Corso Python

Lezione A01 – Introduzione a Python

Alberto Montresor

Università di Trento

2021/01/25

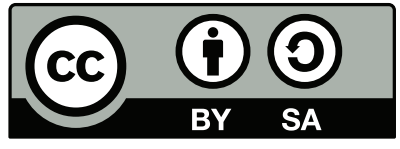


Table of contents

Materiali

http://tiny.cc/python-unitn

- Slide
- Libri

Date

- 15/1: Introduzione
- 19/1: Strutture di controllo (per non informatici)
- 26/1: Slicing, funzioni, programmi
- 5/2: OOP
- 9/2: Numpy,Matplotlib,Pandas
- 12/2: Argomenti avanzati

Un po' di storia

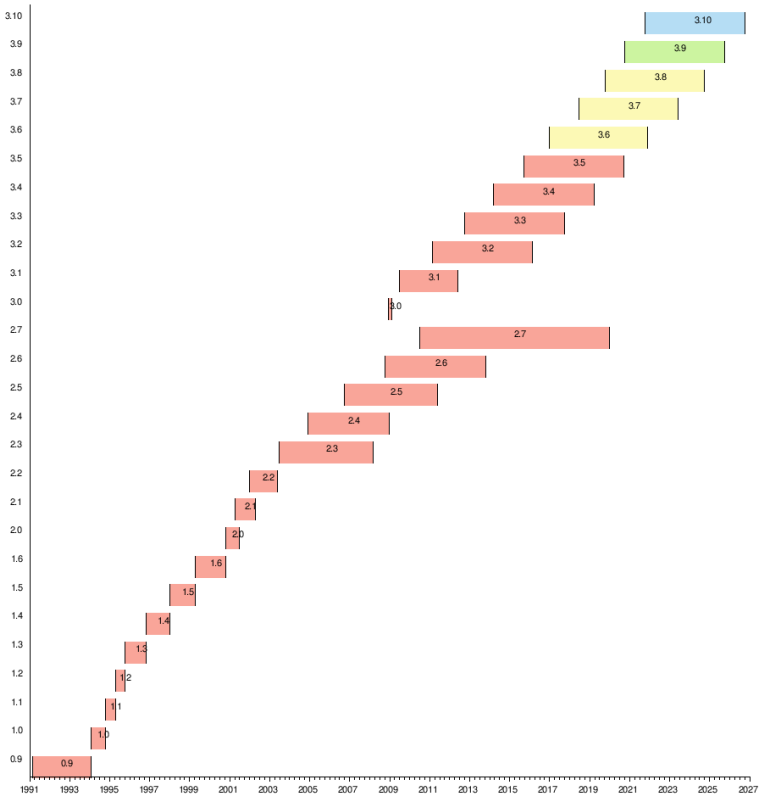
Python è un linguaggio di programmazione di alto livello ampiamente utilizzato per programmazione general-purpose, creato da Guido van Rossum.

1991	Python 1.0. Obsoleta
1995	Guido van Rossum procolamato BDFL (Benevolent Dictator for Life)
2000	Python 2.0 – 2.7.17 (2019/10)
2008	Python 3.0 - 3.9.1 (2020/12)
2018	Guido stepped down from BDFL
2020	End-of-life – 2.7.18 (2020/04)

https://en.wikipedia.org/wiki/Guido_van_Rossum#/media/File:Guido_van_Rossum_OSCON_2006.jpg

Discussione: 2.x vs 3.x

Versioni



- $2.x$ vs $3.y$: codice $2.x$ non funziona in $3.y$
- $3.x$ vs $3.y, x < y$: codice $2.x$ funziona in $3.y$, non è detto il viceversa
- $3.x.y$ vs $3.x.z$: compatibilità piena
- `https://www.python.org/doc/versions/`
  - Esempio: 3.5 type annotations
  - Esempio: 3.6 new dict implementation

`https://en.wikipedia.org/wiki/History_of_Python`

The *pythonic* way

Python ha una filosofia di progettazione che enfatizza la leggibilità del codice

Try: `import this`

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

...

https://inventwithpython.com/blog/2018/08/17/the-zen-of-python-explained/

Hello World: Di quanti concetti avete bisogno?

