Elementi del linguaggio Python

Informatica@SEFA 2018/2019 - Lezione 5

Massimo Lauria <massimo.lauria@uniroma1.it>
http://massimolauria.net/courses/infosefa2018/

Mercoledì, 3 Ottobre 2018

Capitoli del libro su Python

Il capitolo 1 accenna alle cose viste nelle lezioni 1 e 2.

Oggi vediamo il contenuto del capitolo 3.1-3.3

- tipi di dati numerici e operazioni aritmetiche
- usare python come una calcolatrice

Compiti per casa:

- leggere tutto il capitolo 3.
- leggere queste slides.

Scelta di argomenti

Queste slide coprono tutto il capitolo 3 ma sono molto nozionistiche

- userò alcune cose senza spiegarle (chiedete/usate il libro)
- provate le cose da soli usando Python
- imparate a leggere i messaggi di errore

Tipi numerici e calcoli

In Python ogni dato ha un tipo

```
type(5)          # il tipo dell'espressione 5                    1
type('ciao')     # il tipo dell'espressione 'ciao'               2
type(3.2)        # il tipo dell'espressione 3.2                  3
type(5.0)        # il tipo dell'espressione 5.0                  4
3.2 + 5          # somma tra dati di tipo diverso                5
type(3.2 + 5)    # il tipo del risultato                         6
5 + 'ciao'       # altra somma tra dati di tipo diverso          7
```

```
<class 'int'>
<class 'str'>
<class 'float'>
<class 'float'>
8.2
<class 'float'>
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Numeri naturali e interi

I numeri naturali $\mathbb{N}$ sono $0, 1, 2, 3, \ldots$

- in alcuni libri lo zero non è incluso, in altri sì.

I numeri interi $\mathbb{Z}$ sono $\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots$

- contengono numeri negativi.

Gli interi sono codificati con elementi di tipo int

```
print( type(-3) )
print( type(0) )
print( type(100) )
```

```
<class 'int'>
<class 'int'>
<class 'int'>
```

Python come una calcolatrice

Le operazioni comuni +,-,* sono supportate

```
print(15 + 3)
print(10-25)
print(3*7)
```

```
18
-15
21
```

Naturalmente i risultati sono di tipo int

```
print( type( 3 +  12) )
print( type(15 -  24) )
print( type(2*4 + 17 - 24) )
```

```
<class 'int'>
<class 'int'>
<class 'int'>
```

Numeri non interi

I matematica l'insieme dei numeri reali è denotato da $\mathbb{R}$.

- alcuni hanno rappresentazioni posizionali finite

$$13,24 \quad 0,57$$

- la maggior parte di essi non ne ha

$$\frac{4}{3} \quad \pi \quad 2^{\pi^2} \quad \sqrt[\pi]{\frac{3}{7}}$$

Rappresentazione dei numeri reali

I numeri reali sono rappresentati come sequenze finite di cifre prima e dopo la virgola:

- E.g. 123,2441 ; 3,2123 ; 0,0000321 ; 1232,2
- E.g. $4/3$ o π non sono rappresentabili

```
print( type(12.5) )
print( - 12.5 + 1.7 )
print( 23.1 * -2 )
print( type(-4) )
print( type(-4.0) )
```

```
<class 'float'>
-10.8
-46.2
<class 'int'>
<class 'float'>
```

Floating point numbers (`float`)

La rappresentazione mantiene alcune cifre decimali, le più significative, “spostando” la virgola.

$$12340000000000000000.0 = 1.234 \times 10^{19}$$

$$0.00000000000000001234 = 1.234 \times 10^{-16}$$

```
print(12340000000000000000.0)
print(0.00000000000000001234)
```

```
1.234e+19
1.234e-16
```

notazione scientifica: NeE invece di $N \times 10^{E}$

`float` è una rappresentazione approssimata

Vengono perse le cifre meno significative.

```
print(1234000000000000000000001000000.0)
print(0.123400000000000000000000001000000)

# 10**20 + 10**(-20) uguale a 10**20 ?
print(1e20 + 1e-20 == 1e20)

print( (3.0**(1/100))**100 )
```

```
1.234e+32
0.1234
True
2.999999999999994
```

Questo può portare ad errori se non si gestisce l'approssimazione.

Conversione di tipi: da `float` a `int`

Se `x` è un `float` allora `int(x)` è ottenuto troncando i decimali

```
print(int(12.5) )                                    1
print(int(-12.5))                                    2
print(int(1.28475e+13))                              3
print(int(0.54) )                                    4
```

```
12
-12
12847500000000
0
```

Conversione di tipi: da `int` a `float`

Se `x` è un `int` allora `float(x)` è ottenuto prendendo le cifre più significative

```
print ( float(12) )                                          1
print ( float(0) )                                           2
print ( float(120000000000000000000000001) )                 3
```

