Introduzione a Python

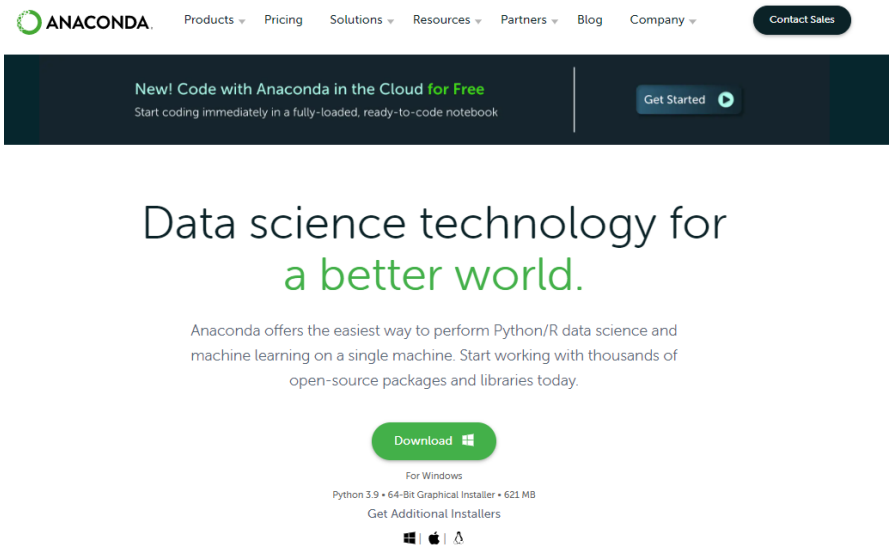
Andrea Sebastiani
andrea.sebastiani3@unibo.it

