Programming Lab

Parte 2

Intro su Python: tipi dati, costrutti, funzioni, moduli, be pythonic.

Stefano Alberto Russo

Python

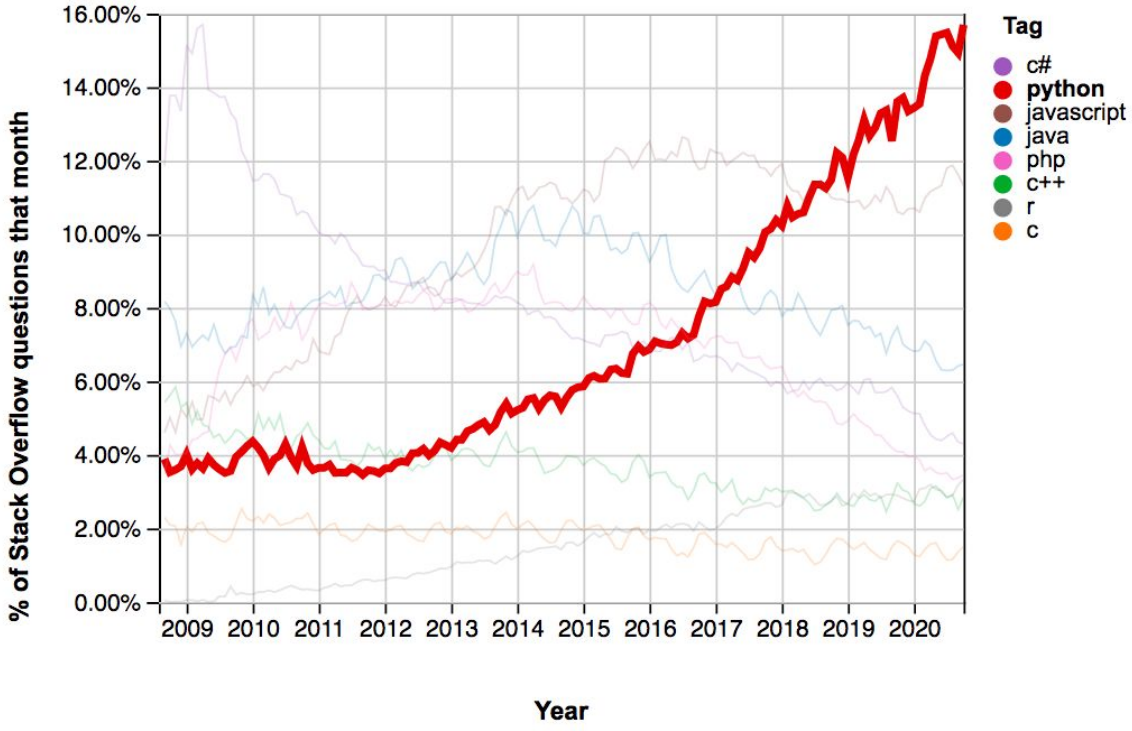
Sarà il linguaggio di riferimento del laboratorio

→ *Versione 3, in particolare >= 3.6*

E’ un linguaggio che nasce a oggetti

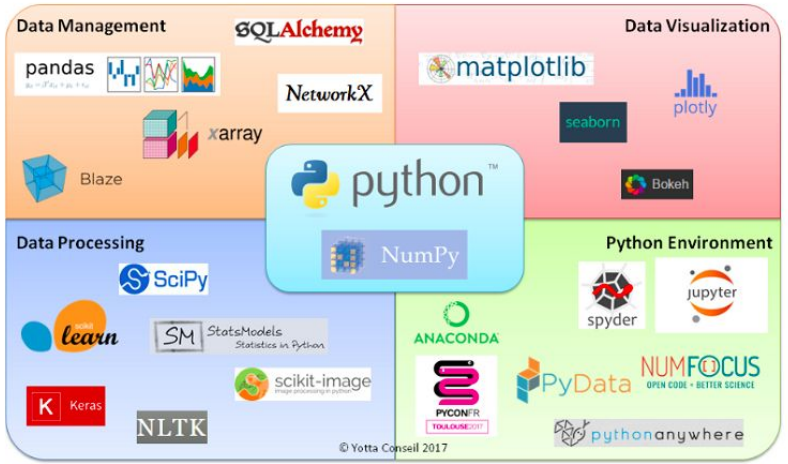
Vi verrà spiegato anche al corso programmazione, focalizzandosi sulla teoria ed in particolare quella relativa ai linguaggi ad oggetti.

Perchè Python?



Perchè Python?

