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

Comparazioni 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 veroofalso(x):
    if x:
        print(repr(x) + ' è come True')
    else:
        print(repr(x) + ' è come False')

veroofalso('')   # stringa vuota è falso, le altre vere
veroofalso(0)    # 0 è falso, gli altri interi sono veri
veroofalso(-3)   # 0 è falso, gli altri interi sono veri
veroofalso('0')
veroofalso(0.0)
veroofalso(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 veroofalso(x):
    if bool(x):
        print(repr(x) + ' è come True')
    else:
        print(repr(x) + ' è come False')

veroofalso('')   # stringa vuota è falso, le altre vere
veroofalso(0)    # 0 è falso, gli altri interi sono veri
veroofalso(-3)   # 0 è falso, gli altri interi sono veri
veroofalso('0')
veroofalso(0.0)
veroofalso(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):
      if sconto < 0 or sconto > 100:
            print("Errore nell'input")
            print("Lo sconto deve essere tra 0 e 100")
            return

      percentuale = 100 - sconto
      prezzo_scontato = prezzo * percentuale / 100
      return prezzo_scontato

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

```
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. É 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

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

```
x = 10

def gruppo_istruzioni():
    print("Tizio")
    print('Caio')
    print("Sempronio")
  gruppo_istruzioni()
```

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

help(math)
```

```
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

def temp():
    return 5

print(type(temp))
```

```
<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)
import math
print(math.pi)
```

```
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))
print(cos(0.4))
from math import cos
print(math.cos(0.4))
print(cos(0.4))
```

```
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
                                                    3
def cubo(x):                                        4
    return x**3                                     5
                                                    6
print('codice che prova il modulo')                 7
print(cubo(2)+quadrato(3))                          8
```

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

Usiamo `primomodulo.py`

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

Descrizione lunga: La docstring del modulo appare
generalmente prima di qualunque istruzione.
"""
def 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.
    """
    if not (type(x)==type(y)==type(z)==float):
        raise TypeError("Si richiedono tre numeri 'float")
    if x <= 0 or y <= 0 or z <= 0:
        raise ValueError("Si richiedono tre numeri positivi")
    return (x < y + z) and (y < x + z) and (z < x + y)

if __name__ == '__main__':
    print(latitriangolo(3.0,4.0,5.0))
    print(latitriangolo(-3.0,4.0,3.0))
```

Testiamo le docstrings

```
import lecture05

help(lecture05.latitriangolo)
```

```
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
                                                   4
lista[0] = 1000                                    5
print(1000)                                        6
                                                   7
stringa[0] = 'a'                                   8
                                                   9
tupla[0]=1000                                      10
```

```
1000
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Identità

La funzione `id()` associa un numero ad ogni oggetto.

Due oggetti che esistono simultaneamente non hanno mai lo stesso numero.

```
print(id(4))                                    1
lista1 = [1,2,4]                                2
lista2 = [1,2,4]                                3
print(id(lista1))                               4
print(id(lista2))                               5
lista1.append(6)                                6
print(lista1)                                   7
print(id(lista1))                               8
```

```
4416093552
4419722120
4419722184
[1, 2, 4, 6]
4419722120
```

id(), tuple,stringhe e liste

L'operatore += estende le sequenze.

```
lista    = [1,2,3]                                                    1
tupla    = (1,2,3)                                                    2
stringa  = "abc"                                                      3
print("L:", id(lista)," T:", id(tupla), " S:", id(stringa))           4
                                                                      5
lista   += [4,5,6]                                                    6
tupla   += (4,5,6)                                                    7
stringa += "def"                                                      8
print("L:", id(lista)," T:", id(tupla), " S:", id(stringa))           9
                                                                      10
lista   = lista    + [7,8,9]                                          11
tupla   = tupla    + (7,8,9)                                          12
stringa = stringa  + "ghi"                                            13
print("L:", id(lista)," T:", id(tupla), " S:", id(stringa))           14
```

```
L: 4523905800  T: 4523762312  S: 4522217232
L: 4523905800  T: 4523506856  S: 4523904560
L: 4523911560  T: 4523501512  S: 4523924016
```

Mutabilità: liste vs variabili numeriche

```
lista    = [1,2,3]                                  1
var      = 5                                        2
print("L:", id(lista)," N:", id(var))               3
                                                    4
lista   += [4,5,6]                                  5
var     += 5                                        6
print("L:", id(lista)," N:", id(var))               7
                                                    8
lista   = lista    + [7,8,9]                        9
var     = var + 5                                   10
print("L:", id(lista)," N:", id(var))               11
```

```
L: 4508525320  N: 4505087376
L: 4508525320  N: 4505087536
L: 4508531080  N: 4505087696
```

Tuple vs Liste

Liste

- una serie di dati omogenei
- serie temporale
- una lista di osservazioni

Tupla

- un singolo dato, anche non omogeneo
- una voce in una rubrica telefonica
- un punto cartesiano `(x,y)`
- un colore RGB `(r,g,b)`

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
```

Letture

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.