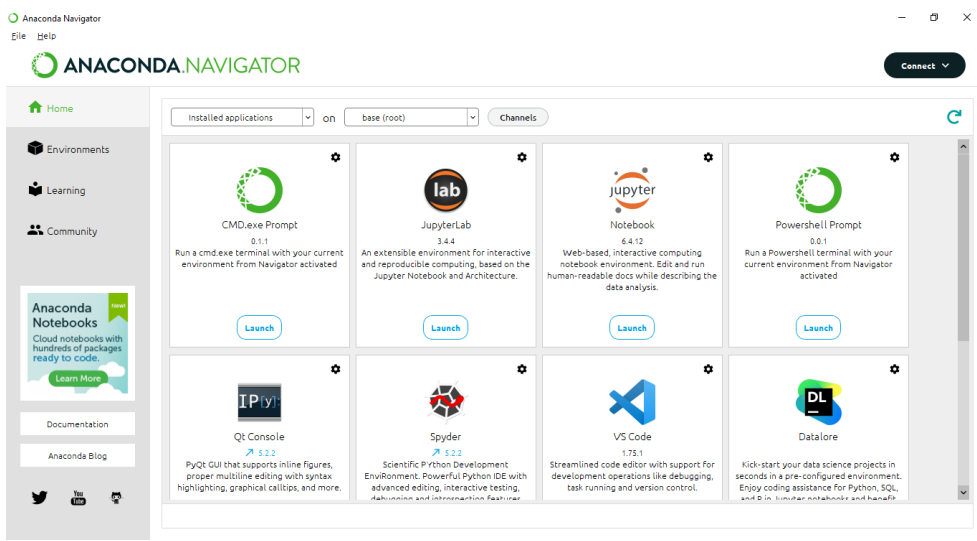
Università di Bologna

28 Settembre 2023

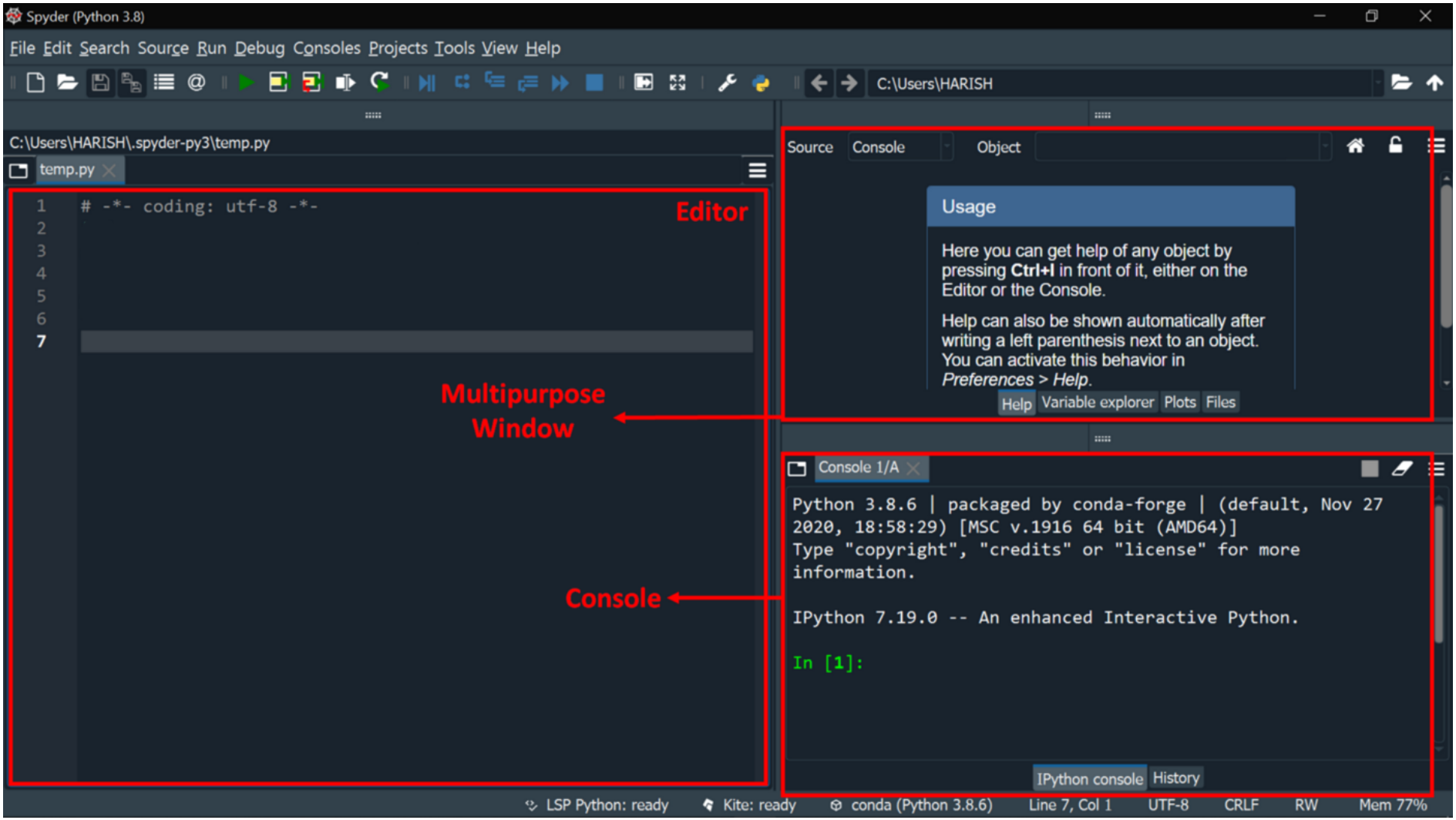
Installazione

1. Scaricare ed installare **Anaconda**
2. Aprire **Spyder** da Anaconda Navigator (o da qualche scorciatoia)



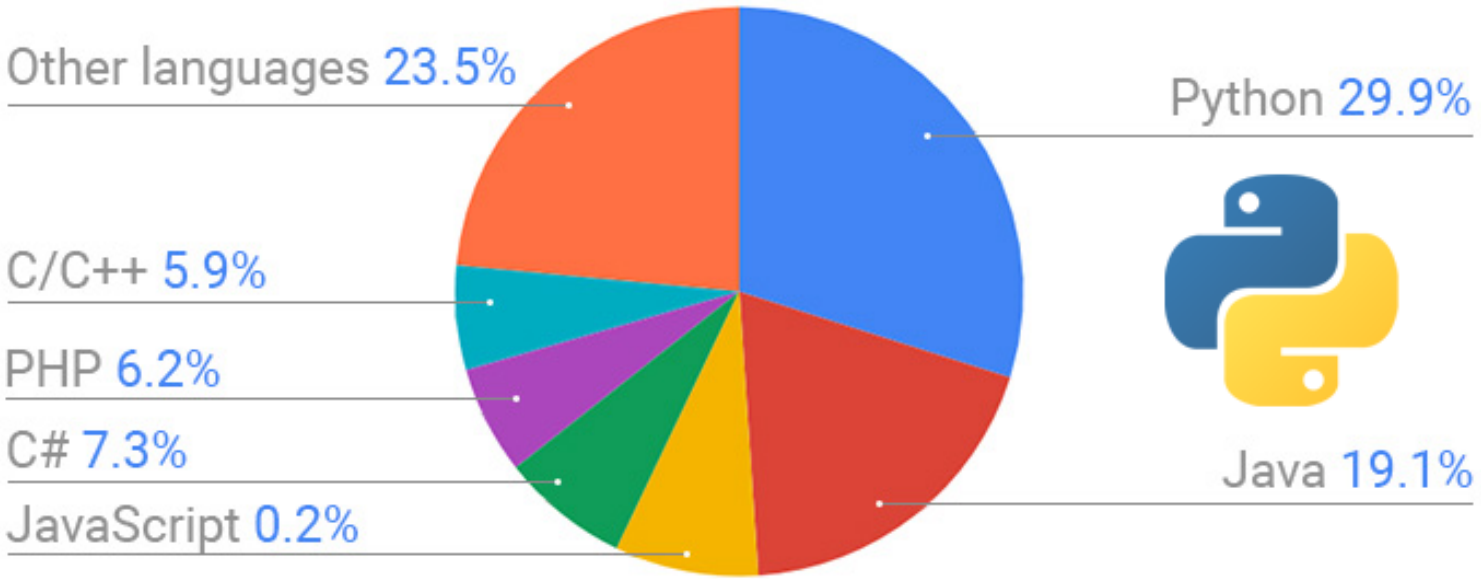


Interfaccia Spyder



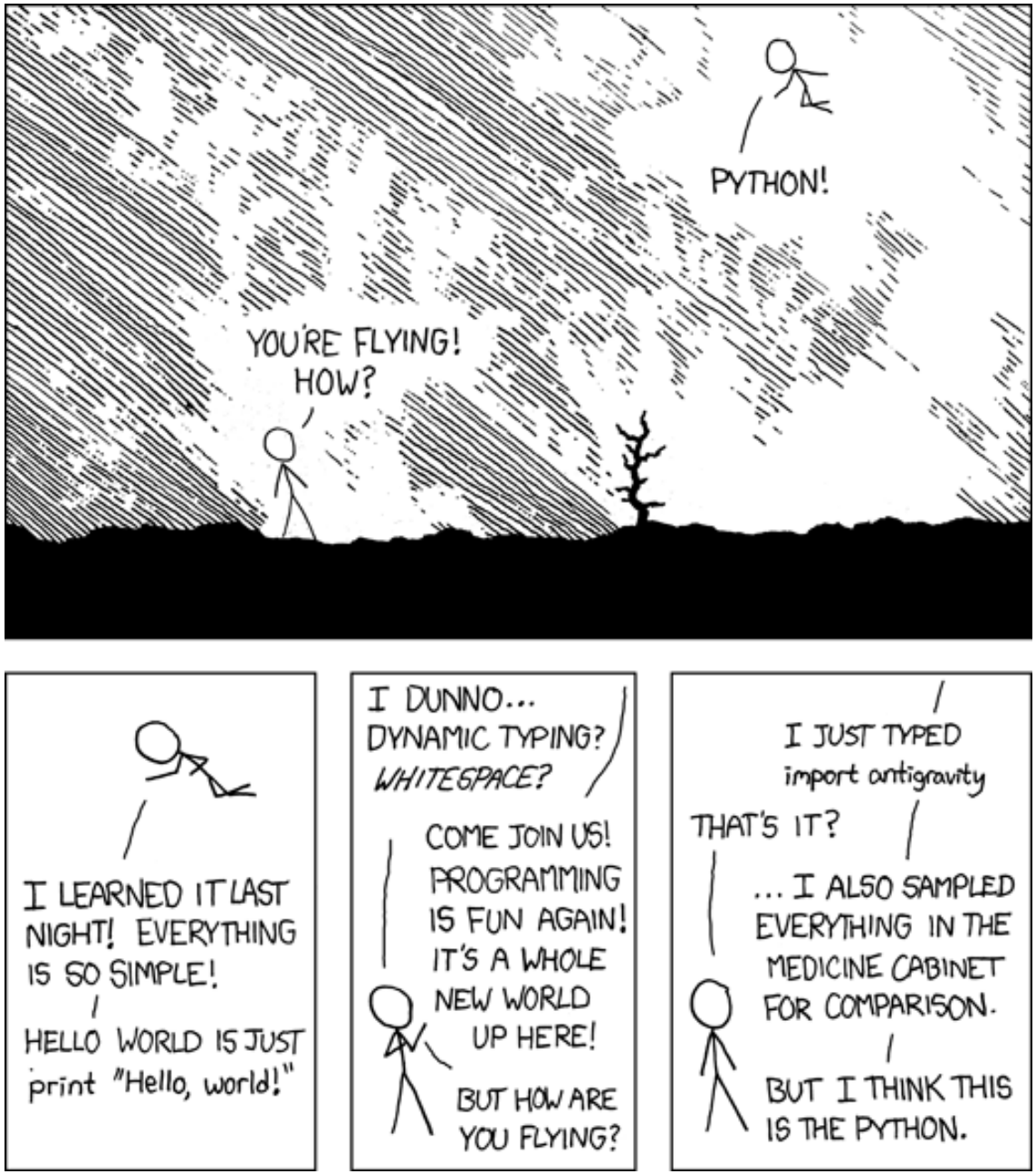
Python

- Linguaggio interpretato ad alto livello.
- Applicazioni in numerosi settori: sviluppo web, scripting, calcolo scientifico, business analytics e intelligenza artificiale.
- Linguaggio più diffuso e richiesto nel mercato del lavoro.
- Usato da organizzazioni quali Google, NASA, Instagram e Netflix.
- ChatGPT, Llama, Stable Diffusion!



