

Uso dei Moduli

Informatica@SEFA 2017/2018 - Lezione 5

Massimo Lauria <massimo.lauria@uniroma1.it>
<http://massimolauria.net/courses/infosefa2017/>

Venerdì, 6 Ottobre 2017

Ancora su operatori booleani

Confronti tra stringhe

```
print('Mario' == 'Bruno')           1
print('Mar' < 'Mario' and 'Mar' < 'Marco') 2
print('A' < 'B')                     3
print('Z' < 'a')                     4
print('0' < '9' < 'A' < 'Z' < 'a' < 'z') 5
print('Mario' > 'Bruno')             6
```

```
False
True
True
True
True
True
```

Ordine lessicografico

Dato un ordine tra caratteri (che può essere arbitrario),

```
stringa1 < stringa2
```

1

quando

- ▶ nella prima posizione in cui differiscono, il carattere di `stringa1` è più piccolo di quello di `stringa2`
- ▶ `stringa1` è un prefisso di `stringa2`

Condizione non booleana

```
def verooofalso(x):
    if x:
        print(repr(x) + ' è come True')
    else:
        print(repr(x) + ' è come False')

verooofalso('')    # stringa vuota è falso, le altre vere
verooofalso(0)     # 0 è falso, gli altri interi sono veri
verooofalso(-3)   # 0 è falso, gli altri interi sono veri
verooofalso('0')
verooofalso(0.0)
verooofalso(0.00000001)
```

```
' ' è come False
0 è come False
-3 è come True
'0' è come True
0.0 è come False
1e-08 è come True
```

Implicitamente la condizione è bool

```
def veroolalso(x):
    if bool(x):
        print(repr(x) + ' è come True')
    else:
        print(repr(x) + ' è come False')

veroolalso('')    # stringa vuota è falso, le altre vere
veroolalso(0)     # 0 è falso, gli altri interi sono veri
veroolalso(-3)   # 0 è falso, gli altri interi sono veri
veroolalso('0')
veroolalso(0.0)
veroolalso(0.00000001)
```

```
' ' è come False
0 è come False
-3 è come True
'0' è come True
0.0 è come False
1e-08 è come True
```

Indentazione

Indentazione

L'inserimento di spazio vuoto all'inizio della riga, per

- identificare blocchi logici di codice
- rendere il codice più leggibile

Esempio di indentazione annidata

```
def scontato(prezzo, sconto):           1
    if sconto < 0 or sconto > 100:     2
        print("Errore nell'input")     3
        print("Lo sconto deve essere  4
        tra 0 e 100")
        return                          5
                                        6

    percentuale = 100 - sconto           7
    prezzo_scontato = prezzo *          8
    percentuale / 100
    return prezzo_scontato              9
                                        10

print(scontato(1000,20))                11
print(scontato(500,15))                 12
```

```
800.0
425.0
```

In Python l'indentazione è importante

Le istruzioni nello stesso blocco devono essere **allineate**

```
print("Prima riga")           1
    print("seconda riga")     2
```

```
Prima riga
File "<stdin>", line 1
    print("seconda riga")
    ^
IndentationError: unexpected indent
```

```
print("Prima riga")           1
print("seconda riga")         2
```

```
Prima riga
seconda riga
```

Quanto indentare

Io suggerisco 2, 3 o 4 spazi. È possibile impostare l'editor per aiutarvi a tenere le righe indentate correttamente.

- ▶ la lunghezza dell'indentazione è facoltativa
- ▶ non compromettete la leggibilità

```
x = 12 1
print("Primo livello di indentazione, 0 spazi") 2
if x > 0: 3
    print("Secondo livello di indentazione, 2 spazi") 4
    if x<100: 5
        print("Terzo livello di indentazione, 1 spazio") 6
else: 7
    print("Secondo livello di indentazione, 5 spazi") 8
```

De-indentare

Ridurre l'indentazione comunica al Python che la nuova istruzione fa parte del blocco di codice più esterno. Quindi questa **deve** essere allineata.

```
x = 10 1
def gruppo_istruzioni(): 2
    print("Tizio") 3
    print('Caio') 4
    print("Sempronio") 5
gruppo_istruzioni() 6 7
```

Usare e scrivere moduli Python

Modulo

Un file python è un **modulo**, ovvero un'unità che contiene funzioni e variabili pronte per essere riutilizzate.

```
import math 1  
print(math.pi * math.sin(0.4)) 2
```

```
1.2233938033699718
```

I moduli python sono documentati

```
import math 1  
help(math) 2  
3
```

Help on module math:

NAME

math

MODULE REFERENCE

<https://docs.python.org/3.6/library/math>

The following documentation is automatically generated from the Python source files. It may be incomplete, incorrect or include features that are considered implementation detail and may vary between Python implementations. When in doubt, consult the module reference at the location listed above.

<.. TANTE ALTRE INFORMAZIONI....>

Anche le funzioni sono documentate

```
import math 1  
help(math.log) 2
```

Help on built-in function log in module math:

```
log(...)  
    log(x[, base])
```

Return the logarithm of x to the given base.
If the base not specified, returns the natural
logarithm (base e) of x.

