

Programmazione avanzata a.a. 2021-22
A. De Bonis

Introduzione a Python
**Il parte della IV lezione e I parte della
V lezione**

0

Namespace

- Quando si utilizza un identificativo si attiva un processo chiamato risoluzione del nome (*name resolution*) per determinare il valore associato all'identificativo
- Quando si associa un valore ad un identificativo tale associazione è fatta all'interno di uno scope
- Il **namespace** (spazio dei nomi) gestisce tutti i nomi definiti in uno scope (ambito)

1

Namespace

- Python implementa il namespace tramite un dizionario che mappa ogni identificativo al suo valore
- Uno scope può contenere al suo interno altri scope
- **Non c'è nessuna relazione tra due identificatori che hanno lo stesso nome in due namespace differenti**
- Tramite le funzioni `dir()` e `vars()` si può conoscere il contenuto del namespace dove sono invocate
 - `dir` elenca gli identificatori nel namespace
 - `vars` visualizza tutto il dizionario

Programmazione Avanzata a.a. 2021-22
A. De Bonis

2

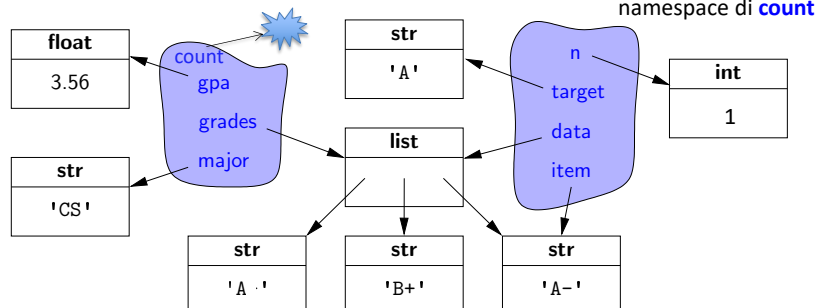
2

Esempio

```
grades = ['A', 'B+', 'A-']
gpa = 3.56
major = 'CS'
count(grades, 'A')
```

```
def count(data, target):
    n=0
    for item in data:
        if item == target:
            n += 1
    return n
```

namespace dove è chiamata `count`



Programmazione Avanzata a.a. 2021-22
A. De Bonis

3

3

Esempio

```
def count(data, target):
    n=0
    for item in data:
        if item == target:
            n += 1
    return n
```

```
grades = ['A', 'B+', 'A-']
gpa = 3.56
major = 'CS'
```

```
count(grades, 'A')
```

```
print(dir())
```

```
[
    '__annotations__', '__builtins__',
    '__cached__', '__doc__', '__file__',
    '__loader__', '__name__', '__package__',
    '__spec__', 'count', 'gpa', 'grades', 'major'
]
```

4

I moduli in Python

- Un modulo è un particolare script Python
 - È uno script che può essere utilizzato in un altro script
 - Uno script incluso in un altro script è chiamato modulo
- Sono utili per decomporre un programma di grande dimensione in più file, oppure per riutilizzare codice scritto precedentemente
 - Le definizioni presenti in un modulo possono essere importate in uno script (o in altri moduli) attraverso il comando **import**
 - Il nome di un modulo è il nome del file script (esclusa l'estensione '.py')
 - All'interno di un modulo si può accedere al suo nome tramite la variabile globale `__name__`

5

Moduli esistenti

- Esistono vari moduli già disponibili in Python
 - Alcuni utili moduli sono i seguenti

Existing Modules	
Module Name	Description
array	Provides compact array storage for primitive types.
collections	Defines additional data structures and abstract base classes involving collections of objects.
copy	Defines general functions for making copies of objects.
heapq	Provides heap-based priority queue functions (see Section 9.3.7).
math	Defines common mathematical constants and functions.
os	Provides support for interactions with the operating system.
random	Provides random number generation.
re	Provides support for processing regular expressions.
sys	Provides additional level of interaction with the Python interpreter.
time	Provides support for measuring time, or delaying a program.

Programmazione Avanzata a.a. 2021-22
A. De Bonis

6

6

Utilizzare i moduli

- All'interno di un modulo/script si può accedere al nome del modulo/script tramite l'identificatore `__name__`
- Per utilizzare un modulo deve essere incluso tramite l'istruzione **import**
 - **import math**
- Per far riferimento ad una funzione del modulo importato bisogna far riferimento tramite il nome qualificato completamente
 - `math.gcd(7,21)`

Programmazione Avanzata a.a. 2021-22
A. De Bonis

7

7

Utilizzare i moduli

- Con l'istruzione **from** si possono importare singole funzioni a cui possiamo far riferimento direttamente con il loro nome
 - **from** math **import** sqrt
 - **from** math **import** sqrt, floor

```
import math
print(math.gcd(7,21))

from math import sqrt
print(sqrt(3))
```



```
7
1.7320508075688772
```

from math **import** * tutte le funzioni di **math** sono importate

Programmazione Avanzata a.a. 2021-22
A. De Bonis

8

8

Caricamento moduli

- Ogni volta che un modulo è caricato in uno script è eseguito
- Il modulo può contenere funzioni e codice *libero*
- Le funzioni sono *interpretate*, il codice libero è eseguito
- Lo script che importa (eventualmente) altri moduli ed è eseguito per primo è chiamato dall'interprete Python `__main__`
- Per evitare che del codice *libero* in un modulo sia eseguito quando il modulo è importato dobbiamo inserire un controllo nel modulo sul nome del modulo stesso. Se il nome del modulo è `__main__` allora il codice libero è eseguito; altrimenti il codice non viene eseguito.