Funzione help e librerie

- La maggior parte delle azioni si effettuano richiamando funzioni: definite dall'utente o presenti in librerie built-in/importate.
  `print('Hello world!')`
- Per conoscere la sintassi di una funzione si utilizza il comando `help(nome_funzione)`.
- Per importare una libreria si utilizza il comando `import nome_libreria`.
- Filosofia **Don't reinvent it, just use it**.


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

Variabili

- Le variabili possono essere chiamate in qualunque modo, purché il primo carattere del nome della variabile sia una lettera.
- Due variabili `pippo` e `Pippo` sono considerate diverse.
- I valori delle variabili vengono assegnati tramite = (da non confondere con l'operatore logico ==).
- Le variabili assegnate vengono salvate nel workspace, nell'angolo in alto a destra di Spyder, oppure utilizzando il comando `%who` o `%whos`.
- Per cancellare una variabile dal workspace, si può usare il comando `del`.

Tipi di dati

- **Numeric:** Integer `x = 3` e Floating point `x = 1.5`
- **Complex** `y = 2 + 3j`
- **Strings** `a = 'pippo'`
- **Boolean** `b = True` o `b = False`
- **None** tipo di dato riservato a `None`, rappresenta una variabile vuota

Data una variabile assegnata `x`, la funzione `type(x)` restituisce il tipo di `x`.

```
> x = 3
> type(3)
[1] float
```

```
> a = "pippo"
> type(a)
[1] str
```

```
> b = True
> type(b)
[1] bool
```

Sintassi operatori

Operatori aritmetici

+	Addizione
-	Sottrazione
*	Moltiplicazione
/	Divisione
**	Elevamento a potenza
%	Resto della divisione
//	Divisione intera

Operatori relazionali

==	Uguale a
!=	Diverso da
>	Maggiore
>=	Maggiore o uguale
<	Minore
<=	Minore o uguale

Operatori logici

not	negazione
and	congiunzione AND
or	congiunzione OR inclusivo
is	identità
in	appartenenza

Tuple

- Le tuple sono tipi di dato (immutabili), che possono contenere al loro interno oggetti di tipo diverso contemporaneamente.
- Una tuple si costruisce con la funzione `tuple()` e si accede al suo elemento i-esimo usando `[i]`.

```
> L = (1, 2, 3)
> len(L)
[1] 3
> L[-1]
[1] 1
> L[0]
[1] 1
> L[1:]
[1] (2,3)
```

Liste

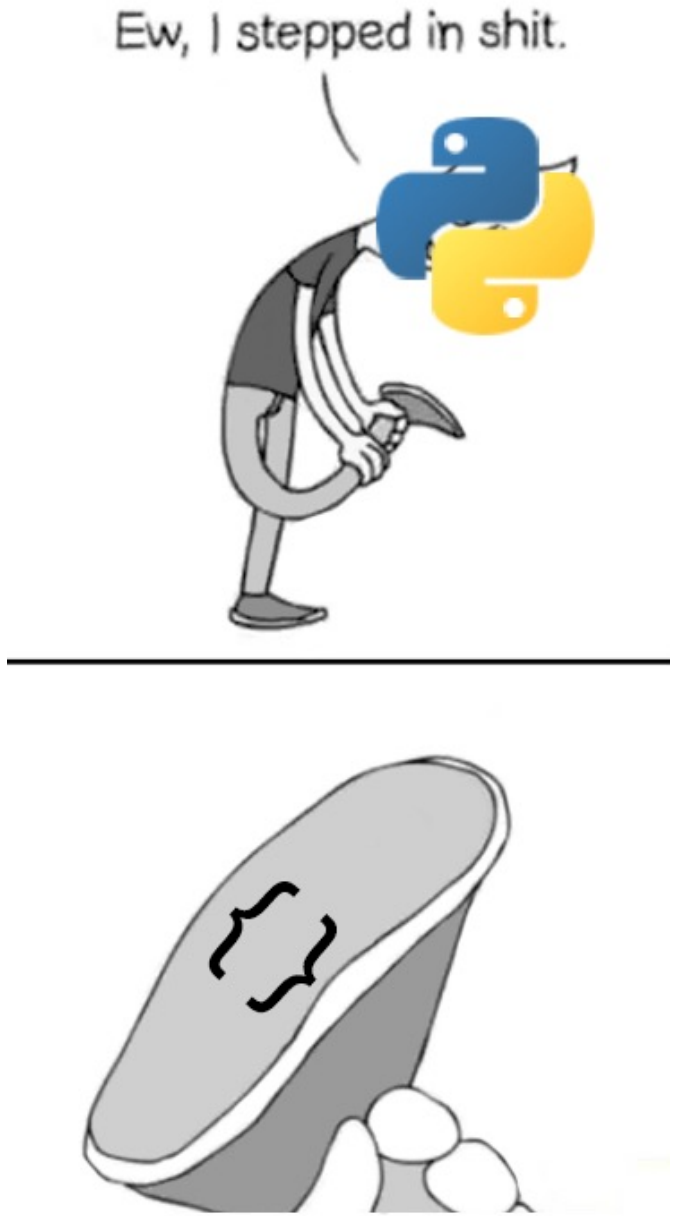
- Le liste sono tipi di dato, che possono contenere al loro interno oggetti di tipo diverso contemporaneamente.
- Una lista si costruisce con la funzione `list()` e si accede al suo elemento i-esimo usando `[i]`.
- A differenza delle tuple le liste sono oggetti mutabili.

```
> L = ['pippo', 1]
> L.append([1, 3, 1])
> L[-1]
[1] 1 3 1
> L[0]
[1] 'pippo'
> len(L)
[1] 3
```

Condizione If-else

- Il comando `if` serve per eseguire codice solamente quando è verificata una condizione.
- La condizione può essere un'espressione logica o un valore booleano.
- Il comando `else` serve per eseguire codice nel caso in cui non sia verificata la condizione.
- Si possono concatenare più `if` con il comando `elif`.

```
if (condizione1):
   espressione1
elif (condizione2):
   espressione2
else:
   espressione3
```


Ew, I stepped in shit.
{ }

Librerie utili

- `math` fornisce tutta una serie di funzioni e costanti matematiche.
- `random` fornisce tutta una serie di generatori pseudorandom.

```
import math
print(math.sin(0))

from math import sqrt
print(sqrt(9))

from math import pi
print(pi)
```

```
import random
random.seed(42)

x = random.random()
y = random.randint(1,10)
```

Cicli while e for

- Con il ciclo `while` si esegue l'espressione finché è verificata la condizione.

```
while (condizione):
    espressione
```

- Con il ciclo `for`, dato una lista/vettore `v`, si esegue l'espressione facendo scorrere l'indice `i` in `v`.

```
for i in v:
    espressione
```

- Il comando `range(n)` crea una 'lista' di numeri che vanno da 0 ad n-1.

Funzioni user-defined

- Spesso si ripete più volte una serie di comandi in uno stesso script.
- In questo caso, è opportuno definire una funzione per alleggerire e rendere più leggibile il codice .

```
def nome_function(x1,x2,...):
    comandi
    return output
```

- Per richiamare la funzione è necessario scrivere `f(x1, x2, ...)`.
- Le variabili definite all'interno della funzioni sono locali e vengono cancellate una volta terminata l'esecuzione della stessa.