```
12.0                                                         1
0.0                                                          2
1.2e+26                                                      3
```

Operazioni tra `int` e `float`

Le operazioni aritmetiche tra `int` e `float` sono fatte

1. convertendo l'operando intero a `float`
2. eseguendo l'operazione

Anche se il risultato è intero

```
print(15.7 + 3)                                                    1
print(18.0 * 5)                                                    2
print(type(18.0 * 5))                                              3
```

```
18.7
90.0
<class 'float'>
```

Divisione ‘intera’ // e resto (int)

```
print(5 // 3)                                                    1
print(6 // 3)                                                    2
print(5 % 3)                                                     3
print(6 % 3)                                                     4
```

```
1
2
2
0
```

Divisione ‘intera’ `//` e resto (`float`)

```
print(5.2 // 3.0)
print(5 // 3.0)
print(7.1 // 3.3)
print(7.1 % 3.3)
```

```
1.0
1.0
2.0
0.5
```

Resto è sempre positivo

```
print(5 // 3)
print(-5 // 3)
print(5 % 3)
print(-5 %  3)
print(6.3 // 3.2)
print(-6.3 // 3.2)
print(6.3 %  3.2)
print(-6.3 %  3.2)
```

```
1
-2
2
1
1.0
-2.0
3.0999999999999996
0.10000000000000053
```

Divisione esatta /

La divisione esatta è sempre un `float`

```
print(2 / 3)                                                    1
print(4 / 2)                                                    2
print(2.0 / 5)                                                  3
print(4.0 / 1.3)                                                4
```

```
0.6666666666666666
2.0
0.4
3.0769230769230766
```

Divisioni per zero `int`

```
>>> 2 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero

>>> 2 // 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero

>>> 2 % 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

Divisioni per zero `float`

```
>>> 2.0 / 0.0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: float division by zero

>>> 2.0 // 0.0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: float divmod()

>>> 2.0 % 0.0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: float modulo
```

Elevazione a potenza

```
print( 2**8 )                                                    1
print( 2   ** 8.0 )                                              2
print( 2.0 ** 8.0 )                                              3
print( 2**0.5 )                                                  4
print( 2**100 )                                                  5
print( 2.0**100 )                                                6
print( 2**(-3))                                                  7
```

```
256
256.0
256.0
1.4142135623730951
1267650600228229401496703205376
1.2676506002282294e+30
0.125
```

- se un operando è `float`, il risultato è `float`
- se base e esponente sono interi:
  - potenza positiva `int`
  - potenza negativa `float`

Precedenza degli operatori

1. `**` con associatività a destra
2. `/`, `//`, `%` con associatività a sinistra
3. `+`, `-` come operatori aritmetici
4. Eccezioni e altri operatori nella documentazione

```
print( 3 * 2 ** -2  + 5 * 2 // 5  )                    1
print( (3 * 2) ** -2  + 5 * ( 2 // 5 ) )               2
```

```
2.75
0.02777777777777776
```

Le parentesi non necessarie migliorano la leggibilità.

Modulo matematico (I)

```
import math

print( math.pi )
print( math.sin(0.0) )
print( math.sin(math.pi / 2) )
print( math.sin(math.pi) )

print( math.e )
print( math.log(math.e * math.e) )
```

```
3.141592653589793
0.0
1.0
1.2246467991473532e-16
2.718281828459045
2.0
```

Modulo matematico (II)

```
import math                                                    1
                                                               2
print( math.log10(10.0)   )                                    3
print( math.log10(100.0)  )                                    4
print( math.log10(1.0e32) )                                    5
                                                               6
print( math.log2(2**10) )                                      7
print( math.log2(1/2)   )                                      8
```

```
1.0                                                            1
2.0                                                            2
32.0                                                           3
10.0                                                           4
-1.0                                                           5
```

Altre conversioni da `float` a `int`

```
import math                                                    1
print( int(5.32)    )      # troncamento                       2
print( round(5.32) )       # arrotonda all'intero più vicino   3
print( round(5.5) )        # arrotonda all'intero più vicino   4
print( round(5.9) )        # arrotonda all'intero più vicino   5
print( math.floor(5.3) ) # intero minore più vicino            6
print( math.floor(-5.3) )# intero minore più vicino            7
print( math.ceil(5.3)   ) # intero maggiore più vicino         8
print( math.ceil(-5.3) ) # intero maggiore più vicino          9
```

```
5
5
6
6
5
-6
6
-5
```

Commenti nel codice

I commenti servono ad aumentare

- leggibilità
- manutenibilità

```
# Copyright 2017 Massimo Lauria <massimo.lauria@uniroma1.it>
#
# 2017/5/12 - supporto per input codificati Latin-1

3 / 2   # un commento può essere anche dopo del codice

print("ciao")  # i commenti non hanno nessun effetto sul
    codice
```

Variabili

Variabili

L’associazione di un nome al valore di un espressione.

```
nome_variable = espressione
```