Java

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

C

```
#include <stdio.h>
int main()
{
  printf("Hello   World!");
}
```

Python

```
print("Hello, World")
```

Qualcosa di più di Hello World

C

```c
# include <stdio.h>
int main(int argc, char *argv[])
{
  char *colors[] = { "red", "green", "blue", };
    for (int i = 0; i < 3; ++i) {
      printf("%s\n", colors[i]);
    }
}
```

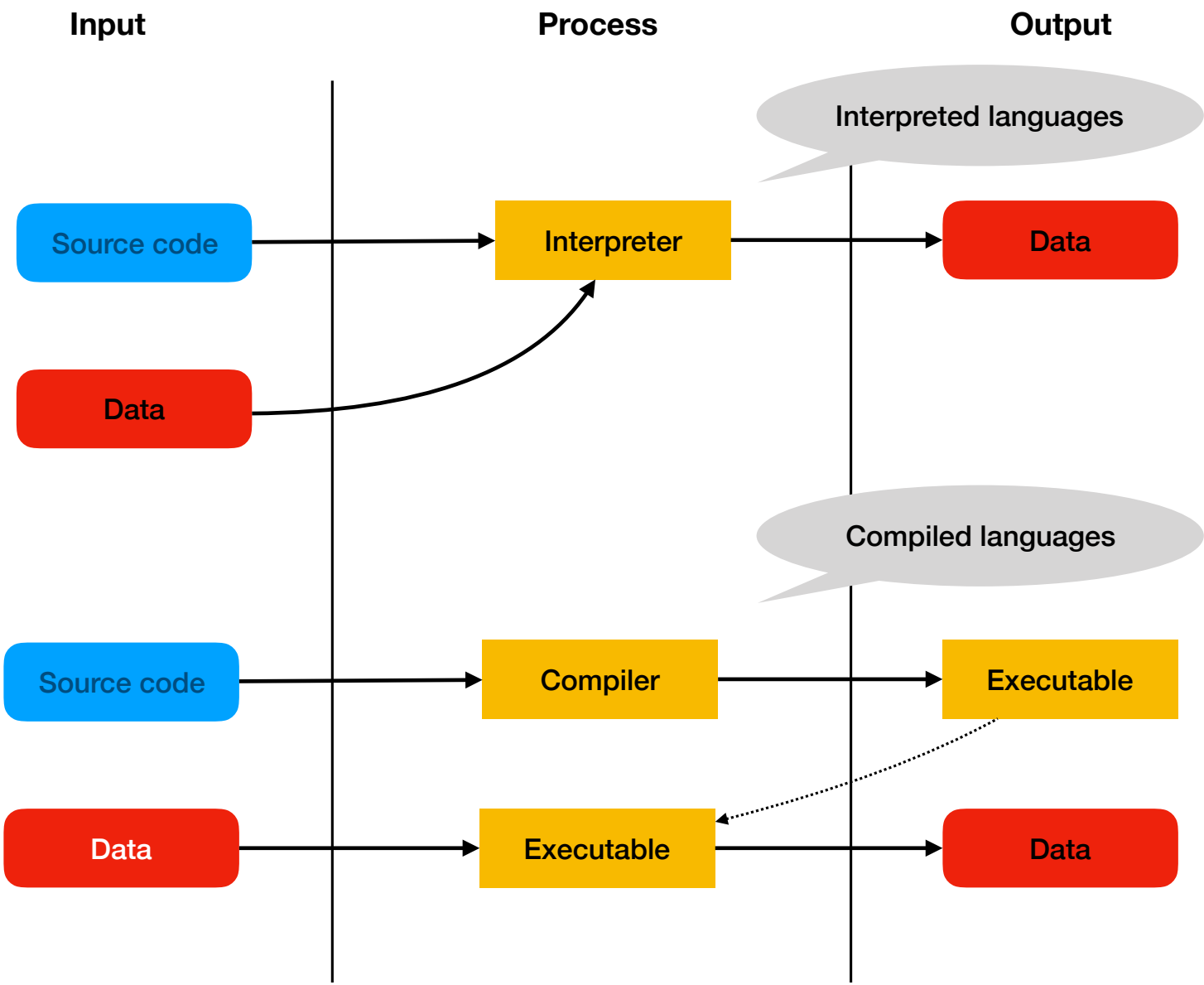
Python

```python
colors = ["red", "green", "blue"]
for color in colors:
    print(color)
```

Python vs altri linguaggi: quale scelta?

