui argomenti sono concatenati: `m.update(a); m.update(b)` equivale a `m.update(a+b)`.

`digest``()`
Restituisce il digest delle stringhe passate fino a quel momento al metodo `update()`. Questo è una stringa da 16 byte che può contenere caratteri non ASCII, compresi i bytes null.

`hexdigest``()`
Equivalente a `digest()` tranne per il fatto che il digest viene restituito come una stringa di dimensione 32 contenente solamente cifre esadecimali. Può essere usato nelle email o in altri ambienti non binari per modificarne il valore senza rischi .

`copy``()`
Restituisce una copia ("clone") dell'oggetto md5. Può essere usato per calcolare efficientemente i digest di quelle stringhe che condividono una medesima sottostringa iniziale.

Vedete anche:

Modulo `sha` (sezione 15.3):
Un modulo simile implementa il Secure Hash Algorithm (SHA). L'algoritmo SHA è considerato un hash più sicuro.

15.3 sha — Algoritmo dei messaggi digest SHA-1

Questo modulo implementa l'interfaccia per l'algoritmo dell'hash sicuro NIST, noto come SHA-1. SHA-1 è una versione migliorata dell'algoritmo originale di hash SHA. Viene usato nello stesso modo del modulo `md5`: si usa `new()` per creare un oggetto sha, quindi si passano a questo oggetto stringhe diverse usando il metodo `update()` potendo, in qualsiasi momento, ottenerne il digest della concatenazione delle stringhe fino a quel momento passate. I digest SHA-1 sono da 160 bit invece dei 128 di quelli MD5.

`new``([`*string*`])`
: Restituisce un nuovo oggetto sha. Se è presente *string*, viene eseguita la chiamata al metodo `update(`*string*`)`.

I seguenti valori sono forniti come costanti nel modulo e come attributi degli oggetti sha restituiti da `new()`:

`blocksize`
: Dimensione dei blocchi passati alla funzione hash; è sempre `1`. Questa dimensione viene usata per permettere ad una qualsiasi stringa di diventare un hash.

`digest_size`
: Dimensione in byte del digest risultante. È sempre `20`.

Un oggetto sha possiede gli stessi metodi di un oggetto md5:

`update``(`*arg*`)`
: Aggiorna l'oggetto sha con la stringa *arg*. Chiamate ripetute equivalgono ad una singola chiamata i cui argomenti sono concatenati: `m.update(a); m.update(b)` equivale a `m.update(a+b)`.

`digest``()`
: Restituisce il digest delle stringhe passate fino a quel momento al metodo `update()`. Questo è una stringa da 20 byte che può contenere caratteri non ASCII, compresi i byte NULL.

`hexdigest``()`
: Equivalente a `digest()` tranne per il fatto che il digest viene restituito come una stringa di lunghezza 40 contenente solo cifre esadecimali. Può essere usato nelle email o in altri ambienti non binari per modificarne il valore senza rischi.

`copy``()`
: Restituisce una copia ("clone") dell'oggetto sha. Può essere usato per calcolare efficientemente i digest di quelle stringhe che condividono una medesima sottostringa iniziale.

Vedete anche:

Secure Hash Standard
(http://csrc.nist.gov/publications/fips/fips180-1/fip180-1.txt)
: Il Secure Hash Algorithm è definito nel documento NIST FIPS PUB 180-1: *Secure Hash Standard*, pubblicato nell'aprile del 1995. È disponibile online come puro testo (con l'omissione di almeno un diagramma) e come PDF presso http://csrc.nist.gov/publications/fips/fips180-1/fip180-1.pdf.

Cryptographic Toolkit (Secure Hashing)
(http://csrc.nist.gov/encryption/tkhash.html)
: Collegamenti da NIST verso varie informazioni sull'hashing sicuro.

15.4 mpz — Interi GNU di dimensioni arbitrarie

Deprecato dalla versione 2.2. Vedere i riferimenti alla fine di questa sezione per informazioni sui package che forniscono funzionalità simili. Questo modulo verrà rimosso in Python 2.3.

Questo è un modulo opzionale. È disponibile solo quando Python è configurato per includerlo, il che richiede l'installazione del software GNU MP.

Questo modulo implementa l'interfaccia di una parte della libreria GNU MP, che definisce interi a precisione arbitraria e routine aritmetiche sui numeri razionali. Vengono fornite solo le interfacce per le routine *integer* (`mpz_*()`). Se non definito altrimenti, può essere applicata la descrizione della documentazione GNU MP.

In Python può essere implementato il supporto per i numeri razionali. Per un esempio vedere il modulo `Rat`, fornito come 'Demos/classes/Rat.py' nella distribuzione dei sorgenti di Python.

In genere i numeri *mpz* possono essere usati come gli altri numeri standard di Python, ad esempio si possono usare gli operatori built-in come `+`, `*`, etc., tanto quanto le funzioni built-in standard come `abs()`, `int()`, ..., `divmod()`, `pow()`. **Notare bene:** l'operazione *bit per bit xor* è stata implementata come un gruppo di *and*, *invert* e *or*, poiché nella libreria manca una funzione `mpz_xor()`, che poi non serve.

Si crea un numero *mpz* chiamando la funzione `mpz()` (vedere più sotto per una descrizione precisa). Un numero *mpz* risulta stampato così: `mpz(`*value*`)`.

`mpz``(`*value*`)`
Crea un nuovo numero mpz. *value* può essere un intero, un long, un altro numero mpz, e perfino una stringa. Se è una stringa, viene interpretato come un array di cifre con radice 256, con prima la cifra meno significativa, dando come risultato un numero positivo. Vedere anche il metodo `binary()`, descritto sotto.

`MPZType`
Il tipo degli oggetti restituiti da `mpz()` e dalla maggior parte delle funzioni di questo modulo.

In questo modulo vengono definite svariate funzioni *extra*. Argomenti non mpz vengono prima convertiti in valori mpz, quindi le funzioni restituiscono numeri mpz.

`powm``(`*base, exponent, modulus*`)`
Restituisce `pow(`*base, exponent*`) % `*modulus*. Se *exponent* `== 0`, restituisce `mpz(1)`. A differenza della funzione della libreria C, questa versione può gestire esponenti negativi.

`gcd``(`*op1, op2*`)`
Restituisce il massimo comun divisore tra *op1* e *op2*.

`gcdext``(`*a, b*`)`
Restituisce una tupla `(`*g, s, t*`)` tale che *a*`*`*s* `+` *b*`*`*t* `==` *g* `==` `gcd(`*a, b*`)`.

`sqrt``(`*op*`)`
Restituisce la radice quadrata di *op*. Il risultato viene arrotondato verso zero.

`sqrtrem``(`*op*`)`
Restituisce una tupla `(`*root, remainder*`)`, tale che *root*`*`*root* `+` *remainder* `==` *op*.

`divm``(`*numerator, denominator, modulus*`)`
Restiuisce un numero *q* tale che *q* `*` *denominator* `%` *modulus* `==` *numerator*. È possibile implementare questa funzione in Python usando `gcdext()`.

Un numero mpz possiede un solo metodo:

`binary``()`
Converte questo numero mpz in una stringa binaria, dove il numero viene conservato come un array di cifre con radice 256, con all'inizio la cifra meno significativa.

Il numero mpz deve avere un valore maggiore o uguale a zero, altrimenti verrà sollevata un'eccezione `ValueError`.

Vedete anche:

General Multiprecision Python
(http://gmpy.sourceforge.net/)
Questo progetto sta creando nuovi tipi numerici che supportino l'aritmetica a precisone arbitraria. I loro primi sforzi si basano anche sulla libreria GNU MP.

mxNumber — Extended Numeric Types for Python
(http://www.egenix.com/files/python/mxNumber.html)
Un altro lavoro che ha a che fare con la libreria GNU MP, che comprende il porting di tale libreria su piattaforma Windows.

15.5 `rotor` — Crittazione e decrittazione di tipo Enigma

Deprecato dalla versione 2.3. L'algoritmo di crittazione non è sicuro.

Questo modulo implementa un algoritmo di crittazione basato su rotor, dovuto a Lance Hellinghouse. Il progetto deriva dal dispositivo Enigma, una macchina usata durante la II Guerra Mondiale per cifrare i messaggi. Un rotor è semplicemente una permutazione. Ad esempio, se il carattere 'A' è l'origine del rotor, allora un dato rotor può mappare 'A' con 'L', 'B' con 'Z', 'C' con 'G' e così via. Per crittare si scelgono numerosi rotor differenti e si impostano le origini dei rotor sulle posizioni conosciute; la loro posizione iniziale è la chiave di cifratura. Per cifrare un carattere si permuterà il carattere originale con il primo rotor, quindi si applicherà al risultato la permutazione del secondo rotor. Si andrà avanti fino ad aver applicato tutti i rotor. Il carattere risultante è il nostro testo cifrato. A questo punto si cambia l'origine dell'ultimo rotor di una posizione, da 'A' a 'B'; se l'ultimo rotor ha compiuto un giro completo, allora si ruota di una posizione il penultimo rotor, applicando ripetutamente la stessa procedura. In altre parole, dopo aver cifrato un carattere, si fanno avanzare i rotor allo stesso modo del contachilometri di un auto. La decodifica funziona nello stesso modo, tranne per il fatto che si invertono le permutazioni e le si applicano in ordine inverso.

In questo modulo le funzioni disponibili sono:

newrotor(*key*[, *numrotors*])
: Restituisce un oggetto rotor. *key* è una stringa contenente la chiave di cifratura dell'oggetto; può contenere dati binari arbitrari ma non byte null. La chiave verrà usata per generare casualmente le permutazioni del rotor e le loro posizioni iniziali. *numrotors* è il numero delle permutazioni del rotor dell'oggetto restituito. Se omesso verrà usato il valore predefinito 6.

Gli oggetti rotor possiedono i seguenti metodi:

setkey(*key*)
: Imposta *key* come chiave del rotor. *key* non dovrà contenere byte null.

encrypt(*plaintext*)
: Reinizializza l'oggetto rotor e critta *plaintext*, restituendo una stringa contenente il testo cifrato. Il testo cifrato è sempre di lunghezza uguale al testo in chiaro *plaintext*.

encryptmore(*plaintext*)
: Critta *plaintext* senza resettare l'oggetto rotor, e restituisce una stringa contenente testo cifrato.

decrypt(*ciphertext*)
: Reinizializza l'oggetto rotor e decritta *ciphertext*, restituendo una stringa contenente il testo in chiaro. La stringa di testo in chiaro avrà sempre la stessa lunghezza del testo cifrato.

decryptmore(*ciphertext*)
: Decritta *ciphertext* senza resettare l'oggetto rotor e restituisce una stringa contenente il testo in chiaro.

Ecco un esempio:

```
>>> import rotor
>>> rt = rotor.newrotor('key', 12)
>>> rt.encrypt('bar')
'\xab4\xf3'
>>> rt.encryptmore('bar')
'\xef\xfd$'
>>> rt.encrypt('bar')
'\xab4\xf3'
>>> rt.decrypt('\xab4\xf3')
'bar'
>>> rt.decryptmore('\xef\xfd$')
'bar'
>>> rt.decrypt('\xef\xfd$')
'l(\xcd'
>>> del rt
```

Il codice del modulo non è l'esatta simulazione del dispositivo Enigma originale; lo schema di crittazione del rotor è implementato in modo diverso dall'originale. La principale differenza è che nell'Enigma originale si avevano in funzione solo 5 o 6 rotor diversi che venivano applicati due volte per ogni carattere; la chiave di cifratura era l'ordine in cui essi venivano posizionati nella macchina. Il modulo `rotor` di Python usa la chiave fornita per inizializzare un generatore di numeri casuali; le permutazioni dei rotor e le loro posizioni iniziali sono quindi generate casualmente. L'apparecchio originale cifrava solo le lettere dell'alfabeto, mentre questo modulo può

gestire qualsiasi dato binario a 8 bit e binario può essere anche il suo output. Inoltre questo modulo può lavorare con un indefinito numero di rotor.

Il codice Enigma originale fu infranto nel 1944. La versione qui implementata ha un algoritmo probabilmente più impegnativo da rompere (specialmente se si usano più rotor), ma non sarebbe impossibile violarne il codice da parte di un attaccante molto abile e ben determinato. Quindi, se si vuole tenere l'NSA alla larga dai file, questo codice rotor potrebbe non essere sicuro ma, probabilmente, andrà più che bene per scoraggiare eventuali sbirciate ai file, garantendo un po' più di sicurezza rispetto all'uso del comando UNIX **crypt**.

CAPITOLO

SEDICI

Interfaccia Utente Grafica con Tk

Tk/Tcl è da molto tempo una parte integrante di Python. Fornisce un robusto windowing toolkit indipendente dalle piattaforme, che è disponibile per i programmatori Python usando il modulo `Tkinter` e la sua estensione, il modulo `Tix`.

Il modulo `Tkinter` è un sottile strato orientato agli oggetti sopra a Tcl/Tk. Non è necessario scrivere codice Tcl per usare `Tkinter`, ma sarà necessario consultare la documentazione di Tk e, occasionalmente, quella di Tcl. `Tkinter` è un insieme di involucri che implementano i Tk widget come classi di Python. Inoltre il modulo interno `_tkinter` fornisce un meccanismo threadsafe che permette a Python e a Tcl di interagire.

Non è l'unica GUI per Python, ma è quella usata più comunemente; vedere la sezione **??** "Altri moduli di interfaccia utente e pacchetti", per maggiori informazioni su altri strumenti per GUI disponibili in Python.

`Tkinter`	Interfaccia a Tcl/Tk per interfacce utenti grafiche
`Tix`	Widgets che estendono Tk per Tkinter
`ScrolledText`	Widget per testi con barra di scorrimento verticale.
`turtle`	Un ambiente per la grafica della tartaruga.

16.1 `Tkinter` — Interfaccia Python per Tcl/Tk

Il modulo `Tkinter` ("Tk interface") è l'interfaccia standard di Python al Tk GUI toolkit. Sia Tk che `Tkinter` sono disponibili in molte piattaforme UNIX, così come nei sistemi Windows e Macintosh (Tk non è parte di Python; è mantenuto in efficienza da Activestate).

Vedete anche:

Risorse Python Tkinter
(http://www.python.org/topics/tkinter/)
: La guida agli argomenti di Tkinter fornisce una grande quantità di informazioni sull'uso di Tk da Python e rimanda ad altre sorgenti di informazioni su Tk.

An Introduction to Tkinter
(http://www.pythonware.com/library/an-introduction-to-tkinter.htm)
: Materiale di riferimento in linea di Fredrik Lundh.

Tkinter reference: a GUI for Python
(http://www.nmt.edu/tcc/help/pubs/lang.html)
: Materiale di riferimento in linea.

Tkinter per JPython
(http://jtkinter.sourceforge.net)
: L'interfaccia a Tkinter di Jython.

Python and Tkinter Programming
(http://www.amazon.com/exec/obidos/ASIN/1884777813)
: Il libro di John Grayson (ISBN 1-884777-81-3).

16.1.1 Moduli Tkinter

Per la maggior parte del tempo, il modulo `Tkinter` è tutto quello che realmente vi serve, ma comunque è disponibile un buon numero di moduli aggiuntivi. L'interfaccia Tk si trova in un modulo binario chiamato `_tkinter`. Questo modulo contiene l'interfaccia di basso livello a Tk e non deve mai essere usato direttamente dai programmatori di applicazioni. È normalmente una libreria condivisa (o DLL), ma può, in alcuni casi, essere collegata staticamente con l'interprete Python.

In aggiunta al modulo di interfaccia a Tk, `Tkinter` comprende un buon numero di moduli Python. I due moduli più importanti sono `Tkinter` stesso ed un modulo chiamato `Tkconstants`. Il primo importa automaticamente il secondo, così che per usare Tkinter tutto quello che si deve fare è importare un solo modulo:

```
import Tkinter
```

O, più spesso:

```
from Tkinter import *
```

class `Tk`(*screenName=None, baseName=None, className='Tk'*)
La classe `Tk` viene istanziata senza argomenti. Questo crea un widget di livello base che, di solito, è la finestra principale di un'applicazione. Ogni istanza ha associato il suo proprio Interprete Tcl. Modificato nella versione 2.4: È stato aggiunto il parametro *useTk*.

`Tcl`(*screenName=None, baseName=None, className='Tk', useTk=0*)
La funzione `Tcl` è una funzione factory che crea un oggetto molto simile a quello realizzato dalla classe `Tk`, tranne per il fatto che non inizializza il sottosistema Tk. Questo spesso è molto utile quando viene usato un interprete tcl in un ambiente dove non si vuole creare una finestra toplevel estranea o dove non si può (ad esempio in sistemi UNIX/Linux senza il server X). Un oggetto creato mediante l'oggetto `Tcl` può avere una finestra toplevel creata (ed inizializzando il sottosistema Tk) mediante la chiamata al metodo `loadtk`. Nuovo nella versione 2.4.

Altri moduli che il supporto di Tk fornisce includono:

`ScrolledText` Widget di testo con una barra di scorrimento verticale integrata.

`tkColorChooser` Finestra di dialogo che permette di scegliere un colore.

`tkCommonDialog` Classe base per le finestre di dialogo definite negli altri moduli elencati qui.

`tkFileDialog` Finestra di dialogo che permette all'utente di specificare un file da aprire o da salvare.

`tkFont` Strumento che aiuta il lavoro con i font dei caratteri.

`tkMessageBox` Accesso alle finestre standard di dialogo di Tk.

`tkSimpleDialog` Funzioni base per le finestre di dialogo e funzioni di comodo

`Tkdnd` Supporto al Drag-and-drop per Tkinter. È sperimentale e non verrà più accettato quando sarà sostituito dal Tk DND.

`turtle` La grafica della tartaruga in una finestra Tk.

16.1.2 Tkinter salvagente

Questa sezione non è pensata per essere una guida esaustiva a Tk o Tkinter. Piuttosto è intesa a colmare una lacuna fornendo alcune indicazioni introduttive al sistema.

Ringraziamenti:

- Tkinter è stato scritto da Steen Lumholt e Guido van Rossum.
- Tk è stato scritto da John Ousterhout mentre era a Berkeley.

- Questo Salvagente è stato scritto da Matt Conway presso l'università della Virginia.
- La traduzione html ed alcune edizioni libere, sono state prodotte da una versione FrameMaker da Ken Manheimer.
- Frederik Lundh ha elaborato e revisionato le descrizioni delle interfacce delle classi, per ottenere la versione corrente con Tk 4.2.
- Mike Clarkson ha tradotto la documentazione in LaTeX e compilato il capitolo dell'interfaccia utente del manuale di riferimento.

Come usare questa sezione

Questa sezione è progettata in due parti: la prima metà (approssimativamente) riguarda il materiale di base, mentre la seconda può essere intesa come manuale di riferimento per l'utente.

Quando si tenta di rispondere a domande del tipo: "come faccio a fare qualcosa", è spesso meglio cercare come fare "qualcosa" direttamente in Tk e poi convertirlo nella corrispondente chiamata `Tkinter`. I programmatori Python possono spesso supporre il corretto comando Python guardando la documentazione di Tk. Quindi per usare Tkinter, si deve conoscere un po' di Tk. Non essendo questo lo scopo del presente documento, il massimo che possiamo fare è indicare la miglior documentazione che esiste. Alcuni suggerimenti:

- Gli autori raccomandano di procurarsi una copia delle pagine di manuale di Tk. In particolare le pagine di manuale nella directory `mann` sono molto utili. Le pagine `man3` descrivono l'interfaccia C alla libreria Tk e questo non è particolarmente utile per chi scrive programmi.
- Addison-Wesley pubblica un libro intitolato *Tcl and the Tk Toolkit* di John Ousterhout (ISBN 0-201-63337-X) che è una buona introduzione a Tcl e Tk per i principianti. Il libro non è completo e per molti dettagli rimanda alle pagine di manuale.
- 'Tkinter.py' è, per molti, l'ultima risorsa, ma può essere un buon posto dove cercare quando nient'altro sembra avere senso.

Vedete anche:

La Home Page di ActiveState Tcl
(http://tcl.activestate.com/)
Lo sviluppo di Tk/Tcl è ampiamente trattato presso ActiveState.

Tcl and the Tk Toolkit
(http://www.amazon.com/exec/obidos/ASIN/020163337X)
Il libro di John Ousterhout, l'inventore di Tcl.

Practical Programming in Tcl and Tk
(http://www.amazon.com/exec/obidos/ASIN/0130220280)
Il libro enciclopedico di Brent Welch.

```
from Tkinter import *

class Application(Frame):
    def say_hi(self):
        print "Ciao a tutti!"

    def createWidgets(self):
        self.QUIT = Button(self)
        self.QUIT["text"] = "QUIT"
        self.QUIT["fg"]   = "red"
        self.QUIT["command"] =  self.quit

        self.QUIT.pack({"side": "left"})

        self.hi_there = Button(self)
        self.hi_there["text"] = "Hello",
        self.hi_there["command"] = self.say_hi

        self.hi_there.pack({"side": "left"})

    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.pack()
        self.createWidgets()

app = Application()
app.mainloop()
```

16.1.3 Una (molto) rapida occhiata a Tcl/Tk

La gerarchia delle classi sembra complicata, ma nella pratica attuale i programmatori delle applicazioni possono quasi sempre riferirsi alle classi che si trovano negli ultimi nodi della gerarchia.

Note:

- Queste classi vengono fornite allo scopo di organizzare certe funzioni in uno spazio dei nomi. Non sono pensate per essere istanziate indipendentemente.
- La classe `Tk` è pensata per essere istanziata una sola volta in una applicazione. I programmatori di applicazioni non hanno bisogno di istanziarne una esplicitamente, il sistema ne crea una ogni volta che una delle altre classi viene istanziata.
- La classe `Widget` non è pensata per essere istanziata, è pensata solo per derivarne delle sottoclassi per fare dei widget “reali” (in C++ si direbbe una ‘classe astratta’).

Per fare uso di questo materiale di riferimento, alcune volte ci sarà bisogno di saper leggere brevi passaggi di Tk e di identificare le varie parti di un comando Tk. Vedere la sezione 16.1.4 per l’equivalente in Tkinter di quanto segue.

Gli script di Tk sono programmi Tcl. Come tutti i programmi Tcl, gli script di Tk sono liste di pezzi separati da spazi. Un widget Tk è formato dalla sua *classe*, dalle *opzioni* che aiutano a configurarlo e dalle *azioni* che gli fanno fare qualcosa di utile.

Per fare un widget in Tk, il comando è sempre nella forma:

```
classCommand newPathname options
```

classCommand denota quale tipo di widget costruire (un bottone, un'etichetta, un menu, ...)

newPathname é il nome per questo widget. Tutti i nomi in Tk devono essere unici. Per aiutare a far rispettare questo, i widget in Tk sono chiamati con un *pathname* (NdT: percorso), proprio come i file in un file system. Il widget di livello più alto, *root*, la radice, è chiamato `.` (punto) ed i discendenti vengono delimitati da altri punti. Per esempio, `.myApp.controlPanel.okButton` può essere il nome di un widget.

options configurano l'aspetto del widget e, in alcuni casi, il suo comportamento. Le opzioni sono date in forma di lista di opzioni e valori. Le opzioni sono precedute da un '-', come le opzioni delle shell UNIX ed i valori sono messi tra virgolette se sono costituiti da più di una parola.

Per esempio:

```
button     .fred     -fg red -text "hi there"
   ^         ^       \_____________________/
   |         |                  |
 Comando    nuovo           options
di classe  widget    (-opt val -opt val ...)
```

Una volta creato, il pathname del widget diventa un nuovo comando. Questo nuovo *comando di widget* è lo strumento del programmatore per fare in modo che il nuovo widget esegua qualche *azione*. In C andrebbe scritto così: unAzione(fred, opzioni), in C++ lo si scriverebbe: fred.unAzione(opzioni), e in Tk:

```
.fred someAction someOptions
```

Notare che il nome dell'oggetto, `.fred`, inizia con un punto.

Come ci si può aspettare, i valori accettabili per *unAzione* dipenderanno dalla classe del widget: `.fred disable` funziona se fred è un bottone (fred diventa grigiolino), ma non funziona se fred è un'etichetta (la disabilitazione di un'etichetta non è supportata in Tk).

I valori accettabili di *opzioni* dipendono dall'azione. Alcune azioni come `disable`, non richiedono argomenti, altre, come il comando `delete` in una casella di introduzione di testo, richiederà argomenti che specifichino quale parte di testo cancellare.

16.1.4 Mappatura dei comandi di base di Tk in Tkinter

I comandi di classe in Tk corrispondono ai costruttori di classi in Tkinter.

```
button .fred                =====>  fred = Button()
```

Il proprietario di un oggetto è implicito nel nuovo nome datogli al momento della creazione. In Tkinter, il proprietario è specificato esplicitamente.

```
button .panel.fred          =====>  fred = Button(panel)
```

Le opzioni di configurazione in Tk vengono date in liste di parole precedute dal segno meno seguite da valori. In Tkinter le opzioni vengono specificate come argomenti dati con parole chiave nell'istanza del costruttore, come argomenti dati con parole chiave nella chiamata del metodo config o come indici dell'istanza, nello stile dei dizionari, per le istanze stabilite. Vedere la sezione 16.1.6 su come impostare le opzioni.

```
button .fred -fg red        =====>  fred = Button(panel, fg = "red")
.fred configure -fg red     =====>  fred["fg"] = red
                            OR ==>  fred.config(fg = "red")
```

In Tk, per eseguire un'azione in un widget, si usa il nome del widget come un comando, seguito dal nome dell'azione ed eventualmente da argomenti (le opzioni). In Tkinter, si chiamano i metodi dell'istanza di una classe per eseguire un'azione in un widget. Le azioni (i metodi) che un dato widget può eseguire sono elencate nel modulo Tkinter.py.

```
.fred invoke                =====>  fred.invoke()
```

Per passare un widget al packer (il gestore della disposizione), si può chiamare pack con argomenti opzionali. In Tkinter, la classe Pack contiene tutte queste funzionalità e le varie forme dei comandi di pack sono implementati come metodi. Tutti i widget in `Tkinter` sono sottoclassi di Packer e così ereditano tutti i suoi metodi. Vedere la documentazione del modulo `Tix` per ulteriori informazioni sul gestore della geometria delle Form.

```
pack .fred -side left       =====>  fred.pack(side = "left")
```

16.1.5 Come sono relazionati Tk e Tkinter

Note: Questo è stato derivato da un'immagine grafica; l'immagine verrà usata più direttamente in una seguente versione di questo documento.

Dal generale al particolare:

La vostra applicazione qui (Python) Un'applicazione Python produce delle chiamate a `Tkinter`.

Tkinter (Modulo di Python) Questa chiamata (vedere, per esempio, la creazione di un bottone), è implementata nel modulo *Tkinter*, che è scritto in Python. Questa funzione Python analizzerà il comando e gli argomenti e li convertirà in una forma che li fa simili a come sarebbero se invece che da uno script di Python venissero da uno script di Tk.

tkinter (C) Questi comandi e i loro argomenti verranno passati ad una funzione C nel modulo di estensione di *tkinter* - notare le minuscole.

Tk Widgets (C e Tcl) Questa funzione C è in grado di eseguire delle chiamate ad altri moduli C, incluse le funzioni di C che formano la libreria Tk. Tk è implementata in C e in parte in Tcl. La parte Tcl dei widget di Tk viene usata per costruire certi comportamenti predefiniti ed eseguita solo nel momento in cui il modulo `Tkinter` di Python viene importato (l'utente non vedrà mai questa fase).

Tk (C) La parte di Tk dei Widget di Tk implementa la mappatura finale a ...

Xlib (C) la libreria Xlib per disegnare elementi grafici sullo schermo.

16.1.6 Manuale di riferimento

Le opzioni di impostazione

Le opzioni controllano cose come il colore o la larghezza dei bordi di un widget. Le opzioni possono essere impostate in tre maniere:

Nel momento della creazione dell'oggetto, usando argomenti a parola chiave :

```
fred = Button(self, fg = "red", bg = "blue")
```

Dopo la creazione dell'oggetto, trattando il nome dell'opzione come la chiave di un dizionario :

```
fred["fg"] = "red"
fred["bg"] = "blue"
```

Usando il metodo config() per modificare più attributi dopo la creazione dell'oggetto :

```
fred.config(fg = "red", bg = "blue")
```

Per una completa esposizione delle possibili opzioni e del loro comportamento, vedere le pagine di manuale di Tk per il widget in questione.

Notare che le pagine di manuale elencano le OPZIONI STANDARD e le OPZIONI SPECIFICHE DEL WIDGET per ogni widget. Le prime sono una lista delle opzioni comuni a molti widget, le seconde sono le opzioni specifiche di un particolare widget. Le opzioni standard vengono documentate nella pagina di manuale di *options*(3).

In questo documento non viene fatta distinzione tra le opzioni standard e quelle specifiche per un dato widget. Alcune opzioni non si applicano ad alcuni tipi di widget. Se un dato widget risponda o meno ad una particolare opzione, dipende dalla classe del widget (i bottoni hanno l'opzione `command`, le etichette no).

Le opzioni supportate da un dato widget sono elencate nella pagina di manuale di questo widget o possono essere trovate nel momento dell'esecuzione chiamando il metodo `config()` senza argomenti o chiamando il metodo `keys()` di questo widget. Il valore restituito da queste chiamate è un dizionario dove la chiave è il nome dell'opzione sotto forma di stringa (ad esempio `'relief'`) ed il suo valore è una tupla di 5 elementi.

Alcune opzioni, come `bg` sono sinonimi di altre opzioni con nomi lunghi (`bg` è l'abbreviazione di background). Passando al metodo `config()` il nome abbreviato di un'opzione, questo restituirà una coppia di tuple non una quintupla. La coppia di tuple restituita conterrà il nome del sinonimo e l'opzione "real" (NdT: reale) (come `(('bg', 'background'))`).

Indice	Significato	Esempio
0	nome dell'opzione	`'relief'`
1	nome dell'opzione per la ricerca nel database	`'relief'`
2	classe dell'opzione per la ricerca nel database	`'Relief'`
3	valore predefinito	`'raised'`
4	valore attuale	`'groove'`

Esempio:

```
>>> print fred.config()
{'relief' : ('relief', 'relief', 'Relief', 'raised', 'groove')}
```

Naturalmente il dizionario stampato includerà tutte le opzioni disponibili ed i loro valori. Questo è solo un esempio.

Il Packer

Il packer è uno dei meccanismi di gestione della geometria di Tk. Vedere anche *l'interfaccia della classe Packer*.

I gestori della geometria vengono usati per specificare la posizione reciproca dei widget all'interno dei loro contenitori - i loro comuni proprietari, *master*. Contrariamente al più scomodo posizionatore, *placer*, (che viene

utilizzato meno comunemente, e di cui non trattiamo qui), il packer richiede di specificare relazioni qualitative - *dopo*, *a sinistra di*, *riempimento*, ecc. - e rielabora ogni cosa fino a determinare le coordinate della posizione esatta.

La dimensione di ogni widget *proprietario* viene determinata dalla dimensione dei widget contenuti al suo interno. Il packer viene usato per controllare dove i widget contenuti appaiono all'interno del proprietario. Si possono inserire widget dentro a frame e frame entro altri frame per ottenere il tipo di disposizione desiderata. Inoltre la disposizione degli oggetti viene regolata dinamicamente in modo da adattarsi a cambiamenti successivi della configurazione, che potrebbero avvenire dopo l'impacchettamento.

Notare che i widget non appaiono finché non hanno avuto la loro geometria specificata da un gestore geometrico. È un errore comune dimenticare la specificazione della geometria ed essere sorpresi quando il widget viene creato ma non appare. Un widget apparirà solamente dopo che gli è stato applicato, ad esempio, il metodo `pack()`.

Il metodo pack() può essere chiamato con coppie di chiavi-valori che controllano dove il widget deve apparire all'interno del suo contenitore e come deve comportarsi quando la finestra dell'applicazione proprietaria viene ridimensionata. Alcuni esempi:

```
fred.pack()                     # in modo predefinito di fianco = "top"
fred.pack(side = "left")
fred.pack(expand = 1)
```

Opzioni di Packer

Per maggiori informazioni sul packer e sulle opzioni che può ricevere, vedere le pagine di manuale e la pagina 183 del libro di John Ousterhout.

anchor Tipo anchor. Indica dove il packer sistemerà ogni oggetto contenuto nel suo spazio.

expand Booleano, `0` o `1`.

fill Valori validi sono: `'x'`, `'y'`, `'both'`, `'none'`.

ipadx and ipady Una distanza - disegna un contorno interno su ogni lato del widget contenuto.

padx and pady Una distanza - disegna un contorno esterno su ogni lato del widget contenuto.

side Valori validi sono: `'left'`, `'right'`, `'top'`, `'bottom'`.

Associazione tra widget e variabili

Il valore corrente di alcuni widget (come la casella di introduzione di testo) può essere connesso direttamente a variabili dell'applicazione usando speciali opzioni. Queste opzioni sono `variable`, `textvariable`, `onvalue`, `offvalue` e `value`. Questo collegamento lavora in entrambe le direzioni: se la variabile, per qualunque motivo, cambia, allora il widget collegato verrà modificato per riflettere il nuovo valore.

Sfortunatamente nell'implementazione corrente di `Tkinter` non è possibile passare una variabile Python arbitraria ad un widget attraverso l'opzione `variable` o `textvariable`. L'unico tipo di variabili che possono svolgere questo lavoro sono variabili che derivano da una classe chiamata Variable, definita nel modulo `Tkinter`.

Ci sono molte sottoclassi di Variable già definite: `StringVar`, `IntVar`, `DoubleVar` e `BooleanVar`. Per leggere il valore attuale di una tale variabile, chiamare il suo metodo `get()` e, per cambiarlo, il metodo `set()`. Se si segue questo protocollo il widget si adeguerà sempre al valore della variabile senza ulteriori interventi.

Per esempio:

```
class App(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.pack()

        self.entrythingy = Entry()
        self.entrythingy.pack()

        self.button.pack()
        # Qui c'è la variabile dell'applicazione
        self.contents = StringVar()
        # Le viene dato un qualche valore
        self.contents.set("this is a variable")
        # Si fa in maniera che il widget di immissione veda questa variabile
        self.entrythingy["textvariable"] = self.contents

        # e qui si mette la chiamata ad una funzione quando l'utente preme return.
        # faremo in modo che il programma stampi il valore della
        # variabile dell'applicazione quando l'utente preme return.
        self.entrythingy.bind('<Key-Return>',
                              self.print_contents)

    def print_contents(self, event):
        print "Ciao. Il contenuto della casella di immissione ora è ---->", \
              self.contents.get()
```

Il window manager

In Tk c'è un comando di utilità, `wm`, per interagire con il window manager (NdT: gestore di finestre). Le opzioni del comando `wm` permettono di controllare cose come il titolo, la posizione, la bitmap dell'icona e cose simili. In `Tkinter`, questi comandi vengono implementati come metodi della classe `Wm`. I widget di Toplevel sono classi derivate dalla classe `Wm` e così possono chiamare i metodi di `Wm` direttamente.

