

Marco Barisione

I moduli Python

Cosa sono i moduli

- Quando il programma cresce è necessario suddividere lo script in più parti
- In C non esiste il concetto di modulo
 - I file vengono compilati separatamente
 - I diversi file conoscono ciò che si trova negli altri includendo gli header
 - È necessario tenere gli header aggiornati con i sorgenti
- In Python i moduli sono normali sorgenti
- Per poter usare le funzioni, classi e variabili definite in un modulo è necessario importarlo

Come scrivere un modulo

- Per scrivere un modulo è semplicemente necessario scrivere il codice in un file

```
"Questa è la doc-string del modulo somma.py"
```

```
def somma(a, b):
```

```
    return a + b
```

```
print "È stato importato il modulo"
```

- Se la prima riga del modulo è una stringa questa diventa la doc-string
- Il codice all'interno del modulo viene eseguito
 - Questo non vuol dire che la funzione è richiamata!
 - Generalmente ciò che è fuori dalle funzioni e dalle classi serve per inizializzare il modulo o stampare informazioni si debug

Importare un modulo (1)

- “import modulo”
 - Per riferirsi ad un membro del modulo è necessario accedervi con “modulo.membro”
 - È il metodo più usato per importare
 - Si evitano collisioni di nomi (fatto frequente in C)

```
>>> import somma
```

```
>>> print somma
```

```
<module 'somma' from 'somma.py'>
```

```
>>> print somma.__doc__
```

```
Questa è la doc-string del modulo somma.py
```

```
>>> print somma.somma(5, 3)
```

```
8
```

Importare un modulo (2)

- “from modulo import funzione1, funzione2, classe1”
 - Importa dal modulo solo poche funzioni o classi
 - Se usato per le variabili non si ottiene il risultato aspettato
 - Si crea una variabile locale con lo stesso nome e che si riferisce allo stesso oggetto di quella del modulo
 - L'assegnando di un nuovo valore alla variabile è locale
 - Scomodo se bisogna importare molti membri del modulo

```
>>> from somma import somma
```

```
>>> print somma(5, 3)
```

```
8
```

Importare un modulo (3)

- “from modulo import *”
 - Importa tutti i membri del modulo eccetto quelli che iniziano per “_”
 - “Sporca” il namespace locale, rischiando di creare conflitti
 - Sconsigliato se non per alcuni moduli che usano altri metodi per evitare conflitti
 - Ad esempio tutti i membri iniziano con lo stesso prefisso

```
>>> from somma import *  
>>> print somma(1, 3)  
4
```

Come viene trovato un modulo

- Il modulo “sys” contiene la variabile “path”
- Python cerca in tutte le directory indicate nella in “sys.path” fino a quando trova un modulo con il nome corrispondente
- “sys.path” è una normale lista
 - È possibile aggiungere nuove directory di ricerca a runtime
 - `sys.path.append("~/moduli")`
- All'avvio di Python “sys.path” è inizializzato per contenere:
 - La directory corrente
 - Le directory indicate nella variabile d'ambiente “PYTHONPATH”
 - Alcune directory di default dipendenti dall'installazione

Byte-compilazione

- Quando un file viene importato per la prima volta viene creato un file con lo stesso nome ma con estensione “.pyc”
 - Nel file viene inserito il modulo “byte-compilato”
 - L'esecuzione di questo file non è più veloce, è solo più veloce la fase di caricamento
- Alla successiva esecuzione (se il file “.py” non è stato modificato) viene importato il file “.pyc”
- Un'applicazione può essere distribuita solo con i file compilati, senza i file sorgente

Moduli compilati in linguaggio macchina

- Alcuni moduli possono essere scritti in C (o altri linguaggi) e compilati in linguaggio macchina
 - Su Unix sono shared object con estensione “.so”
 - Su Windows sono DLL con estensione “.pyd”
- In caso di conflitto dovuto all'esistenza di due moduli con lo stesso nome, uno compilato e l'altro sorgente viene importato quello compilato

Package

- Con package si intende una collezione di moduli
- Quando i moduli crescono è naturale raggrupparli assieme
- Un esempio di possibile struttura è la seguente:

```
suoni/  
    __init__.py  
    formati/  
        __init__.py  
        wav.py  
        mp3.py  
    effetti/  
        __init__.py  
        eco.py
```

- All'interno di ogni directory del package è necessario un file “__init.py__”

