Linguaggio Python - Introduzione

Mirko Cesarini - Dario Pescini
nome.cognome@unimib.it

Università di Milano Bicocca

Linguaggio di programmazione del corso: Python

- Python
  - Un linguaggio interpretato creato da Guido van Rossum
  - Prende il nome dalla serie televisiva "Monty Python's Flying Circus" (e non dal nome di un serpente)
- Domande
  - Perché python e non un linguaggio di programmazione più blasonato (es. Java, C, C++, ...)?
  - perché non dei linguaggi di programmazione sviluppati appositamente per la statistica (es. R, SAS, ...)?

Contesto

- Come statistici dovrete imparare diversi linguaggi di programmazione
  - R
  - SAS
  - ...
- Problema
  - Non tutti dovrete imparare gli stessi linguaggi
  - Non è possibile insegnarvi tutti questi linguaggi in un corso singolo
- “Houston we have a problem" !!!

Imparare a programmare

- Cosa occorre?
  - Acquisire i fondamenti della programmazione strutturata
    - seppur declinati diversamente tra un linguaggio e l'altro, i concetti basilari sono molto simili tra loro
    - dopo aver imparato un primo linguaggio, apprenderne un altro è semplice
  - Imparare "un certo modo" di ragionare (comune a diversi linguaggi)
  - E' sufficiente concentrarsi su un unico linguaggio 
- In questo corso si insegna Python
  - Python è un linguaggio molto potente, ma con una curva di apprendimento adatta a principianti e con tutte le funzionalità utili ad esperti
  - Consente di focalizzare maggiormente l'attenzione sul ragionamento e meno sui problemi sintattici

Vantaggi di Python

- E' adatto per risolvere velocemente problemi
- Facilita l'evoluzione da soluzioni "quick and dirty" verso progetti strutturati di maggiore complessità
- L'uso di Python si sta diffondendo notevolmente in ambito accademico, economico, finanziario, ...
- ... Provate a fare una verifica (usando un motore di ricerca)
- Compromesso
  - La documentazione di Python a volte usa una terminologia non standard rispetto ad altri linguaggi del mondo dell'informatica
  - Dato che per voi Python un punto di passaggio per altri mondi, per quanto possibile sarà usata una terminologia comune ad altri linguaggi di programmazione

I componenti principali di un linguaggio di programmazione

- Variabili
- Istruzioni
  - Istruzioni di input e output
  - Istruzioni di assegnamento (per eseguire elaborazioni)
  - Istruzioni condizionali
  - Istruzioni di iterazione
- Librerie

Python: due differenti modalità di lavoro

- Script
  - File contenente un insieme di istruzioni
  - Programma sorgente e script sono sinonimi
  - L'esecuzione di uno script implica l'esecuzione ordinata di tutte le istruzioni contenute al suo interno
- La "riga di comando" (chiamata anche "prompt" o "interactive window")
  - Esegue un'istruzione per volta
  - permette di valutare velocemente l'effetto di un'istruzione
- I programmi complessi sono composti da molte istruzioni $\rightarrow$, conviene utilizzare gli script

Alcuni script

- Primo esempio. E' una tradizione visualizzare *hello world* nel primo programma che uno scrive

```
1 print('Hello world')
```

- Questo è ciò che viene visualizzato

```
Hello world
```

Alcuni script

- Altro esempio di script

```
1  # Questo e' un commento
2  # i commenti come questo non vengono ...
3  # ... considerati dall'interprete python
4  print("Dato un rettangolo di lati") #l'istruzione print
5                                       # visualizza un messaggio a video
6  print(10) # i numeri non vanno racchiusi tra apici
7  print(5)
8  print("L'area e'") # le stringhe vanno racchiuse a
9                     # scelta tra apici o virgolette
10 print(10*5)
```

```
Dato un rettangolo di lati
10
5
L'area e'
50
```

- Vediamo come questo script può essere migliorato

Le variabili

- Necessità di algoritmi generici (algoritmi in grado di compiere più volte le stesse operazioni ma lavorando su dati diversi)
- Le variabili lo rendono possibile
- Analogia con il calcolo letterale:

$$\left.\begin{matrix} 5^2 + 3 * 5 \\ 16^2 + 3 * 16 \end{matrix}\right| \rightarrow Y^2 + 3 * Y$$

- Variabile: nome assegnato ad una cella di memoria
- Le variabili permettono di costruire algoritmi la cui esecuzione può essere ripetuta più volte su dati diversi

- Lo script precedente può essere riscritto in questo modo ...

```
a=10 # a e' una variabile
b=5  # b e' una variabile
print("Dato un rettangolo di lati")
print(a)
print(b)
print("L'area e'")
area = a*b # area e' un'altra variabile
print(area)
```

- e l'output dell'esecuzione è sempre

```
Dato un rettangolo di lati
10
5
L'area e'
50
```