Per scoprire la finestra di toplevel che contiene un dato widget, è possibile, spesso, semplicemente far riferimento al master del widget. Naturalmente se il widget è stato inserito in un frame il master non rappresenta la finestra di toplevel. Per scoprire la finestra di toplevel che contiene un widget arbitrario, si può chiamare il metodo `_root()`. Questo metodo inizia con il simbolo di sottolineatura per indicare che questa funzione è parte dell'implementazione e non un'interfaccia alle funzionalità di Tk.

Questi sono alcuni tipici esempi d'uso:

```
import Tkinter
class App(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.pack()


# Crea l'applicazione
myapp = App()

#
# Qui ci sono le chiamate di metodo alla classe window manager
#
myapp.master.title("My Do-Nothing Application")
myapp.master.maxsize(1000, 400)

# Inizio del programma
myapp.mainloop()
```

I tipi di dato delle opzioni Tk

anchor I valori ammessi sono i punti cardinali: `n`, `ne`, `e`, `se`, `s`, `sw`, `w`, `nw`, ed anche `center`.

bitmap Ci sono otto built-in bitmap predefinite: `'error'`, `'gray25'`, `'gray50'`, `'hourglass'`, `'info'`, `'questhead'`, `'question'`, `'warning'`. Per specificare un certo nome di file di bitmap dell'interfaccia grafica, si deve fornire l'intero percorso del file preceduto da un `@`, come in `@/usr/contrib/bitmap/gumby.bit`.

boolean Si possono passare gli interi 0 o 1 o le stringhe `yes` o `no`.

callback Questa è una qualunque funzione Python che non presenta argomenti. Per esempio:

```
def print_it():
        print "Ciao a tutti"
fred["command"] = print_it
```

color I colori possono essere dati con i nomi dei colori dell'interfaccia grafica presenti nel file rgb.txt o come stringhe rappresentanti i valori RGB in una risoluzione a 4 bit: `#RGB`, 8 bit: `#RRGGBB`, 12 bit `#RRRGGGBBB` o 16 bit `#RRRRGGGGBBBB`, dove R, G, B qui rappresentano ogni valore valido di numeri esadecimali. Vedere la pagina 160 del libro di Ousterhout per i dettagli.

cursor I nomi dei cursori standard dell'interfaccia grafica che si trovano in 'cursorfont.h' possono essere usati senza il prefisso `XC_`. Per esempio per ottenere un cursore a manina (`XC_hand2`), usare la stringa `hand2`. Si può anche usare una bitmap e un mask file vostro. Vedere la pagina 179 del testo di Ousterhout.

distance Le distanze sullo schermo possono essere specificate sia in pixel sia in valori assoluti. I pixel vengono passati come numeri ed i valori assoluti come stringhe terminanti con un carattere che indica l'unità di misura: `c` per centimetri, `i` per pollici, `m` per millimetri, `p` per punti della stampante. Per esempio 3.5 pollici è espresso con `3.5i`.

font Tk usa un formato di lista dei nomi delle font di caratteri, come `{courier 10 bold}`. La dimensione dei font, identificata da numeri positivi viene misurata in punti; le dimensioni identificate con numeri negativi indicano che sono misure in pixel.

geometry È una stringa nella forma '*larghezza*x*altezza*', dove larghezza e altezza sono misurate in pixel per la maggior parte dei widget (in caratteri per i widget che visualizzano testo). Per esempio: `fred[ geometry] = 200x100`.

justify I valori accettati sono le stringhe: `left`, `center`, `right` e `fill`.

region Questa è una stringa con quattro elementi separati da uno spazio, ognuno dei quali è una distanza (vedere sopra). Per esempio: `2 3 4 5`, `3i 2i 4.5i 2i` e `3c 2c 4c 10.43c` sono regioni accettabili.

relief Determina qual è lo stile del bordo di un widget. I valori accettati sono: `raised`, `sunken`, `flat`, `groove` e `ridge`.

scrollcommand Questo è quasi sempre il metodo `set()` di qualche widget scrollbar, ma può essere ogni metodo di un widget che prende un singolo argomento. Vedere il file 'Demo/tkinter/matt/canvas-with-scrollbars.py' nel sorgente della distribuzione Python per un esempio.

wrap: Deve essere una delle seguenti stringhe: `none`, `char` o `word`.

Binding ed eventi

Il metodo collegato al comando di un widget permette di monitorare certi eventi e di far scattare la chiamata ad una funzione quando accade quel tipo di evento. La forma del metodo bind è:

```
def bind(self, sequence, func, add=''):
```

dove:

sequence è una stringa che indica il tipo di evento considerato. Vedere la pagina di manuale di bind e la pagina 201 del testo di John Ousterhout per i dettagli.

func è una funzione Python, che prende un argomento e che viene chiamata quando l'evento accade. Un'istanza di Event verrà passata come argomento. Le funzioni utilizzate in questo modo sono, di solito, dette *callbacks*.

add è facoltativa, può valere '' o '+'. Passando una stringa vuota questo binding (NdT: legame) sostituirà ogni altro binding che è associato a questo evento. Passando la stringa '+' si indica che la funzione va aggiunta alla lista delle funzioni collegate a questo tipo di evento.

Per esempio:

```
def turnRed(self, event):
    event.widget["activeforeground"] = "red"

self.button.bind("<Enter>", self.turnRed)
```

Notare come qui si accede al campo widget di event nella chiamata `turnRed()`. Questo campo contiene il widget che ha intercettato l'evento nell'interfaccia grafica. La seguente tavola elenca gli altri campi di event a cui si può accedere e come sono indicati in Tk, cosa che può essere utile quando si fa riferimento alle pagine di manuale di Tk.

Tk	Campi Evento Tkinter	Tk	Campi Evento Tkinter
%f	focus	%A	char
%h	height	%E	send_event
%k	keycode	%K	keysym
%s	state	%N	keysym_num
%t	time	%T	type
%w	width	%W	widget
%x	x	%X	x_root
%y	y	%Y	y_root

L'indice dei parametri

Un certo numero di wiget richiedono che siano loro passati dei parametri "indice". Vengono usati per indicare uno specifico posto in un Text widget o un particolare carattere in un Entry widget o un particolare elemento di un menu in un Menu widget.

Entry widget indexes (index, view index, etc.) Gli Entry widget hanno opzioni che fanno riferimento alle posizioni dei caratteri nel testo che viene mostrato. Si possono usare queste funzioni di `Tkinter` per accedere a questi particolari punti nei widget di testo:

AtEnd() fa riferimento all'ultima posizione nel testo

AtInsert() fa riferimento al punto in cui si trova il cursore del testo

AtSelFirst() indica il punto in cui inizia un testo selezionato

AtSelLast() indica l'ultimo punto di un testo selezionato e la sua fine

At(x[, y]) fa riferimento al carattere che si trova nella posizione del pixel *x*, *y* (con *y* non utilizzato nel caso di un widget di introduzione di testo che contiene una singola linea di testo).

Text widget indexes La notazione per gli indici dei Text widget è molto ricca ed è ben descritta nelle pagine di manuale di Tk.

Menu indexes (menu.invoke(), menu.entryconfig(), etc.) Alcune opzioni e metodi per i menu gestiscono specifiche voci di menu. Se un indice di menu è necessario per un'opzione o un parametro, si può passare:

- un intero che si riferisce alla posizione di immissione nel widget, contata dall'inizio, partendo da 0;

- la stringa `'active'`, che fa riferimento alla posizione del menu che è attualmente sotto il cursore;
- la stringa `last` che fa riferimento all'ultimo elemento del menu;
- un intero preceduto da `@`, come in `@6`, dove il numero viene interpretato come la coordinata y di pixel nel sistema di coordinate del menu;
- la stringa `none`, che indica nessun elemento di menu, maggiormente usata con menu.activate() per disattivare tutte le voci, ed infine,
- una stringa di testo che viene confrontata con le etichette delle voci di menu percorse dall'alto al basso. Notare che questo indice di menu viene considerato dopo tutti gli altri, questo permette che le ricerche per gli elementi di menu etichettati `last`, `active` o `none` possano invece essere interpretate come le stringhe precedenti.

Immagini

Immagini Bitmap/Pixelmap possono essere create derivando una classe di `Tkinter.Image`:

- `BitmapImage` può essere usata per dati bitmap di X11.
- `PhotoImage` può essere usata per bitmap GIF e colori PPM/PGM.

Entrambi i tipi di immagini vengono create sia per mezzo di opzioni di `file` che di `dati` (altre opzioni sono comunque disponibili).

L'oggetto immagine può anche essere usato ogni qualvolta un widget supporta un'opzione `image` (ad esempio etichette, bottoni, menu). In questi casi, Tk non manterrà un riferimento all'immagine. Quando l'ultimo riferimento Python all'oggetto immagine viene rimosso, i dati dell'immagine verranno pure cancellati e Tk visualizzerà un rettangolo vuoto dove veniva usata l'immagine.

16.2 `Tix` — Widgets che estendono Tk

Il modulo `Tix` (Tk Interface Extension) fornisce un ulteriore, ricco, insieme di widget. Sebbene la libreria standard Tk abbia molti utili widget, è lontana dall'essere completa. La libreria `Tix` fornisce la maggior parte dei widget utilizzati comunemente e non presenti in Tk: `HList`, `ComboBox`, `Control` (cioè SpinBox) ed un assortimento di widget scorrevoli. `Tix` include anche molti più Widget di quanti servano generalmente in un vasto ambito di applicazioni: `NoteBook`, `FileEntry`, `PanedWindow`, etc.; ce ne sono più di 40.

Con tutti questi nuovi widget si possono introdurre nuove tecniche di interazione nelle applicazioni, creando interfacce più utili e più intuitive. Si possono progettare applicazioni scegliendo i widget più appropriati per venire incontro alle particolari necessità delle proprie applicazioni e dei propri utenti.

Vedete anche:

Homepage di Tix
(http://tix.sourceforge.net/)
: La pagina Internet di `Tix`. Include collegamenti ad ulteriore documentazione e risorse da scaricare.

Tix Man Pages
(http://tix.sourceforge.net/dist/current/man/)
: La versione on-line delle pagine di manuale e del materiale di riferimento.

Tix Programming Guide
(http://tix.sourceforge.net/dist/current/docs/tix-book/tix.book.html)
: La versione on-line del manuale di riferimento di programmazione.

Applicazioni Tix per lo sviluppo
(http://tix.sourceforge.net/Tide/)
: Le applicazioni Tix per lo sviluppo di programmi Tix e Tkinter. Le applicazioni Tide lavorano sotto Tk o Tkinter ed includono **TixInspect**, un inspector per modificare e correggere da remoto applicazioni Tix/Tk/Tkinter.

16.2.1 Usare Tix

class **`Tix`**(*screenName*[, *baseName*[, *className*]])
Widget di top level di Tix che rappresenta generalmente la finestra principale di un'applicazione. Ha un interprete Tcl associato.

Le classi nel modulo `Tix` sono sottoclassi delle classi del modulo `Tkinter`. Il primo modulo importa il secondo, così che per usare `Tix` con Tkinter, tutto quello che serve è importare un solo modulo. In generale si può semplicemente importare `Tix` e sostituire la chiamata a `Tkinter.Tk` con `Tix.Tk`:

```
import Tix
from Tkconstants import *
root = Tix.Tk()
```

Per usare `Tix` si devono avere i widget `Tix` installati, normalmente a fianco della propria installazione dei widget Tk. Per controllare la propria installazione eseguire il seguente programma:

```
import Tix
root = Tix.Tk()
root.tk.eval('package require Tix')
```

Se questo comando fallisce ci sono dei problemi nell'installazione di Tk, problemi che devono essere risolti prima di procedere ulteriormente. Usare la variabile d'ambiente TIX_LIBRARY per indicare la directory dove è installata la libreria `Tix` ed assicurarsii di avere la libreria dinamica ('tix8183.dll' o 'libtix8183.so') nella stessa directory che contiene la libreria dinamica Tk ('tk8183.dll' o 'libtk8183.so'). Le directory che contengono le librerie dinamiche devono anche contenere un file chiamato 'pkgIndex.tcl' (attenzione a minuscole e maiuscole), che contenga la riga:

```
package ifneeded Tix 8.1 [list load "[file join $dir tix8183.dll]" Tix]
```

16.2.2 I widget di Tix

Tix aggiunge più di 40 classi di widget al repertorio di `Tkinter`. C'è una demo di tutti i widget di `Tix` nella directory 'Demo/tix' della distribuzione standard.

I widget di base

class **`Balloon`**()
Un fumetto che appare sopra un widget per fornire aiuto. Quando l'utente muove il cursore in un widget al quale è legato un widget Balloon, viene mostrata sullo schermo una piccola finestra a scomparsa con un messaggio descrittivo.

class **`ButtonBox`**()
Il widget ButtonBox crea un contenitore di bottoni, così come si usa comunemente per i bottoni `Conferma` `Annulla`.

class **`ComboBox`**()
Il widget ComboBox è simile al controllo combo box di MS Widows. L'utente può selezionare una scelta tra tante, scrivendo in un sottowidget di inserimento, o selezionando in un sottowidget di tipo listbox.

class **`Control`**()
Il widget Control è anche conosciuto come widget `SpinBox`. L'utente può modificare il valore premendo i due bottoni freccia o digitando direttamente il valore nel campo di inserimento. Il nuovo valore verrà controllato rispetto ai limiti superiore ed inferiore definiti dall'utente.

class **`LabelEntry`**()

Il widget LabelEntry riunisce un widget di inserimento ed un'etichetta in un superwidget. Può essere usato per semplificare la creazione di maschere di inserimento dati, del tipo "entry-form".

class `LabelFrame()`
Il widget LabelFrame riunisce un widget frame ed un'etichetta in un superwidget. Per creare widget dentro ad un LabelFrame, si creano dei nuovi widget dipendenti dal widget `frame`, ovvero sottowidget di `frame`, e poi si gestiscono al suo interno.

class `Meter()`
Il widget Meter può essere utilizzato per visualizzare il progresso di un lavoro in background che può avere bisogno di molto tempo per essere eseguito.

class `OptionMenu()`
OptionMenu crea un bottone menu di opzioni.

class `PopupMenu()`
Il widget PopupMenu può essere usato al posto del comando `tk_popup`. Il vantaggio del widget `Tix PopupMenu` è nel richiedere meno codice dell'applicazione per manipolarlo.

class `Select()`
Il widget Select è un contenitore di bottoni. Può essere usato per fornire selezioni di opzioni in stile radio-box o check-box per l'utente.

class `StdButtonBox()`
Il widget StdButtonBox è un gruppo di bottoni standard per finestre di dialogo stile Motif.

Selezionatori di file

class `DirList()`
Il widget DirList widget mostra la lista di una directory, la sua directory madre e le sue sottodirectory. L'utente può scegliere una delle directory visualizzate nella lista o spostarsi su un'altra directory.

class `DirTree()`
Il widget DirTree visualizza un albero di directory, la sua directory madre e le sue sottodirectory. L'utente può scegliere una delle directory visualizzate nella lista o spostarsi su un'altra directory.

class `DirSelectDialog()`
Il widget DirSelectDialog presenta le directory del file system in una finestra di dialogo. L'utente può usare questa finestra di dialogo per navigare attraverso il file system per selezionare la directory desiderata.

class `DirSelectBox()`
`DirSelectBox` è simile al directory-selection box di Motif(TM). È generalmente usato dall'utente per scegliere una directory. DirSelectBox immagazzina le directory selezionate più di recente in un widget `ComboBox`, in modo da poterle selezionare nuovamente in modo rapido.

class `ExFileSelectBox()`
Il widget ExFileSelectBox viene normalmente inserito in un widget tixExFileSelectDialog. Fornisce all'utente un modo comodo per selezionare file. Lo stile del widget `ExFileSelectBox` è molto simile al file dialog standard per i file di MS Windows 3.1.

class `FileSelectBox()`
FileSelectBox è simile al file-selection box standard di Motif(TM). È generalmente usato dall'utente per scegliere un file. FileSelectBox immagazzina i file selezionati più di recente in un widget `ComboBox` così possono essere selezionati nuovamente in modo rapido.

class `FileEntry()`
Il widget FileEntry può essere usato per introdurre il nome di un file. L'utente può scrivere il nome del file manualmente. Oppure può premere il bottone che si trova vicino alla linea di inserimento, il che farà apparire un dialogo di selezione file.

ListBox gerarchiche

class `HList`()
Il widget HList può essere usata per visualizzare qualsiasi dato che abbia una struttura, per esempio l'albero delle directory del file system. Le voci dell'elenco sono indentate e collegate da linee secondo il loro posto nella gerarchia.

class `CheckList`()
Il widget CheckList visualizza una lista di elementi per essere selezionati dall'utente. CheckList funziona in modo simile ai widget ceckbutton o ai radiobutton di Tk, tranne per il fatto che sono in grado di gestire molti più elementi di checkbuttons o radiobuttons.

class `Tree`()
Il widget Tree può essere usato per visualizzare dati gerarchici in forma di albero. L'utente può modificare la vista dell'albero aprendo o chiudendo parti dell'albero.

ListBox tabellare

class `TList`()
Il widget TList può essere usato per per visualizzare dati in forma di tabella. Le voci della lista di un widget `TList` sono simili alle voci del widget listbox in Tk. Le principali differenze sono: 1) il widget `TList` può visualizzare le voci della lista di ingresso in un formato bidimensionale e 2) si possono usare immagini grafiche come diversi colori e font per le voci.

I widget di gestione

class `PanedWindow`()
Il widget PanedWindow permette all'utilizzatore di manipolare le dimensioni di diversi pannelli. I pannelli possono essere sistemati sia verticalmente sia orizzontalmente. L'utente cambia le dimensioni dei pannelli trascinando la maniglia di ridimensionamento tra i due pannelli.

class `ListNoteBook`()
Il widget ListNoteBook è molto simile al widget `TixNoteBook`: può essere usato per visualizzare diverse finestre in uno spazio limitato usando la metafora del taccuino. Il taccuino è diviso in una pila di pagine (finestre). In ogni momento solo una di queste pagine può essere mostrata. L'utente può navigare attraverso queste pagine scegliendo il nome della pagina desiderata nel sottowidget `hlist`.

class `NoteBook`()
Il widget NoteBook può essere usato per visualizzare molte finestre in uno spazio limitato usando la metafora del taccuino. Il taccuino è diviso in una pila di pagine. In ogni momento solo una di queste pagine può essere mostrata. L'utente può navigare attraverso queste pagine scegliendo le "linguette" ("tabs") nella parte alta del widget NoteBook.

I tipi immagine

Il modulo `Tix` aggiunge:

- pixmap capacità di gestire pixmap a tutti i widget `Tix` e `Tkinter` per creare immagini a colori da file XPM.
- Compound Tipi di immagine composta possono essere usate per creare immagini che consistono di linee multiple orizzontali; ogni linea è composta da una serie di elementi (testi, bitmap, immagini o spazi) disposte da sinistra verso destra. Per esempio una immagine composta può essere usata per visualizzare una bitmap ed una stringa di testo contemporaneamente in un widget `Button` di Tk.

Widget vari

class `InputOnly`()
I widget InputOnly servono per ricevere dati dall'utente, e questo può essere fatto con il comando (`bind`) associato (solo in UNIX).

Gestore della geometria del modulo

In aggiunta, `Tix` aumenta `Tkinter` fornendo:

class `Form`()
Il gestore di geometria Form basato sulle regole di unione per tutti i widget di Tk.

16.2.3 I comandi Tix

class `tixCommand`()
I comandi tix forniscono accesso a vari elementi dello stato interno di `Tix` ed al contesto dell'applicazione `Tix`. La maggior parte delle informazioni manipolate da questi metodi riguarda l'intera applicazione, lo schermo o il display piuttosto che una particolare finestra.

Per vedere la configurazione corrente, normalmente si fa:

```
import Tix
root = Tix.Tk()
print root.tix_configure()
```

`tix_configure`([*cnf,*] ***kw*)
Interroga o modifica le opzioni di configurazione del contesto dell'applicazione corrente. Se non vengono specificate opzioni, restituisce un dizionario di tutte le opzioni disponibili. Se viene specificata un'opzione senza il valore, il metodo restituisce una lista che descrive l'opzione citata (questa lista sarà identica alla sottolista del valore restituito se viene specificata l'opzione). Se vengono specificate una o più coppie opzione-valore, allora il metodo modifica l'opzione (le opzioni) assegnando il valore (i valori) dato(i); in questo caso il metodo restituisce una stringa vuota. L'opzione può essere una delle opzioni di configurazione.

`tix_cget`(*option*)
Restituisce il valore corrente dell'opzione di configurazione indicata da *option*. L'opzione deve essere una delle opzioni di configurazione.

`tix_getbitmap`(*name*)
Localizza un file bitmap di nome `name.xpm` o `name` in una delle directory bitmap (vedere il metodo`tix_addbitmapdir()`). Usando `tix_getbitmap()`, si può evitare la codifica pesante del percorso del file di bitmap nella propria applicazione. Se ha successo, restituisce il percorso completo del file di bitmap, preceduto dal carattere '`@`'. Il valore restituito può essere usato per configurare l'opzione `bitmap` dei widget di Tk e Tix.

`tix_addbitmapdir`(*directory*)
Tix mantiene una lista di directory nelle quali i metodi `tix_getimage()` e `tix_getbitmap()` cercheranno i file di immagini. La directory bitmap standard è '`$TIX_LIBRARY/bitmaps`'. Il metodo `tix_addbitmapdir()` aggiunge *directory* a questa lista. Usando questo metodo, i file di immagini di un'applicazione possono anche essere localizzati usando il metodo `tix_getimage()` o `tix_getbitmap()`.

`tix_filedialog`([*dlgclass*])
Restituisce il dialogo di selezione dei file che può essere condiviso tra differenti chiamate da questa applicazione. Questo metodo creerà un widget di dialogo di selezione file quando viene chiamato per la prima volta. Questo dialogo verrà restituito da tutte le successive chiamate a `tix_filedialog()`. Il parametro facoltativo dlgclass può essere passato come stringa per specificare quale tipo di widget di selezione file si desideri. Le possibili opzioni sono `tix`, `FileSelectDialog` o `tixExFileSelectDialog`.

`tix_getimage`(*self, name*)
Colloca un file di immagine di nome '`name.xpm`', '`name.xbm`' o '`name.ppm`' in una delle directory delle

bitmap (vedere il precedente metodo `tix_addbitmapdir()`). Se esiste più di un file con lo stesso nome (ma differente estensione) allora il tipo di immagine viene scelta in accordo alla profondità di colore del display X: xbm sono scelte nei display monocromatici e le immagini a colori vengono scelte nei display a colori. Usando `tix_getimage()`, si può evitare la codifica rigida del percorso dei file di immagini nella propria applicazione. Quando ha successo, questo metodo restituisce il nome della nuova immagine creata, che può essere usato per configurare le opzioni dell'immagine `image` dei widget Tk o Tix.

`tix_option_get`(*name*)
Fornisce le opzioni mantenute dal meccanismo dello schema di Tix.

`tix_resetoptions`(*newScheme, newFontSet*[, *newScmPrio*])
Ripristina lo schema ed il set di font dell'applicazione Tix rispettivamente a *newScheme* e *newFontSet*. Questo influisce solo sui widget creati dopo questa chiamata. Alle volte è utile chiamare il metodo `tix_resetoptions()` prima della creazione di ogni widget in un'applicazione Tix.

Il parametro facoltativo *newScmPrio* può essere passato per ripristinare il livello di priorità delle opzioni Tk impostate dagli schemi di Tix.

A causa del modo nel quale Tk manipola il database delle opzioni di X, dopo che Tix è stato importato e inizializzato, non è possibile ripristinare lo schema di colori e di font usando il metodo `tix_config()`. Deve essere usato il metodo `tix_resetoptions()`.

16.3 `ScrolledText` — Widget per testi con barra di scorrimento

Il modulo `ScrolledText` fornisce una classe con lo stesso nome che implementa un widget di testo di base che ha una barra di scorrimento verticale configurata per apparire sul lato destro. Usare la classe `ScrolledText` risulta molto più semplice che costruire un widget di testo collegato a una barra di scorrimento. Il costruttore è lo stesso di quello della classe `Tkinter.Text`.

Il widget di testo e la barra di scorrimento vengono assemblate in un `Frame` ed i metodi di gestione della geometria `Grid` e `Pack` vengono ereditati dall'oggetto `Frame`. Questo permette di utilizzare direttamente il widget `ScrolledText` per ottenere il più normale comportamento di gestione della geometria.

Se dovessero essere necessari controlli più specifici, sono disponibili i seguenti attributi:

`frame`
La cornice che circonda il testo e la barra di scorrimento.

`vbar`
La barra di scorrimento.

16.4 `turtle` — La grafica della tartaruga per Tk

Il modulo `turtle` fornisce le primitive della grafica della tartaruga sia in forma orientata agli oggetti, sia in forma procedurale. Poiché usa `Tkinter` per la grafica sottostante, è necessario che Python sia installato con il supporto di Tk.

L'interfaccia procedurale usa una penna ed una tela che vengono automagicamente create quando viene chiamata una qualsiasi funzione grafica.

Il modulo `turtle` definisce le seguenti funzioni:

`degrees`()
Imposta a gradi l'unità di misura degli angoli.

`radians`()
Imposta a radianti l'unità di misura degli angoli.

`reset`()
Pulisce lo schermo, riporta la penna al centro e setta le variabili con i valori predefiniti.

`clear`()
Cancella lo schermo.

tracer(*flag*)
Accende o spegne la funzione di tracciatura (a seconda se l'opzione viene impostata a vero o falso). La funzione di tracciatura fa in modo che le linee siano disegnate più lentamente, con l'animazione della freccia lungo il percorso.

forward(*distance*)
Va avanti di *distance* passi.

backward(*distance*)
Va indietro di *distance* passi.

left(*angle*)
Ruota a sinistra di unità *angle*. Le unità predefinite sono gradi, ma possono essere modificate attraverso le funzioni `degrees()` e `radians()`.

right(*angle*)
Ruota a destra di unità *angle*. Le unità predefinite sono gradi, ma possono essere modificate attraverso le funzioni `degrees()` e `radians()`.

up()
Solleva la penna — da questo momento in poi si muove senza disegnare.

down()
Appoggia la penna — da questo momento in poi quando si muove disegna.

width(*width*)
Imposta la larghezza della traccia della penna a *width*.

color(*s*)
color(*(r, g, b)*)
color(*r, g, b*)
Definisce il colore della penna. Nella prima forma, il colore viene indicato da una striga che contiene il nome Tk di un colore. La seconda forma indica il colore come una tupla con i 3 valori RGB, ognuno nell'intervallo [0..1]. Per la terza forma, il colore viene indicato fornendo i valori RGB come tre parametri separati (ognuno nell'intervallo [0..1]).

write(*text*[, *move*])
Scrive il testo *text* nella posizione corrente della penna. Se *move* viene impostato a vero, la penna viene spostata nell'angolo in basso a sinistra del testo. Il valore predefinito di *move* è falso.

fill(*flag*)
La descrizione completa è più complessa, ma si suggerisce di utilizzarlo così: chiamate `fill(1)` prima di disegnare il percorso che volete riempire e chiamate `fill(0)` quando avete finito di tracciare il percorso.

circle(*radius*[, *extent*])
Disegna un cerchio di raggio *radius* il cui centro è *radius* unità a sinistra della tartaruga. *extent* determina quale parte del cerchio viene disegnata: se non indicato viene disegnato un cerchio completo.

Se *extent* non è a cerchio completo, l'estremo finale dell'arco diventa la posizione corrente della penna. L'arco viene disegnato nel verso anti orario, se *radius* è positivo, altrimenti nel verso orario. Nel processo la direzione della tartaruga viene cambiata della quantità indicata da *extent*.

goto(*x, y*)
goto(*(x, y)*)
Va alle coordinate *x*, *y*. Le coordinate possono essere specificate sia con due argomenti separati che con una tupla di due elementi.

Questo modulo importa tutte le funzioni del modulo `math`, vedete la documentazione di `math` per le costanti e le funzioni utili alla grafica della tartaruga.

demo()
Un po' di esempi che utilizzano il modulo.

exception Error
Eccezione sollevata quando un qualunque errore viene causato da questo modulo.

Per esempi, vedete il codice della funzione `demo()`.

Questo modulo definisce le seguenti classi:

class `Pen`()
: Definisce una penna. Tutte le funzioni descritte sopra possono essere chiamate come metodi di una data penna. Il costruttore crea automaticamente una superficie dove disegnare.

class `RawPen`(*canvas*)
: Definisce una penna che disegna sulla superficie di un *canvas*. Questo è utile se volete usare il modulo per creare elementi grafici in un programma "reale".

16.4.1 Gli oggetti Pen e RawPen

Gli oggetti `Pen` e `RawPen` hanno, come metodi per manipolare la penna fornita, tutte le funzioni globali descritte precedentemente, eccetto `demo()`.

L'unico metodo che è più potente come metodo è `degrees()`.

`degrees`([*fullcircle*])
: *fullcircle* ha come valore predefinito 360. Questo permette di dare alla penna qualsivoglia unità di misura per gli angoli: date a fullcircle *fullcircle* 2*π per angoli in radianti o 400 per angoli in gradi centesimali (gradians).

16.5 Idle

Idle è l'IDE di Python, realizzata con il toolkit della GUI `Tkinter`.

IDLE ha le seguenti caratteristiche:

- scritto al 100% in Python, usando il toolkit per la GUI `Tkinter`.
- multi piattaforma: funziona in Windows e UNIX (in Mac OS, attualmente ci sono problemi con Tcl/Tk)
- editor multi finestra con più livelli di annullamento, colorazione sintattica e molte altre funzioni, ad esempio l'indentazione semplificata ed i suggerimenti sulle chiamate
- finestra di shell di Python (cioè interprete interattivo)
- debugger (non completo, ma si possono impostare breakpoint, ispettori ed esecuzione passo passo)

16.5.1 I menu

Questa parte è stata modificata rispetto all'originale in lingua inglese per adeguarla alla versione 1.0.3 di IDLE.

Menu file

New window Crea una nuova finestra di editing di un file

Recent files... Elenco degli ultimi file aperti

Open... Apre un file esistente

Open module... Apre un modulo esistente (lo ricerca in sys.path).

Class browser Mostra classi e metodi del file corrente

Path browser Mostra le directory, i moduli, le classi e i metodi di sys.path

Save Salva la finestra corrente nel file associato (file non salvati hanno un * prima e dopo il titolo della finestra).

Save As... Salva la finestra corrente in un nuovo file che diventa il nuovo file associato

Save Copy As... Salva la finestra corrente in un differente file senza cambiare il file associato

Close Chiude la finestra corrente (chiede di salvarla se modificata)

Exit Chiude tutte le finestre e termina IDLE (chiede di salvare le finestre modificate)

Menu edit

Undo Annulla l'ultima modifica alla finestra corrente (massimo 1000 cambiamenti)

Redo Ripristina l'ultima modifica annullata nella finestra corrente

Cut Copia la selezione corrente negli appunti del sistema; poi cancella la selezione

Copy Copia la selezione negli appunti del sistema

Paste Inserisce nella finestra il contenuto degli appunti del sistema

Select All Seleziona l'intero contenuto del buffer dell'editor

Find... Apre una finestra di dialogo di ricerca con diverse opzioni

Find again Ripete l'ultima ricerca

Find selection Ricerca una stringa in una selezione

Find in Files... Apre una finestra di dialogo di ricerca per cercare all'interno di file

Replace... Apre una finestra di dialogo di ricerca e sostituzione

Go to line Chiede un numero di riga e sposta il cursore su quella riga

Expand word Espande la parola che si è scritto in un'altra parola nello stesso buffer; ripetendolo dà diverse espansioni

Menu Shell (solo nella finestra di Shell)

View Last Restart Rende visibile l'ultimo RESTART

Restart Shell Riavvia l'interprete con l'ambiente ripulito

16.5.2 Menu Format (solo nelle finestre di editing)

Indent region Sposta a destra di 4 spazi le righe selezionate

Dedent region Sposta a sinistra di 4 spazi le linee selezionate

Comment out region Inserisce ## all'inizio delle righe selezionate

Uncomment region Toglie # o ## dall'inizio delle righe selezionate

Tabify region Traduce gli *spazi* iniziali delle righe in tabulazioni

Untabify region Traduce *tutte* le tabulazioni nell'equivalente numero di spazi

Format Paragraph Formatta il paragrafo corrente

16.5.3 Menu Run (solo nelle finestre di editing)

Python Shell Apre o porta in primo piano la finestra della shell di Python

Check Module Esegue un controllo sintattico del modulo

Run Module Esegue il file corrente nello spazio dei nomi *__main__*

16.5.4 Menu options

Configure IDLE Apre un dialogo di configurazione. Si possono modificare i font, l'indentazione, l'azione dei tasti ed i colori. Si possono stabilire le preferenze e specificare altre fonti di Help

Revert to Default Settings Ristabilisce le opzioni originali. Attualmente non viene implementato, semplicemente cancella il file '.idlerc'

16.5.5 Menu Windows

Zoom Height Cambia la dimensione della finestra da normale (24x80) alla massima altezza

Il resto di questo menu elenca i nomi di tutte le finestre aperte; selezionarne una per portarla in primo piano (deiconizzarla se necessario).