`http://disi.unitn.it/~montreso/python/cvspython.pdf`

Linguaggi interpretati vs compilati



Prompt Python vs programmi Python

Prompt Python

```
grannysmith:~ montreso$ python
Python 3.5.3 |Anaconda 4.4.0 (x86_64)| (default, Mar  6 2017, 12:15)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> 2+3
5
>>>
```

Prompt Python vs programmi Python

Programma Python: `firstprogram.py`

```
print("This is my first program")
print(2+3)
```

Esecuzione

```
grannysmith:~ montreso$ python firstprogram.py
```

```
This is my first program
5
```

Cosa utilizzare?

Facili

- Thonny: `https://thonny.org/`
- Colab: `https://colab.research.google.com/`

Linguaggio

- Python base: `https://www.python.org/downloads/`
- Anaconda:
  `https://www.anaconda.com/products/individual`

IDE

- IDLE (Con distribuzione Python base)
- Visual Studio Code (Con Anaconda o standalone)
- Un gazillione di altre possibilità....

Table of contents

Oggetti

Oggetto

Un oggetto è rappresentato da un dato (ad esempio, un numero, testo scritto, una collezione, una funzione, una classe) che un programma manipola.

Gli oggetti sono composti da:

- tipo: il dominio del dato rappresentato dall'oggetto
- valore: il dato stesso

Esempi

Nei programmi visti prima:

- 2, 3 e 5 sono oggetti integer
- "This is my first program" è un oggetto string

Tipi di dato Python (2.x)

Tipo	**Significato**	**Dominio**	**Mutabile?**
`bool`	Booleano	`True`, `False`	No
`int`	Intero	$\{-2^{-63}, \ldots, 2^{63}-1\}$	No
`long`	Intero	$\mathbb{Z}$	No
`float`	Razionale	$\mathbb{Q}$ (più o meno)	No
`str`	Stringa	Text	No
`list`	Sequenza	Collezione di oggetti	Yes
`tuple`	Sequenza	Collezione di oggetti	No
`dict`	Dizionario	Associazione chiave-valore	Yes

Tipi di dato Python (3.x)

Tipo	**Significato**	**Dominio**	**Mutabilie?**
`bool`	Booleano	`True`, `False`	No
`int`	Intero	$\{-2^{-63}, \ldots, 2^{63}-1\}$ $\mathbb{Z}$	No
~~`long`~~	~~Intero~~	~~$\mathbb{Z}$~~	~~No~~
`float`	Razionale	$\mathbb{Q}$ (più o meno)	No
`str`	Stringa	Text	No
`list`	Sequenza	Collezione di oggetti	Yes
`tuple`	Sequenza	Collezione di oggetti	No
`dict`	Dizionario	Associazione chiave-valore	Yes

Variabili

Variabile

- Le variabili sono reference ad oggetti
- Le potete vedere come nomi degli oggetti a cui si riferiscono
- Il tipo di una variabile è dato dal tipo dell'oggetto a cui si riferisce

Esempio

```
pi = 3.1415926536
print(pi)
print(type(pi))
```

```
3.1415926536
<class 'float'>
```

Scelta dei nomi di variabili

- I nomi delle variabili sono una scelta del programmatore
- Devono essere il più significativi possibili

Nomi di variabili

Regole per creare nomi di variabili

- Possono contenere solo lettere, numeri e underscore `a-z,A-Z,0-9,_`
- Non possono essere iniziare con un numero
- Alcune parole sono keyword del linguaggio
- Alcune parole sono funzioni primitive del linguaggio, non vanno ri-usate (ad esempio max)

Esempi – Sintassi non valida

```
76trombones = "big parade"
more$ = 1000000
class = "Computer Science 101"
```