Nomi di Variabili

- Regole per i nomi di variabili
  1. Possono contenere solo lettere, numeri e il simbolo '_' (altri simboli o caratteri di punteggiatura non sono ammessi)
  2. Devono sempre iniziare con una lettera
     - tot → ok
     - somma1 → ok
     - 12esimo_giorno → NO
  3. E' permesso usare lettere sia maiuscole sia minuscole. *Numero*, *NUmEro* e *numero* sono considerate variabili diverse dall'interprete (*case sensitive*)
  4. Non possono essere usati "nomi riservati": print, ... (vedremo più avanti altri esempi)
  5. I nomi delle variabili possono essere lunghi quanto si desidera es., *totaleVenditeMese, i, contatore*, ...
- I programmatori generalmente scelgono dei nomi significativi per le loro variabili, documentando così a che cosa servono

Naming Conventions

- Naming conventions (convenzioni suggerite per scegliere i nomi delle variabili)
  - Non sono obbligatorie
  - Facilitano la comprensione del codice
  - In progetti complessi si dimostrano *molto* utili
- Esempi di naming conventions
  - variabili: nomiConInizialiInterneMaiuscole
  - costanti: TUTTOMAIUSCOLO
    - PI
    - ENEPERO

Dichiarazione delle variabili

- Come si *creano* (dichiarano) le variabili?
- Dichiarare una variabile: informare l'interprete che si intende usare una variabile
- Per dichiarare una variabile in python basta effettuare un assegnamento, es.

```
1 totale=0 # l'operatore di assegnamento e' =
2 cognome="Rossi" # Le stringhe vanno scritte tra virgolette,
3                 # in questo modo la stringa Rossi non viene
4                 # confusa con una variabile
5 studente=cognome
6 nome='Mario' # Al posto delle virgolette possono essere
7              # usati gli apici singoli
8 PI=3.14 # si usa il . per separare la parte intera
9         # dalla parte decimale
```

Tipo di una variabile

- Una variabile oltre ad *ospitare* un valore, contiene anche informazioni sul *tipo* di valore ospitato. Alcuni esempi:

```
1 numAutomobili = 100
2 print(numAutomobili)
3 print(type(numAutomobili))
```

```
100
<type 'int'>
```

```
4 PI=3.14
5 print(PI)
6 print(type(PI))
```

```
3.14
<type 'float'>
```

```
7 cognome = "Rossi"
8 print(cognome)
9 print(type(cognome))
```

```
Rossi
<type 'str'>
```

```
10 lettera = 'a'
11 print(lettera)
12 print(type(lettera))
```

```
a
<type 'str'>
```

Operazioni e tipi di variabili

- Il tipo di una variabile stabilisce le operazioni ammesse sui dati e la semantica (comportamento):

```
1 a=5 # tipo int
2 b=5 # tipo int
3 print(a+b)
```

```
10
```

```
7 c=0.3 # tipo float
8 d=8 # tipo int
9 print(c+d)
```

```
8.3
```

```
4 x='Mario' # tipo str
5 y='Rossi' # tipo str
6 print(x+y)
```

```
'MarioRossi'
```

```
10 z='Numero' # tipo str
11 k=1        # tipo int
12 print(z+k)
```

Cosa viene visualizzato?

```
Traceback ... line 3 ...
TypeError: cannot concatenate 'str' and 'int' obj.
```

- L'operatore +, ha una semantica (effetto, comportamento) diverso a seconda del tipo di variabili a cui è applicato

Non è richiesta la dichiarazione esplicita di tipo

- Python non richiede di dichiarare il tipo delle variabili, a differenza di altri linguaggi (C, Java)
- In Python, ogni variabile nel corso del tempo può ospitare tipi di dati diversi
- Dopo un assegnamento (dietro le quinte) l'interprete Python cerca di determinare il tipo della variabile in base al suo contenuto. Esempio:

```
1 i=100 # tipo int
2 print(type(i)) # int
3 i=10.2
4 print(type(i)) # float, ora il tipo di i e' cambiato
```

- Nota: la variabile *i* assume tipi diversi nel corso della sua esistenza

Tipi elementari (di variabili) in Python

- string

```
1 var1='Arturo "il giullare"'
2 var2="e' arrivato"
```

E' possibile usare sia gli apici singoli ', sia i doppi apici per aprire e chiudere una stringa

- int

```
3 i=10
4 totale=1000
```

- float

```
5 PI=3.14
```

- bool

```
6 condizione=True
7 risposta=False
```

- . . . (e tanti altri)
- E' possibile creare nuovi tipi di dati, vedremo più avanti come

Tipi e operazioni ammesse

- int, float
  - tutte le operazioni aritmetiche: $+, -, *, /, **$ (qs. ultimo è l'elevamento a potenza)
- string
  - $+$ (con la semantica di *unione*)
  - $*$ (con la semantica di *ripetizione*)
  - vedremo altre operazioni più avanti

```
1 saluto='ciao'
2 nome="Mario"
3 messaggio=saluto+'  '+nome
4 print(messaggio)
```

```
ciao  Mario
```

```
5 saluto='ciao_'
6 print(saluto*3)
```

```
ciao_ciao_ciao_
```