Menu Debug (solo nella finestra della Shell di Python)

Go to file/line Ricava dalla riga dove si trova il cursore il nome di un file ed un numero di riga, apre il file e visualizza la riga (NdT: La riga deve essere nella forma: 'File «nome_file», riga <numero>'

Debugger toggle Esegue le istruzioni in una shell all'interno del debugger

Stack viewer Mostra la traccia dello stack dell'ultima eccezione sollevata

Auto-open Stack viewer (toggle) Apre automaticamente la finestra dello stack

16.5.6 Menu Help

About IDLE Versione, copyright, licenza, collaboratori

IDLE Help Visualizza un testo di aiuto per l'uso di IDLE

Python Docs Permette di accedere alla documentazione locale di Python se installata. Altrimenti rimanda al sito ufficiale di Python

16.5.7 Comandi di base per l'editing e la navigazione

- `Backspace` cancella a sinistra; `Canc` cancella a destra
- I tasti freccia e `pag su`/`pag giù` permettono di muoversi nel testo
- `Inizio`/`Fine` mandano all'inizio/fine della riga
- `Ctrl-Inizio`/`Ctrl-Fine` mandano all'inizio/fine del file
- Diversi comandi da tastiera di **Emacs** possono funzionare, inclusi: `C-B`, `C-P`, `C-A`, `C-E`, `C-D`, `C-L`

Indentazione automatica

Dopo un comando che agisce su un blocco, la riga seguente viene indentata di 4 spazi (nella finestra di shell di Python un tabulatore). Dopo certe parole chiave (break, return ecc.) la riga seguente viene de-indentata. Quando il cursore si trova all'inizio di una riga indentata, `Backspace` cancella fino a 4 spazi, se ci sono. `Tab` inserisce da 1 a 4 spazi (nella finestra di shell di Python inserisce un carattere di tabulazione). Vedere anche i comandi indent/dedent di una regione nel menu Edit.

Finestra della shell di Python

- `C-C` interrompe l'esecuzione di un comando
- `C-D` chiude i file; se premuto al prompt '`»> `', chiude la finestra
- `Alt-p` richiama il comando precedente che corrisponde a quanto scritto
- `Alt-n` richiama il comando seguente
- `Return` quando il cursore si trova su un comando precedente richiama quel comando
- `Alt-/` (Espansione di parola) è utile anche qui

16.5.8 Colori della sintassi

La colorazione viene applicata in un "thread" in background, così potreste vedere, occasionalmente, il testo non colorato. Per cambiare lo schema del colore, modificate la sezione `[Colors]` in 'config.txt'.

Colori della sintassi di Python: Keywords arancione

Strings verde

Comments rosso

Definitions blu

Colori della shell: Output della console marrone

stdout blu

stderr verde scuro

stdin nero

Uso da riga di comando

```
idle.py [-c command] [-d] [-e] [-s] [-t title] [arg] ...

-c command  esegue command
-d          abilita il debugger
-e          modalità editor; gli argomenti contengono i nomi dei file
            che verranno modificati
-s          esegue prima $IDLESTARTUP o $PYTHONSTARTUP
-t title    stabilisce il titolo della finestra della shell
```

Se ci sono argomenti:

1. Se viene usata l'opzione **-e**, gli argomenti sono file da aprire per la modifica e `sys.argv` riporta gli argomenti passati a IDLE stesso.

2. Altrimenti se viene usata l'opzione **-c**, tutti gli argomenti vengono posti in `sys.argv[1:...]`, con `sys.argv[0]` posto uguale a `'-c'`.

3. Altrimenti, se non viene utilizzata l'opzione **-e** né **-c**, il primo argomento è uno script che viene eseguito con i rimanenti argomenti in `sys.argv[1:...]` e `sys.argv[0]` posto uguale al nome dello script. Se il nome dello script è '-', non viene eseguito uno script, ma viene avviata una sessione interattiva di Python; gli argomenti sono sempre disponibili in `sys.argv`.

16.6 Altri pacchetti per Interfaccia Grafiche Utenti

C'è un buon numero di insiemi di widget di estensione per `Tkinter`.

Python megawidgets
(http://pmw.sourceforge.net/)
: è un toolkit per costruire widget di alto livello compositi in Python usando il modulo `Tkinter`. Consiste in un insieme di classi base ed in una libreria di megawidget flessibili ed estendibili costruiti su queste basi. Questi megawidget includono notebooks, comboboxes, selection widgets, paned widgets, scrolled widgets, dialog windows, ecc. Con l'interfaccia BLT a Pmw.Blt sono disponibili anche i comandi busy, graph, stripchart, tabset e vector.

  L'idea iniziale di Pmw é stata presa dalle estensioni `itcl` di Tk `[incr Tk]` da Michael McLennan `[incr Widgets]` da Mark Ulferts. Molti di questi megawidget sono traduzioni dirette da itcl a Python. Vi si offrono molti degli insiemi di widgets forniti da `[incr Widgets]`, in modo completo almeno quanto Tix, ma senza il veloce widget di Tix `HList`, per il disegno ad albero.

Tkinter3000 Widget Construction Kit (WCK)
(http://tkinter.effbot.org/)
: è una libreria che permette di scrivere nuovi widget Tkinter in puro Python. La struttura WCK permette un pieno controllo sulla creazione, la configurazione, l'aspetto e la gestione degli eventi dei widget. I widget WCK possono essere molto veloci e leggeri, poiché operano direttamente nelle strutture dati di Python, senza dover trasferire dati attraverso lo strato Tk/Tcl.

Tk non è la sola GUI per Python, ma è comunque quella usata più frequentemente.

wxWindows
(http://www.wxwindows.org)
: è un toolkit di GUI che combina le migliori caratteristiche di Qt, Tk, Motif e GTK+ in un pacchetto potente ed efficiente. È implementato in C++. wxWindows supporta due stili dell'implementazione UNIX: GTK+ e Motif, e sotto Windows ha l'aspetto delle standard Microsoft Foundation Classes (MFC), poiché usa i widget di Win32. È una classe di interfaccia Python indipendente da Tkinter.

  wxWindows è più ricca di widget rispetto a `Tkinter`, con il suo sistema di help, sofisticati wisualizzatori di documenti HTML, di immagini ed altri widget specializzati, estesa documentazione e capacità di stampa.

PyQt
: PyQt è un **sip**-collegamento al toolkit di Qt. Qt è un ampio GUI toolkit in C++ ed è disponibile per UNIX, Windows e Mac OS X. **sip** è uno strumento per generare collegamenti per librerie C++ e classi Python ed è progettato specificamente per Python. È disponibile un manuale on line presso http://www.opendocspublishing.com/pyqt/ (gli errori si trovano presso http://www.valdyas.org/python/book.html).

PyKDE
(http://www.riverbankcomputing.co.uk/pykde/index.php)
: PyKDE è un **sip**-collegamento alle librerie del desktop KDE. KDE è un ambiente desktop per i computer UNIX; i componenti grafici sono basati su Qt.

FXPy
(http://fxpy.sourceforge.net/)
: è un modulo di estensione Python che fornisce un'interfaccia alle GUI *FOX*. FOX è un Toolkit basato su C++, semplice ed efficiente, per sviluppare con facilità ed efficienza Interfacce Grafiche Utente. Offre una collezione di controlli estesa ed ampliabile ed offre lo stato dell'arte per strumenti come drag and drop,

selezione, come pure i widget OpenGL per la manipolazione della grafica 3D. FOX implementa anche icone, immagini e dispositivi di comodo per l'utente, come un help della riga di stato ed i suggerimenti.

Sebbene FOX offra già un'ampia collezione di di controlli, facilita ai programmatori l'uso di C++ per realizzare controlli ed elementi della GUI, semplicemente utlizzando controlli già esistenti e creando classi derivate che aggiungono o ridefiniscono i comportamenti desiderati.

PyGTK
(http://www.daa.com.au/~james/software/pygtk/)
: è un insieme di collegamenti per l'insieme di widget di GTK. Fornisce un'interfaccia orientata agli oggetti che è leggermente più di alto livello del C stesso. Fa automaticamente tutte le trasformazioni di tipo ed i conteggi di riferimento che si dovrebbero fare con le API del C. Ci sono anche bindings a GNOME ed è disponibile un tutorial.

CAPITOLO

DICIASSETTE

Esecuzione limitata

In Python 2.3 questi moduli sono stati disabilitati a seguito di alcuni problemi di sicurezza conosciuti e non facilmente sistemabili. I moduli vengono ancora qui documentati per aiutare la lettura di vecchio codice che utilizza i moduli `rexec` e `Bastion`.

L'esecuzione limitata è l'infrastruttura base in Python che permette l'isolamento di codice affidabile e inaffidabile. L'infrastruttura si basa sul concetto per cui il codice Python affidabile (il *supervisore*) possa creare una "cella di contenimento" (o ambiente) con permessi limitati ed eseguire il codice inaffidabile all'interno di questa cella. Il codice inaffidabile non può uscire dalla sua cella e può iteragire con le risorse di sistema sensibili solamente attraverso le interfacce definite e gestite dal supervisore. Il termine "esecuzione limitata" viene preferito al termine "Python sicuro" poiché la vera sicurezza è difficile da definire e viene determinata dalla maniera in cui viene creato l'ambiente limitato. Notate che gli ambienti limitati possono venire annidati, con le celle interne in grado di creare sotto celle con privilegi minori, mai maggiori.

Un aspetto interessante del modello di esecuzione limitata di Python è che l'interfaccia presentata al codice inaffidabile generalmente possiede gli stessi nomi di quella presentata al codice affidabile. Perciò non è necessario conoscere interfacce speciali per scrivere codice da utilizzare in un ambiente limitato. Poiché l'esatta natura della cella di contenimento viene definita dal supervisore, si possono imporre differenti restrizioni, a seconda dall'applicazione. Per esempio, può essere ritenuto "sicuro" per il codice inaffidabile leggere qualunque file in una directory specifica, ma mai scrivere un file. In questo caso, il supervisore può ridefinire la funzione built-in `open()` in modo che sollevi un'eccezione ogniqualvolta il parametro *mode* sia *'w'*. Può anche effettuare un'operazione tipo `chroot()` sul parametro *filename*, in modo che la root sia sempre relativa ad alcune aree "ingabbiate" sicure del filesystem. In questo caso, il codice inaffidabile vedrebbe ancora nel suo ambiente una funzione built-in `open()`, con la stessa interfaccia di chiamata. Anche le semantiche sarebbero identiche, sollevando l'eccezione `IOError` nel caso il supervisore determini l'utilizzo di un parametro non permesso.

Il run-time di Python determina se un particolare blocco di codice venga eseguito in modalità di esecuzione limitata in base all'identità dell'oggetto `__builtins__` nelle sue variabili globali: se questo è (il dizionario de) il modulo standard `__builtin__`, il codice viene ritenuto come non limitato, altrimenti viene considerato come limitato.

Il codice Python eseguito in modalità limitata deve affronare un certo numero di restrizioni disegnate per prevenire la sua fuga dalla cella di contenimento. Per esempio, l'attributo funzione `func_globals` e l'attributo di istanza e classe `__dict__` non sono disponibili.

Sono due i moduli che forniscono la struttura per impostare gli ambienti di esecuzione limitata:

`rexec`	Infrastruttura di base per l'esecuzione limitata.
`Bastion`	Fornire un accesso limitato agli oggetti.

Vedete anche:

Grail Home Page
(http://grail.sourceforge.net/)

Grail, un browser Internet scritto in Python, utilizza questi moduli per supportare gli applet Python. Maggiori informazioni sull'uso della modalità di esecuzione limitata di Python in Grail è disponibile sul sito web.

17.1 rexec — Infrastruttura per l'esecuzione limitata

Modificato nella versione 2.3: Disabled module.

La documentazione è stata lasciata per aiutare nella lettura di vecchio codice che utilizza questo modulo.

Questo modulo contiene la classe `RExec`, che supporta i metodi `r_eval()`, `r_execfile()`, `r_exec()` e `r_import()`, che sono versioni limitate delle funzioni standard Python `eval()`, `execfile()` e delle istruzioni `exec` ed `import`. Il codice eseguito in questo ambiente limitato avrà accesso solamente ai moduli ed alle funzioni che vengono considerate sicure; potete derivare `RExec` per aggiungere o rimuovere le funzionalità che desiderate.

Anche se il modulo `rexec` è disegnato per comportarsi come descritto sotto, esso possiede alcune vulnerabilità note che potrebbero venire sfruttate da codice scritto a proposito. Perciò non dovreste farvi affidamento in situazioni che richiedono sicurezza "pronta per la produzione". In queste situazioni, può essere necessaria l'esecuzione tramite sotto processi o un attento "cleansing" di entrambi i codici e dei dati da elaborare . Alternativamente, sarà molto apprezzato l'aiuto nel correggere le vulnerabilità note di `rexec`.

Note: La classe `RExec` può impedire che il codice effettui operazioni non sicure come leggere o scrivere file su disco o utilizzare socket TCP/IP. Comunque, non protegge da codice che utilizza grandi quantità di memoria o tempo del processore.

class `RExec`([*hooks*[, *verbose*]])
Restituisce un'istanza della classe `RExec`.

hooks è un'istanza della classe `RHooks` o di una sua sotto classe. Se viene omessa o è `None`, viene istanziata la classe predefinita `RHooks`. Ogni volta che il modulo `rexec` cerca un modulo (anche uno built-in) o legge il codice di un modulo, non utilizza realmente il file system da sè. Piuttosto, richiama i metodi di un'istanza `RHooks` che è stata passata al costruttore o creata da questo. (In realtà l'oggetto `RExec` non effettua queste chiamate — queste vengono eseguite da un oggetto loader a livello di modulo che è parte dell'oggetto `RExec`. Questo permette un diverso livello di flessibilità, che può essere utile quando si cambia il meccanismo di `import` in un ambiente limitato).

Fornento un oggetto `RHooks` alternativo, possiamo controllare gli accessi al file system effettuati per importare un modulo, senza modificare il vero algoritmo che controlla l'ordine in cui questi accessi vengono effettuati. Per esempio, potete sostituire un oggetto `RHooks` che passa tutte le richieste del filesystem ad un file server, attraverso qualche meccanismo RPC, come ILU. Il loader delle applet di Grail utilizza questo tecnica per supportare l'import delle applet da un URL per una directory.

Se *verbose* è vero, vengono generate informazioni addizionali di debug sullo standard output.

È importante sapere che il codice in esecuzione in un ambiente limitato può ancora chiamare la funzione `sys.exit()`. Per non permettere al codice limitato di terminare l'interprete, proteggete sempre le chiamate che eseguono codice limitato con un'istruzione `try`/`except` che intercetti l'eccezione `SystemExit`. Rimuovere la funzione `sys.exit()` dall'ambiente limitato non è sufficiente – il codice limitato può ancora usare `raise SystemExit`. Rimuovere `SystemExit` non è un'opzione ragionevole; alcuni codici di libreria ne fanno uso e non funzionerebbero senza.

Vedete anche:

Grail Home Page
(http://grail.sourceforge.net/)
Grail è un browser web scritto interamente in Python. Utilizza il modulo `rexec` come base per supportare applet Python e può essere visionato come esempio di utilizzo di questo modulo.

17.1.1 Oggetti RExec

Le istanze di `RExec` supportano i seguenti metodi:

`r_eval`(*code*)
code deve essere una stringa contenente un'espressione Python o un oggetto codice compilato, che verrà valutato nel modulo `__main__` dell'ambiente limitato. Viene restituito il valore dell'espressione o dell'oggetto codice.

r_exec(*code*)
code deve essere una stringa contenente una o più righe di codice Python o un oggetto codice compilato, che verrà eseguito nel modulo `__main__` dell'ambiente limitato.

r_execfile(*filename*)
Esegue il codice Python contenuto nel file *filename* nel modulo `__main__` dell'ambiente limitato.

I metodi il cui nome inizia con '`s_`' sono simili alle funzioni che iniziano con '`r_`', ma al codice verrà garantito l'accesso alla versione limitata dei flussi standard di I/O `sys.stdin`, `sys.stderr` e `sys.stdout`.

s_eval(*code*)
code deve essere una stringa contenente un'espressione Python, che verrà valutata nell'ambiente limitato.

s_exec(*code*)
code deve essere una stringa contenente una o più righe di codice Python, che verranno eseguite nell'ambiente limitato.

s_execfile(*code*)
Esegue il codice Python contenuto nel file *filename* nell'ambiente limitato.

Gli oggetti `RExec` devono anche fornire supporto per vari metodi che verranno implicitamente richiamati dal codice eseguito nell'ambiente ristretto. La sovrascrittura di questi metodi in una sotto classe è un tecnica utilizzata per alterare la condotta imposta da un ambiente limitato.

r_import(*modulename*[, *globals*[, *locals*[, *fromlist*]]])
Importa il modulo *modulename*, sollevando un'eccezione `ImportError` se il modulo viene considerato insicuro.

r_open(*filename*[, *mode*[, *bufsize*]])
Metodo eseguito quando la funzione `open()` viene chiamata in un ambiente limitato. Gli argomenti sono identici a quelli di `open()` e deve essere restituito un oggetto di tipo file (o l'istanza di una classe compatibile con gli oggetti file). Il comportamento predefinito di `RExec` è quello di permettere l'apertura di tutti i file in lettura, ma impedire ogni accesso in scrittura. Vedete l'esempio sotto per un'implementazione meno restrittiva di `r_open()`.

r_reload(*module*)
Ricarica l'oggetto modulo *module*, rianalizzandolo e reinizializzandolo.

r_unload(*module*)
Scarica dalla memoria l'oggetto modulo *module* (lo rimuove dal dizionario `sys.modules` dell'ambiente limitato).

E gli equivalenti con l'accesso ai flussi di I/O standard limitati:

s_import(*modulename*[, *globals*[, *locals*[, *fromlist*]]])
Importa il modulo *modulename* sollevando un'eccezione `ImportError` se il modulo viene considerato insicuro.

s_reload(*module*)
Ricarica l'oggetto modulo *module*, rianalizzandolo e reinizializzandolo.

s_unload(*module*)
Scarica dalla memoria il modulo *module*.

17.1.2 Definire gli ambienti limitati

La classe `RExec` possiede i seguenti attributi di classe, che vengono utilizzati dal metodo `__init__()`. La modifica di questi attributi su un'istanza esistente non causa alcun effetto; create invece una sotto classe di `RExec` ed assegnate nuovi valori nella definizione della classe. Le istanze delle nuove classi utilizzeranno i nuovi valori. Tutti questi attributi sono tuple o stringhe.

nok_builtin_names
Contiene i nomi delle funzioni built-in che *non* saranno disponibili ai programmi in esecuzione nell'ambiente limitato. Il valore per `RExec` è `('open', 'reload', '__import__')`. (Questo fornisce le eccezioni, poiché la maggioranza delle funzioni built-in sono potenzialmente dannose. Una sotto classe

che vuole sovrascrivere questa variabile dovrebbe probabilmente iniziare con il valore della classe base e concatenare funzioni proibite addizionali – quando nuove funzioni built-in pericolose verranno aggiunte a Python, verranno aggiunte anche a questo modulo).

`ok_builtin_modules`
Contiene i nomi dei moduli built-in che possono venire importati in modo sicuro. Il valore per `RExec` è `('audioop', 'array', 'binascii', 'cmath', 'errno', 'imageop', 'marshal', 'math', 'md5', 'operator', 'parser', 'regex', 'rotor', 'select', 'sha', '_sre', 'strop', 'struct', 'time')`. Si applica una nota simile circa la sovrascrittura di queste variabili — utilizzate il valore della classe base come punto di partenza.

`ok_path`
Contiene le directory in cui si andrà a cercare quando viene effettuato un `import` nell'ambiente limitato. Il valore per `RExec` è lo stesso di `sys.path` (al momento in cui il modulo viene caricato) del codice non limitato.

`ok_posix_names`
Contiene i nomi delle funzioni nel modulo `os` che saranno disponibili ai programmi in esecuzione nell'ambiente limitato. Il valore per `RExec` è `('error', 'fstat', 'listdir', 'lstat', 'readlink', 'stat', 'times', 'uname', 'getpid', 'getppid', 'getcwd', 'getuid', 'getgid', 'geteuid', 'getegid')`.

`ok_sys_names`
Contiene i nomi delle funzioni e delle variabili nel modulo `sys` che saranno disponibili ai programmi in esecuzione nell'ambiente limitato. Il valore per `RExec` è `('ps1', 'ps2', 'copyright', 'version', 'platform', 'exit', 'maxint')`.

`ok_file_types`
Contiene i tipi di file dai quali è permesso caricare i moduli. Ogni tipo di file è una costante intera definita nel modulo `imp`. I valori più significativi sono `PY_SOURCE`, `PY_COMPILED` e `C_EXTENSION`. Il valore per `RExec` è `(C_EXTENSION, PY_SOURCE)`. Non è consigliabile aggiungere `PY_COMPILED` in una sotto classe; un attaccante potrebbe uscire dalla modalità di esecuzione limitata mettendo un file byte-compilato ('.pyc') forgiato dovunque nel file system, per esempio scrivendolo in '/tmp' o caricandolo nella directory '/incoming' del vostro server FTP pubblico.

17.1.3 Un esempio

Mettiamo di voler imporre una condotta un pò meno rigida rispetto alla classe `RExec` predefinita. Per esempio, se vogliamo permettere la scrittura di file in '/tmp', possiamo derivare la classe `RExec`:

```
class TmpWriterRExec(rexec.RExec):
    def r_open(self, file, mode='r', buf=-1):
        if mode in ('r', 'rb'):
            pass
        elif mode in ('w', 'wb', 'a', 'ab'):
            # controlla il nome del file: deve iniziare con /tmp/
            if file[:5]!='/tmp/':
                raise IOError, "can't write outside /tmp"
            elif (string.find(file, '/../') >= 0 or
                 file[:3] == '../' or file[-3:] == '/..'):
                raise IOError, "'..' in filename forbidden"
        else: raise IOError, "Illegal open() mode"
        return open(file, mode, buf)
```

Notate che il codice sopra proibirà occasionalmente l'apertura di un nome di file perfettamente valido; per esempio, il codice in un ambiente limitato non sarà in grado di aprire un file chiamato '/tmp/foo/../bar'. Per correggere questo difetto, il metodo `r_open()` dovrebbe semplificare il nome del file a '/tmp/bar', che richiederebbe l'estrazione del nome del file e l'esecuzione di varie operazioni su questo. Nei casi in cui è in gioco la sicurezza, può essere preferibile scrivere codice più semplice che sia qualche volta particolarmente restrittivo, invece di un

codice più generale ma anche molto complesso che possa contenere un subdolo problema di sicurezza.

17.2 `Bastion` — Limitare l'accesso agli oggetti

Modificato nella versione 2.3: Disabled module.

La documentazione è stata lasciata per aiutare nella lettura di vecchio codice che utilizza questo modulo.

Come da dizionario, un bastione (NdT: bastion) è "un'area o una posizione fortificata" oppure "qualcosa che viene considerato una fortezza". È un nome adatto per questo modulo, che fornisce un modo per proibire l'accesso ad alcuni attributi di un oggetto. Deve essere sempre utilizzato assieme al modulo `rexec`, in modo da permettere ai programmi che operano in modalità limitata di accedere ad alcuni attributi sicuri di un oggetto, mentre si impedisce l'accesso ad altri attributi non sicuri.

`Bastion`(*object*[, *filter*[, *name*[, *class*]]])

Protegge l'oggetto *object*, restituendo un bastione per l'oggetto. Ogni tentativo di accedere ad uno degli attributi dell'oggetto dovrà venire approvato dalla funzione *filter*; se l'accesso è vietato verrà sollevata un'eccezione `AttributeError`.

Se presente, *filter* deve essere una funzione che accetta una stringa contenente il nome di un attributo e restituisce vero se è consentito l'accesso a quest'ultimo; se *filter* restituisce falso, l'accesso viene proibito. Il filtro predefinito nega l'accesso a tutte le funzioni che iniziano con il carattere di di trattino basso ('_'). La stringa di rappresentazione di bastion sarà '`<Bastion for` *name*`>`' se viene fornito il valore di *name*; altrimenti verrà utilizzato '`repr(`*object*`)`'.

class, se presente, dovrebbe essere una sotto classe di `BastionClass`; vedete il codice in 'bastion.py' per i dettagli. Solo raramente sarà necessaria una sovrascrittura della classe `BastionClass`.

class `BastionClass`(*getfunc*, *name*)

Classe che implementa realmente gli oggetti bastion. Questa è la classe predefinita utilizzata da `Bastion()`. Il parametro *getfunc* è una funzione che quando viene chiamata con il nome di un attributo come unico argomento restituisce il valore dell'attributo da esporre all'ambiente di esecuzione limitata. *name* viene utilizzato per realizzare il metodo `repr()` dell'istanza della classe `BastionClass`.

CAPITOLO

DICIOTTO

Servizi del linguaggio Python

Python fornisce numerosi moduli per aiutarvi a lavorare con il linguaggio Python. Questi moduli supportano l'uso dei simboli, l'analisi del codice, l'analisi sintattica, il disassemblamento del bytecode e varie altre utilità.

Questi moduli includono:

parser	Accesso agli alberi di analisi del codice sorgente Python.
symbol	Costanti che rappresentano i nodi interni degli alberi di analisi di Python.
token	Costanti che rappresentano i nodi terminali degli alberi di analisi di Python.
keyword	Verifica se la stringa è una parola chiave di Python.
tokenize	Scanner lessicale per il codice sorgente Python.
tabnanny	Strumento per rilevare problemi relativi agli spazi vuoti in file sorgenti Python nell'albero di directory.
pyclbr	Supporto all'estrazione di informazioni per il browser delle classi Python.
py_compile	Compilazione di file sorgenti Python in file bytecode.
compileall	Strumento per compilare file sorgenti in bytecode all'interno di un albero di directory.
dis	Disassemblatore per il bytecode Python.
distutils	Supporto per la costruzione e l'installazione dei moduli Python in un'installazione Python già esistente.

18.1 `parser` — Accesso agli alberi di analisi di Python

Il modulo `parser` fornisce un'interfaccia al parser interno ed al compilatore bytecode di Python. Lo scopo primario di questa interfaccia è quello di permettere al codice Python di modificare l'albero di analisi di una espressione Python e di creare da essa del codice eseguibile. Tutto ciò è meglio che provare ad analizzare e modificare un frammento arbitrario di codice di Python come se fosse una stringa, perché l'analisi viene svolta nella medesima maniera di quella del codice che forma l'applicazione. È anche più veloce.

Ci sono poche cose nuove da notare su questo modulo importanti al fine di utilizzare al meglio le strutture dati create. Pur non essendo un tutorial sulla modifica degli alberi di analisi del codice Python vengono comunque presentati alcuni esempi d'uso del modulo `parser`.

Molto importante, viene richiesta una buona comprensione della grammatica di Python elaborata dal parser interno. Per delle informazioni complete sulla sintassi del linguaggio, fate riferimento al *Manuale di riferimento di Python*. Il parser stesso viene creato da una specifica grammaticale definita nella distribuzione standard Python nel file 'Grammar/Grammar'. Quando vengono creati dalle funzioni `expr()` o `suite()` descritte sotto, gli alberi di analisi conservati negli oggetti AST creati da questo modulo rappresentano l'output reale del parser interno. Gli oggetti AST creati da `sequence2ast()` simulano fedelmente queste strutture. Fate attenzione che i valori delle sequenze che vengono considerate "corrette" varieranno da una versione all'altra di Python, come anche la grammatica formale del linguaggio. In ogni caso, trasporre codice da una versione di Python ad un'altra come testo sorgente permetterà sempre la creazione di alberi di analisi corretti nella versione di destinazione, con la sola restrizione che migrando verso una versione più vecchia non verranno supportati molti dei costrutti linguistici più recenti. Gli alberi di analisi non sono in genere compatibili da una versione all'altra, mentre il codice sorgente è stato sempre compatibile con le versione successive.

Ogni elemento delle sequenze restituite da `ast2list()` o `ast2tuple()` ha una forma semplice. Le sequenze che rappresentano elementi non terminali nella grammatica hanno sempre lunghezza più grande di uno. Il primo elemento è un intero che identifica una produzione nella grammatica. Questi interi prendono nomi simbolici

nel file header C ‘Include/graminit.h’ e nel modulo Python `symbol`. Ogni elemento aggiuntivo della sequenza rappresenta un componente della produzione come riscontrato nella stringa di input: ci sono sempre sequenze che hanno la stessa forma dei genitori. Un aspetto importante di questa struttura che dovrebbe essere notato è che le parole chiave usate per identificare il tipo del nodo genitore, come la parola chiave `if` in un `if_stmt`, vengono incluse nell’albero dei nodi senza uno speciale trattamento. Per esempio, la parola chiave `if` viene rappresentata dalla tupla `(1, 'if')`, dove `1` è il valore numerico associato con tutti i token `NAME`, inclusi i nomi delle variabili e delle funzioni definite dall’utente. In una forma alternativa restituita quando viene richiesta l’informazione su di un numero di riga, lo stesso simbolo può venire rappresentato come `(1, 'if', 12)`, dove il `12` rappresenta il numero di riga in cui è stato trovato il simbolo terminale.

Gli elementi terminali vengono rappresentati nello stesso modo, ma senza nessun elemento figlio e l’aggiunta del testo sorgente con il quale era stato identificato. L’esempio della parola chiave `if` lo descrive. I vari tipi di simboli terminali vengono definiti nel file header C ‘Include/token.h’ ed nel modulo Python `token`.

Non vengono richiesti gli oggetti AST per supportare le funzionalità di questo modulo, ma vengono forniti per tre scopi: permettere ad un’applicazione di ammortizzare il costo dell’elaborazione di alberi di analisi complessi, fornire una rappresentazione dell’albero di analisi che conservi spazio in memoria quando viene confrontato con la rappresentazione di liste o tuple Python, e semplificare la creazione di moduli aggiuntivi in C che manipolano gli alberi di analisi. Si può creare una semplice classe “wrapper” in Python per nascondere l’uso di oggetti AST.

Il modulo `parser` definisce funzioni per pochi scopi distinti. Gli scopi più importanti sono la creazione di oggetti AST e la conversione di oggetti AST in altre rappresentazioni, come alberi di analisi ed oggetti di codice compilato, ma ci sono anche funzioni che servono ad interrogare il tipo di albero di analisi rappresentato da un oggetto AST.

Vedete anche:

Modulo `symbol` (sezione 18.2):
Costanti utili che rappresentano i nodi interni dell’albero di analisi.

Modulo `token` (sezione 18.3):
Costanti utili che rappresentano i nodi foglia dell’albero di analisi e funzioni per testare i valori dei nodi.

18.1.1 Creazione di oggetti AST

Gli oggetti AST possono venire creati dal codice sorgente o da un albero di analisi. Quando si crea un oggetto AST da sorgente, vengono usate differenti funzioni per creare le forme `'eval'` ed `'exec'`.

`expr`(*source*)
La funzione `expr()` analizza il parametro *source* come fosse, per così dire, un input a ‘`compile(`*source*`, 'file.py', 'eval')`’. Se l’analisi va a buon fine, viene creato un oggetto AST per contenere la rappresentazione interna dell’albero di tale analisi, altrimenti viene sollevata un’eccezione appropriata.

`suite`(*source*)
La funzione `suite()` analizza il parametro *source* come fosse, per così dire, un input a ‘`compile(`*source*`, 'file.py', 'exec')`’. Se l’analisi va a buon fine, viene creato un oggetto AST per contenere la rappresentazione interna dell’albero di tale analisi, altrimenti viene sollevata un’eccezione appropriata.

`sequence2ast`(*sequence*)
Questa funzione accetta un albero di analisi rappresentato come una sequenza e ne costruisce, se possibile, una rappresentazione interna. Se inoltre essa verifica che l’albero è conforme alla grammatica Python e che tutti i nodi sono di tipo valido nella versione Python in uso, viene creato un oggetto AST dalla rappresentazione interna e restituito al chiamante. Se si verifica un problema durante la creazione della rappresentazione interna, oppure se l’albero non può essere convalidato, viene sollevata un’eccezione `ParserError`. Un oggetto AST creato in questo modo non dovrebbe essere compilabile correttamente; le normali eccezioni sollevate durante la compilazione potrebbero essere ancora sollevate quando l’oggetto AST viene passato alla funzione `compileast()`. Ciò potrebbe indicare problemi non correlati alla sintassi (come un’eccezione `MemoryError`), ma potrebbe anche essere dovuto a costrutti quale il risultato dell’analisi di `del f(0)`, che sfugge al parser Python ma viene controllata dal compilatore bytecode.

Sequenze che rappresentano simboli terminali possono venire rappresentate come liste di due elementi nella forma `(1, 'name')` oppure come liste di tre elementi nella forma `(1, 'name', 56)`. Se il

terzo elemento è presente, viene considerato come un numero di riga valido. Il numero di riga può essere specificato per qualsiasi sotto insieme dei simboli terminali nell'albero di input.

`tuple2ast`(*sequence*)
Questa funzione è analoga a `sequence2ast()`. Questa chiamata a funzione viene mantenuta per compatibilità con le versioni precedenti.

18.1.2 Conversione di oggetti AST

Gli oggetti AST, indipendentemente dall'input usato per crearli, possono venire convertiti in alberi di analisi rappresentati come alberi di liste o tuple, oppure possono venire compilati in oggetti di codice eseguibile. Alberi di analisi possono venire estratti con o senza le informazioni sul numero di riga.

`ast2list`(*ast*[, *line_info*])
Questa funzione accetta un oggetto AST dal chiamante come parametro *ast* e restituisce una lista Python che rappresenta l'albero di analisi equivalente. La rappresentazione della lista risultante può essere usata per l'ispezione o per la creazione di un nuovo albero di analisi sotto forma di lista. Questa funzione non fallisce finché è disponibile memoria per costruire la rappresentazione della lista. Se l'albero di analisi verrà usato soltanto per l'ispezione, si dovrebbe invece utilizzare `ast2tuple()` per ridurre il consumo e la frammentazione della memoria. Quando viene richiesta la rappresentazione della lista, questa funzione è significativamente più veloce rispetto al recupero della rappresentazione sotto forma di tupla e la conversione di questa in liste annidate.

Se *line_info* è vero, l'informazione relativa al numero di riga verrà inclusa per tutti i simboli terminali come terzo elemento della lista che rappresenta il simbolo. Notate che il numero di riga fornito specifica la riga sulla quale il simbolo *termina*. Questa informazione viene omessa se l'opzione ha un valore falso o viene omessa.

`ast2tuple`(*ast*[, *line_info*])
Questa funzione accetta dal chiamante un oggetto AST come parametro *ast* e restituisce una tupla Python che rappresenta l'albero di analisi equivalente. A parte restituire una tupla invece di una lista, questa funzione è identica ad `ast2list()`.

Se *line_info* è vero, l'informazione relativa al numero di riga verrà inclusa per tutti i simboli terminali come terzo elemento della lista che rappresenta il simbolo. Questa informazione viene omessa se l'opzione ha valore falso o viene omessa.

