

Elementi del linguaggio Python

Informatica@SEFA 2017/2018 - Lezione 3

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

Venerdì, 29 Settembre 2017

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 $0, 1, 2, 3, \dots$

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

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

- contengono numeri negativi.

Gli interi sono codificati in Python con elementi di tipo **int**

```
type(-3)           1
type(0)            2
type(100)          3
```

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

Python come una calcolatrice

Le operazioni comuni +,-,* sono supportate

```
15 + 3 1
# Out: 18 2
10 - 25 3
# Out: -15 4
3*7 5
# Out: 21 6
```

Naturalmente i risultati sono di tipo **int**

```
type( 3 + 12) 1
type(15 - 24) 2
type(2*4 + 17 - 24) 3
```

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

Numeri non interi

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

- ▶ alcuni hanno rappresentazioni posizionali **finite**

13,24 0,57

- ▶ la maggior parte di essi non ne ha

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

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

```
type(12.5)           1
- 12.5 + 1.7        2
23.1 * -2           3
type(-4)            4
type(-4.0)          5
```

```
<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.

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

$$0.0000000000000001234 = 1.234 \times 10^{-16}$$

```
12340000000000000000.0  
0.0000000000000001234
```

1
2

```
1.234e+19
```

```
1.234e-16
```

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

Conversione di tipi: da float a int

Se x è un float allora $\text{int}(x)$ è ottenuto troncando i decimali

<code>int(12.5)</code>	1
<code>int(-12.5)</code>	2
<code>int(1.28475e+13)</code>	3
<code>int(0.54)</code>	4

12	1
-12	2
12847500000000	3
0	4

Conversione di tipi: da int a float

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

<code>float(12)</code>	1
<code>float(0)</code>	2
<code>float(12000000000000000000000000000001)</code>	3

12.0	1
0.0	2
1.2e+26	3

Operazioni tra int e float

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

- convertendo l'operando intero a `float`
- eseguendo l'operazione

Anche se il risultato è intero

<code>15.7 + 3</code>	<code>1</code>
<code># Out: 18.7</code>	<code>2</code>
<code>18.0 * 5</code>	<code>3</code>
<code># Out: 90.0</code>	<code>4</code>

Divisione 'intera' // e resto (int)

5 // 3	1
6 // 3	2
5 % 3	3
6 % 3	4

1
2
2
0

Divisione 'intera' // e resto (float)

```
5.2 // 3.0
```

1

```
5 // 3.0
```

2

```
7.1 // 3.3
```

3

```
7.1 % 3.3
```

4

```
1.0
```

```
1.0
```

```
2.0
```

```
0.5
```

Resto è sempre positivo

5 // 3	1
-5 // 3	2
5 % 3	3
-5 % 3	4
6.3 // 3.2	5
-6.3 // 3.2	6
6.3 % 3.2	7
-6.3 % 3.2	8

1
-2
2
1
1.0
-2.0
3.09999999999999996
0.100000000000000053

Divisione esatta /

La divisione esatta è sempre un float

2 / 3	1
4 / 2	2
2.0 / 5	3
4.0 / 1.3	4

```
0.6666666666666666  
2.0  
0.4  
3.0769230769230766
```

Divisioni per zero int

```
2 / 0 1
2 // 0 2
2 % 0 3
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
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 1  
2.0 // 0.0 2  
2.0 % 0.0 3
```

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

Elevazione a potenza

<code>2**8</code>	1
<code>2 ** 8.0</code>	2
<code>2.0 ** 8.0</code>	3
<code>2**0.5</code>	4
<code>2**100</code>	5
<code>2.0**100</code>	6

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

- ▶ 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

<code>3 * 2 ** -2 + 5 * 2 // 5</code>	1
<code>(3 * 2) ** -2 + 5 * (2 // 5)</code>	2

<code>2.75</code> <code>0.027777777777777776</code>
--

Le parentesi non necessarie migliorano la leggibilità.

Modulo matematico (I)

```
import math 1
2
math.pi 3
math.sin(0.0) 4
math.sin(math.pi / 2) 5
math.sin(math.pi) 6
7
math.e 8
math.log(math.e * math.e) 9
```

```
3.141592653589793 1
2
0.0 3
1.0 4
1.2246467991473532e-16 5
2.718281828459045 6
2.0 7
```

Modulo matematico (II)

<code>math.log10(10.0)</code>	1
<code>math.log10(100.0)</code>	2
<code>math.log10(1.0e32)</code>	3
	4
<code>math.log2(2**10)</code>	5
<code>math.log2(1/2)</code>	6

1.0	1
2.0	2
32.0	3
10.0	4
-1.0	5

Altre conversioni da float a int

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

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 variabile viene distrutta

Uso e riuso di variabili

```
pigreco = 3.14 1
# area di un cerchio di raggio 10 2
raggio = 10 3
area = pigreco * raggio ** 2 4
print(area) 5
# Out: 314.0 6
7
8
# ricalcolo dell'area con raggio 20 9
raggio = 20 10
area = pigreco * raggio ** 2 11
print(area) 12
# Out: 1256.0 13
```

Il tipo di una variabile

Tipo della variabile = tipo del dato memorizzato

Può variare durante il programma

```
approx_pigreco = 3           1
print(type(approx_pigreco)) 2
                               3
# meglio usare un'approssimazione migliore 4
approx_pigreco = 3.141592    5
print(type(approx_pigreco))  6
```

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

Name not defined

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

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

1

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

Stringhe di testo

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

```
'ciao' 1
# Out: 'ciao' 2
print('ciao') 3
# Out: ciao 4
5
print("L'altra mattina") 6
# Out: L'altra mattina 7
8
# stringa vuota 9
'' 10
# Out: '' 11
print('') 12
# Out: 13
```

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")      1
print('una stringa "protetta" da apici singoli')  2
```

```
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
```

Organizzazione del codice

Astrazioni

Astrazione: l'atto di non tenere in considerazione una o più proprietà di un oggetto complesso, così da poter analizzarne altre.

Funzioni

Variabili: riuso del **valore di espressioni**

Funzioni: riuso di una **sequenze di istruzioni**

```
def nome_funzione(parametro1, parametro2, ..., parametroN): 1
    istruzione_1 2
    istruzione_2 3
    ... 4
    istruzione_M 5
    return espressione # opzionale 6
```

Per usare una funzione già creata

```
nome_funzione(valore1, valore2, ..., valoreN) 1
```

Funzioni: esempio

```
def area_cilindro(raggio, altezza):           1
    pigreco = 3.14159                         2
    area = pigreco * raggio ** 2             3
    circonferenza = 2 * pigreco * raggio     4
    return 2 * area + altezza * circonferenza 5
                                                6

print(area_cilindro(10, 5))                 7
# Out: 942.477                              8

print(area_cilindro(20, 10))                9
# Out: 3769.908                             10
```

- ▶ raggio, altezza sono i **parametri formali**
- ▶ (10,5) e (20,10) sono i **parametri effettivi**

Nomi legali per variabili e funzioni

- può contenere lettere maiuscole o minuscole
- può contenere il carattere _
- può contenere numeri

Non può iniziare con un numero

```
variable1 = 0           1
variabile_con_nome_lungo = 0  2
NomeMaiuscolo = 0      3
def _my_print():       4
    pass                5
```

Lecture

- Capitolo 3
- Paragrafi 4.1, 4.2