Versioni di Python

- Esistono due (macro) versioni di python in questo momento
  - Python 2
  - Python 3
- Come si individua la versione di Python?
- Quando avvierete l'interprete, nella prima riga troverete scritto

```
Python 2.7.10 (default, ...)
...
Type "help", "copyright", ...
>>>
```

  - I 3 numeri in 2.7.10, sono rispettivamente da sinistra verso destra, la major, minor e micro version. La major version ci dice se abbiamo a che fare con Python 2 (come in qs caso) o Python 3
  - Più avanti vedrete i diversi modi con cui l'interprete può essere avviato

Quale versione usiamo in questo corso?

- *Python 2*, il 1/gen/2020 ha raggiunto la EOL (End Of Life). Cosa significa?
  - Gli strumenti esistenti continuano a funzionare, ma non saranno più supportati dalla community di sviluppatori che gestisce Python
  - Non ci saranno più miglioramenti, bug fixing o nuove versioni di Python 2
- In questo corso faremo riferimento a Python 3
  - Per il vostro livello (principianti ) le differenze tra Py2 e Py3 sono poche
  - Vi indicheremo cosa cambia tra Python 2 e Python 3 quando incontreremo comandi con differenze significative
  - Tuttavia, dovete imparare ad usare entrambe le varianti del linguaggio. C'è in giro ancora tanto codice scritto per Python 2

Python 2

```
1 a=7
2 b=2
3 print(a/b)
```

```
3
```

```
4 c=4.5
5 d=3
6 print(c/d)
```

```
1.5
```

```
7 e=7.0
8 f=2
9 print(e//f)
```

```
3.0
```

Python 3

```
10 g=7
11 h=2
12 print(g/h)
```

```
3.5
```

```
13 i=7
14 l=2
15 print(i//l)
```

```
3
```

- Qual è la regola di funzionamento di / e //? In Py2 e Py3?

/ e // in Py2 e Py3

- **Divisione intera**: la parte intera della divisione tra 2 numeri (la parte decimale, se presente, è eliminata o azzerata)
- **Divisione reale**: il risultato della divisione tra due numeri (sia la parte intera sia la parte decimale)

In Python 2

- il risultato di x/y dipende dai tipi di x e y
  - se x e y sono entrambi interi, il risultato è la divisione intera
  - se almeno uno dei due tra x e y è un float, il risultato è la divisione reale
- il risultato di $x//y$ è sempre il risultato della divisione intera

In Python 3

- il risultato di z/k è sempre il risultato della divisione reale tra z e k, es. $5/2 = 2.5$
- il risultato di $z//k$ è sempre il risultato della divisione intera tra z e k. Es. $5//2 = 2$, $5.0//2 = 2.0$

Conversioni forzate di tipi

- Le funzioni int(), float(), str() possono essere usate per forzare la conversione di una variabile in uno specifico tipo

Esempio

```
1  a="10" # <type 'str'>
2  b=int(a)
3  print('*')
4  print(b)
5  print(type(b))
6  c=float(a)
7  print('**')
8  print(c)
9  print(type(c))
```

```
*
10
<type 'int'>
**
10.0
<type 'float'>
```

Attenzione

```
10 d="fila" # <type 'str'>
11 e=int(d)
12 print(e)
```

Che cosa viene visualizzato?

```
ValueError: invalid literal
for int() with base 10: 'fila'
```

```
12 f="dieci,cinque" # <type 'str'>
13 g=float(f)
```

```
ValueError: could not
convert string to float:
dieci,cinque
```

Conversioni di tipo, esempi

Uso corretto

```
1 msg='Sei il visitatore '
2 n=71
3 print(msg+str(n))
```

```
Sei il visitatore 71
```

```
4 ad1=10
5 ad2="5" # <type 'str'>
6 print(ad1 + int(ad2))
```

```
15
```

Domanda ...

```
7 msg='Sei il visitatore '
8 n=71
9 print(msg+n)
```

```
TypeError: cannot
concatenate
'str' and 'int'
objects
```

```
10 ad1=10
11 ad2="5" # <type 'str'>
12 print(ad1 + ad2)
```

Cosa viene visualizzato?

```
TypeError: unsupported
operand type(s) for +:
'int' and 'str'
```

Conversioni di tipo, esempi 2

Uso corretto

```
1 ad3=10.23
2 ad4="7.4" # <type 'str'>
3 print (ad3 + float(ad4))
```

```
17.63
```

Cosa sarebbe successo se ...

```
1 ad3=10.23
2 ad4="7.4" # <type 'str'>
3 print (ad3 + ad4)
```

```
TypeError: unsupported
operand type(s) for +:
'float' and 'str'
```