Importare da un package

- Normali forme di importazione da package sono:
 - `import suoni.formati.mp3`
 - `from suoni.formati import mp3`
 - `from suoni.formati.mp3 import Mp3File`
- Per riferirsi a “Mp3File” si usano rispettivamente:
 - `suoni.formati.mp3`
 - `mp3.Mp3File`
 - `Mp3File`
- “from suoni.formati import *” non può funzionare a causa delle limitazioni di alcuni file system
 - Il problema può essere risolto inserendo in “suoni/formati/__init__.py” una lista chiamata “__all__” con l’elenco di tutti i moduli contenuti nel package
 - `__all__ = ["mp3", "wav"]`

__main__

- Spesso è utile poter importare un file ma poterlo eseguire anche separatamente
- È quindi necessario avere una sezione di codice eseguita solo se il modulo non è stato importato
- La variabile “__name__” contiene il nome del modulo
- Se il modulo è quello principale (non è stato importato) il suo nome è “__main__”

```
def foo(): ...  
if __name__ == "__main__":  
    # eseguito solo se è il modulo principale
```

OS

- “environ” un dizionario contenente le variabili d’ambiente
- “popen(cmd, mode)” apre una pipe da o per “cmd”
- “chdir(path)” cambia la directory corrente
- “getcwd()” ritorna la directory corrente
- “chmod(path, mode)” cambia il modo di “path” al modo numerico “mode”
- “listdir(path)” ritorna la lista di file e directory in “path”
- “mkdir(path[, mode])” crea la directory “path”, il default di “mode” è “0777”
- “remove(path)” cancella il file “path”
- “rmdir(path)” cancella la directory (vuota) “path”
- “rename(old, new)” rinomina il file o directory “old” a “new”
- “fork()” ritorna “0” nel processo figlio, l’id del processo figlio nel genitore (solo Unix)
- “system(cmd)” esegue il comando “cmd”

os.path

- “isfile(path)”, “isdir(path)” ritornano vero se “path” è, rispettivamente, un file normale o una directory
- “exists(path)” vero se “path” esiste
- “split(path)” ritorna una tupla contenente il nome della directory di path e il nome del file
- “dirname(path)” equivalente a “split(path)[0]”
- “basename(path)” equivalente a “split(path)[1]”
- “abspath(path)” ritorna il percorso assoluto di “path”
- “splitext(path)” ritorna una tupla contenente nome del file ed estensione (“.” compreso)

shutil

- “copyfile(src, dst)” copia “src” su “dst”
- “copytree(src, dst)” copia la directory “src” su “dst”
- “rmtree(path)” cancella ricorsivamente la directory “path”

sys

- “argv” una lista contenente gli argomenti sulla linea di comando
- “exit([arg])” esce dal programma, equivalente a lanciare un’eccezione SystemExit
- “path” lista di directory in cui cercare i moduli
- “stdin”, “stdout”, “stderr” stream standard, possono essere sostituiti con altri file
- “__stdin__”, “__stdout__”, “__stderr__”, valori originali degli stream standard

glob

- “glob(pathname)” ritorna una lista di file che corrispondono a “pathname”
 - Può contenere “*”, “?” e intervalli di caratteri
 - Non viene eseguita l’espansione di “~”

```
>>> os.listdir('.')
['1.gif', '2.txt', 'card.gif']
>>> import glob
>>> glob.glob('[0-9]*')
>>> glob.glob('*.gif')
['1.gif', 'card.gif']
>>> glob.glob('?.gif')
['1.gif']
```

math

- Contiene le funzioni matematiche principali
 - “pi” pi greco (π) ~ “3.1415926535897931”
 - “e” ~ “2.7182818284590451”
 - “floor(n)” approssima “n” all’intero inferiore
 - “ceil(n)” approssima “n” all’intero superiore
 - “round(n)” approssima “n” all’intero più vicino
 - “fabs(n)” ritorna il valore assoluto (a virgola mobile) di “n”
 - Esiste la funzione globale “abs” che ha comportamento simile
 - Le altre funzioni sono simili a quelle per il C contenute in “math.h”

exceptions

- Contiene la classe “Exception” dalla quale normalmente sono derivate le eccezioni
- Il costruttore di “Exception” accetta come argomento la descrizione dell’errore
- Generalmente è sufficiente derivare da “Exception” e aggiungere una descrizione

```
from exceptions import Exception
```

```
class MioErrore(Exception):
```

```
    def __init__(self, descr="Blah blah..."):  
        Exception.__init__(self, descr)
```

```
raise MioErrore
```

```
raise MioErrore, "Messaggio di errore..."
```

exceptions

Gestire le eccezioni

```
try:
    ...
except Exception as ex:
    ...
else:
    ...
finally:
    ...
```