Programmazione modulare: funzioni, moduli e package

Definizione di funzione

- **La definizione di funzioni avviene per mezzo della parola chiave def**
- ***Esempio – funzione fib:***

```
# Stampa la serie di Fibonacci da 1 fino a n
def fib(n):
    a, b = 0, 1
    print(a)
    while b <= n:
        print(b)
        a, b = b, a + b
```

Generazione dei numeri di Fibonacci: ciascun numero è la somma dei due numeri precedenti nella serie, partendo dai numeri 0 e 1
Serie: 0 1 1 2 3 5 8 13

Definizione di funzione

→ESEMPI
fib.py
fibReturn.py

```
# Stampa la serie di Fibonacci da 1 fino a n
def fib(n):
    a, b = 0, 1
    print(a)
    while b <= n:
        print(b)
        a, b = b, a + b
```

def è seguito da:
- *nome funzione*
- *lista parametri* tra () **passati per valore** e di cui **non si specifica il tipo**
- **Corpo della funzione** opportunamente indentato

Il **valore di ritorno** della funzione è:
- L'argomento dello statement return (se presente)
- None se non c'è return o se return non ha parametri (default)

Argomenti opzionali

→ESEMPI
default.py

- **In Python possiamo facilmente scrivere funzioni con un numero variabile di argomenti**
- **Utilizziamo degli argomenti opzionali**
  - **Nella definizione della funzione, un valore di default è associato all'argomento opzionale**
  - **Se non esplicito l'argomento al momento della chiamata, il valore assegnato è quello di default**
- **Esempio con 2 parametri opzionali**

```
def ask_ok(prompt, retries = 4, complaint = 'Yes or no, please!'):
```

Regole di scope

- Il modello di scoping del Python è statico (determinato a tempo di compilazione)
  - Se una variabile è definita in un blocco, è locale per quel blocco
  - Se una variabile è definita al di fuori delle funzioni, è considerata globale per un modulo
- Le variabili locali "sovrascrivono" temporaneamente le variabili omonime globali
  - Attenzione: assenza di dichiarazione
- Se non dichiarata diversamente Python assume che le variabili siano locali
  - Var globali: generalmente sconsigliate (limitano modularità, meglio usare parametri)

Regole di scope

→ESEMPI
scope2.py
scope.py

- L'uso di una variabile globale dentro ad un blocco di codice o ad una funzione richiede una dichiarazione esplicita
- Parola chiave global: permette di riferirsi alla variabile nel blocco top-level del file da blocchi di codice interni

Es.

```
first = 1            #var globale
def double():
    global first  #riferimento alla var globale
    first *=2     #raddoppio il valore di first
```

Modulo

- Unità di base per salvare codice che potrà essere riutilizzato
- In pratica: un file contenente definizioni di variabili e funzioni
  - Le definizioni possono essere importate e usate da altri moduli
- Il nome del file è lo stesso del modulo con estensione .py
- All'interno del modulo, il nome del modulo stesso è ottenibile tramite l'attributo speciale __name__

Importazione dei simboli

- Un modulo può importare le funzionalità di un altro modulo attraverso la clausola import
  - import *name*: cerca il modulo name nel *percorso dei moduli* e lo importa (notazione puntata per accesso a simboli – name.simbolo)
  - from *name* import *elenco*: cerca il modulo name nel *percorso dei moduli* e ne importa i simboli definiti in elenco (* per tutti i simboli del modulo)
- Percorso dei moduli:
  - Directory corrente
  - Elenco di directory in PYTHONPATH
  - Elenco di directory di sistema (/usr/lib/pythonX.X)

Modulo Sys

Modulo built-in in Python che dà accesso a entità a disposizione dell'interprete, come variabili d'ambiente e parametri d'ingresso

- **sys.path = lista del percorso dei moduli (include la var d'ambiente PYTHONPATH)**
  - **sys.path.append('/home/claudia/lib/mylib')**
- **sys.argv = lista degli argomenti passati a linea di comando a uno script / comando Python**
  - **argv[0] = nome di script/comando python**
  - **argv[1] = primo parametro d'ingresso**
  - **argv[2] = secondo parametro d'ingresso**
  - **….**
  - **I parametri vengono letti come stringhe → può essere necessaria conversione esplicita**