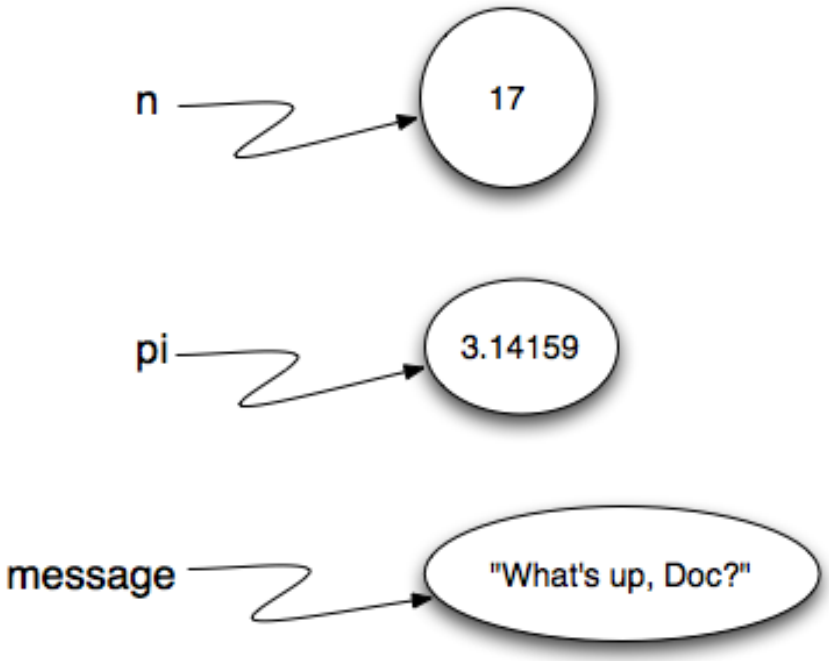
Parole riservate

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Variabili e memoria

Gli oggetti vivono nelle celle di memoria del computer e le variabili sono riferimenti a queste celle

```
message = "What's up, Doc?"
n = 17
pi = 3.14159
```



https://runestone.academy/runestone/static/thinkcspy/SimplePythonData/Variables.html

Definizione variabili

Le variabili devono essere inizializzate prima di essere utilizzate

```
print(r*r*3.14)
r = 2
```

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

Aggiornare variabili

Uno delle più comuni forme di riassegnamento è un update dove il nuovo valore dipende da quello vecchio.

```
x = 6          # initialize x
print(x)                                  6
x = x + 1      # update x
print(x)                                  7
```

https://runestone.academy/runestone/static/thinkcspy/SimplePythonData/UpdatingVariabili.html

Tipi multipli!

Le variabili possono avere tipi multipli durante la loro vita
Questo non è normalmente una buona idea!

```
var = 3
print(var*2)
print(type(var))
```

```
6
<class 'int'>
```

```
var = 3.1415926536
print(var*2)
print(type(var))
```

```
6.2831853072
<class 'float'>
```

```
var = "3.14"
print(var*2)
print(type(var))
```

```
3.143.14
<class 'str'>
```

Tipi numerici

Come scrivere valori numerici

```
x = 10          # Integer
y = 123.45      # Float
z = 1.2345e2    # Float
```

Operatori numerici

+,-,*	sum,difference,product
/	division
//	integer division
%	remainder
**	power

Differenze fra 2.x e 3.x

```
# Python 2.x
>>> 2/3
0
>>> 2//3
0
```

```
# Python 3.x
>>> 2/3
0.6666666666666666
>>> 2//3
0
```

Conversione di tipo

Conversione di tipo

- Conversioni automatiche
- Da `float` a `int`: `int(2.5)`
- Da `int` a `float`: `float(2)`

```
print(1+1.0)
print(type(1+1.0))
print(int(3.0))
print(float(1))
print("The value is " + str(1))
```

```
2.0
<class 'float'>
3
1.0
The value is 1
```

Valori booleani (`True`, `False`) e operatori

Tabelle di verità

and	False	True
False	False	False
True	False	True

or	False	True
False	False	True
True	True	True

a	not a
False	True
True	False

Tabelle di verità

a	b	a and b	a or b
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

Confronti

Confronti

`a == b`	`True` if and only if $a = b$
`a != b`	`True` if and only if $a \neq b$
`a < b`	`True` if and only if $a < b$
`a > b`	`True` if and only if $a > b$
`a <= b`	`True` if and only if $a \leq b$
`a >= b`	`True` if and only if $a \geq b$

Boolean: Short Circuiting

Annihilator

Se la prima parte di un'espressione booleana decide il risultato finale, il la valutazione del resto può essere saltata (short circuit)

```
False and a == False
```

```
True or a == True
```