`compileast`(*ast*[, *filename* `= '<ast>'`])
Il compilatore di bytecode Python può venire invocato su un oggetto AST per produrre oggetti di codice che possano essere utilizzati come parti di un'istruzione `exec` o di una chiamata alla funzione built-in `eval()`. Questa funzione fornisce l'interfaccia al compilatore, passando l'albero di analisi interno dal parametro *ast* al parser, usando il nome del file sorgente specificato dal parametro *filename*. Il valore predefinito fornito per *filename* indica che il sorgente era un oggetto AST.

Compilare un oggetto AST può generare eccezioni relative alla compilazione; un esempio sarebbe un'eccezione `SyntaxError` provocata dall'albero di analisi per `del f(0)`: questa istruzione viene considerata corretta all'interno della grammatica formale di Python, ma non lo è come costrutto del linguaggio. L'eccezione `SyntaxError` sollevata per questa condizione viene di solito realmente generata dal compilatore bytecode di Python, il che è la ragione per cui possa venire sollevata a questo punto dal modulo `parser`. La maggior parte delle cause degli errori di compilazione possono essere diagnosticate sistematicamente ispezionando l'albero di analisi.

18.1.3 Interrogazioni su oggetti AST

Vengono fornite due funzioni che consentono ad un'applicazione di determinare se un AST sia stato creato come espressione o come suite. Nessuna di esse può venire impiegata per determinare se un AST sia stato creato da codice sorgente tramite `expr()` o `suite()` o da un albero di analisi mediante `sequence2ast()`.

`isexpr`(*ast*)
Questa funzione restituisce vero quando *ast* rappresenta una forma `'eval'`, altrimenti assume come valore falso; cosa utile, visto che per ottenere queste informazioni non si possono interrogare gli oggetti codice

tramite le funzioni built-in esistenti. Notate che non si possono interrogare così nemmeno oggetti codice creati da `compileast()`, uguali a quelli creati con la funzione built-in `compile()`.

`issuite`(*ast*)
Questa funzione ricopia `isexpr()` nel riportare se un oggetto AST rappresenti un modello `'exec'`, comunemente noto come "suite." Non è prudente supporre che questa funzione equivalga a '`not isexpr(`*ast*`)`', dato che in futuro potrebbero venire supportati dei frammenti sintattici aggiuntivi.

18.1.4 Eccezioni e gestione degli errori

Il modulo parser definisce una singola eccezione, ma può passare anche altre eccezioni built-in da altre porzioni dell'ambiente di esecuzione di Python. Vedete ogni funzione per informazioni sulle eccezioni che può sollevare.

exception `ParserError`
Eccezione sollevata quando avviene un errore all'interno del modulo parser. Questo generalmente accade per errori nella validazione piuttosto che mediante il sollevamento dell'eccezione built-in `SyntaxError` durante la normale analisi. L'argomento dell'eccezione può essere una stringa che descrive la ragione del fallimento o una tupla contenente la sequenza che ha causato il fallimento dall'albero di analisi passato a `sequence2ast()` ed una stringa esplicativa. Le chiamate a `sequence2ast()` devono necessariamente essere in grado di gestire entrambi i tipi di eccezioni, mentre chiamate ad altre funzioni nel modulo avranno bisogno unicamente di essere preparate a ricevere una stringa.

Notate che le funzioni `compileast()`, `expr()` e `suite()` possono sollevare eccezioni generalmente causate dal processo di analisi e di compilazione. Queste includono le eccezioni built-in `MemoryError`, `OverflowError`, `SyntaxError` e `SystemError`. In tali casi hanno gli stessi significati normalmente associati ad esse. Per maggiori informazioni, vedete la descrizione di ogni funzione.

18.1.5 Oggetti AST

Vengono supportati i confronti di ordine e di uguaglianza fra oggetti AST. Viene supportata anche la serializzazione degli oggetti AST (usando il modulo `pickle`).

`ASTType`
Il tipo degli oggetti restituiti da `expr()`, `suite()` e `sequence2ast()`.

Gli oggetti AST possiedono i seguenti metodi:

`compile`(*[filename]*)
Equivalente a `compileast(`*ast*`, `*filename*`)`.

`isexpr`()
Equivalente a `isexpr(`*ast*`)`.

`issuite`()
Equivalente a `issuite(`*ast*`)`.

`tolist`(*[line_info]*)
Equivalente a `ast2list(`*ast*`, `*line_info*`)`.

`totuple`(*[line_info]*)
Equivalente a `ast2tuple(`*ast*`, `*line_info*`)`.

18.1.6 Esempi

Il modulo parser consente di compiere operazioni sull'albero di analisi del codice sorgente Python prima che venga generato il bytecode, e mette a disposizione l'ispezione dell'albero di analisi al fine di raccogliere informazioni. Vengono riportati due esempi. Quello semplice dimostra l'emulazione della funzione built-in `compile()`, mentre quello complesso mostra l'uso di un albero di analisi per la ricerca d'informazioni.

Emulazione di `compile()`

Anche se si possono eseguire molte operazioni utili fra l'analisi e la generazione del bytecode, la cosa più semplice è non fare nulla: a tal proposito, l'uso del modulo `parser` per produrre una struttura di dati intermedia equivale al codice:

```
>>> code = compile('a + 5', 'file.py', 'eval')
>>> a = 5
>>> eval(code)
10
```

L'operazione equivalente, utilizzando il modulo `parser`, richiede leggermente più tempo e permette all'albero di analisi interno intermedio di essere gestito come un oggetto AST.

```
>>> import parser
>>> ast = parser.expr('a + 5')
>>> code = ast.compile('file.py')
>>> a = 5
>>> eval(code)
10
```

Un'applicazione che richieda sia gli oggetti AST che gli oggetti codice può raggruppare questo codice in funzioni già disponibili:

```
import parser

def load_suite(source_string):
    ast = parser.suite(source_string)
    return ast, ast.compile()

def load_expression(source_string):
    ast = parser.expr(source_string)
    return ast, ast.compile()
```

Recupero delle informazioni

Alcune applicazioni possono beneficiare dell'accesso diretto all'albero di analisi. La parte rimanente di questa sezione dimostrerà come l'albero di analisi fornisca l'accesso alla documentazione del modulo definita nella docstring, senza la necessità che il codice sotto analisi venga caricato in un interprete in esecuzione tramite la direttiva `import`. Questo risulta molto utile nell'analisi di codice inaffidabile.

In generale, l'esempio dimostrerà come l'albero di analisi possa venire attraversato per recuperare informazioni interessanti. Sono state sviluppate due funzioni ed un insieme di classi per permettere un accesso via programma alle funzioni di alto livello ed alle definizioni di classi fornite dal modulo. Le classi estraggono le informazioni dall'albero di analisi e ne forniscono l'accesso attraverso ad un livello semantico utile, una funzione fornisce una semplice funzionalità di corrispondenze tra modelli a basso livello, mentre l'altra funzione definisce un'interfaccia ad alto livello alle classi gestendo le operazioni sui file secondo le necessità del chiamante. Tutti i file sorgenti qui citati che non siano parte dell'installazione di Python sono situati nella directory 'Demo/parser/' della distribuzione.

La natura dinamica di Python consente al programmatore una grande flessibilità, ma la maggior parte dei moduli ne richiedono solamente una piccola parte quando si tratta di definire delle classi, delle funzioni o dei metodi. In questo esempio, le sole definizioni che verranno considerate sono quelle definite al livello più alto del loro contesto, ad esempio una funzione definita da un'istruzione `def` alla colonna zero di un modulo, ma non una funzione definita all'interno di un ramo di un costrutto `if ... else`, anche se ci sono dei buoni motivi per far questo in certe situazioni. L'annidamento di definizioni verrà gestito dal codice sviluppato nell'esempio.

Per costruire i metodi di estrazione del livello più alto, dobbiamo conoscere l'aspetto della struttura dell'albero di

analisi e quanto di tale struttura è utile ai nostri scopi. Python utilizza un albero di analisi abbastanza profondo, quindi esistono numerosi nodi intermedi. È importante leggere e capire la grammatica formale usata da Python. Questa viene definita nel file 'Grammar/Grammar' nella distribuzione. Considerate il caso di interesse più semplice quando si ricerca una docstrings: un modulo che contenga solamente una doctring e nient'altro (vedete il file 'docstring.py').

```
"""Un po' di documentazione
"""
```

Utilizzando l'interprete per dare un'occhiata all'albero di analisi, troveremo una sconcertante massa di numeri e parentesi, con la documentazione sepolta profondamente in mezzo a tuple annidate.

```
>>> import parser
>>> import pprint
>>> ast = parser.suite(open('docstring.py').read())
>>> tup = ast.totuple()
>>> pprint.pprint(tup)
(257,
 (264,
  (265,
   (266,
    (267,
     (307,
      (287,
       (288,
        (289,
         (290,
          (292,
           (293,
            (294,
             (295,
              (296,
               (297,
                (298,
                 (299,
                  (300, (3, '"""Un po' di documentazione.\n"""')))))))))))))))),
   (4, ''))),
 (4, ''),
 (0, ''))
```

I numeri come primo elemento di ogni nodo nell'albero costituiscono i tipi dei nodi; si riferiscono direttamente a simboli terminali e non terminali della grammatica. Sfortunatamente, questi vengono indicati nella rappresentazione interna con degli interi e le strutture Python generate non li modificano. Comunque i moduli `symbol` e `token` forniscono nomi simbolici per i tipi dei nodi e dizionari che mappano per ogni tipo di nodo gli interi in nomi simbolici .

Nell'output presentato sopra, la tupla più esterna contiene quattro elementi: l'intero `257` e tre tuple addizionali. Il tipo di nodo `257` ha il nome simbolico `file_input`. Ognuna delle tuple più interne contiene un intero come primo elemento; questi interi, `264`, `4` e `0`, rappresentano rispettivamente i tipi di nodi `stmt`, `NEWLINE` e `ENDMARKER`. Fate attenzione che questi valori potrebbero variare a seconda della versione di Python in uso; consultate 'symbol.py' e 'token.py' per i dettagli su come vengono mappati. Dovrebbe essere abbastanza chiaro che il nodo più esterno referenzia principalmente la sorgente di ingresso piuttosto che il contenuto del file e può essere trascurato per il momento. Il nodo `stmt` è molto più interessante. In particolare, tutte le docstring si trovano in sotto alberi che vengono formati nella stessa maniera in cui viene formato questo nodo, con la sola differenza della stringa stessa. L'associazione tra la docstring in tale albero e l'entità definita (classe, funzione o modulo) che questa descrive viene data dalla posizione del sotto albero della docstring all'interno dell'albero che definisce la struttura descritta.

Sostituendo la docstring reale con qualcosa che rappresenti una componente variabile dell'albero, può permettere ad un semplice approccio di corrispondenza fra modelli di controllare per ogni dato sotto albero l'equivalenza delle docstring con modelli generici. Dato che l'esempio mostra l'estrazione delle informazioni, possiamo richiedere

in maniera sicura che l'albero sia in forma di tupla piuttosto che in forma di lista, permettendo ad una semplice rappresentazione di una variabile di essere nella forma `['nome_variabile']`. La corrispondenza tra modelli può venire implementata con una semplice funzione ricorsiva che restituisca un valore booleano ed un dizionario di corrispondenze tra nomi di variabile e valori (vedete il file 'example.py').

```
from types import ListType, TupleType

def match(pattern, data, vars=None):
    if vars is None:
        vars = {}
    if type(pattern) is ListType:
        vars[pattern[0]] = data
        return 1, vars
    if type(pattern) is not TupleType:
        return (pattern == data), vars
    if len(data) != len(pattern):
        return 0, vars
    for pattern, data in map(None, pattern, data):
        same, vars = match(pattern, data, vars)
        if not same:
            break
    return same, vars
```

Utilizzando questa semplice rappresentazione per le variabili sintattiche e i tipi di nodo simbolici, il modello per i sotto alberi di docstring candidati diventa abbastanza leggibile (vedete il file 'example.py').

```
import symbol
import token

DOCSTRING_STMT_PATTERN = (
    symbol.stmt,
    (symbol.simple_stmt,
     (symbol.small_stmt,
      (symbol.expr_stmt,
       (symbol.testlist,
        (symbol.test,
         (symbol.and_test,
          (symbol.not_test,
           (symbol.comparison,
            (symbol.expr,
             (symbol.xor_expr,
              (symbol.and_expr,
               (symbol.shift_expr,
                (symbol.arith_expr,
                 (symbol.term,
                  (symbol.factor,
                   (symbol.power,
                    (symbol.atom,
                     (token.STRING, ['docstring'])
                     ))))))))))))),
     (token.NEWLINE, '')
     ))
```

Utilizzando la funzione `match()` con questo modello, è facile estrarre il modulo docstring dall'albero di analisi creato in precedenza:

```
>>> found, vars = match(DOCSTRING_STMT_PATTERN, tup[1])
>>> found
1
>>> vars
{'docstring': '"""Some documentation.\n"""'}
```

Una volta che i dati specifici possono essere estratti da una locazione dove ci si aspetta siano presenti, la domanda di dove si possano trovare le informazioni necessita di una risposta. Quando ci si occupa delle docstring, la risposta è semplice: la docstring è il primo nodo `stmt` in un blocco di codice (tipi di nodo `file_input` o `suite`). Un modulo consiste in un singolo nodo `file_input`, le definizioni di classe e di funzione contengono ognuna esattamente un nodo `suite`. Classi e funzioni vengono prontamente identificate come sotto alberi di nodi dei blocchi di codice che iniziano con `(stmt, (compound_stmt, (classdef, ...` o `(stmt, (compound_stmt, (funcdef, ...`. Notate che questi sotto alberi non possono essere verificati da `match()` visto che essa non supporta la ricerca su nodi fratelli multipli senza riguardo per il numero. Per superare questa limitazione può venire usata una funzione di ricerca più elaborata, ma questo è sufficiente per l'esempio.

Avuta l'abilità di determinare se un'istruzione possa essere una docstring ed estarre la stringa reale dall'istruzione, è necessario compiere delle operazioni di percorrimento dell'albero di analisi per un intero modulo, estrarre l'informazione sui nomi definiti in ogni contesto del modulo ed associare ogni docstring ai nomi. Il codice che esegue questo lavoro non è complicato, ma necessita di alcune spiegazioni.

Questa interfaccia pubblica per le classi è chiara e dovrebbe essere probabilmente un po' più flessibile. Ogni blocco "major" del modulo viene descritto da un oggetto che fornisce alcuni metodi per la ricerca e da un costruttore che accetta come minimo il sotto albero dell'albero di analisi completo che lo descrive. Il costruttore `ModuleInfo` accetta un parametro *name* facoltativo poiché esso non può determinare altrimenti il nome del modulo.

Le classi pubbliche includono `ClassInfo`, `FunctionInfo` e `ModuleInfo`. Tutti gli oggetti forniscono i metodi `get_name()`, `get_docstring()`, `get_class_names()` e `get_class_info()`. Gli oggetti `ClassInfo` supportano `get_method_names()` e `get_method_info()` mentre le altre classi forniscono `get_function_names()` e `get_function_info()`.

All'interno di ogni forma di blocco di codice che le classi pubbliche rappresentano, la maggior parte delle informazioni si trova nella stessa forma e vi si accede nello stesso modo, con la differenza che alle funzioni definite al livello più alto in queste classi ci si riferisce come "methodi". Da quando la differenza nella terminologia riflette una reale distinzione semantica dalle funzioni definite fuori dalla classe, l'implementazione necessita di mantenere questa distinzione. Da questo punto in poi, la maggior parte delle funzionalità delle classi pubbliche può venire implementata in una classe base comune, `SuiteInfoBase`, con gli accessori per le funzioni e le informazioni sui metodi forniti altrove. Notate che c'è una sola classe che rappresenta le informazioni su funzioni e metodi; questo è parallelo all'uso dell'istruzione `def` per definire entrambi i tipi di elemento.

La maggior parte delle funzioni accessorie vengono dichiarate in `SuiteInfoBase` e non necessitano di essere sovrascritte nelle sotto classi. Più importante, l'estrazione della maggior parte delle informazioni da un albero di analisi viene gestita attraverso un metodo chiamato dal costruttore `SuiteInfoBase`. Il codice di esempio per la maggior parte delle classi è chiaro quando viene letto dal lato della grammatica formale, ma il metodo che crea ricorsivamente nuovi oggetti informativi richiede un'ulteriore verifica. Qui ci sono le parti rilevanti della definizione di `SuiteInfoBase` da 'example.py':

```
class SuiteInfoBase:
    _docstring = ''
    _name = ''

    def __init__(self, tree = None):
        self._class_info = {}
        self._function_info = {}
        if tree:
            self._extract_info(tree)

    def _extract_info(self, tree):
        # estrazione della docstring
        if len(tree) == 2:
            found, vars = match(DOCSTRING_STMT_PATTERN[1], tree[1])
        else:
            found, vars = match(DOCSTRING_STMT_PATTERN, tree[3])
        if found:
            self._docstring = eval(vars['docstring'])
        # rileva le definizioni
        for node in tree[1:]:
            found, vars = match(COMPOUND_STMT_PATTERN, node)
            if found:
                cstmt = vars['compound']
                if cstmt[0] == symbol.funcdef:
                    name = cstmt[2][1]
                    self._function_info[name] = FunctionInfo(cstmt)
                elif cstmt[0] == symbol.classdef:
                    name = cstmt[2][1]
                    self._class_info[name] = ClassInfo(cstmt)
```

Dopo l'inizializzazione di alcuni stati interni, il costruttore chiama il metodo `_extract_info()`. Questo metodo esegue la verifica dell'estrazione dell'informazione che avviene nell'intero esempio. L'estrazione ha due fasi distinte: la localizzazione della docstring nell'albero di analisi passato e la scoperta di definizioni aggiuntive all'interno del blocco di codice rappresentato dall'albero di analisi.

L'`if` iniziale determina se la suite annidata sia per la "forma breve" o per la "forma lunga". La forma abbreviata viene usata quando il blocco di codice è sulla stessa riga della sua definizione, come in

```
def square(x): "Eleva al quadrato un argomento."; return x ** 2
```

mentre la forma lunga utilizza un blocco indentato e permette l'annidamento delle definizioni:

```
def make_power(exp):
    "Crea una funzione che eleva un argomento all'esponente 'exp'."
    def raiser(x, y=exp):
        return x ** y
    return raiser
```

Quando viene usata la forma abbreviata, il blocco di codice può contenere una docstring come primo, e possibilmente unico, elemento `small_stmt`. L'estrazione di una docstring simile è un pò differente e richiede soltanto una parte del modello completo usato nel caso più comune. Appena implementata, la docstring verrà trovata soltanto se c'è un singolo nodo `small_stmt` nel nodo `small_stmt`. Dal momento che la maggior parte delle funzioni e dei metodi che usano la forma abbreviata non forniscono una docstring, ciò potrebbe essere considerato sufficiente. L'estrazione della docstring procede usando la funzione `match()` come descritto sopra ed il valore della docstring viene conservato come attributo dell'oggetto `SuiteInfoBase`.

Dopo l'estrazione della docstring, si attiva un semplice algoritmo per rintracciare la definizione sui nodi `stmt`

del nodo `suite`. Il caso speciale della forma abbreviata non viene testato; dal momento che non ci sono nodi `stmt` nella forma abbreviata, l'algoritmo salta tacitamente il singolo nodo `simple_stmt` e giustamente non rileva alcuna definizione annidata.

Ogni istruzione nel blocco di codice viene classificata come una definizione di classe, una definizione di funzione o metodo, o come qualcos'altro. Per le istruzioni di definizione, viene estratto il nome dell'elemento definito e creato un oggetto che rappresenta in modo appropriato la definizione, con il sotto albero definente passato come argomento al costruttore. Tali oggetti vengono conservati in variabili d'istanza e possono venire recuperati per nome utilizzando i metodi d'accesso appropriati.

Le classi pubbliche forniscono qualsiasi accesso necessario che sia più specifico di quelli forniti dalla classe `SuiteInfoBase`, ma l'algoritmo di estrazione effettivo rimane comune a tutte le forme di blocco di codice. Una funzione d'alto livello può essere usata per estrarre l'insieme completo di informazioni da un file sorgente (vedete il file 'example.py').

```
def get_docs(fileName):
    import os
    import parser

    source = open(fileName).read()
    basename = os.path.basename(os.path.splitext(fileName)[0])
    ast = parser.suite(source)
    return ModuleInfo(ast.totuple(), basename)
```

Questa funzione fornisce un'interfaccia per la documentazione di un modulo di facile utilizzo. Se l'informazione richiesta non viene estratta dal codice di questo esempio, il codice può venire esteso in punti ben definiti per fornire ulteriori funzionalità.

18.2 `symbol` — Costanti usate con gli alberi di analisi di Python

Questo modulo fornisce le costanti che rappresentano i valori numerici dei nodi interni dell'albero di analisi. Diversamente dalla maggior parte delle costanti Python, i loro nomi sono in minuscolo. Fate riferimento al file 'Grammar/Grammar' della distribuzione Python per le definizioni dei nomi nel contesto della grammatica del linguaggio. I valori numerici specifici ai quali i nomi corrispondono possono differire tra le varie versioni di Python.

Questo modulo fornisce inoltre un ulteriore oggetto per i dati:

`sym_name`
Dizionario che mappa i valori numerici delle costanti definite in questo modulo con le stringhe dei nomi, permettendo di generare una rappresentazione degli alberi di analisi umanamente comprensibile.

Vedete anche:

Modulo `parser` (sezione 18.1):
Il secondo esempio per il modulo `parser` mostra come usare il modulo `symbol`.

18.3 `token` — Costanti usate con gli alberi di analisi di Python

Questo modulo fornisce le costanti che rappresentano i valori numerici dei nodi foglia dell'albero di analisi (simboli terminali). Fate riferimento al file 'Grammar/Grammar' della distribuzione Python per le definizioni dei nomi nel contesto della grammatica del linguaggio. I valori numerici specifici che vengono mappati possono cambiare tra le diverse versioni di Python.

Questo modulo fornisce inoltre un oggetto di dati ed alcune funzioni. Le funzioni rispecchiano le definizioni nei file header C di Python.

`tok_name`
Dizionario che mappa i valori numerici delle costanti definite in questo modulo con le corrisponden-

ti stringhe dei nomi, permettendo di generare una rappresentazione dell'albero di analisi umanamente comprensibile.

ISTERMINAL(*x*)
Restituisce vero per valori che sono simboli terminali.

ISNONTERMINAL(*x*)
Restituisce true per valori che non sono simboli terminali.

ISEOF(*x*)
Restituisce vero se *x* è il marcatore che indica la fine dell'input.

Vedete anche:

Modulo `parser` (sezione 18.1):
Il secondo esempio del modulo `parser` mostra come usare il modulo `symbol`.

18.4 `keyword` — Verifica le parole chiave di Python

Questo modulo consente ad un programma Python di determinare se una stringa è una parola chiave di Python.

iskeyword(*s*)
Restituisce vero se *s* è una parola chiave di Python.

kwlist
Sequenza che contiene tutte le parole chiave definite nell'interprete. Sono parimenti incluse quelle parole chiave che si è stabilito saranno attive quando entreranno in funzione particolari istruzioni `__future__`.

18.5 `tokenize` — Elaboratore di simboli per il sorgente Python

Il modulo `tokenize` fornisce uno scanner lessicale per il codice sorgente Python, implementato in Python. Lo scanner di questo modulo restituisce sia commenti che simboli, rendendolo utile nell'implementazione di stampe in forma elegante, inclusi decoratori per schermate a video.

Il punto di partenza iniziale è un generatore:

generate_tokens(*readline*)
Il generatore `generate_tokens()` richiede un argomento, *readline*, che deve essere un oggetto richiamabile, il quale fornisce la medesima interfaccia del metodo `readline()` incluso negli oggetti file built-in (vedete la sezione 2.3.9). Ogni chiamata alla funzione dovrebbe restituire una riga di input sotto forma di stringa.

Il generatore produce tuple di 5 elementi con questi membri: il tipo di simbolo; la stringa del simbolo; una tupla formata da 2 elementi (*srow*, *scol*) di numeri interi che specificano riga e colonna nel codice in cui il simbolo inizia; una tupla formata da 2 elementi (*erow*, *ecol*) di numeri interi che specificano riga e colonna nel codice in cui il simbolo finisce; e la riga in cui viene trovato il simbolo. La riga passata è quella *logica*; sono comprese le righe di continuazione. Nuovo nella versione 2.2.

Per retrocompatibilità viene mantenuto un vecchio punto di partenza:

tokenize(*readline*[, *tokeneater*])
la funzione `tokenize()` accetta due parametri: uno che rappresenta il flusso in input ed uno che fornisce un meccanismo di output per `tokenize()`.

Il primo parametro, *readline*, deve essere un oggetto richiamabile che fornisce la stessa interfaccia del metodo `readline()` incluso negli oggetti file built-in (vedete la sezione 2.3.9). Ogni chiamata alla funzione dovrebbe restituire una riga di input sotto forma di stringa.

Il secondo parametro, *tokeneater*, deve essere anch'esso un oggetto richiamabile. Viene chiamato una volta per ogni simbolo, con cinque argomenti che corrispondono alle tuple generate da `generate_tokens()`.

Tutte le costanti provenienti dal modulo `token` vengono esportate anche da `tokenize`, inoltre ci sono due valori aggiuntivi di tipi di simbolo che possono venire passati alla funzione *tokeneater* da `tokenize()`:

COMMENT
Valore di simbolo utilizzato per indicare un commento.

NL
Valore di simbolo utilizzato per indicare un fine riga non terminante. Il simbolo NEWLINE indica la fine di una riga logica di codice Python; i simboli NL vengono generati quando una riga logica di codice continua su più righe fisiche.

18.6 tabnanny — Rilevatore di indentazioni ambigue

A tutt'oggi questo modulo si presume venga chiamato come script. È comunque possibile importarlo in un IDE ed usare la funzione `check()` descritta più avanti.

Avvertenze: l'API fornita da questo modulo è suscettibile di cambiamenti nelle prossime versioni; tali cambiamenti potrebbero non essere compatibili con le versioni precedenti.

check(*file_or_dir*)
Se *file_or_dir* è una directory e non un collegamento simbolico verrà percorso ricorsivamente l'albero delle directory nominato da *file_or_dir* controllando tutti i file '.py' incontrati. Se *file_or_dir* è un file sorgente di Python ordinario, verrà esaminato alla ricerca di problemi relativi agli spazi vuoti. I messaggi diagnostici vengono scritti sullo standard output mediante l'istruzione print.

verbose
Opzione che indica di scrivere messaggi prolissi. Se il modulo viene invocato come script essa verrà incrementata dall'opzione `-v`.

filename_only
Opzione che indica se stampare solo i nomi dei file che hanno problemi legati agli spazi vuoti. Se chiamato come script viene impostata a vero dall'opzione `-q`.

exception NannyNag
Sollevata da `tokeneater()` se viene riscontrata un'indentazione ambigua. Catturata e gestita con `check()`.

tokeneater(*type, token, start, end, line*)
Questa funzione viene usata da `check()` come parametro di richiamo per la funzione `tokenize.tokenize()`.

Vedete anche:

Modulo `tokenize` (sezione 18.5):
Scanner lessicale per codice sorgente Python.

18.7 pyclbr — Supporto al browser delle classi Python

`pyclbr` può essere usato per determinare delle informazioni limitate riguardo classi, metodi e funzioni di alto livello definite in un modulo. Le informazioni fornite sono sufficienti per implementare un browser tradizionale delle classi in tre sezioni. Le informazioni vengono estratte dal codice sorgente piuttosto che importando il modulo, così lo stesso modulo è utilizzabile anche con codice sorgente insicuro. Questa restrizione rende impossibile l'uso di questo modulo con altri non implementati in Python, compresi molti moduli di estensione standard e facoltativi.

readmodule(*module*[, *path*])
Legge un modulo e restituisce un dizionario che mappa i nomi delle classi con i rispettivi oggetti descrittori di classe. Il parametro *module* dovrebbe essere il nome di un modulo sotto forma di stringa; può essere il nome di un modulo incluso in un package. Il parametro *path* dovrebbe essere una sequenza e viene utilizzata per aumentare il valore di `sys.path`, utilizzato per localizzare il codice sorgente del modulo.

readmodule_ex(*module*[, *path*])
Simile a `readmodule()`, ma il dizionario restituito, in aggiunta alla mappatura dei nomi delle classi con gli oggetti descrittori di classe, mappa anche i nomi delle funzioni di alto livello con gli oggetti descrittori

di funzione. Inoltre, se il modulo letto è un package, la chiave `'__path__'` nel dizionario restituito ha come valore una lista che contiene il percorso di ricerca del package.

18.7.1 Oggetti descrittori di classe

Gli oggetti descrittori di classe, usati come valori nel dizionario restituito da `readmodule()` e `readmodule_ex()`, forniscono i seguenti membri di dati:

`module`
Il nome del modulo che definisce la classe definita dal descrittore di classe.

`name`
Il nome della classe.

`super`
Una lista di descrittori di classe che definiscono le classi base immediate della classe appena definita. Le classi che sono state nominate come superclassi ma che non sono rintracciabili da `readmodule()`, vengono esposte sotto forma di stringa con il nome della classe al posto del descrittore di classe.

`methods`
Un dizionario che mappa i nomi dei metodi con i numeri di riga.

`file`
Nome del file contenente l'istruzione `class` che definisce la classe.

`lineno`
Il numero di riga dell'istruzione `class` all'interno del file definito da `file`.

18.7.2 Oggetti descrittori di funzione

Gli oggetti descrittori di funzione usati come valori nel dizionario restituito da `readmodule_ex()` forniscono i seguenti membri di dati:

`module`
Il nome del modulo che definisce la funzione definita dal descrittore di funzione.

`name`
Il nome della funzione.

`file`
Nome del file contenente l'istruzione `def` che definisce la funzione.

`lineno`
Il numero di riga dell'istruzione `def` all'interno del file definito da `file`.

18.8 `py_compile` — Compilazione di file sorgenti Python

Il modulo `py_compile` fornisce una funzione per generare file bytecode da un file sorgente ed un'altra funzione utilizzata quando il file sorgente del modulo viene invocato come script.

Sebbene non utilizzata spesso, questa funzione può essere utile durante l'installazione di moduli condivisi, specialmente se alcuni utenti non hanno il permesso di scrittura su file di cache bytecode nella directory contenente il codice sorgente.

exception `PyCompileError`
Eccezione sollevata quando viene rilevato un errore nel tentativo di compilare il file.

`compile`(*file*[, *cfile*[, *dfile*[, *doraise*]]])
Compila un file sorgente generando bytecode che viene scritto in un file di cache. Il codice sorgente viene caricato dal file identificato dal parametro *file*. Il bytecode viene scritto nel file *cfile*, il valore predefinito di tale parametro è *file* `+ 'c'` (`'o'` se è attiva l'opzione di ottimizzazione nell'interprete in esecuzione). Se viene definito il parametro *dfile*, esso viene utilizzato come nome del file sorgente nei messaggi di errore al

posto di *file*. Se *doraise* = True, viene sollevata un'eccezione `PyCompileError` se si verifica un errore durante la compilazione di *file*. Se *doraise* = False (valore predefinito), viene scritta su sys.stderr una stringa di errore, ma non viene sollevata alcuna eccezione.

`main`([*args*])
Compila diversi file sorgenti. I file specificati in *args* (o sulla riga di comando, se *args* non viene definito) vengono compilati ed il bytecode risultante viene messo nella cache alla solita maniera. Questa funzione non ricerca in una struttura di directory per trovare i file sorgenti; compila solamente i file definiti esplicitamente.

Quando questo modulo viene eseguito come script, viene utilizzata `main()` per compilare tutti i file elencati sulla riga di comando.

Vedete anche:

Modulo `compileall` (sezione 18.9):
Utilità per compilare file sorgenti Python in un albero di directory.

18.9 `compileall` — Compila in bytecode le librerie Pyhton

Questo modulo fornisce delle funzioni di supporto all'installazione delle librerie Python. Queste funzioni compilano i file sorgenti di Python all'interno di un albero di directory, permettendo agli utenti che non hanno i diritti di scrittura nelle librerie di sfruttare il vantaggio dei file bytecode presenti nella cache.

Il file sorgente di questo modulo può venire utilizzato come script per compilare i sorgenti Python in directory specificate sulla riga di comando o in `sys.path`.

`compile_dir`(*dir*[, *maxlevels*[, *ddir*[, *force*[, *rx*[, *quiet*]]]]])
Percorre ricorsivamente in discesa l'albero di directory specificato dal parametro *dir*, compilando tutti i file '.py' che incontra. Il parametro *maxlevels* viene utilizzato per limitare la profondità della ricorsione; il suo valore predefinito è `10`. Se viene specificato *ddir*, viene utilizzato come percorso di base dal quale verranno generati i nomi di file utilizzati nei messaggi di errore. Se *force* ha valore vero, i moduli vengono ricompilati anche se già aggiornati.

Se viene specificato *rx*, esso definisce un'espressione regolare dei nomi di file da escludere dalla ricerca; questa espressione viene valutata in tutto il percorso.

Se *quiet* ha valore vero, nulla viene stampato sullo standard output durante l'esecuzione normale.

`compile_path`([*skip_curdir*[, *maxlevels*[, *force*]]])
Compila in bytecode tutti i file '.py' trovati in `sys.path`. Se il parametro *skip_curdir* ha valore vero (predefinito), la directory corrente non viene inclusa nella ricerca. I parametri *maxlevels* e *force*, aventi come valore predefinito `0`, vengono passati alla funzione `compile_dir()`.

Vedete anche:

Modulo `py_compile` (sezione 18.8):
Compila in bytecode un singolo file sorgente.

18.10 `dis` — Disassemblatore per il bytecode Python

Il modulo `dis` permette l'analisi del bytecode Python tramite la sua disassemblatura. Dato che non esiste un assemblatore Python, questo modulo definisce il linguaggio assembly Python. Il bytecode di Python che questo modulo utilizza come input viene definito nel file 'Include/opcode.h' ed utilizzato sia dal compilatore che dall'interprete.

Esempio: Data la funzione `myfunc`:

```
def myfunc(alist):
    return len(alist)
```

il seguente comando può venire utilizzato per ottenere il codice disassemblato di `myfunc()`:

```
>>> dis.dis(myfunc)
  2           0 LOAD_GLOBAL              0 (len)
              3 LOAD_FAST                0 (alist)
              6 CALL_FUNCTION            1
              9 RETURN_VALUE
             10 LOAD_CONST               0 (None)
             13 RETURN_VALUE
```

(il "2" è il numero di riga).