Statement esterni a definizioni

- **Nei moduli possono esserci statement esterni a definizioni di funzioni o di altre entità (es. classi)**

  Es. Nel modulo fibReturn.py
- **In caso di importazione del modulo**

  >>> import fibReturn

gli statement esterni vengono eseguiti

- **Dietro alla clausola import, in Python c'è una chiamata alla funzione *exec()***
- **Analogo nel Perl: si usa eval()**

- **L'esecuzione avviene solo alla prima importazione**
- **Statement esterni dovrebbero essere inseriti solo per fare inizializzazioni del modulo**

Uso di moduli: importati o indipendenti

- **Un stesso modulo può essere usato in due modi:**
  - **da un altro modulo che lo importa**
  - **come programma principale indipendente (main)**
- ***Come impostare correttamente il doppio uso?***
- **Uso dell'attributo speciale __name__**
  - **Assume valore "__main__" se il modulo è usato come programma principale**
  - **Assume valore uguale al nome del modulo se il modulo è usato come modulo importato**

>>> import FibMod2

FibMod2 impostato correttamente
Non esegue nulla se importato

$ python FibMod2.py 100
$ python usa_FibMod2.py

Parametro passato al modulo

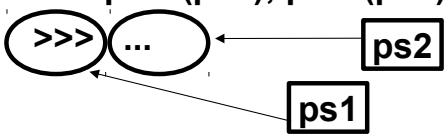
→ESEMPI
FibMod2.py
usa_FibMod2.py

Namespace di un modulo

- **Namespace: insieme dei simboli visibili in un certo punto del programma**
- **dir(): funzione built-in di Python per esplorare il namespace**
- **Se invocata senza argomenti, dir() stampa i simboli visibili dall'interprete in quel punto del codice, dove "simboli" sono:**
  - **funzioni (built-in e definite da utente)**
  - **variabili (built-in e definite da utente)**
  - **moduli (precedentemente importati)**
- **Usata come dir(modulo) elenca tutti i simboli (namespace) del modulo indicato**

Namespace

```
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__']
>>> import FibMod2; dir()
['FibMod2', '__builtins__', '__doc__', '__name__', '__package__']
>>> from FibMod2 import *; dir()
['FibMod2', '__builtins__', '__doc__', '__name__', '__package__',
'fib2']
>>> import sys; dir(sys)
… '__name__', '__doc__', 'maxint', 'ps1', 'ps2', …
>>> from sys import * ; dir()
>>> print(ps1); print(ps2)
```



Package, namespace e moduli

- Il concetto di package permette di strutturare in modo gerarchico i moduli (e i loro namespace) usando una notazione puntata
  - A.B : indica un modulo detto B contenuto in un package denominato A
- Divisione in package: moduli diversi possono avere lo stesso nome se in package diversi
- Il compilatore Python crea per ciascun modulo un namespace (contenitore di simboli)
  - *Namespace di default*: main (contiene variabili e funzioni del programma principale)

Package, namespace e moduli

- ***Struttura gerarchica*: ogni package consiste in una collezione di sub-package e moduli**
  - **Corrispondenza fisica sul file system con directory e file ordinati gerarchicamente**
  - **Directory → package**
  - **(sub-)directory → (sub-)package**
  - **File → moduli**

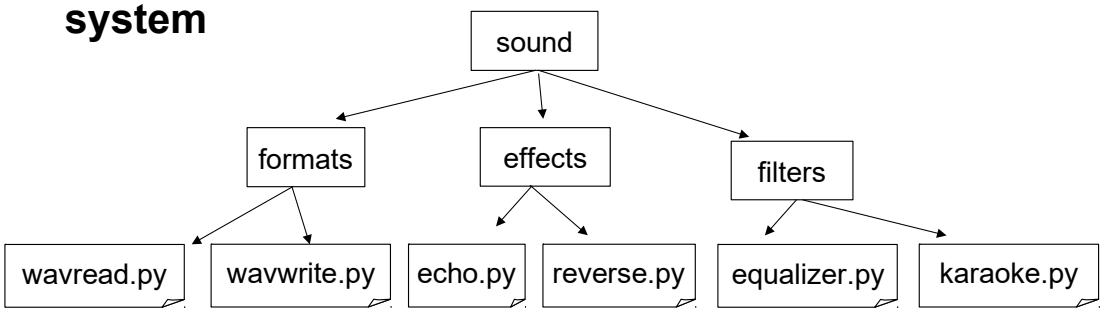
→ *(sub-)package = (sub-)directory contenente file che implementano moduli*

