Moduli e Package

Cosa sono i moduli Python?

- Le definizioni di funzioni e variabili **non vengono salvate** quanto usciamo dall'interprete Python
- I **moduli** consentono di salvare le definizioni per accedervi in un secondo momento
- I moduli consentono inoltre di eseguire le istruzioni come script eseguibili
- I moduli sono **file con estensione .py**

Importare un modulo

- I moduli vengono importati utilizzando il comando incorporato import, senza l'estensione .py

```
>>> import example
```

- Per rendere più facile l'accesso, si possono importare singole definizioni all'interno di un modulo

```
>>> from function import func1, func2
```

- Quando i moduli sono importati, **tutte le dichiarazioni e le definizioni saranno eseguite**

Accedere ai componenti di un modulo

- Per utilizzare le funzioni definite nel modulo, digitare il nome del modulo seguito dal punto

```
>>> example.func(3)
```

- Se la funzione nel modulo è stata importata individualmente con ***from***, il nome del modulo e il punto possono essere omessi

```
>>> func(3)
```

Moduli come script

- Come sappiamo, gli script Python con estensione .py possono essere eseguiti da un terminale con

  `python example.py <arguments>`

- I moduli importati (utilizzando **`import`**) e quelli eseguiti direttamente (**`python nome_modulo.py`**) possono essere distinti accedendo alla variabile globale **`__name__`**

La "funzione" `main()` e i moduli

- Generalmente la funzione **`main()`** è il punto di ingresso principale di un programma
- Tuttavia l'interprete Python **esegue il codice fin dalla prima riga di un codice sorgente**, a prescindere dal fatto che ci sia o meno la definizione della funzione `main()`
- In Python non esiste una funzione main()
  - Quando viene dato all'interprete il comando di esecuzione di un programma Python, viene eseguito il codice che si trova già al livello 0 di indentazione
- La variabile built-in **`__name__`** valuta il nome del **modulo** corrente
  - Se il file ***.py*** viene **eseguito come programma principale**, l'interprete imposta la variabile `__name__` in modo che abbia il valore **`__main__`**
  - Se il file .py viene **importato da un altro modulo**, `__name__` sarà impostato sul **nome del modulo**.

Esempio di `main() \1`

```
# Python program to demonstrate main() function
print("Hello")
# Defining main function
def main():
        print("hey there")

# Using the special variable
# __name__
if __name__=="__main__":
        main()
```

- Output

```
Hello
hey there
```

Esempio di `main() \2`

```python
# File1.py

print("File1 __name__ = %s" %__name__)

if __name__ == "__main__":
    print("File1 is being run directly")
else:
    print("File1 is being imported")
```

Output:

```
File1 __name__ = __main__
File1 is being run directly
```

```python
# File2.py

import File1

print("File2 __name__ = %s" %__name__)

if __name__ == "__main__":
    print("File2 is being run directly")
else:
    print("File2 is being imported")
```

Output:

```
File1 __name__ = File1
File1 is being imported
File2 __name__ = __main__
File2 is being run directly
```

Search path per i moduli

- L'interprete cerca tutti i moduli importati in determinati percorsi designati finché il modulo non viene trovato
- L’ordine seguito per la ricerca è il seguente:
  - Directory corrente
  - L'elenco delle directory nella variabile d'ambiente **`PYTHONPATH`**
  - Percorso predefinito dipendente dall'installazione

Search path per i moduli

- La variabile d'ambiente **PYTHONPATH** è un elenco di directory
- La directory corrente, la variabile d'ambiente **PYTHONPATH**, e il *default path* (dipendente dall'installazione), sono tutte informazioni memorizzate nella variabile **sys.path**

```
$ python
Python 3.9.7 (default, Sep 16 2021, 08:50:36)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> print(sys.path)
['', '/Users/pinco/opt/anaconda3/lib/python39.zip',
'/Users/pinco/opt/anaconda3/lib/python3.9', '/Users/pinco/opt/anaconda3/lib/python3.9/lib-
dynload', '/Users/pinco/opt/anaconda3/lib/python3.9/site-packages',
'/Users/pinco/opt/anaconda3/lib/python3.9/site-packages/aeosa',
'/Users/pinco/opt/anaconda3/lib/python3.9/site-packages/locket-0.2.1-py3.9.egg']
>>>
```