Esempio

```
a = 0
b = 2
print(a > 0 and 12/a >= 0)
print(b > 0 and 12/b == 6)
print(a == 0 and 12/a >= 0)
```

Boolean: Short Circuiting

Annihilator

Se la prima parte di un'espressione booleana decide il risultato finale, il la valutazione del resto può essere saltata (short circuit)

```
False and a == False
```

```
True or a == True
```

Esempio

```
a = 0
b = 2
print(a > 0 and 12/a >= 0)
print(b > 0 and 12/b == 6)
print(a == 0 and 12/a >= 0)
```

```
False
True
ZeroDivisionError
```

Precedenza operatori

`**`	Potenza (Precedenza più alta)
`+,-`	Più, meno unario
`* / // %`	Moltiplicazione, divisione, divisione intera, modulo
`+ -`	Somma e sottrazione
`<= < > >=`	Operatori di confronto
`== !=`	Operatori di uguaglianza
`not or and`	Operatori logici (Precedenza più bassa)

Esempio

```
2+3*4**2 == 23+3**3 and 3*-1**2+7 != 10
```

Precedenza operatori

`**`	Potenza (Precedenza più alta)
`+,-`	Più, meno unario
`* / // %`	Moltiplicazione, divisione, divisione intera, modulo
`+ -`	Somma e sottrazione
`<= < > >=`	Operatori di confronto
`== !=`	Operatori di uguaglianza
`not or and`	Operatori logici (Precedenza più bassa)

Esempio

```
2+3*4**2 == 23+3**3 and 3*-1**2+7 != 10

(2+(3*(4**2))) == (23+(3**3)) and (((3*(-(1**2)))+7) != 10)
```

Espressioni vs Istruzioni

Espressioni

- Un'espressione è una combinazione di valori, variabili, operatori, e chiamate a funzioni
- Quando si scrive un'espressione sul prompt, l'interprete la valuta

Esempio

```
>>> (2+3)*5
25
>>> 42
42
>>> n = 17
>>> n+25
42
```

Espressioni vs Istruzioni

Istruzioni

- Un'istruzione è un'unità di codice che ha un effetto, come modificare lo stato della memoria o produrre un output
- Le espressioni non sono istruzioni

Esempio

```
a=(2+3)*5
a*2
print(a)
```

```
25
```

Assegnamento

Istruzione di assegnamento

Istruzione	Prima	Dopo
`a = 3`	?	3

Istruzioni composte di assegnamento

Istruzione	Prima	Dopo
`a += 3`	3	6
`a -= 3`	6	3
`a *= 3`	3	9
`a /= 3`	9	3
`a **= 3`	3	27

Funzioni e metodi

Funzioni

Una funzione prende zero o più oggetti come input (argomenti), esegue qualche operazione su di essi, e restituisce zero o più oggetti (i suoi risultati).

Funzione

Si invoca una funzione scrivendo il nome della funzione, seguito da parentesi contenenti oggetti per ognuno dei parametri. È possibile raccogliere il risultato di una funzione

```
result = f(par1, par2, ...)
```

(Alcune) funzioni built-in

`abs()`	Ritorna il valore assoluto di un numero
`max()`	Ritorna il massimo di due o più valori, liste, insiemi, etc.
`min()`	Ritorna il minimo di due o più valori, liste, insiemi, etc.
`round()`	Arrotonda un numero in virgola mobile al numero desiderato di cifre
`print()`	Stampa gli argomenti

Esempio

```
val = abs(-3)
print(val, max(2,3), round(3.1415926536, 2))
```

```
3 3 3.14
```

Input semplice

`input()`

Si può usare la funzione built-in `input()` per leggere input dall'utente. Non ha parametri di input e restituisce una singola stringa. La stringa deve essere convertita in un numero intero, se necessario.

Esempio

```
val = int(input())
print(val*val)
```

Metodi

Metodi

Un metodo è esattamente come una funzione, a parte che è fornito da un certo tipo ed è applicato ad una specifica instanza di quel tipo.

Esempio

Il tipo `string` fornisce un metodo `upper()` che restituisce una versione upper-case della stringa originale. La stringa originale non viene modificata.

```
s = "hello world"
print(s)
print(s.upper())
print(s)
```

```
hello world
HELLO WORLD
hello world
```