36. Introduzione al Python

Corso di Informatica

Corso di Laurea in Matematica (D.M. 270/04) - A.A. 2020/2021
Angelo Cardellicchio
angelo.cardellicchio@uniba.it
11/01/2021

Outline

- Papere e serpenti
- L’interprete Python
- Calcoli e numeri
- Stringhe
- Liste

Papere e serpenti (1)

- L'**interprete** Python offre una tipizzazione **dinamica**
  - *Ciò significa che l'interprete valuta il tipo di ogni variabile a runtime, e che questo può cambiare*
- Ciò significa che il programmatore può evitare di specificare il tipo delle variabili, in quanto inferito direttamente dall'interprete
- È un vantaggio? **Dipende…**
  - *Se da un lato la complessità della scrittura del codice diminuisce, dall'altro infatti occorre porre particolare attenzione al tipo assunto dalle variabili*
- L'interprete Python infatti adopera il principio chiamato **duck typing**

Papere e serpenti (2)

If it walks like a duck, and quacks like a duck, then it must be a duck.

- L'interprete Python inferisce il tipo di una variabile **a partire dal suo comportamento** nel contesto del nostro programma
  - *Ad esempio, se una variabile si comporta da intero, allora sarà per forza di cose un intero!*
- Se applicato alle classi, il duck typing implica che **la definizione di una classe è meno importante dei metodi definiti al suo interno**
  - *Se un oggetto si comporta come una* `Persona`*, allora sarà una* `Persona`*!*
- Il duck typing, se ben utilizzato, facilita di molto la scrittura di un programma…
  - *…ma, se mal utilizzato, ne rompe inesorabilmente il funzionamento!*

L’interprete Python

- È giunto il momento del primo approccio a Python
- Useremo direttamente l’interprete, usando la console interattiva da riga di comando
- Assicuriamoci di aver completato la procedura di installazione, e lanciamo l’interprete digitando, da riga di comando, l’istruzione **python**

```
C:\>python
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Calcoli e numeri (1)

- In primis, proviamo ad usare l'interprete come una semplice calcolatrice
  - *Proviamo, ad esempio, una somma, una moltiplicazione ed una semplice espressione*

```
>>> 2 + 2
4
>>> 3 * 5
15
>>> 10 - 2 * 4
2
```

- Per le divisioni, otteniamo il valore esatto con `/`, il quoziente intero con `//` ed il modulo con `%`

```
>>> 16 / 3
5.333333333333333
>>> 16 // 3
5
>>> 16 % 3
1
```

Calcoli e numeri (2)

- Per calcolare una potenza, dobbiamo usare l'operatore **

```
>>> 3 ** 2
9
>>> 2 ** 8
256
```

- Oltre ai classici tipi **`int`** e **`float`**, Python ne supporta di altri. Ad esempio, ha un supporto integrato ai numeri complessi, espressi usando il prefisso **`j`** o **`J`** per la parte immaginaria

```
>>> a = 1 + 4j
>>> b = 2 + 3j
>>> print(a + b)
(3+7j)
```

Stringhe (1)

- Python supporta le stringhe, che possono essere indifferentemente racchiuse tra singole e doppie virgolette

```
>>> "una stringa"
'una stringa'
>>> 'un\'altra stringa'
"un'altra stringa"
```

- Sono ovviamente supportati gli escape character, che possono essere però ignorati andando ad aggiungere una `r` prima dell'inizio della stringa.

```
>>> print('C:\nuova_cartella')
C:
uova_cartella
>>> print(r'C:\nuova_cartella')
C:\nuova_cartella
```

Stringhe (2)

- Le stringhe possono anche estendersi su più righe, usando le **triple quotes**

```
>>> print("""Questo è un esempio\
        di
       riga multipla\
       """)
   Questo è un esempio di
   riga multipla
```

- È anche possibile concatenare più stringhe. Per farlo, si può usare l'operatore **+**

```
>>> stringa_a = "Prima stringa"
>>> stringa_b = "Seconda stringa"
>>> print(stringa_a + " -
 " + stringa_b)
Prima stringa - Seconda stringa
```

Stringhe (3)

- Possiamo anche concatenare due **string literals** semplicemente mettendoli l'uno dopo l'altro

```
>>> "Py" "thon"
'Python'
```

- Le stringhe sono considerate come degli array (o, più propriamente, **liste**). Possono essere quindi indicizzate.

```
>>> stringa = 'Python'
>>> stringa[0]
'P'
```

Stringhe (4)

- Python supporta gli indici **negativi**, ovvero quelli che vanno da destra verso sinistra

```
>>> stringa[-1]
'n'
```

- È possibile usare anche le operazioni di **slicing** per selezionare delle sottostringhe

```
>>> stringa[0:2]
'Py'
>>> stringa[2:5]
'tho’
>>> stringa[1:]
'ython'
>>> stringa[:5]
'Pytho'
```

Stringhe (5)

- Per ottenere la lunghezza di una stringa, occorre usare la funzione **len()**

```
>>> len(stringa)
6
```

- Le stringhe sono **immutabili**: ciò significa che non possiamo ridefinirne uno o più elementi, acceduti mediante indicizzazione o slicing

```
>>> stringa[0] = 'C'    # Errore!
```

Liste (1)

- Python **non** supporta nativamente gli array (si affida a librerie terze come **NumPy**)
- Le **liste** sono i container che vengono utilizzati per rappresentare degli insiemi di variabili
- Le liste sono estremamente versatili: **non sono confinate, come nel C e nel C++, ad ospitare elementi di un solo tipo**
  - *Possiamo addirittura avere liste miste ad elementi primitivi nidificate all'interno di altre liste!*
- Le stringhe sono considerate alla stregua di liste, per cui le operazioni che abbiamo visto in precedenza valgono anche su queste ultime
- Le liste, però, sono **mutabili**

Liste (2)

```
>>> lista = [1, 2, 3, 4, 5]      # Dichiarazione di una lista di interi
[1, 2, 3, 4, 5]

>>> lista[0]                     # Slicing, indicizzazione e concatenazione
1
>>> lista[2:]
[3, 4, 5]
>>> lista_due = [6,7]
>>> lista + lista_due
[1, 2, 3, 4, 5, 6, 7]
>>> lista + [6]
[1, 2, 3, 4, 5, 6]
```

Liste (3)

```
>>> lista[0] = 99                  # Mutabilità
>>> lista
[99, 2, 3, 4, 5]

>>> lista[4:] = []                 # Eliminazione di un elemento di una lista
>>> lista
[99, 2, 3, 4]

>>> lista.append([1,2,3])          # Aggiunta di elementi ad una lista
>>> lista
[99, 2, 3, 4, [1, 2, 3]]

>>> lista [0] = stringa
>>> lista
['Python', 2, 3, 4, [1, 2, 3]]
```

Domande?

42