Il modulo `dis` definisce le seguenti funzioni e costanti:

dis([*bytesource*])
Disassembla l'oggetto *bytesource*. *bytesource* può denotare un modulo, una classe, un metodo, una funzione oppure un oggetto codice. Nel caso di un modulo vengono disassemblate tutte le funzioni. Nel caso di una classe vengono disassemblati tutti i metodi. Per una singola sequenza di codice stampa una riga per ogni istruzione bytecode. Se nessun oggetto viene passato come parametro, disassembla l'ultima traceback.

distb([*tb*])
Disassembla la funzione in cima alla pila della traceback, utilizzando l'ultima traceback se non ne viene specificata nessuna. Viene indicata l'istruzione che ha causato l'eccezione.

disassemble(*code*[, *lasti*])
Disassembla un oggetto codice, indicando l'ultima istruzione se *lasti* viene definito. L'output viene suddiviso nelle seguenti colonne:

1. il numero di riga, per la prima istruzione di ogni riga
2. l'istruzione corrente, indicata da '`->`',
3. un'istruzione con etichetta, indicata da '`>>`',
4. l'indirizzo dell'istruzione,
5. il nome del codice dell'operazione,
6. i parametri dell'operazione, e
7. l'interpretazione dei parametri fra parentesi.

Il parametro interpretazione identifica nomi di variabili globali e locali, valori di costanti, destinazioni di operazioni di salto e operatori di confronto.

disco(*code*[, *lasti*])
Sinonimo della funzione `disassemble`. È più facile da scrivere e mantiene la compatibilità con le versioni precedenti di Python.

opname
Sequenza di nomi di operazioni, indicizzabile usando il bytecode.

cmp_op
Sequenza di tutti i nomi delle operazioni di confronto.

hasconst
Sequenza di bytecode che hanno un parametro costante.

hasfree
Sequenza di bytecode che accedono ad una variabile libera.

hasname
Sequenza di bytecode che accede ad un attributo tramite il nome.

hasjrel
Sequenza di bytecode che ha una salto verso una destinazione relativa.

hasjabs
Sequenza di bytecode che ha una salto verso una destinazione assoluta.

haslocal
Sequenza di bytecode che accede ad una variabile locale.

hascompare
Sequenza di bytecode di operazioni booleane.

18.10.1 Istruzioni Byte Code di Python

Attualmente il compilatore Python genera le seguenti istruzioni bytecode.

STOP_CODE
Indica la fine del codice al compilatore, non viene utilizzato dall'interprete.

POP_TOP
Rimuove l'oggetto in cima alla pila (NdT: TOS, in Inglese: top-of-stack).

ROT_TWO
Scambia i 2 oggetti in cima alla pila.

ROT_THREE
Sposta gli oggetti al secondo e terzo posto della pila in alto di una posizione, sposta poi il primo oggetto alla terza posizione nella pila.

ROT_FOUR
Sposta gli oggetti al secondo, terzo e quarto posto della pila in alto di una posizione, sposta poi il primo oggetto alla quarta posizione nella pila.

DUP_TOP
Duplica il riferimento in cima alla pila.

Le operazioni unitarie prendono il primo oggetto della pila, applicano l'operazione e rimettono il risultato nella pila.

UNARY_POSITIVE
Implementa `TOS = +TOS`.

UNARY_NEGATIVE
Implementa `TOS = -TOS`.

UNARY_NOT
Implementa `TOS = not TOS`.

UNARY_CONVERT
Implementa `TOS = 'TOS'`.

UNARY_INVERT
Implementa `TOS = ~TOS`.

GET_ITER
Implementa `TOS = iter(TOS)`.

Le operazioni binarie rimuovono il primo (TOS) ed il secondo (TOS1) elemento dalla cima della pila. Effettuano l'operazione e rimettono il risultato sulla pila.

BINARY_POWER
Implementa `TOS = TOS1 ** TOS`.

BINARY_MULTIPLY
Implementa `TOS = TOS1 * TOS`.

BINARY_DIVIDE
Implementa `TOS = TOS1 / TOS` quando non è in attivo `from __future__ import division`.

BINARY_FLOOR_DIVIDE
Implementa `TOS = TOS1 // TOS`.

BINARY_TRUE_DIVIDE
Implementa `TOS = TOS1 / TOS` quando non è in attivo `from __future__ import division`.

BINARY_MODULO
Implementa `TOS = TOS1 % TOS`.

BINARY_ADD
Implementa `TOS = TOS1 + TOS`.

BINARY_SUBTRACT
Implementa `TOS = TOS1 - TOS`.

BINARY_SUBSCR
Implementa `TOS = TOS1[TOS]`.

BINARY_LSHIFT
Implementa `TOS = TOS1 << TOS`.

BINARY_RSHIFT
Implementa `TOS = TOS1 >> TOS`.

BINARY_AND
Implementa `TOS = TOS1 & TOS`.

BINARY_XOR
Implementa `TOS = TOS1 ^ TOS`.

BINARY_OR
Implementa `TOS = TOS1 | TOS`.

Le operazioni sul posto sono come le operazioni binarie, in quanto rimuovono TOS e TOS1 e rimettono sulla pila il risultato, ma l'operazione viene fatta sul posto quando TOS1 lo supporta ed il TOS risultante può essere (ma non per forza) il TOS1 originale.

INPLACE_POWER
Implementa l'operazione sul posto `TOS = TOS1 ** TOS`.

INPLACE_MULTIPLY
Implementa l'operazione sul posto `TOS = TOS1 * TOS`.

INPLACE_DIVIDE
Implementa l'operazione sul posto `TOS = TOS1 / TOS` quando non è attivo `from __future__ import division`.

INPLACE_FLOOR_DIVIDE
Implementa l'operazione sul posto `TOS = TOS1 // TOS`.

INPLACE_TRUE_DIVIDE
Implementa l'operazione sul posto `TOS = TOS1 / TOS` quando non è attivo `from __future__ import division`.

INPLACE_MODULO
Implementa l'operazione sul posto `TOS = TOS1 % TOS`.

INPLACE_ADD
Implementa l'operazione sul posto `TOS = TOS1 + TOS`.

INPLACE_SUBTRACT
Implementa l'operazione sul posto `TOS = TOS1 - TOS`.

INPLACE_LSHIFT
Implementa l'operazione sul posto `TOS = TOS1 << TOS`.

INPLACE_RSHIFT
Implementa l'operazione sul posto `TOS = TOS1 >> TOS`.

INPLACE_AND
Implementa l'operazione sul posto `TOS = TOS1 & TOS`.

INPLACE_XOR
Implementa l'operazione sul posto `TOS = TOS1 ^ TOS`.

INPLACE_OR

Implementa l'operazione sul posto `TOS = TOS1 | TOS`.

I codici di operazione di affettamento accettano fino a tre parametri.

`SLICE+0`
Implementa `TOS = TOS[:]`.

`SLICE+1`
Implementa `TOS = TOS1[TOS:]`.

`SLICE+2`
Implementa `TOS = TOS1[:TOS]`.

`SLICE+3`
Implementa `TOS = TOS2[TOS1:TOS]`.

L'assegnazione delle slice necessita anche di un parametro addizionale. Come ogni istruzione, non mettono nulla sulla pila.

`STORE_SLICE+0`
Implementa `TOS[:] = TOS1`.

`STORE_SLICE+1`
Implementa `TOS1[TOS:] = TOS2`.

`STORE_SLICE+2`
Implementa `TOS1[:TOS] = TOS2`.

`STORE_SLICE+3`
Implementa `TOS2[TOS1:TOS] = TOS3`.

`DELETE_SLICE+0`
Implementa `del TOS[:]`.

`DELETE_SLICE+1`
Implementa `del TOS1[TOS:]`.

`DELETE_SLICE+2`
Implementa `del TOS1[:TOS]`.

`DELETE_SLICE+3`
Implementa `del TOS2[TOS1:TOS]`.

`STORE_SUBSCR`
Implementa `TOS1[TOS] = TOS2`.

`DELETE_SUBSCR`
Implementa `del TOS1[TOS]`.

Codici di operazione vari.

`PRINT_EXPR`
Implementa l'istruzione di espressione per la modalità interattiva. Il TOS viene rimosso dalla pila e stampato. Nella modalità non interattiva, una dichiarazione di espressione finisce con `POP_STACK`.

`PRINT_ITEM`
Stampa il TOS nell'oggetto simile a file alla fine di `sys.stdout`. C'è un'istruzione del genere per ogni elemento nella dichiarazione `print`.

`PRINT_ITEM_TO`
Come `PRINT_ITEM`, ma stampa il secondo elemento dal TOS verso l'oggetto simile a file in TOS. Questo viene usato dall'istruzione `print` estesa.

`PRINT_NEWLINE`
Stampa una nuova riga in `sys.stdout`. Questa viene generata come ultima operazione di un'istruzione `print`, a meno che l'istruzione non finisca con una virgola.

`PRINT_NEWLINE_TO`
Come `PRINT_NEWLINE`, ma stampa la nuova riga nell'oggetto simile a file sul TOS. Questo viene usato dall'istruzione `print` estesa.

BREAK_LOOP
Esce da un ciclo a causa di un'istruzione `break`.

CONTINUE_LOOP *target*
Continua un ciclo a causa di un'istruzione `continue`. *target* è l'indirizzo in cui saltare(il quale dovrebbe essere una istruzione `FOR_ITER`).

LOAD_LOCALS
Mette un riferimento alle variabili locali dello spazio di visibilità (NdT: scope) corrente sulla pila. Questo viene usato nel codice per una definizione di classe: dopo che il corpo della classe viene valutato le variabili locali vengono passate alla definizione di classe.

RETURN_VALUE
Restituisce con il TOS al chiamante della funzione,

YIELD_VALUE
Rimuove il `TOS` e lo restituisce da un generatore.

IMPORT_STAR
Carica tutti i simboli che non iniziano con '_' direttamente dal modulo TOS nello spazio dei nomi locale. Il modulo viene rimosso dopo aver caricato tutti i nomi. Questo codice di operazione implementa `from module import *`.

EXEC_STMT
Implementa `exec TOS2,TOS1,TOS`. Il compilatore riempie i parametri facoltativi mancanti con `None`.

POP_BLOCK
Rimuove un blocco dallo pila dei blocchi. Per ogni frame, esiste una pila di blocchi, che indica cicli annidati, istruzioni try ed altro.

END_FINALLY
Termina una clausola `finally`. L'interprete ricorda se l'eccezione deve essere nuovamente sollevata, o se la funzione deve restituire e continua con il prossimo blocco esterno.

BUILD_CLASS
Crea una nuovo oggetto classe. Il TOS è il dizionario dei metodi, il TOS1 è la tupla dei nomi delle classi base ed il TOS2 il nome della classe.

Tutti i codici di operazione seguenti si aspettano argomenti. Un argomento è di due byte, con l'ultimo byte più significativo.

STORE_NAME *namei*
Implementa `name = TOS`. *namei* è l'indice di *name* nell'attributo `co_names` dell'oggetto codice. Il compilatore prova ad usare, se è possibile, `STORE_LOCAL` o `STORE_GLOBAL`.

DELETE_NAME *namei*
Implementa `del name`, dove *namei* è l'indice nell'attributo `co_names` dell'oggetto codice.

UNPACK_SEQUENCE *count*
Spacchetta TOS in *count* valori individuali, che vengono inseriti sopra lo stack da destra verso sinistra.

DUP_TOPX *count*
Duplica *count* elementi, prendendoli nello stesso ordine. A causa di limiti implementativi, *count* dovrebbe essere tra `1` e `5` inclusi.

STORE_ATTR *namei*
Implementa `TOS.name = TOS1`, dove *namei* è l'indice di *name* in `co_names`.

DELETE_ATTR *namei*
Implementa `del TOS.name`, usando *namei* come indice in `co_names`.

STORE_GLOBAL *namei*
Funziona come `STORE_NAME`, ma registra il nome come globale.

DELETE_GLOBAL *namei*
Funziona come `DELETE_NAME`, ma cancella un nome globale.

LOAD_CONST *consti*

Inserisce ‘co_consts[*consti*]’ in cima alla pila.

LOAD_NAME *namei*
Inserisce il valore associato con ‘co_names[*namei*]’ in cima alla pila.

BUILD_TUPLE *count*
Crea una tupla consumando *count* elementi dalla pila ed inserendo la tupla risultante in cima alla pila.

BUILD_LIST *count*
Funziona come BUILD_TUPLE, ma crea una lista.

BUILD_MAP *zero*
Inserisce un nuovo oggetto dizionario vuoto in cima alla pila. L’argomento viene ignorato ed impostato a zero dal compilatore.

LOAD_ATTR *namei*
Sostituisce il TOS con getattr(TOS, co_names[*namei*]).

COMPARE_OP *opname*
Svolge un’operazione Booleana. Il nome dell’operazione può essere trovato in cmp_op[*opname*].

IMPORT_NAME *namei*
Importa il modulo co_names[*namei*]. L’oggetto modulo viene inserito in cima alla pila. Lo spazio dei nomi corrente non ne viene influenzato: per una dichiarazione import adeguata, una successiva istruzione STORE_FAST modifica lo spazio dei nomi.

IMPORT_FROM *namei*
Carica gli attributi co_names[*namei*] dal modulo trovato in TOS. L’oggetto risultante viene inserito in cima alla pila, per essere successivamente immagazzinato da una istruzione STORE_FAST.

JUMP_FORWARD *delta*
Incrementa il contatore del bytecode di *delta*.

JUMP_IF_TRUE *delta*
Se il TOS è vero, incrementa il contatore del bytecode di *delta*. Il TOS viene lasciato nella pila.

JUMP_IF_FALSE *delta*
Se il TOS è falso, incrementa il bytecode di *delta*. Il TOS non viene modificato.

JUMP_ABSOLUTE *target*
Imposta il contatore del bytecode a *target*.

FOR_ITER *delta*
TOS è un iteratore. Viene chiamato il suo metodo next(). Se questo genera un nuovo valore, lo inserisce sopra la pila (lasciando l’iteratore sotto di esso). Se l’iteratore che lo indica è esaurito il TOS viene rimosso ed il contatore del bytecode viene incrementato di *delta*.

LOAD_GLOBAL *namei*
Carica l’attributo globale chiamato co_names[*namei*] sopra la pila.

SETUP_LOOP *delta*
Mette un blocco per un ciclo sopra la pila dei blocchi. Il blocco si estende dalla istruzione corrente con la misura di *delta* byte.

SETUP_EXCEPT *delta*
Mette un blocco try da una clausola try-except sulla pila dei blocchi. *delta* punta al primo blocco except.

SETUP_FINALLY *delta*
Mette un blocco try da una clausola try-except sulla pila dei blocchi. *delta* punta al blocco finally.

LOAD_FAST *var_num*
Mette un riferimento al co_varnames[*var_num*] locale sopra la pila.

STORE_FAST *var_num*
Immagazzina il TOS nel co_varnames[*var_num*] locale.

DELETE_FAST *var_num*
Cancella il co_varnames[*var_num*] locale.

LOAD_CLOSURE *i*
Mette un riferimento alla cella contenuta nello slot *i* della cella e libera la variabile memorizzata. Il nome della variabile è `co_cellvars[`*i*`]` se *i* è minore della lunghezza di *co_cellvars*. Altrimenti assume il valore `co_freevars[`*i* `- len(co_cellvars)]`.

LOAD_DEREF *i*
Carica la cella contenuta nello slot *i* della cella e libera la variabile memorizzata. Inserisce un riferimento per l'oggetto cella contenuto nella pila.

STORE_DEREF *i*
Immagazzina il TOS nella cella contenuta nello slot *i* della cella e libera la variabile memorizzata.

SET_LINENO *lineno*
Questo codice di operazione è obsoleto.

RAISE_VARARGS *argc*
Solleva un'eccezione. *argc* indica il numero di parametri per l'istruzione `raise`, con un intervallo da `0` a `3`. Il gestore troverà la traceback come TOS2, il parametro come TOS1, e l'eccezione come TOS.

CALL_FUNCTION *argc*
Chiama una funzione. Il byte meno significativo di *argc* indica il numero di parametri posizionali, il byte più significativo il numero dei parametri a parola chiave. Sopra la pila, il codice di operazione trova prima i parametri a parola chiave. Per ogni argomento a parola chiave, il valore si trova all'inizio della chiave. Sotto i parametri a parola chiave, si trovano i parametri posizionali, con in cima il parametro più a destra. Sotto i parametri si trova l'oggetto funzione da chiamare.

MAKE_FUNCTION *argc*
Mette un nuovo oggetto funzione sopra la pila. TOS è il codice associato con quella funzione. L'oggetto funzione viene definito per avere *argc* parametri predefiniti, che si trovano sotto il TOS.

MAKE_CLOSURE *argc*
Crea una nuovo oggetto funzione, imposta il suo slot *func_closure* e lo mette sopra la pila. TOS è il codice associato con la funzione. Se l'oggetto codice possiede N variabili libere, i prossimi N elementi sulla pila sono le celle per queste variabili. La funzione ha anche *argc* parametri predefiniti, che si trovano prima delle celle.

BUILD_SLICE *argc*
Mette un oggetto fetta sopra lo stack. *argc* deve essere `2` o `3`. Se è `2`, viene inserito `slice(TOS1, TOS)`; se è `3` viene inserito `slice(TOS2, TOS1, TOS)`. Vedete la funzione built-in `slice()` per maggiori informazioni.

EXTENDED_ARG *ext*
Applica un prefisso ad ogni codice di operazione che ha un argomento troppo grande per adattarsi nei due byte predefiniti. *ext* contiene due byte addizionali che, presi insieme all'argomento seguente del codice di operazione, comprende un argomento a 4 byte, *ext* rappresenta i due byte più significativi.

CALL_FUNCTION_VAR *argc*
Chiama una funzione. *argc* viene interpretato come in `CALL_FUNCTION`. L'elemento in cima alla pila contiene l'elenco degli argomenti a variabile, seguito dagli argomenti a parola chiave e da quelli posizionali.

CALL_FUNCTION_KW *argc*
Chiama una funzione. *argc* viene interpretato come in `CALL_FUNCTION`. L'elemento in cima alla pila contiene il dizionario degli argomenti a parola chiave seguito dagli argomenti a parola chiave esplicita e da quelli posizionali.

CALL_FUNCTION_VAR_KW *argc*
Chiama una funzione. *argc* viene interpretato come in `CALL_FUNCTION`. L'elemento in cima alla pila contiene il dizionari degli argomenti a parola chiave seguito dalla tupla degli argomenti a variabile, ulteriormente seguito dagli argomenti a parola chiave esplicita e da quelli posizionali.

18.11 `distutils` — Costruzione ed installazione di moduli Python

Il package `distutils` fornisce il supporto per la costruzione e l'installazione di moduli aggiuntivi all'interno di un'installazione Python. I nuovi moduli possono essere scritti in 100% puro Python, possono essere moduli di estensione scritti in C, oppure potrebbero essere raccolte di pacchetti Python che includono moduli scritti sia in Python che in C.

Questo pacchetto viene descritto in due documenti separati, inclusi nella documentazione di Python. Per saperne di più riguardo la distribuzione dei nuovi moduli con l'utilizzo delle funzioni di `distutils`, vedete il documento *Distribuire moduli Python*; questo documento comprende la descrizione di tutto il necessario per estendere distutils. Per sapere come installare moduli Python e per sapere se l'autore abbia utilizzato o meno il pacchetto `distutils`, vedete il documento *Installare moduli Python*.

Vedete anche:

Distribuire moduli Python
(../dist/dist.html)
: Il manuale per sviluppatori e creatori di pacchetti di moduli Python. Questo manuale descrive come preparare pacchetti basati su `distutils`, facili da installare in una distribuzione di Python preesistente .

Installare moduli Python
(../inst/inst.html)
: Un manuale per "amministratori" che include informazioni sull'installazione dei moduli in una installazione di Python preesistente. Non è necessario essere programmatori Python per leggere questo manuale.

CAPITOLO

DICIANNOVE

Il package di compilazione per Python

Il package di compilazione di Python è uno strumento per l'analisi del codice sorgente Python e la generazione di bytecode Python. Il compilatore contiene delle librerie per la generazione di un albero di sintassi astratta a partire dal codice sorgente Python, da cui viene generato il relativo bytecode.

Il package `compiler` è uno strumento scritto in Python per la traduzione dal codice sorgente Python al bytecode. Esso utilizza il parser built-in ed il modulo standard `parser` per la generazione di un albero di sintassi concreta. Questo albero viene usato per generare un albero di sintassi astratta (AST) e in seguito il bytecode Python.

La piena funzionalità del package rispecchia il compilatore built-in fornito con l'interprete Python. È stato concepito per avere una corrispondenza di comportamento identica. Allora perché realizzare un altro compilatore che fa le stesse cose? Il package è utile per svariati scopi. Può venire modificato più facilmente del compilatore built-in. L'albero AST che genera è inoltre utile per analizzare il codice sorgente Python.

Questo capitolo spiega il funzionamento dei vari componenti del package `compiler`. Viene combinato il materiale di riferimento con un tutorial.

Fanno parte del package `compiler` i seguenti moduli:

19.1 L'interfaccia di base

Il livello più alto del package definisce quattro funzioni. Se si importa `compiler`, si avranno a disposizione queste funzioni ed un insieme di moduli contenuti all'interno del package.

`parse`(*buf*)
: Restituisce un albero di sintassi astratta per il codice sorgente Python contenuto in *buf*. Se il codice sorgente presenta degli errori, la funzione solleva l'eccezione `SyntaxError`. Il valore restituito è un'istanza `compiler.ast.Module` che contiene l'albero.

`parseFile`(*path*)
: Restituisce un albero di sintassi astratta per il codice sorgente Python contenuto nel file specificato dal percorso *path*. È equivalente a `parse(open(`*path*`).read())`.

`walk`(*ast, visitor*[, *verbose*])
: Esegue un attraversamento anticipato (pre-order) dell'albero di sintassi astratta *ast*. Chiama il metodo appropriato sull'istanza *visitor* per ogni nodo che incontra.

`compile`(*source, filename, mode, flags=None, dont_inherit=None*)
: Compila la *source*, che può essere una stringa, un modulo Python, un istruzione o espressione, creando un oggetto codice che può venire eseguito dall'istruzione `exec()` o `eval()`. Questa funzione è un rimpiazzo per la funzione built-in `compile()`.

  filename, ovvero il nome del file, viene usato per i messaggi di errore run-time.

  La modalità *mode* può essere 'exec' per compilare un modulo, 'single' per compilare una singola istruzione (interattiva), o 'eval' per compilare un'espressione.

  Gli argomenti *flags* e *dont_inherit* hanno effetto sulle dichiarazioni future, ma non vengono ancora supportati.

compileFile(*source*)
Compila il file sorgente *source* e genera un file .pyc.

Il package `compiler` contiene i seguenti moduli: `ast`, `consts`, `future`, `misc`, `pyassem`, `pycodegen`, `symbols`, `transformer` e `visitor`.

19.2 Limitazioni

Esistono alcuni problemi riguardo il controllo degli errori del package compiler. L'interprete rileva gli errori di sintassi in due fasi distinte. Una prima categoria di errori viene rilevata tramite il parser dell'interprete, mentre la seconda attraverso il compilatore. Il package compiler si affida al parser dell'interprete e prende per buona la prima fase di ricerca degli errori. Esso realizza da sé la seconda fase rendendo l'implementazione incompleta. Per esempio, il package compiler non solleva eccezioni se un nome appare più di una volta nella lista degli argomenti:`def f(x, x): ...`

Una futura versione del compilatore dovrebbe sistemare questi problemi.

19.3 La sintassi astratta di Python

Il modulo `compiler.ast` definisce una sintassi astratta per Python. Nell'albero di sintassi astratta, ogni nodo rappresenta un costrutto sintattico. La radice dell'albero è l'oggetto `Module`.

La sintassi astratta offre un'interfaccia di più alto livello per l'analisi del codice sorgente Python. Il modulo `parser` ed il compilatore scritto in C per l'interprete di Python utilizzano un albero della sintassi concreta. La sintassi concreta è legata strettamente alla descrizione grammaticale usata per il parser di Python. Invece di un singolo nodo per un costrutto ci sono spesso diversi livelli di nodi nidificati introdotti dalle regole di precedenza di Python.

L'albero di sintassi astratta viene creato dal modulo `compiler.transformer`. Il transformer dipende dal parser built-in di Python per generare un albero di sintassi concreta. Esso genera un albero di sintassi astratta da uno di sintassi concreta.

Il modulo `transformer` è stato creato da e Bill Tutt per un compilatore sperimentale da Python a C. La versione corrente contiene un numero di modificazioni e miglioramenti, ma per la forma base della sintassi astratta e del transformer dobbiamo ringraziare Stein e Tutt.

19.3.1 Nodi AST

Il modulo `compiler.ast` viene generato da un file di testo che descrive ogni tipo di nodo ed i suoi elementi. Ogni tipo di nodo viene rappresentato come una classe che eredita dalla classe base astratta `compiler.ast.Node` e definisce un insieme di attributi denominati per i nodi figli.

class Node()
Le istanze `Node` vengono create automaticamente dal generatore del parser. Per le specifiche istanze `Node` si consiglia di utilizzare un'interfaccia che sfrutti gli attributi pubblici per l'accesso ai nodi figli. Un attributo pubblico può essere legato ad un singolo nodo o ad una sequenza di nodi, a seconda del tipo di `Node`. Per esempio l'attributo `bases` di un nodo `Class` è legato a una lista di nodi di classi base, e l'attributo `doc` viene legato ad un singolo nodo.

Ogni istanza di `Node` possiede un attributo `lineno`, che potrebbe essere `None`.

XXX Non sono sicuro che le regole per alcuni nodi avranno un'attributo `lineno`.utile

Tutti gli oggetti `Node` offrono i seguenti metodi:

getChildren()
Restituisce una lista appiattita dei nodi figli nell'ordine in cui appaiono. Specificatamente, l'ordine dei nodi è quello nel quale compaiono nella grammatica di Python. Non tutti i figli sono istanze di `Node`. I nomi delle funzioni e classi, per esempio, sono semplici stringhe.

`getChildNodes()`

Restituisce una lista appiattita di nodi figli nell'ordine in cui si presentano. Il metodo è simile a `getChildren()`, ad eccezione del fatto che restituisce solo quei figli che sono istanze di `Node`.

Questi due esempi illustrano la struttura generale delle classi `Node`. L'istruzione `while` viene definita attraverso la seguente produzione grammaticale:

```
while_stmt:     "while" expression ":" suite
               ["else" ":" suite]
```

Il nodo `While` ha tre attributi: `test`, `body` ed `else_` (se il nome naturale di un attributo è anche una parola riservata di Python allora non può essere usato come nome di un attributo. Una sottolineatura viene aggiunta alla parola per renderla un identificatore valido, quindi `else_` invece di `else`).

L'istruzione `if` è più complicata, perché può includere diversi test.

```
if_stmt: 'if' test ':' suite ('elif' test ':' suite)* ['else' ':' suite]
```

Il nodo `If` definisce soltanto due attributi: `tests` ed `else_`. L'attributo `tests` è una sequenza di espressioni da valutare, coppie di istruzioni consecutive. C'è una coppia per ogni clausola `if`/`elif`. Il primo elemento della coppia è l'espressione da valutare. Il secondo elemento è un nodo `Stmt` che contiene il codice da eseguire se la condizione è vera.

Il metodo `getChildren()` di `If` restituisce una lista appiattita di nodi figli. Se ci sono tre clausole `if`/`elif` e nessuna clausola `else`, allora `getChildren()` restituirà una lista di sei elementi: la prima espressione da valutare, il primo `Stmt`, il secondo testo dell'espressione, etc.

La seguente tabella elenca ognuna delle sottoclassi di `Node`, definite in `compiler.ast` ed ognuno degli attributi pubblici disponibili nelle rispettive istanze. I valori della gran parte degli attributi sono essi stessi istanze di `Node` o sequenze di istanze. Quando il valore è qualcosa di diverso da un'istanza, il tipo viene specificato nel commento a fianco. Gli attributi vengono elencati nell'ordine nel quale vengono restituiti dai metodi `getChildren()` e `getChildNodes()`.

Tipo di Nodo	Attributo	Valore
`Add`	`left`	operando sinistro
	`right`	operando destro
`And`	`nodes`	lista degli operandi
`AssAttr`		*attributo destinazione dell'assegnamento*
	`expr`	espressione a sinistra del punto
	`attrname`	il nome dell'attributo, una stringa
	`flags`	XXX
`AssList`	`nodes`	elenco degli elementi della lista che vengono assegnati
`AssName`	`name`	nome che viene assegnato
	`flags`	XXX
`AssTuple`	`nodes`	lista di elementi della tupla che vengono assegnati
`Assert`	`test`	l'espressione da valutare
	`fail`	il valore di `AssertionError`
`Assign`	`nodes`	una lista degli obiettivi dell'assegnamento, uno per ogni segno di uguale
	`expr`	il valore che viene assegnato
`AugAssign`	`node`	
	`op`	
	`expr`	
`Backquote`	`expr`	
`Bitand`	`nodes`	
`Bitor`	`nodes`	
`Bitxor`	`nodes`	
`Break`		

Tipo di Nodo	Attributo	Valore
`CallFunc`	`node`	espressione per la chiamata
	`args`	una lista di argomenti
	`star_args`	il valore di *-arg esteso
	`dstar_args`	il valore di **-arg esteso
`Class`	`name`	il nome della classe, una stringa
	`bases`	una lista di classi base
	`doc`	doc string, una stringa o `None`
	`code`	il corpo dell'istruzione class
`Compare`	`expr`	
	`ops`	
`Const`	`value`	
`Continue`		
`Dict`	`items`	
`Discard`	`expr`	
`Div`	`left`	
	`right`	
`Ellipsis`		
`Exec`	`expr`	
	`locals`	
	`globals`	
`For`	`assign`	
	`list`	
	`body`	
	`else_`	
`From`	`modname`	
	`names`	
`Function`	`name`	nome usato in def, una stringa
	`argnames`	lista dei nomi degli argomenti, come stringhe
	`defaults`	lista dei valori predefiniti
	`flags`	xxx
	`doc`	doc string, una stringa o `None`
	`code`	il corpo della funzione
`Getattr`	`expr`	
	`attrname`	
`Global`	`names`	
`If`	`tests`	
	`else_`	
`Import`	`names`	
`Invert`	`expr`	
`Keyword`	`name`	
	`expr`	
`Lambda`	`argnames`	
	`defaults`	
	`flags`	
	`code`	
`LeftShift`	`left`	
	`right`	
`List`	`nodes`	
`ListComp`	`expr`	
	`quals`	
`ListCompFor`	`assign`	
	`list`	
	`ifs`	
`ListCompIf`	`test`	
`Mod`	`left`	
	`right`	

Tipo di Nodo	Attributo	Valore
`Module`	`doc`	doc string, una stringa o `None`
	`node`	corpo del modulo, un `Stmt`
`Mul`	`left`	
	`right`	
`Name`	`name`	
`Not`	`expr`	
`Or`	`nodes`	
`Pass`		
`Power`	`left`	
	`right`	
`Print`	`nodes`	
	`dest`	
`Printnl`	`nodes`	
	`dest`	
`Raise`	`expr1`	
	`expr2`	
	`expr3`	
`Return`	`value`	
`RightShift`	`left`	
	`right`	
`Slice`	`expr`	
	`flags`	
	`lower`	
	`upper`	
`Sliceobj`	`nodes`	lista di istruzioni
`Stmt`	`nodes`	
`Sub`	`left`	
	`right`	
`Subscript`	`expr`	
	`flags`	
	`subs`	
`TryExcept`	`body`	
	`handlers`	
	`else_`	
`TryFinally`	`body`	
	`final`	
`Tuple`	`nodes`	
`UnaryAdd`	`expr`	
`UnarySub`	`expr`	
`While`	`test`	
	`body`	
	`else_`	
`Yield`	`value`	

19.3.2 Assegnamento dei nodi

Esiste una raccolta di nodi utilizzati per rappresentare gli assegnamenti. Ogni istruzione di assegnamento nel codice sorgente diventa un singolo nodo `Assign` nell'AST. L'attributo `nodes` è una lista che contiene un nodo per ogni obiettivo dell'assegnamento. Questo è necessario perché l'assegnamento può venire concatenato, per esempio `a = b = 2`. Ogni `Node` nella lista sarà una di queste classi: `AssAttr`, `AssList`, `AssName` o `AssTuple`.

Ciascun nodo obiettivo dell'assegnamento descriverà il tipo di oggetto per il quale è stato assegnato: `AssName` per un nome semplice, per esempio `a = 1`. `AssAttr` per l'assegnamento di un attributo, per esempio `a.x =`

1. `AssList` ed `AssTuple` rispettivamente per l'espansione di una lista o di una tupla, per esempio `a, b, c = a_tuple`.

I nodi obiettivo dell'assegnamento possiedono inoltre un attributo `flags` che indica se il nodo venga utilizzato per un assegnamento o per un'istruzione di cancellazione. `AssName` viene usato anche per rappresentare un'istruzione di cancellazione, ad esempio `del x`.

Quando un'espressione contiene alcuni riferimenti ad attributi, un'istruzione di assegnamento o cancellazione conterrà solo un nodo `AssAttr` – per il riferimento finale all'attributo. Gli altri riferimenti agli attributi verranno rappresentati come nodi `Getattr` nell'attributo `expr` dell'istanza `AssAttr`.

19.3.3 Esempi

Questa sezione mostra vari semplici esempi di AST per codice sorgente Python. Gli esempi mostrano come usare la funzione `parse()`, come si presenta il "repr" di un AST e come accedere agli attributi di un nodo AST.