Programmazione Avanzata a.a. 2021-22
A. De Bonis

9

9

La variabile `__name__`

- Ogni volta che un modulo è importato, Python crea una variabile per il modulo chiamata `__name__` e salva il nome del modulo in questa variabile.
- Il nome di un modulo è il nome del suo file `.py` senza l'estensione `.py`.
- Supponiamo di importare il modulo contenuto nel file `test.py`. La variabile `__name__` per il modulo importato `test` ha valore `"test"`.
- Supponiamo che il modulo `test.py` contenga del codice libero. Se prima di questo codice inseriamo il controllo `if __name__ == '__main__':` allora il codice libero viene eseguito se e solo se `__name__` ha valore `__main__`. Di conseguenza, se importiamo il modulo `test` allora il suddetto codice libero non è eseguito.
- Ogni volta che un file `.py` è eseguito Python crea una variabile per il programma chiamata `__name__` e pone il suo valore uguale a `"__main__"`. Di conseguenza se eseguiamo `test.py` come se fosse un programma allora il valore della sua variabile `__name__` è `__main__` e il codice libero dopo l'if viene eseguito.

Programmazione Avanzata a.a. 2021-22
A. De Bonis

10

10

Esempio

testNolfMain.py

```
def modifica(lista):
    lista.append('nuovo')

lst = [1, 'due']
print('lista =', lst)
modifica(lst)
print('lista =', lst)
```

esecuzione testNolfMain.py

```
lista = [1, 'due']
lista = [1, 'due', 'nuovo']
```

Stesso comportamento se
eseguiti entrambi come
programmi

test.py

```
def modifica(lista):
    lista.append('nuovo')

if __name__ == '__main__':
    lst = [1, 'due']
    print('lista =', lst)
    modifica(lst)
    print('lista =', lst)
```

esecuzione test.py

```
lista = [1, 'due']
lista = [1, 'due', 'nuovo']
```

Programmazione Avanzata a.a. 2021-22
A. De Bonis

11

11

Esempio

importUNO.py

```
import test
lista = [3,9]
print(lista)
test.modifica(lista)
print(lista)
```

esecuzione importUNO.py

```
[3, 9]
[3, 9, 'nuovo']
```

In questo caso l'if presente in test.py evita che vengano eseguite le linee di codice libero presenti in test.py

importDUE.py

```
import testNoIfMain
lista = [3,9]
print(lista)
testNoMain.modifica(lista)
print(lista)
```

esecuzione importDUE.py

```
lista = [1, 'due']
lista = [1, 'due', 'nuovo']
[3, 9]
[3, 9, 'nuovo']
```

In questo caso vengono eseguite anche le linee di codice libero di testNoIfMain.py perché non sono precedute dall'if

Programmazione Avanzata a.a. 2021-22
A. De Bonis

12

12

package

- Modo per strutturare codice Python in moduli, cartelle e sotto-cartelle
- Il package è una collezione di moduli
 - Il package è una cartella in cui, oltre ai moduli o subpackage, è presente il file `__init.py__` che contiene istruzioni di inizializzazione del package (può essere anche vuoto)
 - `__init.py__` serve ad indicare a Python di trattare la cartella come un package

Programmazione Avanzata a.a. 2021-22
A. De Bonis

13

13

sound/	Top-level package
__init__.py	Initialize the sound package
formats/	Subpackage for file format conversions
__init__.py	
wavread.py	In uno script presente nella cartella che contiene sound
wavwrite.py	
aiffread.py	
aiffwrite.py	<code>import sound.effects.echo</code>
auread.py	<code>sound.effects.echo.echofilter(input, output, delay=0.7)</code>
auwrite.py	
...	
effects/	
__init__.py	<code>from sound.effects import echo</code>
echo.py	<code>echo.echofilter(input, output, delay=0.7)</code>
surround.py	
reverse.py	
...	
filters/	
__init__.py	<code>from sound.effects.echo import echofilter</code>
equalizer.py	<code>echofilter(input, output, delay=0.7)</code>
vocoder.py	
karaoke.py	
...	

Programmazione Avanzata a.a. 2021-22
A. De Bonis

14

14

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/	
__init__.py	
echo.py	
surround.py	Per importare moduli in surround.py si usa un import relativo
reverse.py	
...	
filters/	
__init__.py	
equalizer.py	
vocoder.py	
karaoke.py	
...	

Programmazione Avanzata a.a. 2021-22
A. De Bonis

15

15

Importare moduli tra **package**

- Lo script che importa il modulo deve conoscere la posizione del modulo da importare
 - Non è necessario quando
 - il modulo è un modulo di Python
 - il modulo è stato installato
 - La variabile `sys.path` è una lista di stringhe che determina il percorso di ricerca dell'interprete Python per i moduli
 - Occorre aggiungere a `sys.path` il percorso assoluto che contiene il modulo da importare

Programmazione Avanzata a.a. 2021-22
A. De Bonis

16

16

Importare moduli tra **package**

- Quando il modulo `miomodulo` è importato l'interprete prima cerca un modulo built-in con quel nome. Se non lo trova, cerca un file `miomodulo.py` nella lista di directory date dalla variabile **`sys.path`**
- `sys.path` è una lista di stringhe che specifica il percorso di ricerca di un modulo. `sys.path` è inizializzata dalle seguenti locazioni:
 - Contiene nella prima posizione la directory contenente lo script input
 - è inizializzata da `PYTHONPATH` (una lista di nomi di directory con la stessa sintassi della variabile shell `PATH`).
 - Default dipendente dall'installazione

Programmazione Avanzata a.a. 2021-22
A. De Bonis

17

17