- E’ un linguaggio semplice, intuitivo e potente
- Facilmente comprensibile, quasi pseudo-codice
- E’ il linguaggio della Data Science
- Ha un ecosistema di software per il calcolo scientifico / statistico invidiabile



The Python data science ecosystem (source: Yotta Conseil)

Pseudo Codice (parentesi)

Lo pseudo-codice sarà il vostro migliore amico, ancora prima di Python.

Fare pseudo-codice vuol dire scrivere, in linguaggio naturale (Italiano/Inglese), quello che dovrebbe fare il programma, con un minimo di sintassi.

Non ci si focalizza sui dettagli nello pseudo-codice!

Ovvero, non ci si focalizza sul ***come***, ma sul ***cosa*** fare.

Pseudo Codice (parentesi)

Esempio: trova i numeri in una lista minori di 5 e stampali

```
data una lista di numeri;

per ogni elemento della lista:
    se l’elemento è minore di 5:
        stampa l’elemento
```

Python: un codice minimale

Esempio: trova i numeri in una lista minori di 5 e stampali

```
number_list = [13,12,34,4,51,8,27,18]

for item in number_list:
    if item < 5:
        print(item)
```

Python: un linguaggio interpretato

Python è un linguaggio interpretato, non deve essere tradotto in linguaggio macchine come per il c (ovvero, compilato), ma viene eseguito “come sta”

Essendo un linguaggio interpretato, ha anche un interprete interattivo, che potete (e dovreste) usare ogni qualvolta vogliate testare delle cose in rapidità

```
ste@Stes-MacAir:ProgrammingLab (master) $ python
Python 2.7.10 (default, Oct 23 2015, 19:19:21)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+2
4
>>> for item in [1,2,3]: print item
...
1
2
3
>>>
```

Python: una premessa

In questo corso di laboratorio viene assunto che siate già un minimo familiari con i costrutti e gli operatori di base di un linguaggio di programmazione. Per esempio:

- Assegnazione di variabili e stampa a schermo (`=`, `print`)
- Operatori condizionali (`if`, `else`)
- Operatori artimetici (`+`, `-`, `*`, etc.)
- Operatori logici (`and`, `or`)
- Operatori di confronto (`==` , `<`, `>`, etc.)
- Cicli (`for`, `while`, etc.)

Tutorial di supporto extra: https://www.w3schools.com/python/default.asp

Python: hello world!

Nella lezione precedente abbiamo visto un tipico esempio di “ciao mondo” che si usa come esempio in programmazione.

```
print('Hello world!')
```

L’istruzione **print** può essere usata assieme alla formattazione delle stringhe per includere i valori delle variabili che vogliamo stampare a schermo:

```
print('Valore 1: {}, valore 2: {}'.format(my_var_1, my_var_2))
```

..che vuol dire che Python formatterà la stringa da stampare andando a sostituire alle doppie graffe prima la variabile “`my_var_1`”, poi la variabile “`my_var_1`”.

Python: tipi dati

Python non richiede di definire esplicitamente i tipi dati, ovvero una variabile può cambiare contenuto e non deve esserne definito il tipo.

Ecco i tipi principali:

Con i cancelletti si inseriscono i commenti nel codice.
Commentate il più possibile quello che fate!

```
my_var = 1          # Esempio di variabile tipo intero
my_var = 1.1        # Esempio di variabile tipo floating point
my_var = 'ciao'     # Esempio di variabile tipo stringa
my_var = True       # Esempio di variabile tipo booleano
my_var = None       # Il “niente” si rappresenta con il “None”
```

Python: accedere alle stringhe per posizione

Si può accedere all'elemento i-esimo di una stringa così:

```
mia_stringa[2]   # Terzo carattere della stringa
mia_stringa[-1]  # Penultimo carattere della stringa
```

Python: lo “slicing” delle stringhe

Si può tagliare una fetta (“to *slice*”) di una stringa così:

```
mia_stringa[0:50]   # Dal primo al cinquantesimo carattere
mia_stringa[30:50]  # Dal trentunesimo al cinquantesimo carattere
mia_stringa[0:-1]   # Dal primo al penultimo carattere
```

Python: operatori di confronto

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Python: operatori aritmetici

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y

Python: operatori logici

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Python: le liste

Uno dei tipi dati di Python un po' più evoluti è la lista.

```
my_list = [1,2,3]                    # Lista di numeri
my_list = ['Marco', 'irene', 'paolo']  # Lista di stringhe
```

La lista ci introduce anche agli operatori di appartenenza:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

```
if 'Marco' in my_list ...
```

Python: i dizionari

Un altro tipo dati evoluto di Python sono i “dizionari”. Sono coppie chiave-valore:

```
my_dict = {'Trieste': 34100, 'Padova': 35100}   # stringa -> numero
my_dict = {34100: 'Trieste', 35100: 'Padova'}   # numero -> stringa
my_dict = {'Trieste': 'TS', 'Padova': 'PD'}     # stringa -> stringa
```