Il primo modulo definisce una singola funzione. Si assuma che sia memorizzata in '/tmp/doublelib.py'.

```
"""Questo è un modulo di esempio.

Questa è la docstring.
"""

def double(x):
    "Restituisce due volte l'argomento"
    return x * 2
```

Nella sessione interattiva dell'interprete qui sotto, per favorire la leggibilità, vengono riformattate le lunghe repr dell'AST. Le repr AST usano nomi di classi non qualificate. Se si vuole creare un'istanza a partire da una repr, si devono importare i nomi della classe dal modulo `compiler.ast`.

```
>>> import compiler
>>> mod = compiler.parseFile("/tmp/doublelib.py")
>>> mod
Module('Questo è un modulo di esempio.\n\nQuesta è la docstring.\n',
       Stmt([Function('double', ['x'], [], 0, 'Restituisce due volte l'argomento',
       Stmt([Return(Mul((Name('x'), Const(2))))]))]))
>>> from compiler.ast import *
>>> Module('Questo è un modulo di esempio.\n\nQuesta è la docstring.\n',
...    Stmt([Function('double', ['x'], [], 0, 'Restituisce due volte l'argomento',
...    Stmt([Return(Mul((Name('x'), Const(2))))]))]))
Module('Questo è un modulo di esempio.\n\nQuesta è la docstring.\n',
       Stmt([Function('double', ['x'], [], 0, 'Restituisce due volte l'argomento',
       Stmt([Return(Mul((Name('x'), Const(2))))]))]))
>>> mod.doc
'Questo è un modulo di esempio.\n\nQuesta è la docstring.\n'
>>> for node in mod.node.nodes:
...     print node
...
Function('double', ['x'], [], 0, 'Restituisce due volte l'argomento',
         Stmt([Return(Mul((Name('x'), Const(2))))]))
>>> func = mod.node.nodes[0]
>>> func.code
Stmt([Return(Mul((Name('x'), Const(2))))])
```

19.4 Utilizzo dei Visitor per percorrere gli AST

Il modello visitor è ... Il package `compiler` utilizza una variante del modello visitor che sfrutta i vantaggi delle caratteristiche introspettive di Python per eliminare la necessità di gran parte dell'infrastruttura visitor.

Non c'è bisogno che le classi da visitare siano state programmate per accettare i visitor. Il visitor necessita solo di definire i metodi di visita per le classi a cui è specificamente interessato; un metodo di visita predefinito può gestire il resto.

`walk`(*tree, visitor*[, *verbose*])

class `ASTVisitor`()

L'`ASTVisitor` è responsabile dell'attraversamento dell'albero nell'ordine corretto. Il percorrimento inizia con una chiamata a `preorder()`. Per ogni nodo, passa l'argomento *visitor* a `preorder()` alla ricerca un metodo chiamato 'visitNodeType', dove NodeType è il nome della classe del nodo; per esempio, per un nodo `While` dovrebbe essere chiamato un metodo `visitWhile()`. Se il metodo esiste, viene chiamato con il nodo come suo primo argomento.

Il metodo visitor per un tipo particolare di nodo può controllare come i nodi figli vengano visitati durante il percorrimento. L'`ASTVisitor` modifica l'argomento visitor aggiungendo un metodo visit al visitor; questo metodo può essere usato per visitare un nodo figlio particolare. Se nessun visitor viene trovato per un particolare tipo di nodo, viene chiamato il metodo `default()`.

Gli oggetti `ASTVisitor` possiedono i seguenti metodi:

`default`(*node*[, *...*])

`dispatch`(*node*[, *...*])

`preorder`(*tree, visitor*)

19.5 Generazione di bytecode

Il generatore di codice è un visitor che emette bytecode. Ogni metodo visit può chiamare il metodo `emit()` per emettere nuovo bytecode. Il generatore di codice base è specializzato per moduli, classi e funzioni. Un assemblatore converte le istruzioni emesse in formato bytecode a basso livello. Gestisce cose come generatori di liste costanti di oggetti codice ed il calcolo del jump offset.

CAPITOLO

VENTI

Servizi specifici per SGI IRIX

I moduli descritti in questo capitolo forniscono interfacce a funzionalità proprie del sistema operativo IRIX della SGI (versioni 4 e 5).

al	Funzioni audio su piattaforme SGI.
AL	Costanti utilizzate con il modulo al.
cd	Interfaccia al CD-ROM su sistemi Silicon Graphics.
fl	La libreria FORMS per interfacce utente di tipo grafico.
FL	Costanti usate con il modulo fl.
flp	Funzioni per creare FORMS leggendone la specifica da file.
fm	Interfaccia verso il *Font Manager* per workstations SGI.
gl	Funzioni dalla *libreria grafica* della Silicon Graphics.
DEVICE	Costanti usate con il modulo gl.
GL	Costanti usate con il modulo gl.
imgfile	Supporto per file di tipo imglib SGI.
jpeg	Lettura e scrittura di file immagine compressi in formato JPEG.

20.1 al — Funzioni audio su piattaforme SGI

Questo modulo fornisce accesso alle funzionalità audio delle workstation Indy e Indigo di SGI. Vedere la sezione 3A delle pagine di manuale IRIX per i dettagli. È necessario leggere tali pagine per comprendere ciò che queste funzioni fanno! Alcune di queste non sono disponibili nelle versioni di IRIX precedenti la 4.0.5. Si ribadisce di vedere il manuale per controllare se una specifica funzione è disponibile sulla propria piattaforma.

Tutte le funzioni e i metodi definiti in questo modulo sono equivalenti alle funzioni C con le lettere 'AL' prefissate al proprio nome.

Le costanti simboliche contenute nel file d'intestazione C <audio.h> sono definite nel modulo standard AL, vedere di seguito.

Avvertenze: La versione corrente della libreria audio potrebbe provocare un core dump, qualora venissero passati valori errati, invece di restituire un errore. Sfortunatamente, dal momento che le circostanze precise nelle quali questo può accadere non sono documentate e sono difficili da verificare, l'interfaccia Python non può fornire alcuna protezione contro questo tipo di problemi. Un esempio è quello di specificare una dimensione eccessiva per la coda — non c'è alcuna documentazione riguardo al limite superiore.

Il modulo definisce le seguenti funzioni:

openport(*name*, *direction*[, *config*])
: Gli argomenti *name* e *direction* sono stringhe. L'argomento facoltativo *config* è un oggetto di configurazione del tipo restituito da newconfig(). Il valore restituito è un *oggetto porta audio*; i metodi di oggetti porta audio vengono descritti in seguito.

newconfig()
: Il valore di ritorno è un *nuovo oggetto di configurazione audio*; i metodi degli oggetti di configurazione audio vengono descritti in seguito.

queryparams(*device*)

L'argomento device è un intero. Il valore di restituito è una lista di interi contenente i dati restituiti da `ALqueryparams()`.

`getparams`(*device, list*)
L'argomento *device* è un intero. L'argomento *list* è una lista come quella restituita da `queryparams()`; viene modificata sul posto (!).

`setparams`(*device, list*)
L'argomento *device* è un intero. L'argomento *list* è una lista come quella restituita da `queryparams()`.

20.1.1 Oggetti configuration

Gli oggetti configuration restituiti da `newconfig()` hanno i seguenti metodi:

`getqueuesize`()
Restituisce la dimensione della coda.

`setqueuesize`(*size*)
Imposta la dimensione della coda.

`getwidth`()
Ottiene l'ampiezza di campionamento.

`setwidth`(*width*)
Imposta l'ampiezza di campionamento.

`getchannels`()
Ottiene il numero di canali.

`setchannels`(*nchannels*)
Imposta il numero di canali.

`getsampfmt`()
Ottiene il formato di campionamento.

`setsampfmt`(*sampfmt*)
Imposta il formato di campionamento.

`getfloatmax`()
Ottiene il valore massimo per i formati di campionamento nel tipo numerico in virgola mobile.

`setfloatmax`(*floatmax*)
Imposta il valore massimo per i formati di campionamento nel tipo numerico in virgola mobile.

20.1.2 Oggetti port

Gli oggetti di tipo port (NdT: porta audio), come restituiti da `openport()`, hanno i seguenti metodi:

`closeport`()
Chiude la porta.

`getfd`()
Restituisce il descrittore di file come intero.

`getfilled`()
Restituisce il numero di campionamenti riempiti.

`getfillable`()
Restituisce il numero di campionamenti riempibili.

`readsamps`(*nsamples*)
Legge un numero di campionamenti dalla coda, bloccandola se necessario. Restituisce i dati in forma di stringa contenente dati grezzi (ad esempio, 2 byte per campione in ordine big-endian (byte alto, byte basso) se si è impostata l'ampiezza di campionamento a 2 byte).

`writesamps`(*samples*)

Scrive campionamenti nella coda, bloccandola se necessario. I campionamenti vengono codificati come descritto per il valore restituito dal metodo `readsamps()`.

`getfillpoint``()`
Restituisce il 'fill point' (NdT: punto di riempimento).

`setfillpoint``(`*fillpoint*`)`
Imposta il 'punto di riempimento'.

`getconfig``()`
Restituisce un oggetto configuration contenente la configurazione corrente della porta.

`setconfig``(`*config*`)`
Imposta la configurazione usando l'argomento config, un oggetto configuration.

`getstatus``(`*list*`)`
Acquisisce informazioni di stato sull'ultimo errore.

20.2 AL — Costanti utilizzate con il modulo `al`

Questo modulo definisce le costanti simboliche necessarie per usare il modulo integrato `al` (vedere sopra); sono equivalenti a quelle definite nel file C d'intestazione `<audio.h>`, eccetto per il fatto che il prefisso del nome 'AL_' viene omesso. Leggere il sorgente del modulo per un elenco completo dei nomi definiti. Uso consigliato:

```
import al
from AL import *
```

20.3 cd — Accesso al CD-ROM su sistemi SGI

Questo modulo fornisce un'interfaccia alla libreria CD della Silicon Graphics. È disponibile quindi soltanto su sistemi Silicon Graphics.

La libreria opera come descritto di seguito. Un programma apre il dispositivo CD-ROM con `open()` e crea un parser con `createparser()` per analizzare i dati letti dal CD. L'oggetto restituito da `open()` può essere utilizzato per leggere i dati dal CD, ma anche per ottenere informazioni sullo stato del dispositivo CD-ROM, e sul CD stesso, come ad esempio l'elenco dei contenuti. I dati dal CD vengono passati al parser, che analizza i frame, e chiama qualsiasi funzione di callback che sia stata precedentemente aggiunta.

Un CD audio è diviso in *tracce* o *programmi* (i termini sono interscambiabili). Le tracce possono essere suddivise in indici. Un CD audio contiene una *tabella del suo contenuto*, che dà l'inizio per ogni traccia sul CD. L'indice `0` indica solitamente la pausa prima dell'inizio di una traccia. L'inizio della traccia come è dato dalla tabella del contenuto è normalmente l'inizio di indice `1`.

Le posizioni su un CD possono essere rappresentate in due modi. O tramite un numero di frame oppure una tupla di tre valori: minuti, secondi e frame. Gran parte delle funzioni usano la seconda rappresentazione. Le posizioni possono poi essere entrambe relative all'inizio del CD, e all'inizio della traccia.

Il modulo `cd` definisce le seguenti funzioni e costanti:

`createparser``()`
Crea e restituisce un oggetto parser opaco. I metodi di tale oggetto sono descritti di seguito.

`msftoframe``(`*minuti, secondi, frames*`)`
Converte una tripla di valori (*minuti*, *secondi*, *frames*), che indicano una codifica del tempo assoluto, nel corrispondente numero di frame del CD.

`open``([`*dispositivo*`[`, *modalità*`]])`
Apre il dispositivo CD-ROM. Il valore restituito è un oggetto player opaco; i metodi di tale oggetto vengono descritti di seguito. Il *dispositivo* è il nome di un file di dispositivo SCSI, ad esempio

`'/dev/scsi/sc0d4l0'`, oppure `None`. Se omesso oppure `None`, viene consultato l'inventario dell'hardware per localizzare un drive CD-ROM. La *modalità*, se non viene omessa, dovrebbe essere la stringa `'r'`.

Il modulo definisce le seguenti variabili:

exception `error`
Eccezione sollevata in seguito a vari errori.

`DATASIZE`
La dimensione del valore di un singolo frame di dati audio. Questa è la dimensione dei dati audio come viene passata alla funzione di callback di tipo `audio`.

`BLOCKSIZE`
La dimensione di un singolo frame di dati audio non interpretato.

Le seguenti variabili sono gli stati restituiti da `getstatus()`:

`READY`
Il drive è pronto ad operare con un CD audio inserito.

`NODISC`
Il drive non rileva un CD inserito.

`CDROM`
Il drive è caricato con un CD-ROM. Operazioni successive di play o di lettura restituiranno errori di I/O.

`ERROR`
Si è verificato un errore durante il tentativo di lettura del disco o della sua tabella del contenuto.

`PLAYING`
Il drive è in modalità player e suona un CD audio con uscita sui suoi jack audio.

`PAUSED`
Il drive è in modalità CD layer con suono in pausa.

`STILL`
L'equivalente di `PAUSED` su un vecchio modello Toshiba (non 3301). dispositivi CD-ROM. Tali dispositivi non sono mai stati venduti da SGI.

L'equivalente di `PAUSED` su vecchi (non 3301) dispositivi CD-ROM dei modelli Toshiba. Tali dispositivi non sono mai stati venduti da SGI.

`audio`
`pnum`
`index`
`ptime`
`atime`
`catalog`
`ident`
`control`
Interi costanti che descrivono i vari tipi di funzioni callback analizzatrici che possono essere impostate con il metodo `addcallback()` degli oggetti "CD parsers" (come si vedrà più avanti).

20.3.1 Oggetti Player

Gli oggetti di tipo "Player" (restituiti da `open()`) hanno i seguenti metodi:

`allowremoval``()`
Sblocca il bottone di espulsione del lettore CD-ROM permettendo all'utente di espellere il piatto porta CD se lo desidera.

`bestreadsize``()`
Restituisce il miglior valore possibile per il parametro *num_frames* del metodo `readda()`. Per migliore si intende il valore tale da permettere un flusso continuo di dati dal lettore CD-ROM.

close()
Libera le risorse associate all'oggetto "player". Dopo aver chiamato `close()`, i metodi dell'oggetto non dovrebbero più essere usati.

eject()
Espelle il piatto porta CD dal lettore CD-ROM.

getstatus()
Restituisce informazioni riguardanti lo stato corrente del lettore CD-ROM. Le informazioni restituite sono in forma di tupla avente i seguenti elementi: *stato*, *traccia*, *rtime*, *atime*, *ttime*, *first*, *last*, *scsi_audio*, *cur_block*. *rtime* è il tempo relativo all'inizio della traccia corrente; *atime* è il tempo relativo all'inizio del disco; *ttime* rappresenta la durata del disco. Per maggiori informazioni sul significato di questi valori, vedere la pagina di manuale di *CDgetstatus*(3dm). Il valore dello stato può essere uno dei seguenti: `ERROR`, `NODISC`, `READY`, `PLAYING`, `PAUSED`, `STILL` oppure `CDROM`.

gettrackinfo(*track*)
Restituisce informazioni riguardanti la traccia specificata. Le informazioni restituite sono in forma di una tupla di due elementi, corrispondenti al tempo iniziale e alla durata della traccia.

msftoblock(*min, sec, frame*)
Converte una tripla (minuti, secondi, frame) rappresentanti un riferimento temporale, in codice assoluto nel corrispondente numero di blocco logico per quel particolare lettore CD-ROM. È preferibile usare `msftoframe()` piuttosto che `msftoblock()` per confrontare riferimenti temporali. Alcuni lettori di CD fanno si che il numero di blocco logico differisca di un valore costante dal numero di frame.

play(*start, play*)
Inizia a riprodurre la traccia specificata di un CD audio inserito nel lettore. L'uscita audio viene inviata al connettore per cuffie del lettore CD (se presente). La riproduzione si ferma alla fine del disco. L'argomento *start* corrisponde al numero di traccia da cui cominciare a riprodurre il CD; se l'argomento *play* vale zero, il CD verrà messo in uno stato iniziale di pausa. Il metodo `togglepause()` può quindi essere usato per avviare l'effettiva riproduzione.

playabs(*minutes, seconds, frames, play*)
Come `play()`, eccetto per il fatto che il punto d'inizio della riproduzione viene fornito in minuti, secondi e frame invece che con il numero della traccia.

playtrack(*start, play*)
Come `play()`, eccetto per il fatto che la riproduzione si ferma al termine della traccia indicata.

playtrackabs(*track, minutes, seconds, frames, play*)
Come `play()`, eccetto per il fatto che la riproduzione comincia al punto specificato dal riferimento temporale assoluto e termina alla fine della traccia specificata.

preventremoval()
Blocca il bottone di espulsione sul lettore CD-ROM, impedendo in tal modo che l'utente possa arbitrariamente espellere il piatto porta CD.

readda(*num_frames*)
Legge il numero specificato di frame da un CD audio inserito nel lettore. Il valore restituito è una stringa che rappresenta i frame audio letti. Questa stringa può essere passata così com'è al metodo `parseframe()` di un oggetto parser.

seek(*minutes, seconds, frames*)
Imposta il puntatore che indica la posizione di partenza della prossima lettura di dati da un CD-ROM audio. Il puntatore viene impostato alla stessa posizione assoluta specificata dagli argomenti di input in *minuti*, *secondi* e numero di *frame*.

seekblock(*block*)
Modifica il puntatore che indica il punto di partenza della prossima lettura di dati da un CD-ROM audio. Il puntatore viene impostato al valore del numero logico del blocco specificato. Il valore restituito è quello stesso numero logico del blocco.

seektrack(*track*)
Imposta il puntatore che indica il punto di partenza della prossima lettura di dati da un CD-ROM audio. Il puntatore viene impostato al medesimo valore della traccia specificata. Il valore restituito è il numero logico

del blocco corrispondente alla posizione specificata.

`stop``()`
Ferma l'operazione di riproduzione in corso.

`togglepause``()`
Mette in pausa il lettore CD se questi stava riproducendo, avvia la riproduzione se era in pausa.

20.3.2 Oggetti Parser

Gli oggetti di tipo "parser" (restituiti da `createparser()`) hanno i seguenti metodi:

`addcallback``(`*tipo, func, arg*`)`
Aggiunge una funzione di chiamata per l'oggetto parser. Gli oggetti di tipo parser hanno chiamate corrispondenti ad otto tipi diversi di dati che possono essere letti in un flusso dati di audio digitale. Nel modulo `cd` (vedere i paragrafi precedenti) vengono definite delle costanti che identificano questi tipi di dati. Una funzione callback viene chiamata come segue: *func*`(`*arg*`, tipo, data)`, dove *arg* è l'argomento specificato dal codice utente, *tipo* è il particolare tipo di callback chiamata e *data* sono i dati passati a quel tipo di callback. Il tipo di dati dipende dal *tipo* di callback, nel modo che segue:

Tipo	Valore
`audio`	Una stringa che può essere passata immutata ad `al.writesamps()`.
`pnum`	Un numero intero corrispondente al numero di traccia (programma).
`index`	Un numero intero corrispondente al numero di indice.
`ptime`	Una tupla che esprime il tempo del programma in minuti, secondi e numero di frame.
`atime`	Una tupla che esprime il tempo assoluto in minuti, secondi e numero di frame.
`catalog`	Una stringa di 13 caratteri, corrispondente al numero di catalogo del CD.
`ident`	Una stringa di dodici caratteri corrispondente al numero di registrazione IRSC della registrazione. La stringa si compone di due caratteri per il codice della nazione, tre caratteri per il codice del proprietario, due caratteri per l'anno e cinque caratteri corrispondenti ad un numero seriale di identificazione.
`control`	Un numero intero corrispondente ai bit di controllo dei dati dei sotto-codici del CD.

`deleteparser``()`
Elimina l'oggetto parser e libera la memoria che questo stava usando. L'oggetto non dovrebbe più esser usato dopo questa chiamata. Questa chiamata viene effettuata automaticamente quando l'ultimo riferimento all'oggetto viene rimosso.

`parseframe``(`*frame*`)`
Analizza uno o più frame di dati audio digitali letti da un CD, quali quelli restituiti da `readda()`. Stabilisce quali sotto-codici sono presenti nei dati. Se tali sotto-codici sono cambiati rispetto all'ultimo frame, allora `parseframe()` esegue la funzione di chiamata del tipo appropriato, passandogli i dati del sotto-codice trovati nel frame. A differenza della funzione C, è possibile passare a questo metodo più di un frame di dati audio digitali.

`removecallback``(`*tipo*`)`
Rimuove la funzione callback per il *tipo* specificato.

`resetparser``()`
Re-inizializza i campi dell'oggetto parser usati per tener traccia dei sotto-codici, riportandoli ai valori iniziali. Il metodo `resetparser()` dovrebbe essere chiamato quando viene cambiato il CD.

20.4 `fl` — La libreria FORMS per interfacce utente di tipo grafico

Questo modulo fornisce un'interfaccia per la libreria FORMS di Mark Overmars. Il codice sorgente per questa libreria può essere scaricato con un FTP anonimo da '`ftp.cs.ruu.nl`', nella directory 'SGI/FORMS'. Questo modulo è stato testato l'ultima volta con la versione 2.0b di tale libreria.

La maggior parte delle funzioni di questo modulo sono una traduzione letterale delle funzioni equivalenti della libreria C, con l'eccezione che il prefisso '`fl_`' è stato rimosso dai loro nomi. Le costanti usate nella libreria sono definite nel modulo FL descritto più avanti.

La creazione di oggetti in Python è un tantino differente rispetto alla libreria in C: invece di esserci il concetto di 'form corrente' di cui la libreria tiene traccia ed a cui vengono aggiunti nuovi oggetti FORMS, si adotta l'approccio che tutte le funzioni che aggiungono un nuovo oggetto FORMS ad una form (NdT: finestra) esistente, sono metodi dell'oggetto Python corrispondente. Di conseguenza, non ci sono equivalenti Python per le funzioni `fl_addto_form()` e `fl_end_form()`, mentre l'equivalente di `fl_bgn_form()` è stato rinominato `fl.make_form()`.

Fare attenzione a potenziali confusioni di terminologia: parlando della libreria FORMS, si usa la parola *oggetto* per indicare i bottoni, i cursori e simili, che si possono inserire in una form. In Python, il termine 'oggetto' indica un qualsiasi valore. L'interfaccia Python alla libreria FORMS introduce due nuovi tipi di oggetto: gli oggetti form (rappresentanti un intero form) e gli oggetti FORMS (rappresentanti bottoni, cursori ed altro). Speriamo che tutto ciò non crei troppa confusione.

Non ci sono 'oggetti liberi' nell'interfaccia Python per la libreria FORMS, né vi è un modo facile di aggiungere nuove classi di oggetti FORMS scritte in Python. L'interfaccia esposta da FORMS verso la gestione di eventi GL è però disponibile, per cui è possibile combinare FORMS e semplici finestre GL.

Nota: importare il modulo `fl` comporta una chiamata alla funzione GL `foreground()` e alla routine FORMS `fl_init()`.

20.4.1 Funzioni definite nel modulo `fl`

Il modulo `fl` definisce le funzioni elencate di seguito. Per più informazioni circa quello che fanno, vedere la descrizione delle funzioni equivalenti in C, nella documentazione della libreria FORM.

`make_form`(*tipo, larghezza, altezza*)
Crea una form con *tipo*, *larghezza* ed *altezza* pari a quelle specificate dagli argomenti. Questa funzione restituisce un oggetto di tipo *form*, i cui metodi vengono descritti in seguito.

`do_forms`()
Il ciclo principale standard della libreria FORMS. Restituisce un oggetto Python che rappresenta l'elemento FORMS che richiede interazione, o il valore speciale `FL.EVENT`.

`check_forms`()
Controlla gli eventi della libreria FORMS. Restituisce lo stesso tipo di valore della funzione `do_forms()` descritta prima, oppure `None` se non vi sono eventi che richiedono un'immediata interazione.

`set_event_call_back`(*function*)
Imposta la funzione callback per la gestione degli eventi.

`set_graphics_mode`(*rgbmode, doublebuffering*)
Imposta le modalità grafiche.

`get_rgbmode`()
Restituisce il valore corrente del modo rgb. Questo corrisponde al valore della variabile globale C `fl_rgbmode`.

`show_message`(*str1, str2, str3*)
Fa apparire una finestra di dialogo con un messaggio di tre righe di testo ed un bottone OK.

`show_question`(*str1, str2, str3*)
Fa apparire una finestra di dialogo con un messaggio di tre righe e bottoni YES e NO. Restituisce `1` se l'utente preme il bottone YES, `0` se preme il bottone NO.

`show_choice`(*str1, str2, str3, but1*[, *but2*[, *but3*]])
Fa apparire una finestra di dialogo con un messaggio di tre righe e fino a tre bottoni. Restituisce il numero del bottone premuto dall'utente (`1`, `2` or `3`).

show_input(*prompt, default*)
Fa apparire una finestra di dialogo con un messaggio di una riga e un campo testo in cui l'utente può inserire una stringa. Il secondo argomento è il valore predefinito di tale stringa. Restituisce la stringa com'è stata modificata dall'utente.

show_file_selector(*message, directory, pattern, default*)
Fa apparire una finestra di dialogo in cui l'utente può selezionare un file. Restituisce il nome assoluto del file selezionato dall'utente, oppure `None` se l'utente preme il bottone Cancel.

get_directory()
get_pattern()
get_filename()
Queste funzioni restituiscono la directory, la maschera ed il nome di file (solo la parte finale) selezionati dall'utente nell'ultima chiamata alla funzione `show_file_selector()`.

qdevice(*dev*)
unqdevice(*dev*)
isqueued(*dev*)
qtest()
qread()
qreset()
qenter(*dev, val*)
get_mouse()
tie(*button, valuator1, valuator2*)
Queste funzioni sono l'interfaccia della libreria FORMS verso le corrispondenti funzioni GL. Da usare se si vuole gestire direttamente qualche evento GL quando si usa `fl.do_events()`. Quando viene riconosciuto un evento GL che la libreria FORMS non sa gestire, allora `fl.do_forms()` restituisce il valore speciale `FL.EVENT` ed allora è possibile chiamare `fl.qread()` per leggere gli eventi in coda. Si raccomanda di non usare le funzioni GL equivalenti!

color()
mapcolor()
getmcolor()
Vedere la descrizione di queste funzioni nella documentazione della libreria FORMS: `fl_color()`, `fl_mapcolor()` e `fl_getmcolor()`.

20.4.2 Oggetti Form

Gli oggetti form (restituiti dalla funzione `make_form()` descritta in precedenza) hanno i metodi descritti di seguito. Ciascun metodo corrisponde ad una funzione della libreria C il cui nome cominci con '`fl_`' e che prenda come primo parametro un puntatore ad una form; fare riferimento alla documentazione ufficiale della libreria FORMS per una completa descrizione di queste funzioni.

Tutti i metodi `add_*()` restituiscono un oggetto Python rappresentante l'oggetto FORMS. I metodi degli oggetti FORMS vengono descritti in seguito. Molti tipi di oggetti FORMS hanno inoltre qualche metodo specifico di quel tipo; tali metodi sono elencati in questa sezione.

show_form(*placement, bordertype, name*)
Fa apparire il form.

hide_form()
Nasconde il form.

redraw_form()
Ridisegna il form.

set_form_position(*x, y*)
Cambia la posizione del form.

freeze_form()

unfreeze_form()
Sblocca il form.

activate_form()
Attiva il form.

deactivate_form()
Disattiva il form.

bgn_group()
Inizializza un nuovo gruppo di oggetti; restituisce un oggetto che rappresenta il gruppo.

end_group()
Pone termine al gruppo di oggetti correntemente utilizzato.

find_first()
Cerca il primo oggetto nella form.

find_last()
Cerca l'ultimo oggetto nella form.

add_box(*type, x, y, w, h, name*)
Aggiunge un oggetto box (NdT: rettangolo) alla form. Non c'è nessun'altro metodo.

add_text(*type, x, y, w, h, name*)
Aggiunge un oggetto text (NdT:testo) alla form. Non c'è nessun'altro metodo.

add_clock(*type, x, y, w, h, name*)
Aggiunge un oggetto clock (NdT:orologio) alla form.
Metodo: `get_clock()`.

add_button(*type, x, y, w, h, name*)
Aggiunge un oggetto button (NdT:bottone) alla form.
Metodi: `get_button()`, `set_button()`.

add_lightbutton(*type, x, y, w, h, name*)
Aggiunge un oggetto lightbutton (NdT: bottone leggero) alla form.
Metodi: `get_button()`, `set_button()`.

add_roundbutton(*type, x, y, w, h, name*)
Aggiunge un roundbutton (NdT:bottone rotondo) alla form.
Metodi: `get_button()`, `set_button()`.

add_slider(*type, x, y, w, h, name*)
Aggiunge un oggetto slider (NdT:cursore a scorrimento) alla form.
Metodi: `set_slider_value()`, `get_slider_value()`, `set_slider_bounds()`, `get_slider_bounds()`, `set_slider_return()`, `set_slider_size()`, `set_slider_precision()`, `set_slider_step()`.

add_valslider(*type, x, y, w, h, name*)
Aggiunge un oggetto valslider (NdT:cursore a scorrimento con valore) alla form.
Metodi: `set_slider_value()`, `get_slider_value()`, `set_slider_bounds()`, `get_slider_bounds()`, `set_slider_return()`, `set_slider_size()`, `set_slider_precision()`, `set_slider_step()`.

add_dial(*type, x, y, w, h, name*)
Aggiunge un oggetto dial (NdT:indicatore analogico) alla form.
Metodi: `set_dial_value()`, `get_dial_value()`, `set_dial_bounds()`, `get_dial_bounds()`.

add_positioner(*type, x, y, w, h, name*)
Aggiunge un oggetto positioner (NdT:posizionatore) alla form.
Metodi: `set_positioner_xvalue()`, `set_positioner_yvalue()`, `set_positioner_xbounds()`, `set_positioner_ybounds()`, `get_positioner_xvalue()`, `get_positioner_yvalue()`, `get_positioner_xbounds()`, `get_positioner_ybounds()`.

add_counter(*type, x, y, w, h, name*)
Aggiunge un oggetto counter (NdT:contatore) alla form.
Metodi: `set_counter_value()`, `get_counter_value()`, `set_counter_bounds()`, `set_counter_step()`, `set_counter_precision()`, `set_counter_return()`.

add_input(*type, x, y, w, h, name*)
Aggiunge un oggetto input alla form.
Metodi: `set_input()`, `get_input()`, `set_input_color()`, `set_input_return()`.

add_menu(*type, x, y, w, h, name*)
Aggiunge un oggetto menu alla form.
Metodi: `set_menu()`, `get_menu()`, `addto_menu()`.

add_choice(*type, x, y, w, h, name*)
Aggiunge un oggetto choice (NdT:selezione) alla form.
Metodi: `set_choice()`, `get_choice()`, `clear_choice()`, `addto_choice()`, `replace_choice()`, `delete_choice()`, `get_choice_text()`, `set_choice_fontsize()`, `set_choice_fontstyle()`.

add_browser(*type, x, y, w, h, name*)
Aggiunge un oggetto browser (NdT: selettore) alla form.
Metodi: `set_browser_topline()`, `clear_browser()`, `add_browser_line()`, `addto_browser()`, `insert_browser_line()`, `delete_browser_line()`, `replace_browser_line()`, `get_browser_line()`, `load_browser()`, `get_browser_maxline()`, `select_browser_line()`, `deselect_browser_line()`, `deselect_browser()`, `isselected_browser_line()`, `get_browser()`, `set_browser_fontsize()`, `set_browser_fontstyle()`, `set_browser_specialkey()`.

add_timer(*type, x, y, w, h, name*)
Aggiunge un oggetto timer alla form.
Metodi: `set_timer()`, `get_timer()`.

Gli oggetti form hanno gli attributi elencati di seguito; vedere la documentazione della libreria FORMS per una descrizione più dettagliata.

Nome	Tipo C	Significato
`window`	int (in sola lettura)	Identificativo della finestra GL
`w`	float	larghezza della form
`h`	float	altezza della form
`x`	float	ascissa dall'origine della form
`y`	float	ordinata dall'origine della form
`deactivated`	int	è diversa da zero se la form è disattivata
`visible`	int	è diverso da zero se la form è visible
`frozen`	int	è diverso da zero se la form è bloccata
`doublebuf`	int	è diverso da zero se il doppio buffering è attivo

20.4.3 Oggetti FORMS

A parte i metodi per gestire tipi specifici di oggetti FORMS, tutti gli oggetti FORMS hanno i seguenti metodi:

set_call_back(*function, argument*)
Imposta la funzione callback ed il suo argomento. Quando un oggetto richiede interazione, la funzione callback verrà chiamata con due argomenti: il primo è l'oggetto, il secondo è l'argomento specificato nella chiamata a `set_call_back`. Gli oggetti FORMS per cui non viene specificata una funzione callback vengono restituiti dalle funzioni `fl.do_forms()` oppure `fl.check_forms()` quando questi richiedono interazione). Per rimuovere la funzione callback da un oggetto, si può chiamare questo metodo senza argomenti.

delete_object()
Cancella l'oggetto.

show_object()
Mostra l'oggetto.

hide_object()
Nasconde l'oggetto.

redraw_object()
Ridisegna l'oggetto.

freeze_object()
Blocca l'oggetto.

unfreeze_object()
Sblocca l'oggetto.

Gli oggetti FORMS hanno gli attributi descritti di seguito; in proposito vedere la documentazione della libreria FORMS.

Nome	Tipo C	Significato
objclass	int (in sola lettura)	classe dell'oggetto
type	int (in sola lettura)	tipo dell'oggetto
boxtype	int	tipo di oggetto box usato
x	float	ascissa del punto di origine
y	float	ordinata del punto di origine
w	float	larghezza
h	float	altezza
col1	int	colore primario
col2	int	colore secondario
align	int	allineamento
lcol	int	colore dell'etichetta
lsize	float	dimensione del font dell'etichetta
label	string	stringa dell'etichetta
lstyle	int	stile dell'etichetta
pushed	int (in sola lettura)	(vedere la documentazione di FORMS)
focus	int (in sola lettura)	(vedere la documentazione di FORMS)
belowmouse	int (in sola lettura)	(vedere la documentazione di FORMS)
frozen	int (in sola lettura)	(vedere la documentazione di FORMS)
active	int (in sola lettura)	(vedere la documentazione di FORMS)
input	int (in sola lettura)	(vedere la documentazione di FORMS)
visible	int (in sola lettura)	(vedere la documentazione di FORMS)
radio	int (in sola lettura)	(vedere la documentazione di FORMS)
automatic	int (in sola lettura)	(vedere la documentazione di FORMS)

20.5 FL — Costanti usate con il modulo fl

Questo modulo definisce le costanti simboliche necessarie per usare il modulo built-in fl (vedere sopra); sono equivalenti a quelle definite nel file header C <forms.h> tranne per il fatto che viene omesso il prefisso 'FL_' dai nomi. Uso suggerito:

```
import fl
from FL import *
```

20.6 flp — Funzioni per creare FORMS leggendone la specifica da file

Questo modulo definisce le funzioni capaci di leggere le definizioni di forms create dal programma 'form designer' (**fdesign**), che accompagna la libreria FORMS (vedere il modulo `fl` descritto precedentemente.