Moduli standard

- In Python esiste una **libreria standard di moduli** la **Python Library Reference**
- Questi moduli contengono operazioni che non fanno parte del nucleo principale del linguaggio Python
  - Sono comunque integrati, sia per motivi di efficienza che per fornire accesso alle primitive del sistema operativo, come le *system call*
- Alcuni moduli vengono **importati solo se si utilizza un determinato sistema operativo**
- Un modulo di libreria importante è **`sys`**

Il modulo *sys*

- Due variabili contenute in sys, **ps1** e **ps2**, contengono i valori delle stringhe usate nel **prompt primario** e **secondario**. Queste stringhe possono essere modificate a piacimento

```
>>> import sys
>>> sys.ps1 = 'gigi_shell> '
gigi_shell> print("plurt")
plurt
gigi_shell >
```

La funzione *dir()*

- La funzione **`dir()`** restituisce un elenco di nomi (variabili e funzioni) definiti dal modulo passato come argomento

```
import fibo
>>> dir(fibo) ['__name__', 'fib', 'fib2’]
```

- Quando dir viene eseguito senza argomenti, restituisce un elenco di tutti i nomi attualmente definiti, ad eccezione di quelli incorporate

```
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__',
'__name__', '__package__', '__spec__’]
```

Packages

- I ***package*** sono raccolte organizzate di moduli, moduli all'interno di moduli.
- I moduli sono memorizzati in directory, con ogni directory che contiene un file **`__init__.py`**
  - Il file *__init__.py* istruisce l'interprete Python di trattare la directory come se fosse un *package*
  - Il file *__init__.py* può essere semplicemente un **file vuoto**, ma può anche eseguire **codice di inizializzazione** per il *package* o impostare la variabile **__all__** (trattata più avanti)
- Ogni sottodirectory della directory principale del pacchetto è considerata un sottopacchetto

Esempio di package

```
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
              ...
```

Packages

- Il *package* o uno qualsiasi dei suoi sotto-package può quindi essere importato
- E.g.:

```
import currency.conversion.euro
```

  - È ora possibile accedere a qualsiasi definizione o sottopacchetto di **euro**, ma solo utilizzando la notazione punto:
    - **euro.convertfromdollar(10)**

Importazione da un package

- A volte si vuole importare tutto ciò che si trova all'interno di un package o di un sottopackage. In questo caso si può usare questo comando:

```
from example.subexample import *
```

- Questo carica tutti i nomi dei moduli contenuti nella variabile __all__, a sua volta contenuta nel file __init__.py della cartella del pacchetto.

Importazione da un package

- Se la variabile **`__all__`** non è definita, l'importazione tramite l'operatore * da un package di default agirà come di seguito:
  - **Non importerà tutti sottomoduli** dal package (per evitare problematiche di performance)
  - Assicurerà di importare tutto il package (possibilmente eseguendo il codice di inizializzazione contenuto in **`__init__.py`**) e importerà tutti i nomi definiti del package
  - Eventuali sottomoduli non saranno importati, **a meno che non siano esplicitamente importati da `__init__.py`**

Riferimenti Intra-package

- Quando i package sono strutturati in sottopackage, si possono usare le **importazioni assolute** per fare riferimento ai sottomoduli dei package "fratelli"
  - E.g., se il modulo `sound.filters.vocoder` deve usare il modulo `echo` del pacchetto `sound.effects`, può usare `from sound.effects import echo`.
- È anche possibile scrivere **importazioni relative**, con la forma di dichiarazione `from module import name`
  - Queste importazioni utilizzano dei punti iniziali per indicare il pacchetto corrente e quello padre coinvolti nell'importazione relativa
  - E.g.:

```
from . import echo
from .. import formats
from ..filters import equalizer
```

Packages in directory multiple

- La posizione in cui un package cerca il suo `__init__.py` può essere cambiata, modificando la **variabile `__path__`**, che memorizza il valore della cartella in cui si trova **`__init__.py`**