Si accede ai valori di un dizionario così (anche in assegnazione):

```
print('CAP di Trieste: {}'.format(my_dict['Trieste'])
```

Valgono gli operatori di appartenenza della slide precedente, agiscono sulle chiavi

```
if 'Trieste' in my_dict ...
```

Python: istruzioni condizionali, blocchi, indentazione

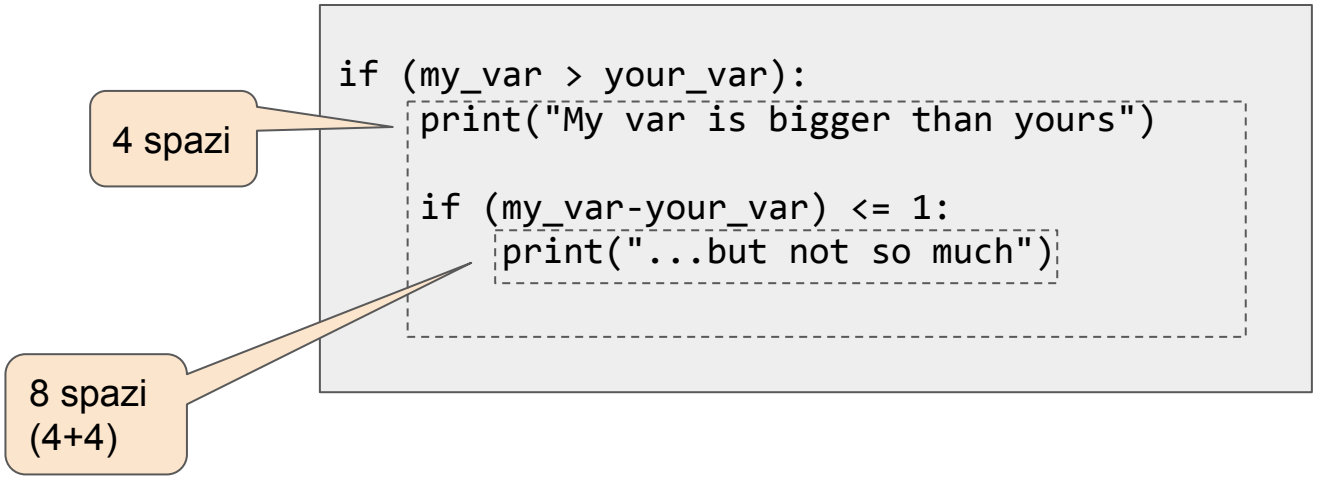
In Python valgono i soliti if-else, che ci introducono ai blocchi e l'indentazione:

```
if (my_var > your_var):
    print("My var is bigger than yours")

    if (my_var-your_var) <= 1:
        print("...but not so much")
```

Python: istruzioni condizionali, blocchi, indentazione

In Python valgono i soliti if-else, che ci introducono ai blocchi e l'indentazione:



Python: istruzioni condizionali, blocchi, indentazione

In Python condizioni aggiuntive si aggiungo con l’ “elif”

```
if (my_var > your_var):

    print("My var is bigger than yours")

    if (my_var-your_var) <= 1:
        print("...but not so much")

    elif (my_var-your_var) <= 5:
        print("...quite a bit")

    else:
        print("...a lot")
```

Python: i cicli

In Python valgono i soliti cicli for e while ma come visto nell'esempio di prima, alcune cose come cilare sugli elementi sono più facili. Esempi:

```
for item in mylist:
    print(item)
```

```
i=0
while i<10:
    print(i)
    i = i+1
```

```
for i in range(10):
    print(i)
```

```
for i, item in enumerate(mylist):
    print("Posizione {}: {}".format(i, item))
```

Python: le funzioni

Una funzione si definisce con:

```
def mia_funzione(argomento1, argomento2):
    print("Argomenti: {} e {}".format(argomento1, argomento2))
```

e si chiama con:

```
mia_funzione("Pippo","Pluto")
```

...che stamperà a schermo:

```
Argomenti: Pippo e Pluto
```

Python: le funzioni

Una funzione può anche (e in genere deve) tornare un valore:

```
def eleva_al_quadrato(numero):
    return numero*numero
```

e si chiama con:

```
eleva_al_quadrato(4)
```

...che tornera' il valore 16, che può essere poi assegnato a una variabile e stampato a schermo:

```
risultato = eleva_al_quadrato(4)
print('Risultato: {}'.format(risultato))
```

Python: le funzioni built-in

Sono funzioni sempre disponibili, un paio le abbiamo già viste:

		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

Python: lo “scope”