Per ora, vedere il file 'flp.doc' del sorgente della libreria di Python per una descrizione.

20.7 `fm` — Interfaccia verso il *Font Manager*

Questo modulo fornisce accesso alla libreria del *Font Manager* del sistema operativo IRIS. Il modulo è disponibile solo su macchine Silicon Graphics. Vedere anche: la *Guida Utente 4Sight*, sezione 1, capitolo 5: "Usare il Font Manager di IRIS."

Questo modulo non fornisce ancora un'interfaccia pienamente funzionale verso il Font Manager. Tra le funzionalità non supportate vi sono: operazioni con matrici, operazioni con la cache, operazioni con caratteri (usare al loro posto le operazioni con stringhe), alcuni dettagli delle informazioni sui font, metriche individuali dei glifi e la funzionalità di printer matching.

Il modulo supporta le seguenti operazioni:

`init``()`
: Funzione di inizializzazione. Chiama `fminit()`. Normalmente non è necessario chiamare questa funzione, giacché viene chiamata automaticamente la prima volta che il modulo `fm` viene importato.

`findfont``(`*fontname*`)`
: Restituisce un oggetto gestore dei font che permette di accedere al font corrispondente al nome dato. Chiama `fmfindfont(`*fontname*`)`.

`enumerate``()`
: Restituisce la lista dei nomi dei font disponibili. Questa funzione è un'interfaccia verso la funzione C `fmenumerate()`.

`prstr``(`*string*`)`
: Visualizza una stringa usando il font corrente (vedere di seguito il metodo `setfont()` degli oggetti gestori di font). Chiama `fmprstr(`*string*`)`.

`setpath``(`*string*`)`
: Imposta il percorso di ricerca dei font. Chiama `fmsetpath(`*string*`)`. (XXX Non funziona !?!)

`fontpath``()`
: Restituisce il percorso corrente di ricerca dei font.

Gli oggetti gestori dei font supportano le seguenti operazioni:

`scalefont``(`*factor*`)`
: Restituisce un gestore per una versione in scalata del font. Chiama `fmscalefont(`*fh*`, `*factor*`)`.

`setfont``()`
: Rende corrente il font rappresentato dall'oggetto. Nota: l'effetto viene annullato senza notifica quando l'oggetto viene cancellato. Chiama `fmsetfont(`*fh*`)`.

`getfontname``()`
: Restituisce il nome del font rappresentato dall'oggetto. Chiama `fmgetfontname(`*fh*`)`.

`getcomment``()`
: Restituisce la stringa di commento associata al font rappresentato dall'oggetto. Solleva un'eccezione se tale stringa non esiste. Chiama `fmgetcomment(`*fh*`)`.

`getfontinfo``()`
: Restituisce una tupla che contiene dati pertinenti al font rappresentato dall'oggetto. Questo metodo è un'interfaccia verso `fmgetfontinfo()`. La tupla restituita contiene i seguenti valori: (*printermatched*, *fixed_width* (indica se è a larghezza fissa), *xorig*, (ascissa dell'origine), *yorig* (ordinata dell'origine), *xsize* (dimensione sull'asse X), *ysize* (dimensione dell'asse y), *height* (altezza), *nglyphs*) (numero del glifo).

`getstrwidth``(`*string*`)`
: Restituisce la larghezza, in pixel, della stringa *string*, qualora venga visualizzata con il font rappresentato dal'oggetto. Chiama `fmgetstrwidth(`*fh*`, `*string*`)`.

20.8 `gl` — interfaccia verso la *libreria grafica*

Questo modulo fornisce l'accesso alla *libreria grafica* Silicon Graphics. Il modulo è disponibile solo su macchine Silicon Graphics.

Avvertenze: In alcuni casi, chiamare in modo improprio le funzioni della libreria GL manda in crash l'interprete Python. In particolare, l'uso di molte chiamate a funzioni GL non è sicuro prima di aprire la prima finestra.

Questo modulo è troppo grande per essere interamente documentato in questa sezione, ma quanto segue dovrebbe essere di aiuto per cominciare ad usarlo. Le convenzioni sui parametri delle funzioni C vengono trasformate in Python come segue:

- Tutti i valori interi (short, long, unsigned) vengono rappresentati da Python come interi.
- Tutti i valori in virgola mobile (float, double) vengono rappresentati dal tipo in virgola mobile di Python. In molti casi, viene anche consentito l'uso di numeri interi.
- Tutti gli array vengono rappresentati da liste Python ad una dimensione. In molti casi, è anche consentito l'uso di tuple.
- Tutti gli argomenti di tipo stringa e carattere vengono rappresentati dal tipo stringa di Python, ad esempio `winopen('Ciao, voialtri!')` e `rotate(900, 'z')`.
- Tutti gli argomenti di tipo intero (short, long, unsigned), come pure i valori restituiti, che siano usati per specificare la lunghezza di un argomento array, vengono omessi. Ad esempio la chiamata a funzione C:

  ```
  lmdef(deftype, index, np, props)
  ```

  diventa in Python

  ```
  lmdef(deftype, index, props)
  ```

- Gli argomenti di output vengono omessi dalla lista degli argomenti; vengono invece restituiti come elementi del valore restituito. Nei casi in cui debbano essere restituiti più di un valore, il valore restituito è una tupla. Se la funzione C ha sia un valore di ritorno regolare (che non venga omesso a causa della regola precedente) che un argomento di output, il valore restituito è sempre il primo elemento della tupla restituita. Ad esempio la chiamata a funzione C:

  ```
  getmcolor(i, &red, &green, &blue)
  ```

  diventa, in python:

  ```
  red, green, blue = getmcolor(i)
  ```

Le seguenti funzioni non sono standard o hanno speciali convenzioni per i loro argomenti:

`varray`(*argument*)
: Equivalente ad una serie di chiamate alla funzione `v3d()`, ma più veloce. L'argomento è una lista (o una tupla) di punti. Ciascun punto deve essere una tupla di coordinate (*x*, *y*, *z*) oppure (*x*, *y*). I punti possono essere bidimensionali o tridimensionali, ma tutti delle stesse dimensioni. I valori in virgola mobile e gli interi possono essere usati insieme. I punti vengono sempre convertiti in punti tridimensionali a doppia precisione assumendo, quando serva, che `z = 0.0` (come indicato dalla pagina di manuale); quindi la funzione `v3d()` viene chiamata per ciascun punto.

nvarray()
: Equivalente ad una serie di chiamate alle funzioni `n3f` e `v3f`, ma più veloce. L'argomento è una array (lista o tupla) di coppie di vettori normali e punti. Ciascuna coppia è una tupla composta di un punto e un vettore normale per quel punto. Ciascun punto o vettore va espresso come tupla di coordinate (*x*, *y*, *z*). È obbligatorio dare tre coordinate. Valori interi ed in virgola mobile possono essere mescolati. Per ciascuna coppia, viene chiamata la funzione `n3f()` per il vettore normale e quindi `v3f()` per il punto.

vnarray()
: Simile a `nvarray()`, ma ogni coppia deve avere come primo elemento il punto e come secondo il vettore normale (NdT: al contrario della funzione precedente).

nurbssurface(*s_k, t_k, ctl, s_ord, t_ord, type*)
: Definisce una superficie nurbs. Le dimensioni di *ctl*`[][]` vengono calcolate come segue: `[len(`*s_k*`) - `*s_ord*`]`, `[len(`*t_k*`) - `*t_ord*`]`.

nurbscurve(*knots, ctlpoints, order, type*)
: Definisce una curva nurbs. La lunghezza di ctlpoints è uguale a `len(`*knots*`) - `*order*.

pwlcurve(*points, type*)
: Definisce una curva lineare a tratti. Il parametro *points* è una lista di punti, il parametro *type* deve avere il valore `N_ST`.

pick(*n*)
select(*n*)
: Il solo argomento di queste funzioni specifica la dimensione voluta per il buffer da usare.

endpick()
endselect()
: Queste funzioni non hanno argomenti. Restituiscono una lista di interi rappresentanti la parte usata del buffer di pickle/select. Non viene fornito alcun metodo per individuare il sovraccarico del buffer.

Quello che segue è un piccolo ma completo esempio di programma Python che usa la Graphic Library:

```
import gl, GL, time

def main():
    gl.foreground()
    gl.prefposition(500, 900, 500, 900)
    w = gl.winopen('CrissCross')
    gl.ortho2(0.0, 400.0, 0.0, 400.0)
    gl.color(GL.WHITE)
    gl.clear()
    gl.color(GL.RED)
    gl.bgnline()
    gl.v2f(0.0, 0.0)
    gl.v2f(400.0, 400.0)
    gl.endline()
    gl.bgnline()
    gl.v2f(400.0, 0.0)
    gl.v2f(0.0, 400.0)
    gl.endline()
    time.sleep(5)

main()
```

Vedete anche:

PyOpenGL: L'interfaccia Python verso la libreria OpenGL
(http://pyopengl.sourceforge.net/)
: È anche disponibile un'interfaccia verso la libreria OpenGL; vedere a questo proposito le informazioni sul progetto **PyOpenGL** presso il sito http://pyopengl.sourceforge.net/. Questa opzione può essere da preferire, se non viene richiesto il supporto per hardware SGI precedente al 1996.

20.9 DEVICE — Costanti usate con il modulo gl

Questo modulo definisce le costanti usate dalla libreria *Graphics Library* della Silicon Graphics, le stesse costanti che i programmatori C trovano nel file header <gl/device.h>. Leggere il codice sorgente del modulo per i dettagli.

20.10 GL — Costanti usate con il modulo gl

Questo modulo contiene costanti usate dalla libreria *Graphics Library* della Silicon Graphics, corrispondenti a quelle del file header C <gl/gl.h>. Leggere il codice sorgente del modulo per i dettagli.

20.11 imgfile — Supporto per file di tipo imglib SGI

Il modulo imgfile permette ai programmi Python di accedere a file immagine con il formato SGI imglib (conosciuti anche come file '.rgb'). Il modulo è tutt'altro che completo, ma è distribuito egualmente poiché contiene sufficienti funzionalità da essere utile in qualche caso. Al momento i file di tipo mappa di colori (colormap) non vengono supportati.

Questo modulo definisce le seguenti variabili e funzioni:

exception error
Questa eccezione viene sollevata su tutti gli errori, come nel caso di tipi di file non supportati, etc. etc....

getsizes(*file*)
Questa funzione restituisce una tupla (*x*, *y*, *z*) dove *x* ed *y* corrispondono alla dimensione dell'immagine in pixel e *z* è il numero di byte per pixel. Al momento vengono supportate solo le immagini RGB con 3 byte per pixel e quelle in scala di grigi con 1 byte per pixel.

read(*file*)
Questa funzione legge e decodifica l'immagine nel file specificato e la restituisce come una stringa Python. La stringa è formata o da un byte per pixel (scale di grigi) o da 4 byte per pixel (formato RGBA). Il pixel in basso a sinistra è il primo della stringa. Questo formato è adatto ad essere passato, per esempio, al metodo gl.lrectwrite() per istanza.

readscaled(*file, x, y, filter*[, *blur*])
Questa funzione è identica a read() ma restituisce un'immagine che viene ridimensionata nel modo specificato dagli argomenti *x* ed *y*. Se gli argomenti *filter* (NdT:filtro) e *blur* (NdT:sfumatura) vengono omessi, il ridimensionamento viene effettuato semplicemente scartando o duplicando i pixel dell'immagine, per cui il risultato potrà non essere di buona qualità, soprattutto per immagini generate dal computer.

In alternativa, si può specificare un filtro da usare per rendere l'immagine più omogenea dopo che è stata ridimensionata. I tipi di filtro supportati sono 'impulse' (NdT: impulso), 'box' (NdT: scatola), 'triangle' (NdT: triangolo), 'quadratic' (NdT: quadratico) e 'gaussian' (NdT: gaussiano). Se l'argomento *filter* viene specificato, allora l'argomento *blur*, facoltativo, indica il livello di sfumatura che il filtro deve applicare. Il valore predefinito è 1.0.

La funzione readscaled() non fa alcun tentativo di mantenere il corretto rapporto X/Y dell'originale, per cui questo è di responsabilità dell'utilizzatore della funzione.

ttob(*flag*)
Questa funzione imposta un'opzione globale che stabilisce se le righe di scansione dell'immagine vengono lette o scritte dal basso in alto (quando l'opzione vale zero; questo è compatibile con la libreria GL della SGI) o dall'alto in basso (quando l'opzione vale uno; questo è compatibile con X). Il valore predefinito è zero.

write(*file, data, x, y, z*)
Questa funzione scrive i pixel in RGB o scala di grigi contenuti nell'argomento *data* nel file immagine di nome *file*. Gli argomenti *x* ed *y* definiscono la dimensione dell'immagine, mentre l'argomento *z* vale 1 per immagini a scale di grigi e 3 per immagini RGB (che vengono scritte con 4 byte per pixel, di cui vengono usati solo i tre byte meno significativi). Questi sono i formati restituiti da gl.lrectread().

20.12 `jpeg` — Lettura e scrittura di file JPEG

Il modulo `jpeg` fornisce accesso al compressore e al decompressore JPEG scritti dall' Independent JPEG Group (IJG). JPEG è uno standard per comprimere immagini; viene definito dal documento ISO 10918. Per dettagli sul formato JPEG o sull'Independent JPEG Group fare riferimento allo standard JPEG o alla documentazione fornita con il software.

Un'interfaccia portabile per gestire i file immagine in formato JPEG è disponibile con la Python Imaging Library (PIL), di Fredrik Lundh. Al sito http://www.pythonware.com/products/pil/. Sono disponibili ulteriori informazioni sulla PIL.

Il modulo `jpeg` definisce un'eccezione ed alcune funzioni.

exception `error`
Eccezione sollevata da `compress()` e `decompress()` in caso di errori.

`compress`(*data, w, h, b*)
Tratta l'argomento data come una matrice di pixel di larghezza *w* e di altezza *h*, con *b* byte per pixel. I pixel vengono elencati nell'ordine compatibile con la libreria GL di SGI, per cui il primo pixel è quello in basso a destra. Questo significa che il valore restituito da `gl.lrectread()` può essere passato immediatamente in input a `compress()`. Al momento, solo pixel di 1 e di 4 bytes sono consentiti, nel primo caso vengono trattati come valori di una scala di grigi, nel secondo caso come valori di colore RGB. La funzione `compress()` restituisce una stringa che contiene l'immagine compressa, in formato JFIF.

`decompress`(*data*)
L'argomento data è una stringa che contiene un'immagine in formato JFIF. La funzione `decompress()` restituisce una tupla (*dati*, *larghezza*, *altezza*, *byte_per_pixel*). Anche in questo caso, il valore restituito in dati è adatto ad essere passato immediatamente alla funzione `gl.lrectwrite()`.

`setoption`(*name, value*)
Imposta varie opzioni. Chiamate successive a `compress()` e `decompress()` faranno uso dei nuovi valori di queste opzioni. Sono disponibili le seguenti opzioni:

Opzione	Effetto
`'forcegray'`	Forza l'immagine in uscita ad essere in scala di grigi, anche se l'immagine in ingresso è di tipo RGB.
`'quality'`	Imposta la qualità dell'immagine compressa ad un valore tra `0` e `100` (il valore predefinito è `75`). Questa opzione ha effetto solo sulla compressione.
`'optimize'`	Effettua le ottimizzazioni della tavola di Huffman. Questo richiede più tempo, ma il risultato è un'immagine maggiormente compressa. Questa opzione ha effetto solo sulla compressione.
`'smooth'`	Effettua lo smorzamento delle differenze tra blocchi di una immagine non compressa. È utile solo con immagini di bassa qualità. Ha effetto solo sulla decompressione.

Vedete anche:

JPEG Still Image Data Compression Standard
Il punto di riferimento canonico per il formato di immagine JPEG, di Pennebaker and Mitchell.

Information Technology - Digital Compression and Coding of Continuous-tone Still Images - Requirements and Guidelines (http://www.w3.org/Graphics/JPEG/itu-t81.pdf)
Lo standard ISO per il formato JPEG, pubblicato anche come ITU T.81. È anche disponibile in rete in formato PDF.

CAPITOLO

VENTUNO

Servizi specifici di SunOS

Il modulo descritto in questo capitolo fornisce interfacce a funzionalità uniche nei sistemi operativi SunOS 5 (anche nota come Solaris versione 2).

21.1 `sunaudiodev` — Accesso all'hardware audio Sun

Questo modulo permette di accedere all'interfaccia audio Sun. L'hardware audio Sun è capace di registrare e riprodurre dati audio in formato u-LAW con una frequenza di campionamento di 8K al secondo. Una descrizione completa può essere trovata nella pagina di manuale *audio*(7I).

Il modulo `SUNAUDIODEV` definisce costanti che possono essere utilizzate con questo modulo.

Questo modulo definisce le seguenti variabili e funzioni:

exception `error`
: Questa eccezione viene sollevata per tutti gli errori. L'argomento è una stringa che descrive cosa non ha funzionato.

`open`(*mode*)
: Questa funzione apre il device audio e restituisce un oggetto Sun audio device. Questo oggetto può essere utilizzato per farci I/O. Il parametro mode può assumere i valori di `'r'` per accessi in sola registrazione, `'w'` per accessi in sola riproduzione, `'rw'` per entrambi e `'control'` per accesso al device di controllo. Poiché solo un processo può avere il registratore o il riproduttore aperto in un dato momento, è una buona idea aprire il device solo per le attività necessarie. Vedere *audio*(7I) per i dettagli.

  Come dalla pagina di manuale, questo modulo prima controlla la variabile d'ambiente `AUDIODEV` per il nome del dispositivo audio base. Se non viene trovata, prova '/dev/audio'. Il dispositivo di controllo viene calcolato aggiungendo "ctl" al dispositivo audio base.

21.1.1 Oggetti dispositivo audio

Gli oggetti dispositivo audio restituiti da `open()` definiscono i seguenti metodi (ad eccezione degli oggetti `control` che forniscono solo `getinfo()`, `setinfo()`, `fileno()` e `drain()`):

`close`()
: Questo metodo chiude esplicitamente il dispositivo. È utile in situazioni dove la cancellazione dell'oggetto non chiude immediatamente l'oggetto, poiché ci sono altri riferimenti ad esso. Un dispositivo chiuso non deve essere riutilizzato.

`fileno`()
: Restituisce il descrittore di file associato al dispositivo. Questo può essere utilizzato per impostare la notifica `SIGPOLL`, come descritto sotto.

`drain`()
: Questo metodo attende finché tutte le unità di output in attesa non vengono elaborate e restituite. Chiamare questo metodo è spesso non necessario: distruggendo gli oggetti verranno automaticamente chiusi tutti i dispositivi audio e questo implica un prosciugamento (NdT: drain) implicito.

flush()
: Questo metodo scarta tutto l'output in attesa. Può essere utilizzato per impedire una risposta lenta ad una richiesta di interruzione dell'utente (dovuto alla bufferizzazione fino ad un secondo del suono).

getinfo()
: Questo metodo recupera informazioni sullo stato come il volume di input e output, etc. e li restituisce nella forma dello stato degli oggetti audio. Questo oggetto non ha metodi, ma contiene vari attributi che descrivono lo stato corrente del dispositivo. Il nome ed il significato degli attributi sono descritti in `<sun/audioio.h>` e nella pagina di manuale *audio*(7I)<sun/audio.h>. I nomi dei membri hanno una sottile differenza dalle loro controparti in C: lo stato dell'oggetto è una singola struttura. I membri della sottostruttura `play` hanno un '`o_`' anteposto al loro nome ed i membri della struttura `record` hanno un '`i_`'. Perciò, il membro C `play.sample_rate` è accessibile come `o_sample_rate`, `record.gain` come `i_gain` e `monitor_gain` linearmente come `monitor_gain`.

ibufcount()
: Questo metodo restituisce il numero totale di campionamenti che vengono bufferizzati lato registrazione, è utile per determinare quanti dati potranno essere letti da una `read()`, senza che il programma si blocchi.

obufcount()
: Questo metodo restituisce il numero di campionamenti bufferizzati lato riproduzione. Sfortunatamente, questo numero non può essere utilizzato per determinare il numero di campionamenti che possono essere scritti senza bloccaggio, poiché la lunghezza della coda di output del kernel sembra essere variabile.

read(*size*)
: Questo metodo legge la dimensione (NdT: *size*) dei campioni dall'input audio e li restituisce come una stringa Python. La funzione è bloccante finché non sono disponibili abbastanza dati.

setinfo(*status*)
: Questo metodo imposta i parametri dello stato del dispositivo audio. Il parametro *status* è un oggetto dispositivo di stato come restituito da `getinfo()` e possibilmente modificato dal programma.

write(*samples*)
: A questo metodo viene passata una stringa Python contenente i campioni audio da riprodurre. Se non c'è abbastanza spazio libero nel buffer, ritorna immediatamente, altrimenti si blocca.

Il device audio supporta la notifica asincrona di vari eventi, attraverso il segnale SIGPOLL. Ecco un esempio di come si può abilitare questo in Python:

```
def handle_sigpoll(signum, frame):
    print 'I got a SIGPOLL update'

import fcntl, signal, STROPTS

signal.signal(signal.SIGPOLL, handle_sigpoll)
fcntl.ioctl(audio_obj.fileno(), STROPTS.I_SETSIG, STROPTS.S_MSG)
```

21.2 SUNAUDIODEV — Costanti usate con `sunaudiodev`

Questo è un modulo di compendio a `sunaudiodev`, che definisce costanti simboliche come `MIN_GAIN`, `MAX_GAIN`, `SPEAKER`, etc.. Il nome delle costanti è lo stesso utilizzato nel file C include `<sun/audioio.h>`, senza la stringa iniziale '`AUDIO_`'.

CAPITOLO

VENTIDUE

Servizi specifici MS Windows

Questo capitolo descrive moduli disponibili solamente su piattaforme MS Windows.

msvcrt	Un insieme di routine utili dalle MS VC++ runtime.
_winreg	Routine ed oggetti per manipolare il registro di Windows.
winsound	Accesso al sistema per la riproduzione del suono in Windows.

22.1 msvcrt – Routine utili dalle MS VC++ runtime

Queste funzioni forniscono accesso ad alcune funzionalità utili su piattaforme Windows. Alcuni moduli di alto livello utilizzano queste funzioni per costruire implementazioni Windows dei loro servizi. Per esempio, il modulo `getpass` utilizza questo nell'implementazione della funzione `getpass()`.

Ulteriore documentazione su queste funzioni può essere trovata nella documentazione della API della piattaforme.

22.1.1 Operazioni su file

locking(*fd, mode, nbytes*)
: Blocca parti di un file basandosi su un descrittore di file *fd* dalla runtime C. Solleva l'eccezione `IOError` in caso di fallimento. La regione bloccata del file si estende dalla posizione corrente per *nbytes* byte e può continuare oltre la fine del file. *mode* deve essere una delle costanti `LK_*` elencate di seguito. Possono essere bloccate molte regioni di uno stesso file allo stesso tempo, ma non possono sovrapporsi. Regioni adiacenti non vengono amalgamate; non possono essere sbloccate separatamente.

LK_LOCK
LK_RLCK
: Blocca gli specifici byte. Se i byte non possono essere bloccati, il programma ritenta dopo 1 secondo. Se, dopo 10 tentativi, i byte non possono essere bloccati, viene sollevata un'eccezione `IOError`.

LK_NBLCK
LK_NBRLCK
: Blocca i byte specifici. Se i byte non possono essere bloccati, solleva l'eccezione `IOError`.

LK_UNLCK
: Sblocca gli specifici byte, che devono essere stati precedentemente bloccati.

setmode(*fd, flags*)
: Imposta il modo di translazione per il terminatore di riga del descrittore di file *fd*. Per impostare il modo testo, *flags* deve essere impostato a `os.O_TEXT`; per il modo binario, deve essere `os.O_BINARY`.

open_osfhandle(*handle, flags*)
: Crea un descrittore di file del runtime C dal file gestore *handle*. Il parametro *flags* deve essere un OR bit-per-bit di `os.O_APPEND`, `os.O_RDONLY` e `os.O_TEXT`. Il descrittore di file risultante può essere utilizzato come parametro per `os.fdopen()` per creare un oggetto di tipo file.

get_osfhandle(*fd*)
: Restituisce un file gestore per il descrittore di file *fd*. Solleva l'eccezione `IOError` se *fd* non viene riconosciuto.

22.1.2 Console I/O

kbhit()
Restituisce vero se c'è un tasto da premere in attesa di lettura.

getch()
Legge un singolo tasto da premere; restituisce il carattere risultante. Non viene mostrato niente in console. Questa chiamata è bloccante se non c'è nessun tasto da premere disponibile, ma non aspetta che venga premuto il tasto `Enter`. Se il tasto premuto è un tasto funzione speciale, questo restituirà `'\000'` o `'\xe0'`; la prossima chiamata restituirà il codice del carattere. La pressione di `Control-C` non può essere letta con questa funzione.

getche()
Simile a `getch()`, ma il tasto da premere viene mostrato sul terminale, se rappresenta un carattere stampabile.

putch(*char*)
Stampa il carattere *char* sulla console senza bufferizzazione.

ungetch(*char*)
Fa sì che il carattere *char* sia reimmesso (NdT: "pushed back") nel buffer della console; sarà il prossimo carattere letto da `getch()` o `getche()`.

22.1.3 Altre funzioni

heapmin()
Forza l'heap della `malloc()` a far pulizia e restituisce i blocchi non utilizzati al sistema operativo. Questo funziona solo su Windows NT. In caso di fallimento, solleva l'eccezione `IOError`.

22.2 `_winreg` – Accesso al registro di Windows

Nuovo nella versione 2.0.

Queste funzioni rendono disponibili le API del registro di Windows a Python. Anziché utilizzare un intero come gestore del registro, viene utilizzato un oggetto gestore per assicurare che i gestori siano chiusi correttamente, anche se il programmatore si dimentica di chiuderli esplicitamente.

Questo modulo rende disponibile un'interfaccia di basso livello al registro di Windows; nel futuro è atteso un nuovo modulo `winreg` che fornisca un'interfaccia di alto livello alle API del registro.

Questo modulo offre le seguenti funzioni:

CloseKey(*hkey*)
Chiude una chiave di registro precedentemente aperta. L'argomento hkey specifica la chiave precedentemente aperta.

Notare che *hkey* non permette di chiudere la chiave di registro se *hkey* è ancora attiva (`handle.Close()` chiude la chiave di registro quando l'oggetto *hkey* viene distrutto da Python).

ConnectRegistry(*computer_name, key*)
Stabilisce una connessione ad un gestore di registro predefinito su un'altro computer e restituisce l'*oggetto gestore*.

computer_name è il nome del computer remoto, nella forma `r\\computername`. Se `None`, viene utilizzato il computer locale.

key è il gestore predefinito a cui connettersi.

Il valore restituito è il gestore della chiave aperta. Se la funzione fallisce, viene sollevata l'eccezione `EnvironmentError`.

CreateKey(*key, sub_key*)
Crea o apre la specifica chiave, restituendo l'*oggetto gestore*.

key è una chiave già aperta oppure una delle costanti predefinite `HKEY_*`.

sub_key è una stringa che indica la chiave che viene aperta o creata da questo metodo.

Se *key* è una delle chiavi predefinite, *sub_key* può essere `None`. In questo caso, il gestore restituito è lo stesso gestore della chiave passato alla funzione.

Se key esiste, questa funzione apre la chiave esistente.

Il valore restituito è il gestore della chiave aperta. Se la funzione fallisce, viene sollevata l'eccezione `EnvironmentError`.

`DeleteKey`(*key, sub_key*)
Cancella la chiave specifica.

key è una chiave già aperta oppure una delle costanti predefinite `HKEY_*`.

sub_key è la stringa che deve essere una sottochiave della chiave identificata dal parametro *key*. Questo valore non deve essere `None` e la chiave può non avere sottochiavi.

Questo metodo non può cancellare chiavi con sottochiavi.

Se il metodo ha successo, l'intera chiave, inclusi tutti i suoi valori, vengono cancellati. Se il metodo fallisce, viene sollevata un'eccezione di tipo `EnvironmentError`.

`DeleteValue`(*key, value*)
Cancella un valore da una chiave di registro.

key è una chiave già aperta oppure una delle costanti predefinite `HKEY_*`.

value è una stringa che identifica il valore da rimuovere.

`EnumKey`(*key, index*)
Numera le sottochiavi di una chiave di registro aperta, restituendo una stringa.

key è una chiave già aperta oppure una delle costanti predefinite `HKEY_*`.

index è un intero the identifica l'indice della chiave da recuperare.

La funzione restituisce il nome di una delle sottochiavi ogni volta che viene chiamata. Viene tipicamente chiamata ripetutamente finché non viene sollevata un'eccezione `EnvironmentError`, ad indicare che non ci sono più valori disponibili.

`EnumValue`(*key, index*)
Numera i valori di una chiave di registro aperta, restituendo una tupla.

key è una chiave già aperta oppure una delle costanti predefinite `HKEY_*`.

index è un intero the identifica l'indice del valore da restituire.

La funzione recupera il nome di una sottochiave ogni volta che viene chiamata. Viene tipicamente chiamata ripetutamente finché non viene sollevata un'eccezione di tipo `EnvironmentError`, ad indicare che non ci sono più valori disponibili.

Il risultato è una tupla di 3 elementi:

Indice	Significato
`0`	Una stringa che identifica il nome del valore
`1`	Un oggetto che contiene i dati del valore il cui tipo dipende dal tipo di registro sottostante
`2`	Un intero che identifica il tipo dei dati del valore

`FlushKey`(*key*)
Scrive tutti gli attributi di una chiave nel registro.

key è una chiave già aperta oppure una delle costanti predefinite `HKEY_*`.

Non è necessario chiamare RegFlushKey per cambiare una chiave. I cambiamenti nel registro vengono salvati su disco dal registro utilizzando il suo pigro sistema di flush. Al contrario di `CloseKey()`, il metodo `FlushKey()` restituisce tutti i suoi dati quando questi sono stati scritti nel registro. Un'applicazione deve chiamare `FlushKey()` solo se deve essere assolutamente certa che i cambiamenti siano stati salvati su disco.

Se non si è sicuri che la chiamata a `FlushKey()` sia necessaria, probabilmente non lo è.

RegLoadKey(*key, sub_key, file_name*)

Crea una sottochiave sotto la specifica chiave ed immagazzina le informazioni di registrazione da uno specifico file nella sottochiave.

key è una chiave già aperta oppure una delle costanti predefinite `HKEY_*`.

sub_key è una stringa che identifica la sottochiave da caricare.

file_name è il nome del file da cui caricare i dati del registro. Questo file deve essere stato creato con la funzione `SaveKey()`. Su un file system FAT (file allocation table), il nome del file può non avere un'estensione.

Una chiamata a `LoadKey()` fallisce se il processo chiamante non ha il privilegio `SE_RESTORE_PRIVILEGE`. Notare che i privilegi sono diversi dai permessi – vedere la documentazione di Win32 per maggiori dettagli al riguardo.

Se la chiave *key* è un gestore restituito da `ConnectRegistry()`, il percorso specificato in *fileName* è relativo al computer remoto.

La documentazione di Win32 dà per scontato che *key* debba essere nell'albero di `HKEY_USER` o `HKEY_LOCAL_MACHINE`. Questo può essere o non essere vero.

OpenKey(*key, sub_key*[, *res = 0*][, *sam = KEY_READ*])

Apre una chiave specifica, restituendo un *oggetto gestore*.

key è una chiave già aperta, oppure una delle costanti predefinite `HKEY_*`.

sub_key è una stringa che identifica la sottochiave da aprire.

res è un intero riservato e deve essere zero. Il valore predefinito è zero.

sam è un intero che specifica una maschera di accesso che descrive l'accesso di sicurezza desiderato per la chiave. Il valore predefinito è `KEY_READ`.

Il risultato è un nuovo gestore per la specifica chiave.

Se la funzione fallisce, viene sollevata un'eccezione `EnvironmentError`.

OpenKeyEx()

Le funzionalità di `OpenKeyEx()` vengono fornite da `OpenKey()`, utilizzando gli argomenti predefiniti.

QueryInfoKey(*key*)

Restituisce informazioni su una chiave, sotto forma di tuple.

key è una chiave già aperta oppure una delle costanti predefinite `HKEY_*`.

Il risultato è una tupla di 3 elementi:

Indice	Significato
`0`	Un intero che fornisce il numero di sottochiavi della chiave corrente.
`1`	Un intero che fornisce il numero di valori di questa chiave.
`2`	Un intero long che indica la data di modifica della chiave, espressa in centinaia di nanosecondi dal primo gennaio 1970.

QueryValue(*key, sub_key*)

Restituisce informazioni su una chiave, sotto forma di stringa.

key è una chiave già aperta oppure una delle costanti predefinite `HKEY_*`.

sub_key è una stringa che contiene il nome della sottochiave con il valore associato. Se questo parametro ha come valore `None` o è vuoto, la funzione recupera il valore impostato dal metodo `SetValue()` per la chiave identificata da *key*.

I valori nel registro hanno nome, tipo e dati componenti. Questo metodo recupera i dati per il primo valore della chiave che ha come nome NULL. Ma la chiamata alle API sottostanti non restituiscono il tipo, NON USARE!!!

QueryValueEx(*key, value_name*)

Restituisce il tipo e i dati di uno nome di un valore specificato, associato ad una chiave di registro aperta.

key è una chiave già aperta oppure una delle costanti predefinite `HKEY_*`.

value_name è una stringa che indica il valore da interrogare.

Il risultato è una tupla di 2 elementi:

Indice	Significato
`0`	Il valore dell'elemento del registro.
`1`	Un intero che specifica il tipo del registro per questo valore.

`SaveKey`(*key, file_name*)
Salva la chiave specificata e tutte le sue sottochiavi nel file specificato.

key è una chiave già aperta oppure una delle costanti predefinite `HKEY_*`.

file_name è il nome del file in cui salvare i dati del registro. Questo file non può esistere. Se il nome del file include un'estensione, non può essere utilizzato in un file system FAT dai metodi `LoadKey()`, `ReplaceKey()` o `RestoreKey()`.

Se *key* rappresenta una chiave su un computer remoto, il percorso descritto da *file_name* è relativo al computer remoto. Il chiamante di questo metodo deve avere il privilegio di sicurezza `SeBackupPrivilege`. Notare che i privilegi sono diversi dai permessi – vedere la documentazione di Win32 per maggiori dettagli.

Questa funzione passa NULL come *security_attributes* alle API.

