Python Avanzato

Agosto 2006

Namespace
- Visibilita' delle variabili

Moduli
- File che contengono definizioni e istruzioni

Package
- Organizzare lo spazio dei nomi

Istruzioni speciali
- Alcune istruzioni speciali

Namespace

Accesso dinamico al nome delle variabili

- I namespace in python sono implementati tramite i dizionari
  in altre parole tutti i nomi di variabili o funzioni sono memorizzati in tali strutture

Visibilita'

- Esistono dizionari per le variabili locali locals(), vars(x) e per quelle globali globals()
  vengono usati a seconda del contesto in cui ci troviamo

Es:

```
a = 1;     b = 5          # il dizionario locale avra' il contenuto seguente:
{'a' : 1 , 'b' : 5, ...}
>> print locals()['a']    # ci restituisce
>> 1
```

- vars(x) si comporta come locals in assenza di parametri mentre con il parametro
  ispeziona il dizionario dell'argomento passato

- Python all'atto dell'assegnazione crea la variabile se non esiste nel dizionario locale!
  Se si vuol usare una variabile esterna bisogna dichiararla globale

```
def prova:
    global a              # dichiaro che la variabile a e' definita esterna alla funzione
    a = 10                # assegna 10 alla variabile che si trova da qualche altra parte
```


Loris Michielutti

Moduli

- Quando i programmi si fanno complessi e' opportuno suddividere le definizioni di funzioni e variabili in piu' files in modo ordinato per tipologia.
- Questi files prendono il nome di moduli e hanno estensione .py
  questi possono contenere anche delle istruzioni che serviranno a inizializzare le variabili quando il modulo e' richiamato la prima volta.

Ricerca

- Quando un modulo e' importato l'interprete segue un percorso di ricerca ben definito:
  - Directory corrente (.)
  - nella lista specificata dalla variabile d'ambiente PYTHONPATH
  - nel percorso predefinito dall'istallazione (es:/usr/local/bin/python)

Compilazione

- Un'accellerazione rilevante dei tempi si ottiene quando si richiama un modulo gia' compilato. Questo si compila la prima volta che si richiama e viene generato un file con stesso nome ma con estensione .pyc
- Se si vuol ottimizzare la compilazione si puo' usare da linea di comando
  - python -O nomeModulo.py
  questo genera un file ottimizzato con estensione .pyo
  eliminando le eventuali istruzioni di assert usate per il debug

Packages

Organizzare i nomi

- Il package e' un metodo per organizzare i moduli in modo strutturato

Directory

- E' il nostro Contenitore principale dove inseriremo il file __init__.py
  responsabile della descrizione della struttura.

Questo serve ad evitare che nomi di Directory si sovrappongono a nomi di moduli reali !

esempio di un Package con File system gerarchico:

```
Sound/                          # contenitore principale
    __init__.py                 # inizializza il package sound
    Formats/
        __init__.py             # inizializza il package formats
        wavRead.py
        wavWrite.py
    Effects/
        __init__.py             # inizializza il package effects
        echo.py
        surround.py
```

- Nel caso piu' semplice __init__.py puo' essere un file vuoto
  altre volte puo' contenere istruzioni di ogni genere (es: inizializzazione dei packages)

Loris Michielutti

Packages

Import dei package

- I moduli dei package vengono importati come dei moduli normali solo che i packages vengono richiamati separandoli con dei punti.

esempio:

```
import Sound.Effects.surround          # carico il modulo surround
```

- un modo alternativo

```
from Sound.Effects import surround     # carico il modulo surround senza prefisso del
                                       #     del package
```

- attenzione !!!

```
from Sound.Effects import *            # idealmente dovrebbe caricare tutti i
                                           sottomoduli presenti nel package
```

- questa istruzione non funziona molto bene in tutte le piattaforme si demanda all'autore di definire e mantenere la lista aggiornata dei moduli da caricare con l'istruzione:

  __all__ = [“echo”, “surround”]

  da inserire nel file

  __init__.py

Loris Michielutti

Special Istructions

exec

- Si usa per eseguire dinamicamente istruzioni python memorizzate in stringhe o files

```
exec 'print “Ciao mondo!”'
```

eval

- Si usa per valutare espressioni memorizzate in una stringa

```
eval ( “2*3”')
```

assert

- Si usa per asserire o imporre che qualcosa sia vero

```
miaLista = [“ciao”]
assert  len(miaLista) >= 1
```

quando qualcuno rimuove l'ultimo elemento dalla lista viene sollevata una eccezione di AssertionError

repr

- Si usa per ottenere una stringa rappresentativa dell'oggetto

```
i = [“elemento']
repr (i)
```

torna la stringa “[“elemento']”