Spazio dei nomi

In ogni punto e momento del programma esiste uno

spazio dei nomi

- ▶ nomi delle variabili e funzioni definite, moduli caricati
- ▶ ad ogni nome corrisponde una sola entità python

```
temp = 4.2                                1
                                           2
def temp():                                3
    return 5                               4
                                           5
print(type(temp))                          6
```

```
<class 'function'>
```

Importare un modulo

Per usare un modulo python, dovete farci riferimento all'interno del programma. Quindi dovete caricarlo nello spazio dei nomi.

```
import <modulo>
```

```
print(math.pi)           1
import math              2
print(math.pi)          3
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'math' is not defined
3.141592653589793
```

Importare solo una funzione del modulo

```
from <modulo> importa <nome_funzione>
```

```
print(math.cos(0.4))      1
print(cos(0.4))          2
from math import cos     3
print(math.cos(0.4))    4
print(cos(0.4))         5
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'math' is not defined
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'math' is not defined
0.9210609940028851
```

Scriviamo il nostro file `primomodulo.py`

```
def quadrato(x):           1
    return x**2           2

def cubo(x):              3
    return x**3           4

print('codice che prova il modulo') 5
print(cubo(2)+quadrato(3))           6
                                     7
                                     8
```

```
$ python3 primomodulo.py
codice che prova il modulo
17
```

Usiamo primomodulo.py

```
import primomodulo 1
print(primomodulo.cubo(3)) 2
3
```

```
codice che prova il modulo
17
27
```

Possiamo riutilizzare le funzioni di `primomodulo.py` !

Ma il codice di prova ci disturba!

La variabile `__name__`

Scriviamo `secondomodulo.py`

```
def quadrato(x):           1
    return x**2           2
                            3

def cubo(x):              4
    return x**3           5
                            6

if __name__ == '__main__': 7
    print('codice che prova il modulo') 8
    print(cubo(2)+quadrato(3)) 9
```

```
$ python3 secondomodulo.py
codice che prova il modulo
17
```

```
import secondomodulo      1
print(secondomodulo.cubo(3)) 2
```

```
27
```

Docstrings, documentazione dei moduli

La documentazione è **molto meglio** dei commenti!

```
"""Descrizione di una riga del modulo                                1
                                                                    2
Descrizione lunga: La docstring del modulo appare                 3
generalmente prima di qualunque istruzione.                       4
"""                                                                    5
def latitriangolo(x,y,z):                                          6
    """Determina se tre lunghezze sono lati di un triangolo      7
                                                                    8
    Dati x,y,z positivi di tipo 'float', determina se può        9
    esistere un triangolo con lati di lunghezza x,y,z. La       10
    funzione restituisce True o False.                            11
    """                                                            12
    if not (type(x)==type(y)==type(z)==float):                    13
        raise TypeError("Si richiedono tre numeri 'float'")      14
    if x <= 0 or y <= 0 or z <= 0:                                15
        raise ValueError("Si richiedono tre numeri positivi")    16
    return (x < y + z) and (y < x + z) and (z < x + y)           17
                                                                    18
if __name__ == '__main__':                                        19
    print(latitriangolo(3.0,4.0,5.0))                             20
    print(latitriangolo(-3.0,4.0,3.0))                            21
```

Testiamo le docstrings

```
import lecture05 1  
help(lecture05.latitriangolo) 2 3
```

Help on function latitriangolo in module lecture05:

latitriangolo(x, y, z)

Determina se tre lunghezze sono lati di un triangolo

Dati x,y,z positivi di tipo 'float', determina se può esistere un triangolo con lati di lunghezza x,y,z. La funzione restituisce True o False.

Sequenze, mutabilità e Identità

Sequenze di dati **indicizzabili**

- tuple
- liste
- stringhe
- range
- ...

```
print(range(3,7))           1
print(list(range(3,7)))    2
print( range(3,7)[3] )     3
print(list(range(6)) )    4
```

```
range(3, 7)
[3, 4, 5, 6]
6
[0, 1, 2, 3, 4, 5]
```

Entità immutabili

Un'entità **immutabile** non può essere modificata dopo la sua definizione

Mutabili

- liste `<class 'list'>`

Immutabili

- stringhe `<class 'str'>`
- tuple `<class 'tuple'>`

Entità immutabili

```
lista = [1,2,3]           1
stringa = "ciao"        2
tupla = (1,2,3)         3

lista[0] = 1000          4
print(1000)             5

stringa[0] = 'a'        6

tupla[0]=1000           7
```

Appendix 1: uso di funzioni

Esempio: Due e zero parametri formali

```
from math import sqrt          1
def diagonale Rettangolo(base, altezza):  2
    return sqrt(base**2+altezza**2)      3
                                        4
print( ( 1 + diagonale Rettangolo(2,1) ) / 2 )  5
```

1.618033988749895

```
import math                    1
def pi_strano():                2
    return 2*math.asin(1)      3
                                4
print( 'Pi greco è ' +str( pi_strano() ))  5
```

Pi greco è 3.141592653589793

Lecture

Lezione di oggi:

- Capitoli 4.5 e 4.6, 5.5, 6

La maggior parte del capitolo 6 è stato trattato alla lavagna, e quindi non ci sono slide.