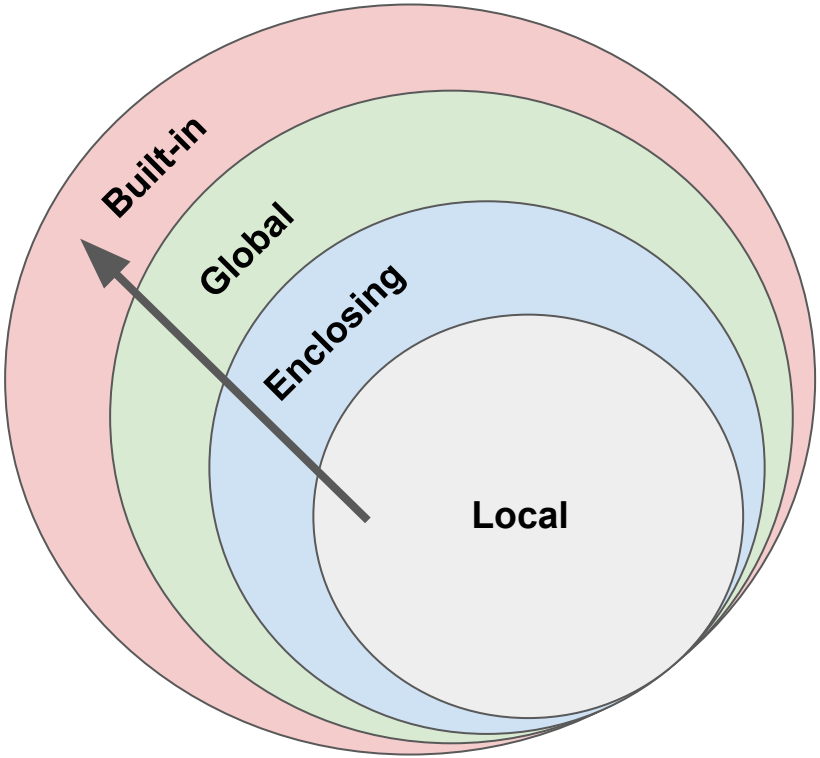
Come viene risolta (trovata) una variabile?

Python segue la regola LEGB:

- Local

  Le cose definite “dove sono”, ad esempio dentro una funzione
- Enclosing

  Ad esempio la funzione esterna se metto due funzioni una dentro l’altra, o il “corpo” esterno del programma
- Global
- Built-in



Come si scrive una buona funzione (1)

Una buona funzione agisce solo su variabili locali

Ovvero, qualsiasi cosa tratto deve “entrare” nella funzione con un argomento.

NO!

```
numero = 5

def eleva_al_quadrato():
    risultato = numero*numero
    return risultato
```

SI :)

```
def eleva_al_quadrato(numero):
    risultato = numero*numero
    return risultato
```

Come si scrive una buona funzione (2)

Una buona funzione torna sempre il suo risultato con un return()

Ovvero, non vado a modificare l'elemento che ho passato con gli argomenti!

NO!

```
risultati = []

def eleva_al_quadrato(numero, risultato):
    risultati.append(numero*numero)
```

SI :)

```
def eleva_al_quadrato(numero):
    risultato = numero*numero
    return risultato

risultati.append(eleva_al_quadrato(...))
```

Python: i moduli

In Python ci sono un sacco di funzionalità (funzioni ed oggetti) già disponibili con la cosiddetta “libreria standard”, ma che non sono “built-in”.

Questo vuol dire che non dovete installare niente per usarle, ma che dovete esplicitamente importare il modulo che le contiene.

Esempio con la radice quadrata:

```
>>> import math
>>> math.sqrt(600)
24.49489742783178
```

oppure

```
>>> from math import sqrt
>>> sqrt(600)
24.49489742783178
```

Cosa vuol dire essere “pythonici”

Iterare sugli elementi di una lista

```
my_list = ["marco", "irene", "paolo"]

for i in range(len(my_list)):
    print(my_list[i])
```

vs.

```
my_list = ["marco", "irene", "paolo"]

for item in my_list:
    print(item)
```

Cosa vuol dire essere “pythonici”

Controllare se Marco è nella lista degli studenti:

```
my_list = ["marco", "irene", "paolo"]

for i in range(len(my_list)):
    if my_list[i] == "marco":
        print("Ho trovato marco!")
```

vs.

```
my_list = ["marco", "irene", "paolo"]

if "marco" in my_list:
    print("Ho trovato marco!")
```

Un primo esercizio

Scrivete una funzione `sum_list(my_list)`
che sommi tutti gli elementi di una lista.

Poi, scaricate il vostro script Python
e testatelo su Autograding

p.s. la lista passata da sommare è vuota, la funzione deve tornare `None` (la somma di una lista vuota non è definita!)