

Programming Lab

Lezione 2

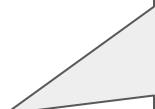
Note pratiche

Prima ora: vediamo la parte teorica (introduzione a Python)

Seconda ora: finiamo la configurazione di Repl dalla lezione precedente e facciamo un' esercizio con Python.

Alla fine di questa lezione dovete tutti sapere:

- 1) Come si usa Repl configurato con Bash
- 2) Come si fa un commit da Repl su Git
- 3) Come si esegue uno script Python dentro Repl



Queste tre cose le userete per tutto il corso e l'esame. Ci saranno delle variazioni (nuovi files, comandi leggermente diversi..) ma lo "zoccolo duro" degli strumenti è questo.

Python

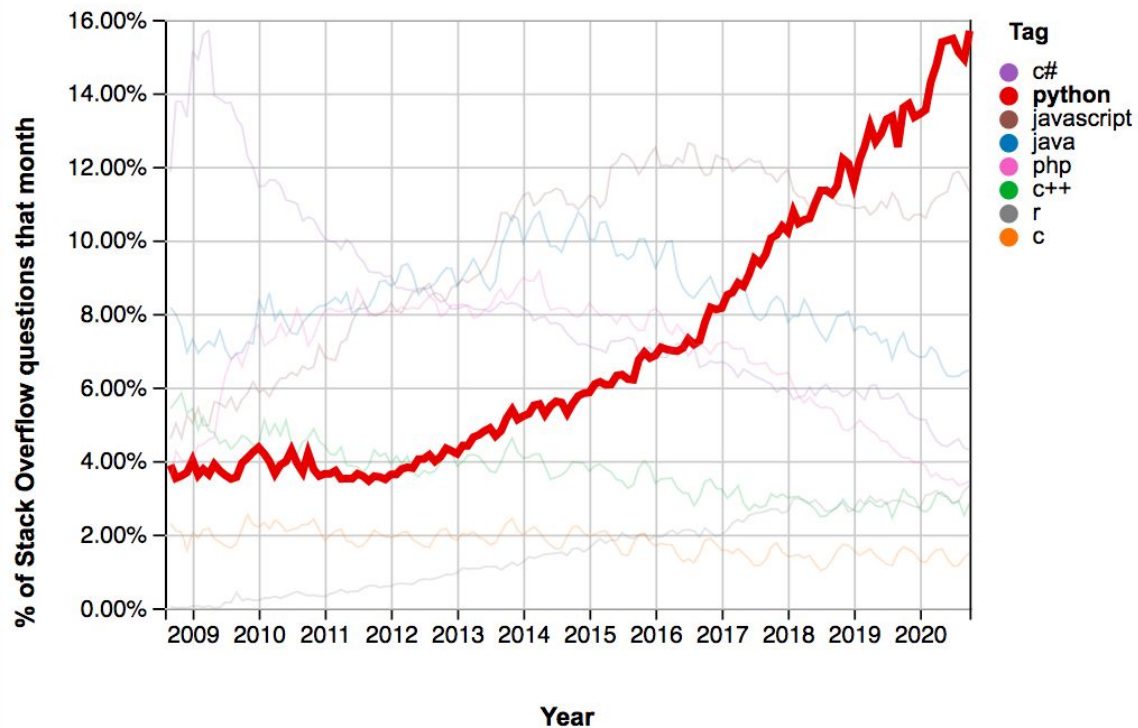
Sarà il linguaggio di riferimento del laboratorio

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.

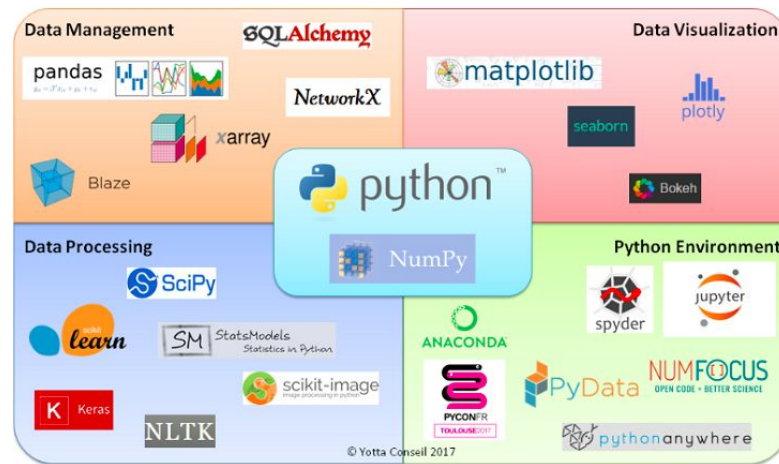


Perchè Python?



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



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 macchina come per il c (ovvero, compilato), ma viene eseguito “come sta”

Essendo un linguaggio interpretato, ha anche un interprete interattivo, che potete (e dovrete) 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:

- Operatori condizionali (if, else)
- Operatori aritmetici (+, -, *, etc.)
- Cicli (for, while, etc.)
- Operatori logici (and, or)
- Operatori di confronto (<, >, == etc.)