Funzioni user-defined

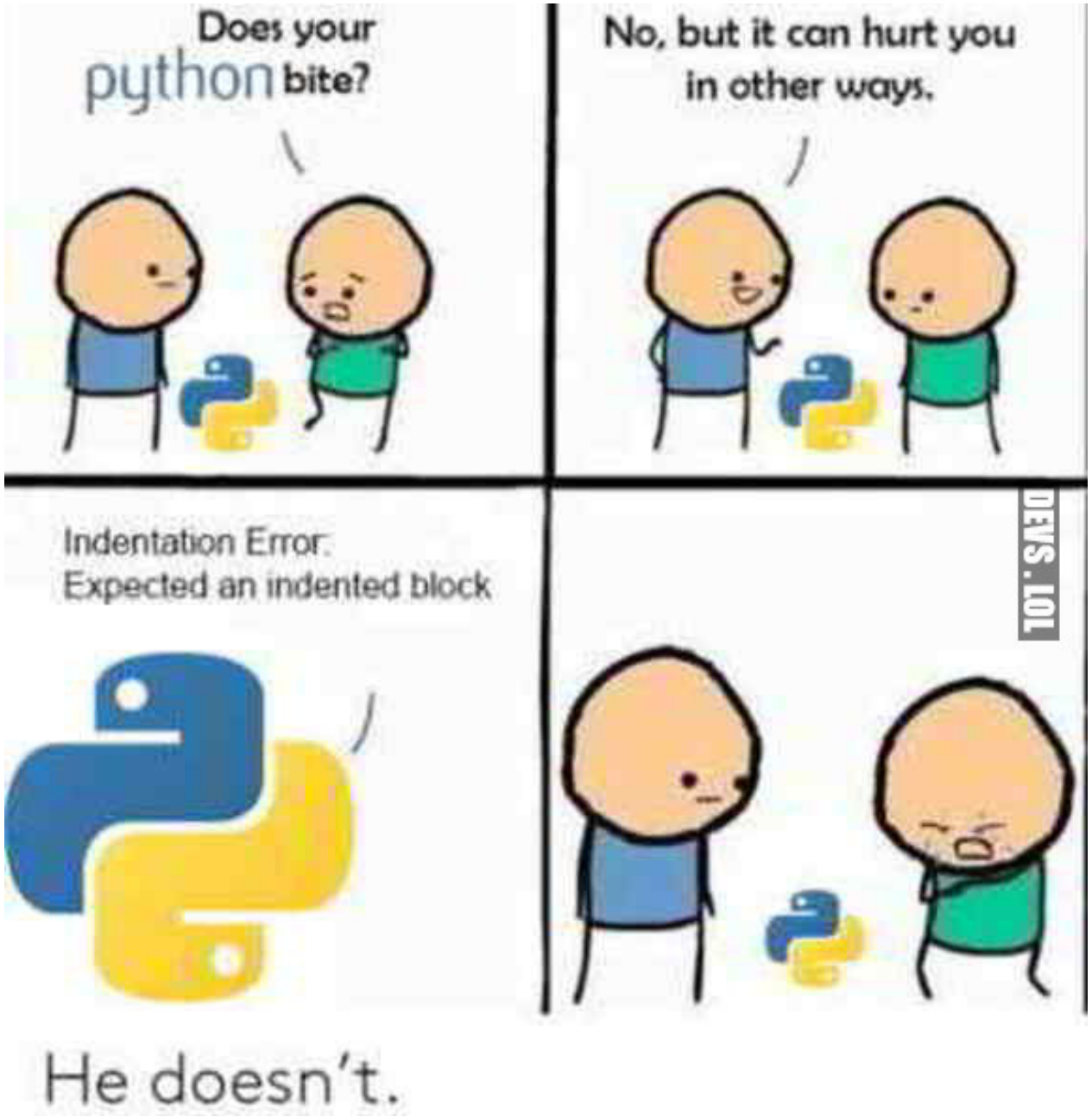
```
> def media(x1, x2):
      m = (x1+x2)/2
      return m
> media(1,5)
[1] 3.0
> m
NameError: name 'm' is not defined
```

Variabili locali immutabili - interi

```
def myrandom1(x):
    x = random()
    return x

def myrandom2():
    a = random()
    return a

a = 2
print('myrandom1: ', myrandom1(a))
print('a = ', a)
print('myrandom2:', myrandom2())
print('a = ', a)
```

Funzioni user-defined

```
v = [i for i in range(2, 9)]
print ('Original v: \t', v)
print(len(v))

def redouble(x):
    for i in range(0, len(x)):
        x[i] = 2*x[i]
    return x

def redouble2(x):
    d = []
    for i in range(0, len(x)):
    d.append(2*x[i])
    return d
```


Does your python bite?
No, but it can hurt you in other ways.
Indentation Error: Expected an indented block
DEVS.LOL
He doesn't.

Numpy

- Libreria fondamentale per il calcolo matriciale e vettoriale.
- Si importa con il comando `import numpy as np`
- Tipo di dato fondamentale `numpy.array`

```
> x = np.array((1,2,3,4))
> x.size
[1] 4
> x.shape
[1] (4,)
> x.ndim
[1] 1
```

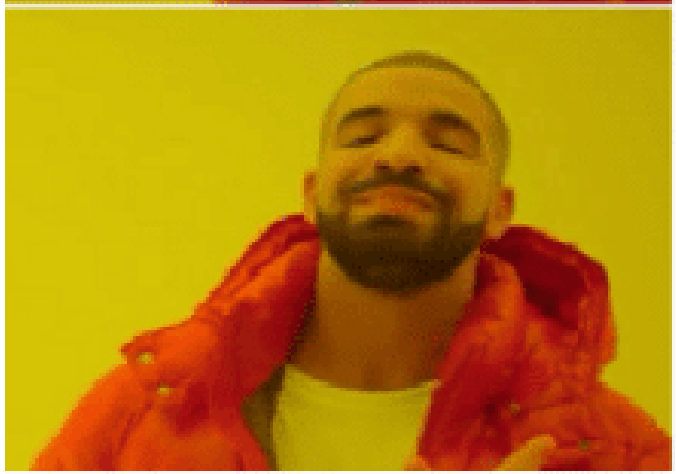
```
> x = np.array([[1,2,3],[4,5,6]])
> x.size
[1] 6
> x.shape
[1] (2, 3)
> x.ndim
[1] 2
```

Funzioni matematiche

Funzioni matematiche base numpy

`cos sin tan`	coseno, seno, tangente
`acos asin atan`	arcoseno, arcocoseno, arcotangente
`cosh sinh tanh`	equivalente iperbolico delle precedenti
`acosh asinh atanh`	idem
`log log10`	Logaritmo naturale, Logaritmo in base 10
`sqrt exp`	Radice quadrata, esponenziale
`sign abs`	Segno, valore assoluto
`round`	Arrotondamento all'intero più vicino
`floor`	Arrotondamento all'intero inferiore

```
1  import numpy
```



```
1  import numpy as np
```