Durante l’esecuzione nel codice

- inizializzata con un valore
- il valore può cambiare nel tempo
- l’informazione nella variabile è riutilizzata
- la variable viene distrutta

Uso e riuso di variabili

```
pigreco = 3.14
# area di un cerchio di raggio 10
raggio = 10
area = pigreco * raggio ** 2
print(area)
 # ricalcolo dell'area con raggio 20
raggio = 20
area = pigreco * raggio ** 2
print(area)
```

```
314.0
1256.0
```

Il tipo di una variabile

Tipo della variabile = tipo del dato memorizzato

Può variare durante il programma

```
approx_pigreco = 3
print(type(approx_pigreco))

# meglio usare un'approssimazione migliore
approx_pigreco = 3.141592
print(type(approx_pigreco))
```

```
<class 'int'>
<class 'float'>
```

Name not defined

Non è possibile utilizzare una variable prima che essa sia definita. Se lo facciamo l'interprete Python darà un errore.

```
print(2 * non_definita)
```

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

Stringhe e testi

Stringhe di testo

Le stringhe sono sequenze di bit (o piuttosto di byte) che codificano del testo.

```
print('ciao')                                    1
print("L'altra mattina")                         2
print('')    # stringa vuota                      3
```

```
ciao
L'altra mattina

```

Apici singoli e doppi

Se nel programma si usano gli apici `'` e `"` per delimitare le stringhe, come si inseriscono questi apici all'interno delle stringhe stesse.

```
print("una stringa che contiene l'apostrofo")
print('una stringa "protetta" da apici singoli')
```

```
una stringa che contiene l'apostrofo
una stringa "protetta" da apici singoli
```

Ma se li voglio mischiare?

Caratteri speciali o non stampabili

Per inserire certi caratteri nelle stringhe del programma esistono le “sequenza escape” `\n \' \" \t \\`

```
print("Sequenze escape\n\t\\n - a capo")                    1
print("\t\\\' - apice singolo\n\t\\\" - apice doppio")      2
print("\t\\\\ - backslash")                                 3
```

```
Sequenze escape
  \n - a capo
  \' - apice singolo
  \" - apice doppio
  \\ - backslash
```

Costruzione di testi

Python ha delle operazioni per l'elaborazione di stringhe

```
nome = "Giorgio"                                    1
cognome = "Rossi"                                   2
print(nome + " " + cognome)  #concatenazione        3
```

```
Giorgio Rossi
```

```
boom = 'tic tac '*5 + 'BOOM!'                       1
print(boom)   # ripetizione                        2
```

```
tic tac tic tac tic tac tic tac tic tac BOOM!
```

Conversione di numeri e stringhe

É possibile usare `str` per convertire numeri in stringhe, ed usare `int` e `float` per la direzione inversa.

```
print('stringa' + str(5))
print(10 + int('5'))
print( - 20 + float('5e3'))
print(float('-5.1232'))
```

```
stringa5
15
4980.0
-5.1232
```

Errori di conversione

```
>>>  int('ciao')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'ciao'

>>>  float('non sono un float')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: could not convert string to float: 'non sono un float'
```

La funzione `print`

L'abbiamo vista in altri esempi. Serve per stampare a video dei testi.

```
print("ciao")                                                    1
print("parola",2.3,54,"a caso")  # una sequenza di valori       2
print("Mario","&","Luigi")                                      3
```

```
ciao
parola 2.3 54 a caso
Mario & Luigi
```

Letture

- Capitolo 3
- Paragrafi 4.1, 4.2
- Queste slide

Esempio (1):

Risolutore per equazione di secondo grado

$$Ax^2 + Bx + C = 0$$

```
import math

def eqsecondogrado(A,B,C):
    """Risolve equazioni di 2o grado A x^2 + B x + C = 0"""
    Delta = B*B - 4*A*C
    if Delta < 0:
        print("Nessuna soluzione")
    else:
        if A==0:
            print("Non è un equazione propria di 2o grado")
        else:
            # Utilizzo la formula standard
            sol1 = ( -B - math.sqrt(Delta) ) / 2*A
            sol2 = ( -B + math.sqrt(Delta) ) / 2*A
            print("Soluzioni: ",sol1,sol2)
```

Esempio (2)

```
eqsecondogrado(1, 0, 0)                                    1
eqsecondogrado(0, 3, 1)                                    2
eqsecondogrado(1, 2, 1.0)                                  3
eqsecondogrado(2, 1, 2)                                    4
                                                           5
help(eqsecondogrado)                                       6
```

```
Soluzioni:  0.0 0.0
Non è un equazione propria di 2o grado
Soluzioni:  -1.0 -1.0
Nessuna soluzione
Help on function eqsecondogrado in module __main__:
eqsecondogrado(A, B, C)
    Risolve equazioni di 2o grado A x^2 + B x + C = 0
```