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

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:

```
myvar = 1      # Esempio di variabile tipo intero
myvar = 1.1    # Esempio di variabile tipo floating point
myvar = "ciao" # Esempio di variabile tipo stringa
myvar = True   # Esempio di variabile tipo booleano
```

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

Python: operatori di confronto

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

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	<code>x < 5 and x < 10</code>
or	Returns True if one of the statements is true	<code>x < 5 or x < 4</code>
not	Reverse the result, returns False if the result is true	<code>not(x < 5 and x < 10)</code>

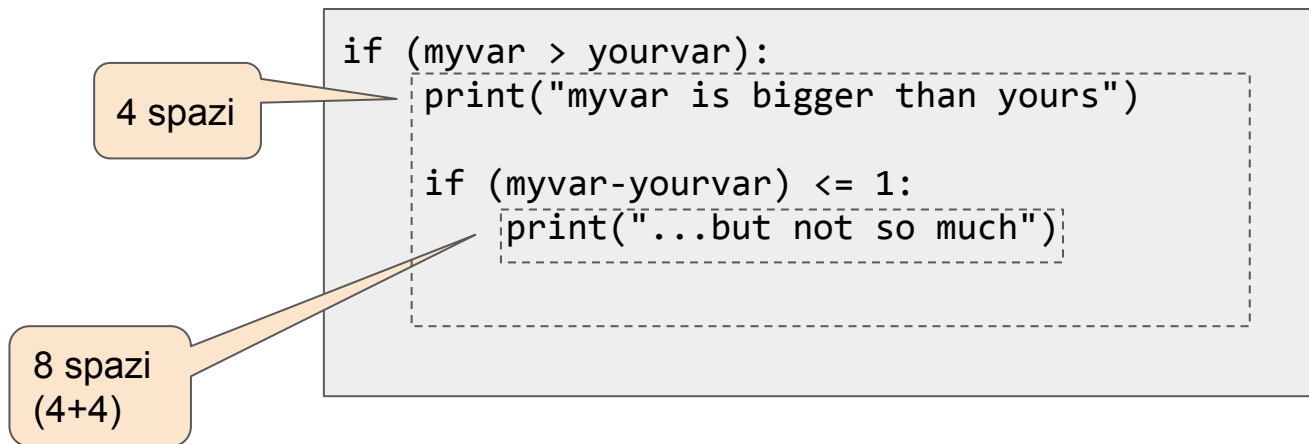
Python: istruzioni condizionali, blocchi, indentazione

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

```
if (myvar > yourvar):  
    print("myvar is bigger than yours")  
  
    if (myvar-yourvar) <= 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 aggiungono con l' "elif"

```
if (myvar > yourvar):  
    print("myvar is bigger than yours")  
  
    if (myvar-yourvar) <= 1:  
        print("...but not so much")  
    elif (myvar-yourvar) <= 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 ciclare 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: nota sul print()

Nella slide precedente c'era questo costrutto:

```
print("Posizione {}: {}".format(i, item))
```

..che vuol dire che Python formatterà la stringa andando a sostituire alle doppie graffe prima la variabile “i”, poi la variabile “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 built-in

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

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

Python: le liste

Uno dei tipi dati di Python un po' più evoluti è la lista, come abbiamo visto.

```
mylist = [1,2,3] # Lista di numeri  
mylist = ["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

Python: cosa vuol dire essere “pythonici”

Controllare se Marco è nella lista degli studenti:

```
mylist = ["marco", "irene", "paolo"]  
  
for student in mylist:  
    if student == "marco":  
        print("Ho trovato marco!")
```

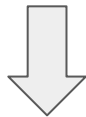
VS.

```
mylist = ["marco", "irene", "paolo"]  
  
if "marco" in mylist:  
    print("Ho trovato marco!")
```

Inizio parte pratica

Setup dell'ambiente (dalla lezione precedente)

- ~~1) Registratevi su GitHub se non lo siete già~~
- ~~2) Cretevi un repo su GitHub chiamato "ProgrammingLab"~~
- ~~3) Registratevi su repl.it~~
- ~~4) Importate il repo ProgrammingLab su repl~~
- 5) Configurare repl (4 spazi indentazione, .replit file con bash come interprete)



```
language = "bash"  
run = ""
```

Primi comandi (dalla lezione precedente)

- 1) Committiamo* il vostro .repl, prima configuriamo user e email per il repo:

```
git config user.name "John Doe"  
git config user.email johndoe@example.com
```

- 2) Creiamo uno script “intro.py” con dentro un: `print("Hello world!")`

- 3) Eseguiamo lo script (`python intro.py`)

- 4) Committiamo* anche lo script

*Reminder su come si committa con Git:

```
git add miofile  
git commit -m “commento”
```

..e adesso.. primo esercizio

Scrivete una funzione che sommi
tutti gli elementi di una lista*.

Poi, committate il file in cui l'avete scritta.

* se non sapete fare qualcosa, usate Google (o Bing o il vostro motore di ricerca preferito), è parte del corso anche che impariate a destreggiarvi!