There is no other way

Vettori

Un **vettore** è una lista di oggetti numerici ordinati.

```
> x = np.array([1,3,2])
> x # crea il vettore di elementi [1 , 3 , 2]
[1] array([1, 3, 2])
> y = np.repeat(x,2)
> y # crea un vettore ripetendo due volte x
[1] array([1, 1, 3, 3, 2, 2])
> k = np.arange(1,11,1)
> k # crea un vettore che contiene i numeri da 1 a 10
[1] array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
> z = np.linspace(0,50,6) # Crea un vettore che contiene 6
elementi equispaziati tra 0 e 50
> z
[1] array([ 0., 10., 20., 30., 40., 50.])
```

Funzioni su vettori

Dato un vettore `x` è possibile estrarne dei valori o una parte utilizzando le parentesi quadre:

- Inserendo un numero, verrà estratto il valore di `x` che si trova in quella posizione.
- Inserendo un vettore, verrà estratto un sottovettore di `x` che ha per elementi gli elementi di `x` con posizione corrispondente a quella descritta dal vettore.

```
> x = np.arange(1,100,10)
> len(x) # length restituisce il numero di elementi di x
[1] 10
> x[3]
[1] 31
> x[1:3]
[1] array([11, 21])
> x[[1, 5, 2[]]
[1]array([11, 51, 21])
```

Funzioni su vettori

- `np.sort()` ordina gli elementi di un vettore in ordine crescente.
- `np.where()` restituisce le posizioni degli elementi di un vettore che rispettano una condizione.
- `np.max()`, `np.min()` restituiscono l'elemento più grande e quello più piccolo di un vettore.

```
> x = np.array([4,1,5,2,3])
> np.sort(x) # Ordina x in ordine crescente
[1] array([1, 2, 3, 4, 5])
> np.where(x >= 3) # Posizione degli elementi di x>=3
[1] (array([0, 2], dtype=int64),)
> np.max(x) # Posizione del max di x
[1] 5
> np.min(x) # Posizione del min di x
[1] 1
```

Matrici

- Le matrici sono un'estensione del concetto di vettore in cui gli elementi sono disposti a forma di tabella.
- Le matrici sono array, come i vettori.
- Una matrice viene costruita con la funzione `np.array()` che prende in ingresso una lista di liste.
- La selezione degli elementi in una matrice avviene in maniera simile ai vettori. Si devono passare due valori, separati da una " ,", che indicano rispettivamente l'indice di riga e l'indice di colonna.

Matrici

```
> x = np.ones(2,2) # Crea la matrice di tutti 1
> x
[1] array([[1., 1.], [1., 1.]])
> x = np.zeros(2,2) # Crea la matrice di tutti 0
> x
[1] array([[0., 0.], [0., 0.]])
> np.diag(x)
[1] array([0., 0.])
> d = np.arange(1,4)
> D = np.diag(d)
> D
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

Funzioni su matrici

- `np.concatenate()` che concatena più matrici (per righe o per colonne con l'opzione `axis`) per costruire una matrice più grande.
- `np.transpose()` che restituisce la trasposta della matrice in input.
- `A.T` che restituisce la trasposta della matrice `A`.
- `A.flatten()` che presa una matrice restituisce il vettore ottenuto appiattendo la matrice.

Slice di array

- Per estrarre la riga i-esima dall'array `A` si usa il comando `A[i,:]`
- Per estrarre più righe consecutive dall'array `A` si usa il comando `A[i:j,:]`
- Per estrarre le colonne la cosa è analoga cambiando gli indici, ossia `A[:,i:j]`
- Per estrarre un singolo elemento in posizione (i,j) dall'array `A` si usa il comando `A[i,j]`
- Per estrarre una sottomatrice si usa il comando `A[i1:i2,j1:j2]`

Plot Introduzione

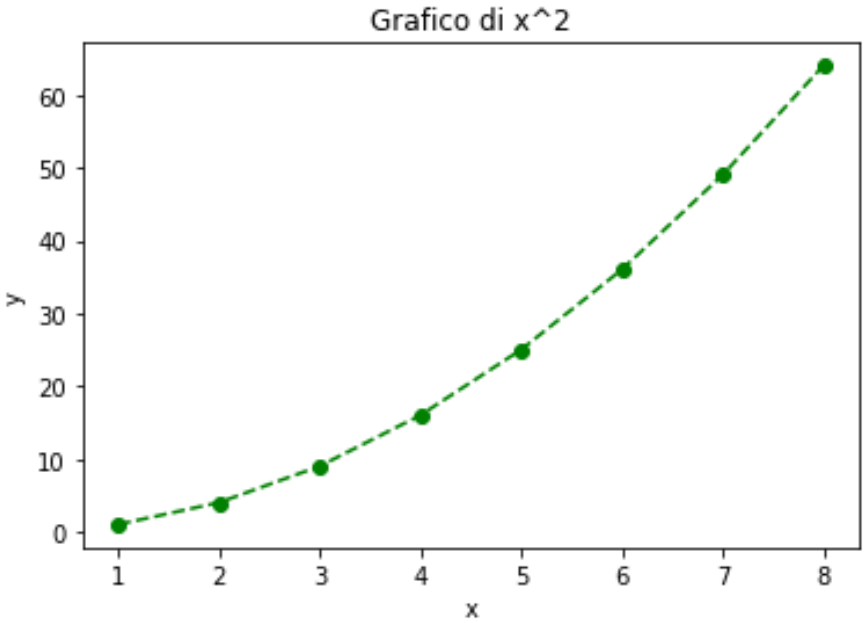
- Per rappresentare i vettori delle osservazioni e controllare come sono distribuiti i dati nel nostro dataset, usiamo Matplotlib.
- Si importa con il comando `import matplotlib.pyplot as plt`

La funzione `plot()` prende come input molti argomenti:

- Un vettore `x` rappresentante le ascisse dei punti da plottare.
- Un vettore `y` rappresentante le ordinate dei punti da plottare.
- `color` che indica il colore del grafico.
- `marker` che indica il marker che rappresenta i punti. [ref]
- `linestyle` che indica il tipo di linea che vogliamo usare.[ref]

Plot Esempio