Struttura di un package

- ***Esempio*: supponiamo di voler realizzare un package di nome sound per la gestione di file e di dati audio**
- **Vari formati audio da manipolare e varie operazioni e filtri da applicare**
  - **sound: package top-level**
  - **sound.formats: sotto-package di gestione dei formati audio (es. .wav, .mp3, .aiff)**
  - **sound.effects: sotto-package di gestione degli effetti sonori (es. surround, reverse)**
  - **sound.filters: sotto-package di gestione dei filtri (es. equalizer, karaoke)**

Struttura di un package

- **Organizzazione gerarchica di directory nel file system**



- **Nella fase di importazione di moduli e package, Python cerca nelle directory specificate in sys.path le directory che corrispondono a package e che contengono i moduli**
- **Come fa a riconoscerle?**

Struttura di un package

Meccanismo per riconoscere automaticamente le directory/package e processarle di conseguenza

- **__init__.py: file che deve essere presente nella directory che implementa un package**
  - **L'uso di __init__.py è obbligatorio, altrimenti il Python non riconosce file e sottocartelle della directory come appartenenti ad un package**
  - **Il file __init__.py può anche essere vuoto**
- **Solitamente __init__.py contiene il codice di inizializzazione di package/moduli relativi**
- **Viene eseguito quando si importa il package**

Struttura del package sound

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
            __init__.py
            wavread.py          wavwrite.py         aiffread.py
            aiffwrite.py        auread.py           auwrite.py
      effects/                  Subpackage for sound effects
            __init__.py
            echo.py             surround.py         reverse.py
      filters/                  Subpackage for filters
            __init__.py
            equalizer.py        vocoder.py          karaoke.py
```


Importazione di simboli

- Importazione del modulo echo (echo.py): import sound.effects.echo
- L'interprete Python cerca:
  - nelle directory di sys.path una cartella di nome sound che contenga __init.py__
  - una sotto-cartella di sound con nome effects e che contenga __init.py__
  - un file di nome echo (o una sotto-cartella di nome echo che contenga __init.py__)
- Tutti i nomi precedenti l'ultimo devono essere package
- L'ultimo può essere un package o un modulo – non una singola entità (var, funzione, …)

Importazione di simboli

import sound.effects.echo

- **Se importato così, devo riferire le funzioni definite nel modulo echo con la notazione completa**

  sound.effects.echo.echofilter(par1,par2,par3)

- **Per evitare l'uso del nome completo, importare direttamente il namespace echo:**
  from sound.effects import echo
  echo.echofilter(par1, par2, par3)
- **O direttamente la funzione desiderata**
  from sound.effects.echo import echofilter

o tutti i simboli da echo
from sound.effects.echo import *
echofilter(par1,par2,par3)

Importazione da un package

- **Che effetto ha from sound.effects import * ?**
- **Potremmo volere che importi tutti i moduli in modo automatico**
- **Sfortunatamente non è così per vari problemi: potrebbe impiegare moltissimo tempo e ci potrebbero essere effetti collaterali**
  - **ES. i nomi di file di Windows possono essere case insensitive**
  - **ECHO.PY deve venire importato come echo, Echo oppure ECHO?**

- **Per ottenere questo effetto è necessario indicare un elenco di simboli esplicito**

Importazione da un package

- Lista __all__ nel file __init__.py del package
- Contiene i simboli da importare in presenza dello statement from package import *
- ES. Se il file sound/effects/__init.py__ contiene

__all__ = ["echo", "surround", "reverse"]

from sound.effects import * importa i moduli indicati

- Se invece __all__ non è definita, non li importa
  - Importa solo il package sound.effects ed esegue il codice di inizializzazione in __init__.py:
    - Importa i simboli definiti e i moduli esplicitamente importati in __init__.py