`SetValue`(*key, sub_key, type, value*)
Associa un valore con la chiave specificata.

key è una chiave aperta oppure una delle costanti predefinite `HKEY_*`.

sub_key è una stringa che identifica la sottochiave a cui è associato il valore.

type è un intero che specifica il tipo dei dati. Attualmente questo deve essere `REG_SZ`, a significare che solo le stringhe vengono supportate. Utilizzare la funzione `SetValueEx()` per supportare altri tipi di dati.

value è una stringa che specifica il nuovo valore.

Se la chiave specificata dal parametro *sub_key* non esiste, la funzione `SetValue()` la crea.

La lunghezza dei valori è limitata dalla memoria disponibile. I valori lunghi (più di 2048 byte) devono essere immagazzinati come file, con il nome del file immagazzinato nel registro di configurazione. Questo aiuta il registro ad essere efficiente.

La chiave identificata dal parametro *key* deve essere aperta con accesso `KEY_SET_VALUE`.

`SetValueEx`(*key, value_name, reserved, type, value*)
Immagazzina dati nel campo valore di una chiave aperta.

key è una chiave già aperta oppure una delle costanti predefinite `HKEY_*`.

sub_key è una stringa che indica la sottochiave a cui il valore è associato.

type è un intero che specifica il tipo dei dati. Questo deve essere una delle seguenti costanti definite in questo modulo:

Costanti	Significato
`REG_BINARY`	Dati binari in qualunque forma.
`REG_DWORD`	Un numero a 32-bit.
`REG_DWORD_LITTLE_ENDIAN`	Un numero a 32 bit in formato little-endian.
`REG_DWORD_BIG_ENDIAN`	Un numero a 32 bit in formato big-endian.
`REG_EXPAND_SZ`	Una stringa null-terminata contenente un riferimento alle variabili d'ambiente ('`%PATH%`').
`REG_LINK`	Un link simbolico Unicode.
`REG_MULTI_SZ`	Una sequenza di stringhe null-terminate, terminate da due caratteri null. Python gestisce queste terminazioni automaticamente.
`REG_NONE`	Nessun tipo di valore definito.
`REG_RESOURCE_LIST`	Una lista di risorse di dispositivi-driver.
`REG_SZ`	Una stringa null-terminata.

reserved può essere qualsiasi cosa; il valore zero viene sempre passato alle API.

value è una stringa che specifica il nuovo valore.

Questo metodo può anche impostare ulteriori valori e tipi di informazioni per la chiave specifica. La chiave identificata dal parametro key deve essere stata aperta con l'accesso `KEY_SET_VALUE`.

Per aprire la chiave, utilizzare i metodi `CreateKeyEx()` o `OpenKey()`.

Le lunghezze dei valori vengono limitate dalla memoria disponibile. I valori lunghi (più di 2048 bytes) devono essere immagazzinati come file, il cui nome verrà immagazzinato nel registro di configurazione. Questo aiuta il registro ad operare in modo efficiente.

22.2.1 Oggetti gestori del registro

Questo oggetto incapsula un oggetto Windows HKEY, chiudendolo in automatico quando questo viene distrutto. Per garantire la pulizia, è possibile chiamare sia il metodo `Close()` dell'oggetto che la funzione `CloseKey()`.

Tutte le funzioni relative al registro di questo modulo restituiscono uno di questi oggetti.

Tutte le funzioni relative al registro di questo modulo che accettano un oggetto gestore, accettano anche un intero; comunque, l'utilizzo dell'oggetto gestore è incoraggiato.

L'oggetto gestore fornisce una semantica per `__nonzero__()`, così:

```
if handle:
    print "Sì"
```

restituirà `Sì` se il gestore è valido (non è stato chiuso o staccato).

Questo oggetto supporta anche la semantica del confronto, così gli oggetti gestori verranno valutati come uguali se si riferiscono al medesimo valore sottostante del gestore Windows.

Gli oggetti gestori possono essere convertiti in un intero (per esempio, utilizzando la funzione builtin `int()`, nel qual caso verrà restituito il valore sottostante del gestore di Windows. É possibile usare anche il metodo `Detach()` che restituirà l'intero gestore, e disconnetterà anche il gestore di Windows dall'oggetto gestore.

`Close``()`
Chiude il gestore di Windows sottostante.

Se il gestore è già chiuso, non viene sollevato alcun errore.

`Detach``()`
Stacca il gestore di Windows dall'oggetto gestore.

Il risultato è un intero (o un long su Windows a 64 bit) the contiene il valore del gestore prima che sia staccato. Se il gestore è già staccato o chiuso, il valore restituito sarà zero.

Dopo la chiamata a questa funzione, il gestore verrà effettivamente invalidato, ma il gestore non verrà chiuso. Si vorrà utilizzare questa funzione quando si ha la necessità che il gestore Win32 sottostante esista oltre la vita dell'oggetto gestore.

22.3 `winsound` — Interfaccia per la riproduzione del suono in Windows

Nuovo nella versione 1.5.2.

Il modulo `winsound` fornisce l'accesso al meccanismo base di riproduzione dei suoni fornito dalle piattaforme Windows. Include funzioni e varie costanti.

`Beep``(`*frequency, duration*`)`
Fa suonare (beep) l'altoparlante del PC. Il parametro *frequency* specifica la frequenza, in hertz del suono e deve essere nell'intervallo da 37 a 32.767. Il parametro *duration* specifica la durata in millisecondi. Se il sistema non è in grado di utilizzare l'altoparlante, viene sollevata l'eccezione `RuntimeError`. **Note:** Sotto Windows 95 e 98, la funzione Windows `Beep()` esiste ma non viene utilizzata (ignora gli argomenti). In questo caso Python la simula manipolando direttamente le porte (aggiunta nella versione 2.1). Non si sa se è in grado di operare su tutti i sistemi. Nuovo nella versione 1.6.

PlaySound(*sound, flags*)

Chiama la funzione sottostante `PlaySound()` dalle API della piattaforma. Il parametro *sound* può essere un nome di file, dati audio sotto forma di stringa o `None`. La sua interpretazione dipende dal valore di *flags*, che può essere una combinazione bit per bit delle costanti definite sotto. Se il sistema indica un errore, viene sollevata un'eccezione `RuntimeError`.

MessageBeep([*type=MB_OK*])

Chiama la funzione sottostante `MessageBeep()` dalle API della piattaforma. Questo riproduce un suono come specificato nel registro. L'argomento *type* specifica quale suono riprodurre; i valori possibili sono `-1`, `MB_ICONASTERISK`, `MB_ICONEXCLAMATION`, `MB_ICONHAND`, `MB_ICONQUESTION` e `MB_OK`, tutte descritte sotto. Il valore `-1` produce un "semplice beep"; questa è l'ultima azione se un suono non può essere riprodotto altrimenti. Nuovo nella versione 2.3.

SND_FILENAME

Il parametro *sound* è il nome del file WAV. Da non utilizzare con `SND_ALIAS`.

SND_ALIAS

Il parametro *sound* è il "suono di un'azione associata ad un nome" dal registro. Se il registro non contiene questo nome, riproduce il suono predefinito di sistema, a meno che non sia specificato `SND_NODEFAULT`. Se non c'è un suono di sistema registrato, solleva l'eccezione `RuntimeError`. Da non utilizzare con `SND_FILENAME`.

Tutti i sistemi Win32 supportano almeno i seguenti suoni; la maggior parte dei sistemi ne supporta di più:

nome **PlaySound()**	**Nome corrispondente nel Pannello di controllo dei suoni**
'SystemAsterisk'	Asterisk
'SystemExclamation'	Esclamazione
'SystemExit'	Uscita da Windows
'SystemHand'	Interruzione critica
'SystemQuestion'	Domanda

Per esempio:

```
import winsound
# Riproduce un suono di Windows di uscita.
winsound.PlaySound("SystemExit", winsound.SND_ALIAS)

#  Probabilmente riproduce il suono predefinito di Windows, se è
#+ registrato (poiché "*" probabilmente non è un nome di un suono
#+ registrato).
winsound.PlaySound("*", winsound.SND_ALIAS)
```

SND_LOOP

Riproduce il suono ripetutamente. L'opzione `SND_ASYNC` deve anche essere utilizzata per evitare il bloccaggio. Non può essere utilizzata con `SND_MEMORY`.

SND_MEMORY

Il parametro *sound* a `PlaySound()` è un'immagine in memoria di un file WAV, sotto forma di stringa.

Note: Questo modulo non supporta la riproduzione di file in memoria in modo asincrono, perciò la combinazione di questa opzione con `SND_ASYNC` solleverà un'eccezione `RuntimeError`.

SND_PURGE

Interrompe la riproduzione di tutte le istanze del suono specificato.

SND_ASYNC

Ritorna immediatamente, permettendo una riproduzione asincrona del suono.

SND_NODEFAULT

Se il suono specificato non può essere trovato, non riproduce il suono predefinito di sistema.

SND_NOSTOP

Non interrompe il suono attualmente in riproduzione.

SND_NOWAIT

Ritorna immediatamente se il driver del suono è occupato.

MB_ICONASTERISK
Riproduce il suono `SystemDefault`.

MB_ICONEXCLAMATION
Riproduce il suono `SystemExclamation`.

MB_ICONHAND
Riproduce il suono `SystemHand`.

MB_ICONQUESTION
Riproduce il suono `SystemQuestion`.

MB_OK
Riproduce il suono `SystemDefault`.

APPENDICE

A

Moduli non documentati

Qui un breve elenco dei moduli che sono attualmente non documentati, ma che devono essere documentati. Sentitevi liberi di contribuire alla documentazione di questi moduli! (mandate un email a docs@python.org).

L'idea ed i contenuti originali di questo capitolo derivano da un post di Fredrik Lundh; i contenuti specifici di questi capitoli sono stati sostanzialmente rivisti.

A.1 Frameworks

I framework sono difficili da documentare, ma ripagano dello sforzo fatto.

`test` — Framework per i test di regressione. Questo viene utilizzato per i test di regressione di Python, ma è utile per altre librerie di Python. Questo è un package piuttosto che un singolo modulo.

A.2 Varie routine utili

Alcune di queste sono molto vecchie e/o non molto robuste; segnate con "hmm".

`bdb` — Una classe generica base di debugger per Python (utilizzata da pdb).

`ihooks` — Supporto per gli import hook (per `rexec`; può essere obsoleto).

`platform` — Questo modulo tenta di recuperare il maggior numero possibile di dati identificativi della piattaforma in uso. Rende le informazioni disponibili attraverso un'API. Se richiamata da riga di comando, stampa le informazioni della piattaforma concatenate in una singola stringa su `sys.stdout`. Il formato di output è utilizzabile come parte di un filename. Nuovo nella versione 2.3.

`smtpd` — Un'implementazione di un demone SMTP che rispecchia i requisiti minimi di conformità all'RFC 821.

A.3 Moduli specifici della piattaforma

Questi moduli vengono utilizzati per implementare il modulo `os.path` e non sono documentati per questa ragione. C'è solo una piccola necessità di documentarli.

`ntpath` — Implementazione di `os.path` su piattaforme Win32, Win64 e OS/2.

`posixpath` — Implementazione di `os.path` su POSIX.

`bsddb185` — Moduli per la retrocompatibilità in sistemi che ancora utilizzano il modulo Berkley DB 1.85. Solitamente è disponibile solo su certi sistemi BSD. Non deve mai essere utilizzato direttamente.

A.4 Multimedia

audiodev — API indipendenti dalla piattaforma per la riproduzione di file audio.

linuxaudiodev — Riproduce dati audio su dispositivi audio Linux. Sostituito in Python 2.3 dal modulo `ossaudiodev`.

sunaudio — Interpreta le intestazioni audio Sun (può diventare obsoleto o un tool/demo).

toaiff — Converte file audio arbitrari in file AIFF; deve probabilmente diventare un tool o una demo. Richiede il programma esterno **sox**.

ossaudiodev — Riproduce dati audio tramite le API Open Sound System. Questo è utilizzabile da Linux, alcuni BSD ed alcune piattaforme UNIX commerciali.

A.5 Obsoleti

Questi moduli non sono normalmente disponibili per l'import; viene richiesto del lavoro aggiuntivo per renderli disponibili.

Quelli scritti in Python verranno installati nella directory 'lib-old/', installati come parte della libreria standard. Per utilizzarli, la directory deve essere aggiunta a `sys.path`, possibilmente utilizzando PYTHONPATH.

I moduli di estensione scritti in C non vengono costruiti in fase di compilazione. Sotto UNIX, questi moduli devono essere abilitati decommentando le righe appropriate in 'Modules/Setup' nell'albero build e occorre ricompilare Python se i moduli sono stati compilati staticamente o ricostruire e reinstallare gli oggetti condivisi se si utilizza il caricamento dinamico delle estensioni.

addpack — Approccio alternativo ai package. Utilizzare invece il supporto nativo per i package.

cmp — Funzioni per il confronto tra file. Utilizzare invece il nuovo modulo `filecmp`.

cmpcache — Versione con cache del modulo obsoleto `cmp`. Utilizzare invece il nuovo modulo `filecmp`.

codehack — Estrae il nome della funzione o il numero di riga da un oggetto codice della funzione (queste sono adesso accessibili come attributi: `co.co_name`, `func.func_name`, `co.co_firstlineno`).

dircmp — Classe per costruire tool per effettuare il diff tra directory (potrà diventare un demo o un tool). **Deprecato dalla versione 2.0.** Il modulo `filecmp` sostituisce `dircmp`.

dump — Stampa il codice Python che ricostruisce una variabile.

fmt — Astrazione per la formattazione del testo (troppo lento).

lockfile — Wrapper attorno al sistema di locking per i file FCNTL (utilizzare invece `fcntl.lockf()`/`flock()`; vedere `fcntl`).

newdir — La nuova funzione `dir()` (il modulo standard `dir()` adesso funziona (in modo accettabile).

Para — Aiuto per `fmt`.

poly — Polinomi.

regex — Supporto per le espressioni regolari in stile Emacs; può ancora essere usato in alcuni vecchi codici (moduli di estensione). Fare riferimento alla *documentazione di Python 1.6* per informazioni.

regsub — Utilità di sostituzione su stringhe basata sulle espressioni regolari, da utilizzare con `regex` (modulo di estensione). Fare riferimento alla *documentazione di Python 1.6* per informazioni.

tb — Stampa i traceback, con un dump delle variabili locali (utilizzare invece `pdb.pm()` o `traceback`).

timing — Misura gli intervalli di tempo in un'alta risoluzione (utilizzare invece `time.clock()`). Questo è un modulo di estensione.

tzparse — Analizza le specifiche di una timezone (non terminato; potrà sparire in futuro e non funziona quando la variabile d'ambiente TZ non è impostata).

util — Funzioni utili che non stavano in altri posti.

whatsound — Riconosce i file audio; utilizzare invece `sndhdr`.

zmod — Computa le proprietà dei "campi".

I seguenti moduli sono obsoleti, ma sono probabilmente da rifare come tool o script:

find — Trova i file che corrispondono ad una determinata corrispondenza in un'albero di directory.

grep — Implementazione di **grep** in Python.

packmail — Crea un'archivio shell UNIX autospacchettante.

I seguenti moduli erano documentati nelle precedenti versioni di questo manuale, ma adesso tali documenti vengono considerati obsoleti. I sorgenti della documentazione sono ancora disponibili come parte degli archivi sorgenti della documentazione.

ni — Importa i moduli in "packages." Il supporto basilare per i package adesso è built-in. Il supporto built-in è molto simile a quello fornito da questo modulo.

rand — Vecchia interfacci per la generazione di numeri casuali.

soundex — Algoritmo per far collassare i nomi che suonano simili in una chiave condivisa. L'algoritmo specifico non sembra corrispondere a nessun algoritmo pubblicato. Questo è un modulo di estensione.

A.6 Moduli di estensione specifici per SGI

I seguenti sono specifici di SGI e possono essere non aggiornati rispetto alla reale versione corrente.

cl — Interfaccia alla libreria di compressione di SGI.

sv — Interfaccia alla lavagna "simple video" su SGI Indigo (hardware obsoleto).

APPENDICE

B

Segnalare degli errori di programmazione

Python è un linguaggio di programmazione maturo che si è guadagnato anche la reputazione di essere stabile. Per mantenere questa reputazione, gli sviluppatori vorrebbero ricevere notizie su errori e deficienze che potreste riscontrare sia in Python che nella sua documentazione.

Prima di inviare un rapporto vi sarà richiesto di loggarvi su SourceForge; questo perché così facendo permettere-rete agli sviluppatori di potervi contattare per ulteriori informazioni, se dovesse tornare utile. Non è prevista la possibilità di fare un rapporto su un bug anonimo.

Tutti i rapporti sui bug dovrebbero essere trasmessi tramite il Python Bug Tracker su SourceForge (http://sourceforge.net/bugs/?group_id=5470). Il tracciatore dei bug fornisce una form web che permette l'immissione di informazioni pertinenti e la loro trasmissione agli sviluppatori.

Il primo passo nell'invio di un rapporto è determinare se quel problema sia stato o meno già segnalato. Il vantaggio di ciò, oltre al fatto di risparmiare tempo agli sviluppatori, è che potreste anche voi conoscere cos'è stato fatto per risolverlo; potrebbe succedere che il problema sia già stato risolto per la prossima release o che siano necessarie ulteriori informazioni (nel qual caso siete i benvenuti a fornirne, se potete!). Perciò, cercate nel database dei bug usando il box di ricerca vicino alla fine della pagina, sul lato sinistro.

Se il problema che vi accingete a segnalare non è già nel tracciatore dei bug, tornate indietro al Python Bug Tracker (http://sourceforge.net/bugs/?group_id=5470). Selezionate il link "Submit a Bug" in cima alla pagina, per aprire la form di segnalazione di bug.

La form di immissione ha un numero di campi. I soli campi richiesti i "Summary" e "Details". Per il Summary, fornite una *molto* breve descrizione del problema; meno di dieci parole va già bene. Nel campo Details, descrivete il problema in dettaglio, includendo cosa voi vi aspettavate avvenisse e cosa invece è avvenuto. Assicuratevi di includere la versione di Python che avete usato, cosiccome ogni modulo di estensione che è stato impiegato e quale tipo di hardware e piattaforma software avete utilizzato (fornendo anche informazioni sulle versioni).

L'unico altro campo che potreste voler definire è "Category", che permette di inserire il rapporto sul bug in una determinata categoria (tipo "Documentation" oppure "Library").

Ogni bug report verrà assegnato ad uno sviluppatore che valuterà cosa dovrà esser fatto per correggere il problema. Voi riceverete un update ogni volta che verrà effettuata una modifica sul bug.

Vedete anche:

How to Report Bugs Effectively
(http://www-mice.cs.ucl.ac.uk/multimedia/software/documentation/ReportingBugs.html)
: Articolo che mostra alcuni dettagli sul come creare un utile rapporto su di un bug. Descrive quale tipo di informazioni sia utile e perché.

Bug Writing Guidelines
(http://www.mozilla.org/quality/bug-writing-guidelines.html)
: Informazioni su come scrivere un buon rapporto su un bug. Alcune sono specifiche del progetto Mozilla ma descrivono molto bene i procedimenti generali.

APPENDICE

C

Storia e licenza

C.1 Storia del software

Python è stato creato agli inizi degli anni 90 da Guido van Rossum al Centro di Matematica di Stichting (CWI, si veda http://www.cwi.nl/) nei Paesi Bassi, come successore di un linguaggio chiamato ABC. Guido rimane l'autore principale di Python, anche se molti altri vi hanno contribuito.

Nel 1995, Guido ha continuato il suo lavoro su Python presso il Centro Nazionale di Ricerca (CNRI, si veda http://www.cnri.reston.va.us/) in Reston, Virginia, dove ha rilasciato parecchie versioni del software.

Nel maggio 2000, Guido e la squadra di sviluppo di Python si spostano in BeOpen.com per formare la squadra PythonLabs di BeOpen. Nell'ottobre dello stesso anno, la squadra di PythonLabs diventa la Digital Creations (ora Zope Corporation; si veda http://www.zope.com/). Nel 2001, viene fondata la Python Software Foundation (PSF, si veda http://www.python.org/psf/), un'organizzazione senza scopo di lucro creata specificamente per detenere la propria proprietà intellettuale di Python. Zope Corporation è un membro sponsorizzato dalla PSF.

Tutti i rilasci di Python sono Open Source (si veda http://www.opensource.org/ per la definizione di "Open Source"). Storicamente la maggior parte, ma non tutti i rilasci di Python, sono stati GPL-compatibili; la tabella seguente ricapitola i vari rilasci.

Versione	Derivata da	Anno	Proprietario	GPL compatibile?
0.9.0 thru 1.2	n/a	1991-1995	CWI	si
1.3 thru 1.5.2	1.2	1995-1999	CNRI	si
1.6	1.5.2	2000	CNRI	no
2.0	1.6	2000	BeOpen.com	no
1.6.1	1.6	2001	CNRI	no
2.1	2.0+1.6.1	2001	PSF	no
2.0.1	2.0+1.6.1	2001	PSF	si
2.1.1	2.1+2.0.1	2001	PSF	si
2.2	2.1.1	2001	PSF	si
2.1.2	2.1.1	2002	PSF	si
2.1.3	2.1.2	2002	PSF	si
2.2.1	2.2	2002	PSF	si
2.2.2	2.2.1	2002	PSF	si
2.2.3	2.2.2	2002-2003	PSF	si
2.3	2.2.2	2002-2003	PSF	si
2.3.1	2.3	2002-2003	PSF	si
2.3.2	2.3.1	2003	PSF	si
2.3.3	2.3.2	2003	PSF	si
2.3.4	2.3.3	2004	PSF	si

Note: GPL-compatibile non significa che viene distribuito Python secondo i termini della GPL. Tutte le licenze di Python, diversamente dalla GPL, permettono di distribuire una versione modificata senza rendere i vostri cambiamenti open source. Le licenze GPL-compatibili permettono di combinare Python con altro software rilasciato sotto licenza GPL; le altre no.

Un ringraziamento ai tanti volontari che, lavorando sotto la direzione di Guido, hanno reso possibile permettere questi rilasci.

C.2 Termini e condizioni per l'accesso o altri usi di Python (licenza d'uso, volutamente non tradotta)

PSF LICENSE AGREEMENT FOR PYTHON 2.3.4

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 2.3.4 software in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 2.3.4 alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright © 2001-2004 Python Software Foundation; All Rights Reserved" are retained in Python 2.3.4 alone or in any derivative version prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or incorporates Python 2.3.4 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 2.3.4.

4. PSF is making Python 2.3.4 available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 2.3.4 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 2.3.4 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 2.3.4, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By copying, installing or otherwise using Python 2.3.4, Licensee agrees to be bound by the terms and conditions of this License Agreement.

BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0
BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization ("Licensee") accessing and otherwise using this software in source or binary form and its associated documentation ("the Software").

2. Subject to the terms and conditions of this BeOpen Python License Agreement, BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use the Software alone or in any derivative version, provided, however, that the BeOpen Python License is retained in the Software, alone or in any derivative version prepared by Licensee.

3. BeOpen is making the Software available to Licensee on an "AS IS" basis. BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

5. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

6. This License Agreement shall be governed by and interpreted in all respects by the law of the State of California, excluding conflict of law provisions. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between BeOpen and Licensee. This License Agreement does not grant permission to use BeOpen trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party. As an exception, the "BeOpen Python" logos available at http://www.pythonlabs.com/logos.html may be used according to the permissions granted on that web page.

7. By copying, installing or otherwise using the software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1

1. This LICENSE AGREEMENT is between the Corporation for National Research Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 ("CNRI"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 1.6.1 software in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 1.6.1 alone or in any derivative version, provided, however, that CNRI's License Agreement and CNRI's notice of copyright, i.e., "Copyright © 1995-2001 Corporation for National Research Initiatives; All Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version prepared by Licensee. Alternately, in lieu of CNRI's License Agreement, Licensee may substitute the following text (omitting the quotes): "Python 1.6.1 is made available subject to the terms and conditions in CNRI's License Agreement. This Agreement together with Python 1.6.1 may be located on the Internet using the following unique, persistent identifier (known as a handle): 1895.22/1013. This Agreement may also be obtained from a proxy server on the Internet using the following URL: http://hdl.handle.net/1895.22/1013."

3. In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 1.6.1.

4. CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis. CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. This License Agreement shall be governed by the federal intellectual property law of the United States, including without limitation the federal copyright law, and, to the extent such U.S. federal law does not apply, by the law of the Commonwealth of Virginia, excluding Virginia's conflict of law provisions. Notwithstanding the foregoing, with regard to derivative works based on Python 1.6.1 that incorporate non-separable material that was previously distributed under the GNU General Public License (GPL), the law of the Commonwealth of Virginia shall govern this License Agreement only as to issues arising under or with respect to Paragraphs 4, 5, and 7 of this License Agreement. Nothing in this License Agreement shall be deemed to

create any relationship of agency, partnership, or joint venture between CNRI and Licensee. This License Agreement does not grant permission to use CNRI trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By clicking on the "ACCEPT" button where indicated, or by copying, installing or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

ACCEPT

CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

C.3 Licenze e riconoscimenti per i programmi incorporati

This section is an incomplete, but growing list of licenses and acknowledgements for third-party software incorporated in the Python distribution.

C.3.1 Mersenne Twister

The `_random` module includes code based on a download from http://www.math.keio.ac.jp/ matumoto/MT2002/emt19937ar.html. The following are the verbatim comments from the original code:

```
A C-program for MT19937, with initialization improved 2002/1/26.
Coded by Takuji Nishimura and Makoto Matsumoto.

Before using, initialize the state by using init_genrand(seed)
or init_by_array(init_key, key_length).

Copyright (C) 1997 - 2002, Makoto Matsumoto and Takuji Nishimura,
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

 1. Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.

 2. Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

 3. The names of its contributors may not be used to endorse or promote
    products derived from this software without specific prior written
    permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


Any feedback is very welcome.
http://www.math.keio.ac.jp/matumoto/emt.html
email: matumoto@math.keio.ac.jp
```

C.3.2 Sockets

The `socket` module uses the functions, `getaddrinfo`, and `getnameinfo`, which are coded in separate source files from the WIDE Project, http://www.wide.ad.jp/about/index.html.

```
Copyright (C) 1995, 1996, 1997, and 1998 WIDE Project.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. Neither the name of the project nor the names of its contributors
   may be used to endorse or promote products derived from this software
   without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE PROJECT AND CONTRIBUTORS ‘‘AS IS’’ AND
GAI_ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.  IN NO EVENT SHALL THE PROJECT OR CONTRIBUTORS BE LIABLE
FOR GAI_ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON GAI_ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN GAI_ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
```

C.3.3 Floating point exception control

The source for the `fpectl` module includes the following notice:

```
 ----------------------------------------------------------------------
/                      Copyright (c) 1996.                             \
|          The Regents of the University of California.                 |
|                      All rights reserved.                            |
|                                                                      |
|   Permission to use, copy, modify, and distribute this software for  |
|   any purpose without fee is hereby granted, provided that this en-  |
|   tire notice is included in all copies of any software which is or  |
|   includes  a  copy  or  modification  of  this software and in all  |
|   copies of the supporting documentation for such software.          |
|                                                                      |
|   This  work was produced at the University of California, Lawrence  |
|   Livermore National Laboratory under  contract  no.  W-7405-ENG-48  |
|   between  the  U.S.  Department  of  Energy and The Regents of the  |
|   University of California for the operation of UC LLNL.             |
|                                                                      |
|                              DISCLAIMER                              |
|                                                                      |
|   This  software was prepared as an account of work sponsored by an  |
|   agency of the United States Government. Neither the United States  |
|   Government  nor the University of California nor any of their em-  |
|   ployees, makes any warranty, express or implied, or  assumes  any  |
|   liability  or  responsibility  for the accuracy, completeness, or  |
|   usefulness of any information,  apparatus,  product,  or  process  |
|   disclosed,   or  represents  that  its  use  would  not  infringe  |
|   privately-owned rights. Reference herein to any specific  commer-  |
|   cial  products,  process,  or  service  by trade name, trademark,  |
|   manufacturer, or otherwise, does not  necessarily  constitute  or  |
|   imply  its endorsement, recommendation, or favoring by the United  |
|   States Government or the University of California. The views  and  |
|   opinions  of authors expressed herein do not necessarily state or  |
|   reflect those of the United States Government or  the  University  |
|   of  California,  and shall not be used for advertising or product  |
\  endorsement purposes.                                              /
 ----------------------------------------------------------------------
```

C.3.4 MD5 message digest algorithm

The source code for the `md5` module contains the following notice:

```
Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All
rights reserved.

License to copy and use this software is granted provided that it
is identified as the "RSA Data Security, Inc. MD5 Message-Digest
Algorithm" in all material mentioning or referencing this software
or this function.

License is also granted to make and use derivative works provided
that such works are identified as "derived from the RSA Data
Security, Inc. MD5 Message-Digest Algorithm" in all material
mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this
documentation and/or software.
```

C.3.5 rotor – Enigma-like encryption and decryption

The source code for the `rotor` contains the following notice:

```
Copyright 1994 by Lance Ellinghouse,
Cathedral City, California Republic, United States of America.

                        All Rights Reserved

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in
supporting documentation, and that the name of Lance Ellinghouse
not be used in advertising or publicity pertaining to distribution
of the software without specific, written prior permission.

LANCE ELLINGHOUSE DISCLAIMS ALL WARRANTIES WITH REGARD TO
THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS, IN NO EVENT SHALL LANCE ELLINGHOUSE BE LIABLE FOR ANY SPECIAL,
INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING
FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION
WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
```

C.3.6 Asynchronous socket services

The `asynchat` and `asyncore` modules contain the following notice:

```
Copyright 1996 by Sam Rushing

                        All Rights Reserved

Permission to use, copy, modify, and distribute this software and
its documentation for any purpose and without fee is hereby
granted, provided that the above copyright notice appear in all
copies and that both that copyright notice and this permission
notice appear in supporting documentation, and that the name of Sam
Rushing not be used in advertising or publicity pertaining to
distribution of the software without specific, written prior
permission.

SAM RUSHING DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN
NO EVENT SHALL SAM RUSHING BE LIABLE FOR ANY SPECIAL, INDIRECT OR
CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS
OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
```

C.3.7 Cookie management

The `Cookie` module contains the following notice:

```
Copyright 2000 by Timothy O'Malley <timo@alum.mit.edu>

               All Rights Reserved

Permission to use, copy, modify, and distribute this software
and its documentation for any purpose and without fee is hereby
granted, provided that the above copyright notice appear in all
copies and that both that copyright notice and this permission
notice appear in supporting documentation, and that the name of
Timothy O'Malley  not be used in advertising or publicity
pertaining to distribution of the software without specific, written
prior permission.

Timothy O'Malley DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS, IN NO EVENT SHALL Timothy O'Malley BE LIABLE FOR
ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
```

C.3.8 Profiling

The `profile` and `pstats` modules contain the following notice:

```
Copyright 1994, by InfoSeek Corporation, all rights reserved.
Written by James Roskind

Permission to use, copy, modify, and distribute this Python software
and its associated documentation for any purpose (subject to the
restriction in the following sentence) without fee is hereby granted,
provided that the above copyright notice appears in all copies, and
that both that copyright notice and this permission notice appear in
supporting documentation, and that the name of InfoSeek not be used in
advertising or publicity pertaining to distribution of the software
without specific, written prior permission.  This permission is
explicitly restricted to the copying and modification of the software
to remain in Python, compiled Python, or other languages (such as C)
wherein the modified or derived code is exclusively imported into a
Python module.

INFOSEEK CORPORATION DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS. IN NO EVENT SHALL INFOSEEK CORPORATION BE LIABLE FOR ANY
SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER
RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF
CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
```

C.3.9 Execution tracing

The `trace` module contains the following notice:

```
portions copyright 2001, Autonomous Zones Industries, Inc., all rights...
err...  reserved and offered to the public under the terms of the
Python 2.2 license.
Author: Zooko O'Whielacronx
http://zooko.com/
mailto:zooko@zooko.com

Copyright 2000, Mojam Media, Inc., all rights reserved.
Author: Skip Montanaro

Copyright 1999, Bioreason, Inc., all rights reserved.
Author: Andrew Dalke

Copyright 1995-1997, Automatrix, Inc., all rights reserved.
Author: Skip Montanaro

Copyright 1991-1995, Stichting Mathematisch Centrum, all rights reserved.


Permission to use, copy, modify, and distribute this Python software and
its associated documentation for any purpose without fee is hereby
granted, provided that the above copyright notice appears in all copies,
and that both that copyright notice and this permission notice appear in
supporting documentation, and that the name of neither Automatrix,
Bioreason or Mojam Media be used in advertising or publicity pertaining to
distribution of the software without specific, written prior permission.
```

C.3.10 UUencode and UUdecode functions

The uu module contains the following notice:

```
Copyright 1994 by Lance Ellinghouse
Cathedral City, California Republic, United States of America.
                       All Rights Reserved
Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in
supporting documentation, and that the name of Lance Ellinghouse
not be used in advertising or publicity pertaining to distribution
of the software without specific, written prior permission.
LANCE ELLINGHOUSE DISCLAIMS ALL WARRANTIES WITH REGARD TO
THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS, IN NO EVENT SHALL LANCE ELLINGHOUSE CENTRUM BE LIABLE
FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT
OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Modified by Jack Jansen, CWI, July 1995:
- Use binascii module to do the actual line-by-line conversion
  between ascii and binary. This results in a 1000-fold speedup. The C
  version is still 5 times faster, though.
- Arguments more compliant with python standard
```

C.3.11 XML Remote Procedure Calls

The xmlrpclib module contains the following notice:

```
    The XML-RPC client interface is

Copyright (c) 1999-2002 by Secret Labs AB
Copyright (c) 1999-2002 by Fredrik Lundh

By obtaining, using, and/or copying this software and/or its
associated documentation, you agree that you have read, understood,
and will comply with the following terms and conditions:

Permission to use, copy, modify, and distribute this software and
its associated documentation for any purpose and without fee is
hereby granted, provided that the above copyright notice appears in
all copies, and that both that copyright notice and this permission
notice appear in supporting documentation, and that the name of
Secret Labs AB or the author not be used in advertising or publicity
pertaining to distribution of the software without specific, written
prior permission.

SECRET LABS AB AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD
TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANT-
ABILITY AND FITNESS.  IN NO EVENT SHALL SECRET LABS AB OR THE AUTHOR
BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY
DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE
OF THIS SOFTWARE.
```

INDICE DEI MODULI

S

T

U

W

X

Z

INDICE

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z