```
> x = np.arange(1,9)
> y = x**2
> plt.plot(x, y, color='green', marker='o', linestyle='dashed'
> plt.xlabel('x')
> plt.ylabel('y')
> plt.title('Grafico di x^2')
> plt.show()
```

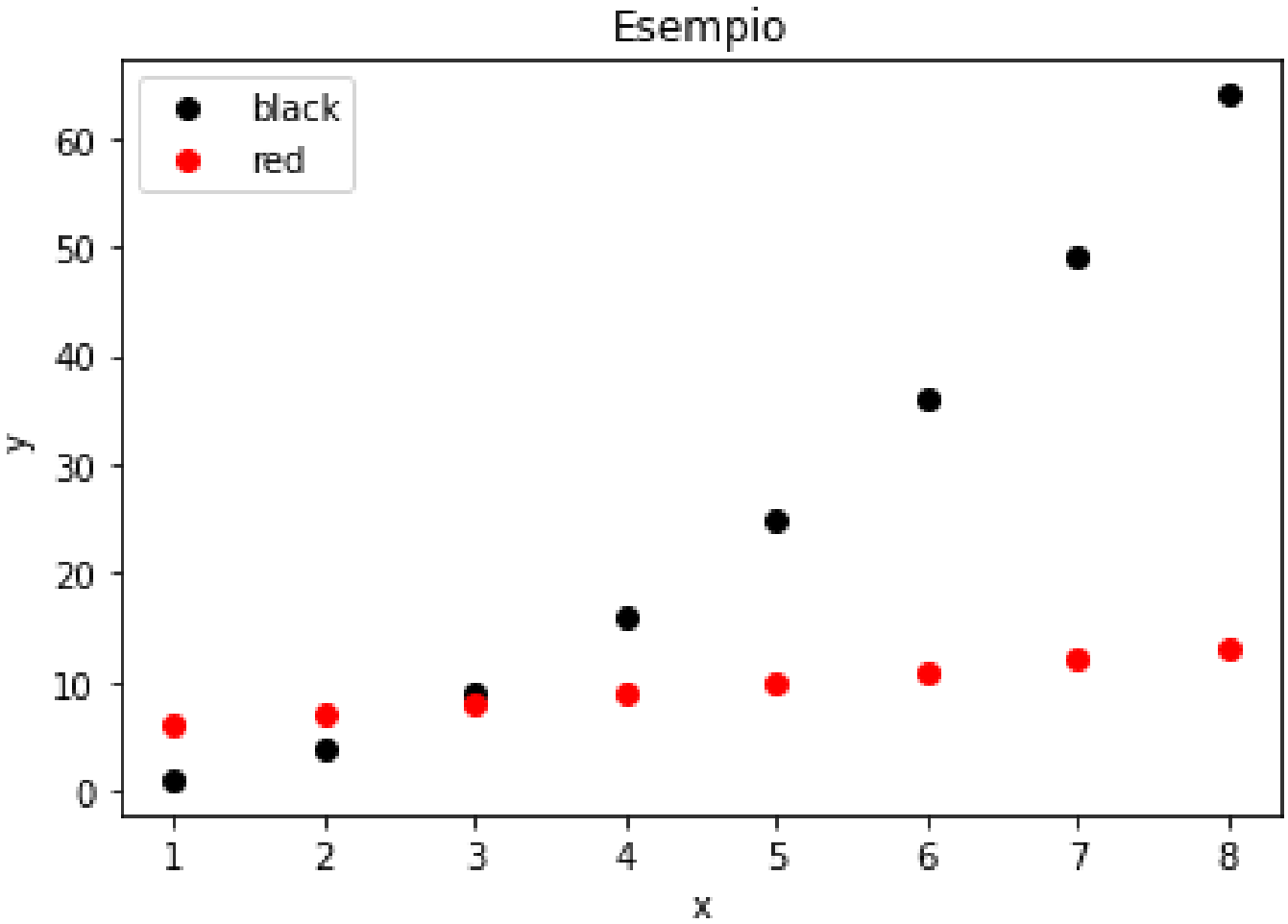


Plot(s)

- La funzione `plot()` disegna il nuovo grafico nella stessa figura.

```
> x = np.arange(1,9)
> y = x**2
> y1 = x + 5
> plt.plot(x, y, 'ko')
> plt.plot(x,y1,'ro')
> plt.xlabel('x')
> plt.ylabel('y')
> plt.legend(['black','red'])
> plt.title('Esempio')
> plt.show()
```

Plot(s)

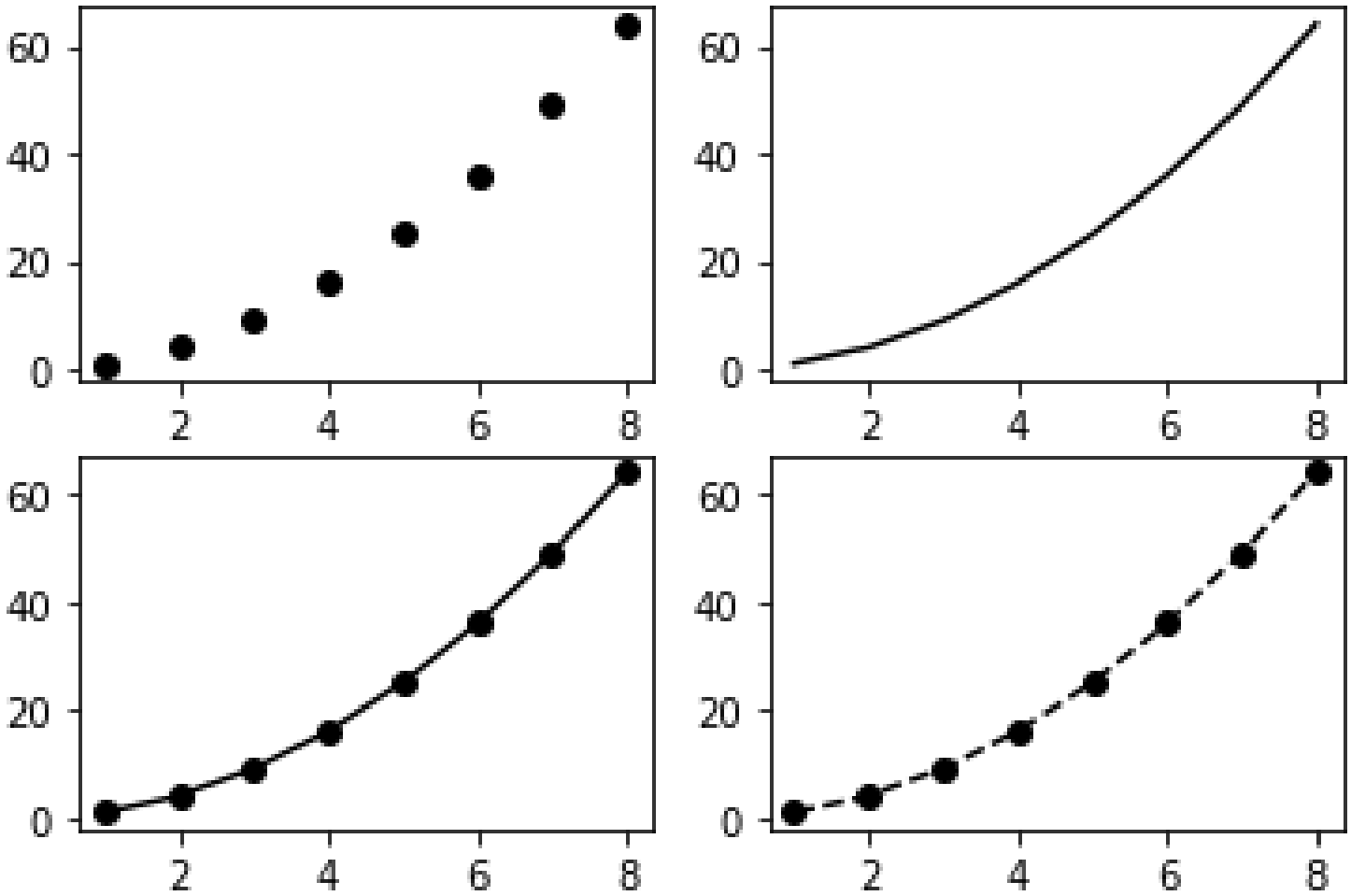


Subfigure

- Per disegnare più plot nella stessa schermata si usa la funzione `plt.subplots(nrows=rnum, ncols=cnum)`, dove `rnum` rappresenta il numero di plot che si vuole visualizzare in riga, mentre `cnum` rappresenta il numero di plot che si vuole visualizzare in colonna.

```
> fig, ax = plt.subplots(nrows=2, ncols=2)
> ax[0,0].plot(x, y, 'ko')
> ax[0,1].plot(x, y, 'k-')
> ax[1,0].plot(x, y, 'k-o')
> ax[1,1].plot(x, y, 'k--o')

> plt.show()
```

Subfigure

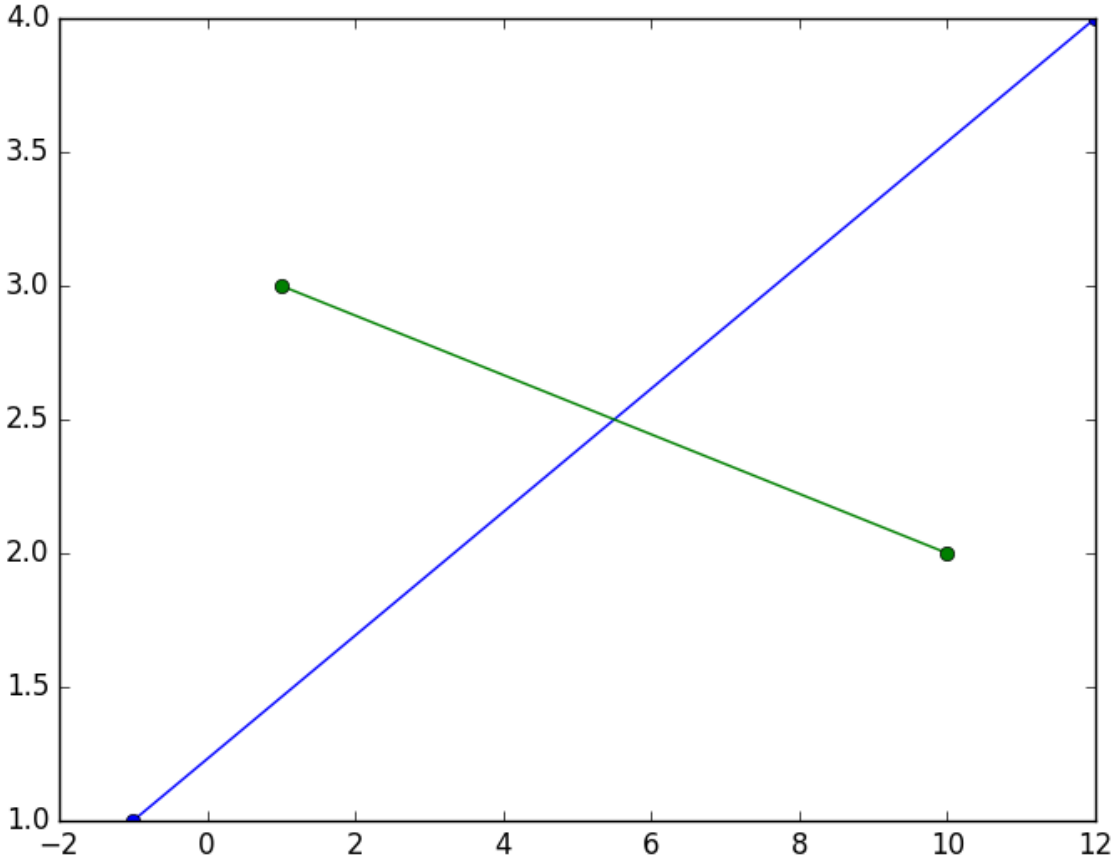


Segmenti e linee

- Per disegnare una retta si usa il comando `axline(xy1, xy2=None, slope=None)`
  - `xy1, xy2` punti per cui passa la retta.
  - `slope` coefficiente angolare retta (se non si specifica `xy2`)
- Si possono disegnare i segmenti prendendo in input le coordinate dei due punti estremi.

```
> x1, y1 = [-1, 12], [1, 4]
> x2, y2 = [1, 10], [3, 2]
> plt.plot(x1, y1, x2, y2, marker = 'o')
> plt.show()
```

Segmenti e linee

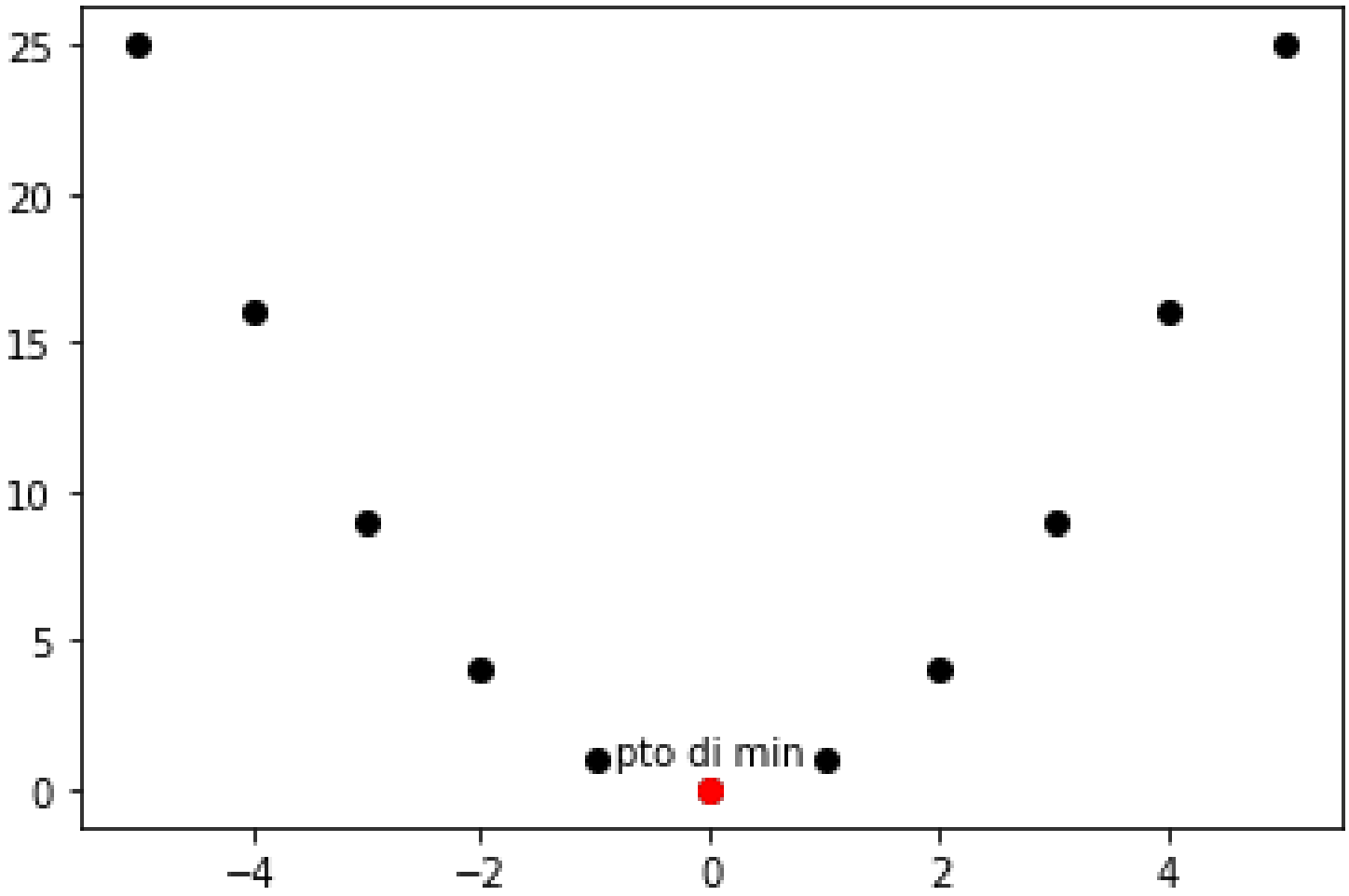


Text

La funzione `text()` serve per aggiungere dei label sul grafico.
Utile quando si vuole aggiungere il nome ad uno specifico punto del grafico.

```
> x = np.arange(-5,6)
> y = x**2

> plt.plot(x,y,'ko')
> plt.plot(0,0,'ro')
> plt.text(0,0.8, 'pto di min', horizontalalignment='center')
> plt.show()
```

Text



Estrazioni casuali: `random.choice()`

- Dato un vettore `v` la funzione `random.choice()` estrae da `v` (con reinserimento) *n* valori.

```
> v = np.arange(1,11)
> np.random.choice(v,10)
[1] 2 9 5 9 1 9 2 6 5 9
```

- Per estrarli con reinserimento

```
> np.random.choice(v,8, replace=False)
[1] 2 9 1 7 3 6 8 4
```

- È possibile specificare la probabilità di estrarre ogni singolo elemento.

```
> v = np.arange(1,4)
> np.random.choice(v, 10, replace=True, p=(0.7 , 0.2 , 0.1
[1] 1 1 2 3 1 1 2 3 1 1
```

Estrazioni casuali: normale Gaussiana

La funzione `random.normal()` prende in input

- la media
- la deviazione standard
- la lunghezza del vettore di output

restituisce un vettore che ha dimensione scelta e contiene elementi estratti con una distribuzione Gaussiana con media e deviazione assegnate.

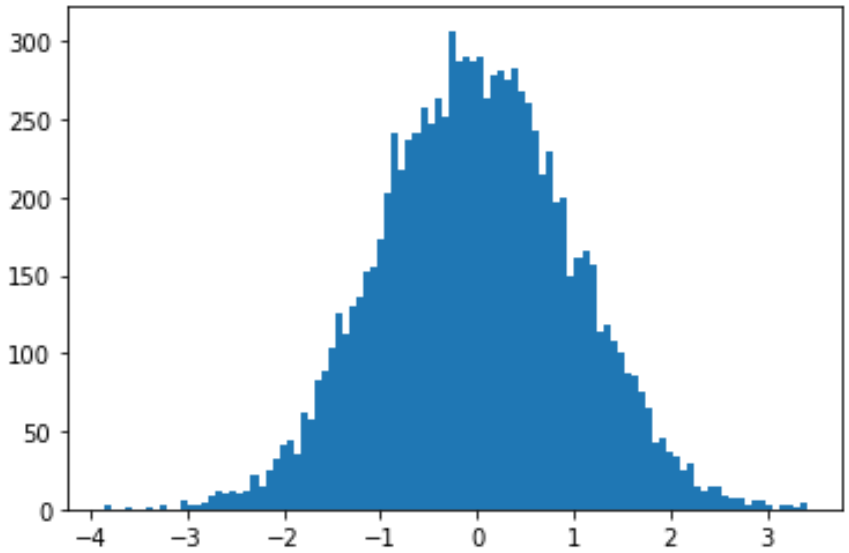
```
> x = random.normal(loc=0.0, scale=1.0, size=10)
> x # Normale standard
[1] 0.35315767 0.81645816 0.27591868 -1.09573384
1.63935188 0.80787089 -0.07381423 0.63730397
-0.61066481 -0.89062814
```

- textttrandom.randn() stessa cosa senza specificare media e deviazione standard (0,1).

Istogrammi

Per disegnare un istogramma si utilizza la funzione `hist()` che prende come input un vettore (del quale plottare le frequenze) e un parametro opzionale `bins`, che indica in quanti intervalli dividere i valori dell'array in input.

```
> x = np.random.randn(10000)
> plt.hist(x , bins=100)
```



Barplot & Piechart

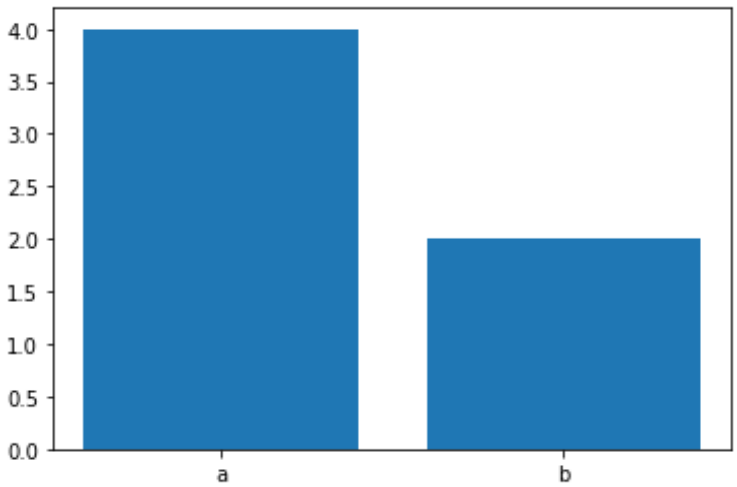
Per rappresentare un vettore di tipo qualitativo si utilizzano le seguenti funzioni

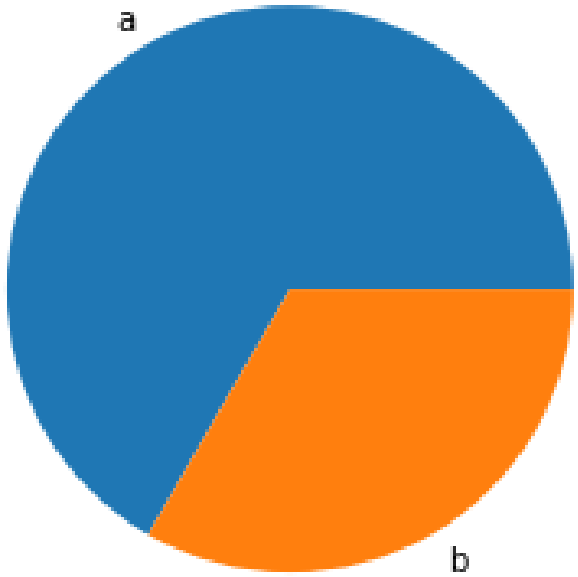
- `bar()` per rappresentare dei grafici a barre
- `piechart()` per rappresentare dei grafici a torta

Prima è necessario contare le occorrenze di ogni possibile valore.

```
> x = np.array(("a" , "a" , "a" , "b" , "a" , "b" ))
> unique = np.unique(x)
> count = [np.sum(x==el) for el in unique]
> plt.bar(unique,count)
> plt.pie(count,labels=unique)
```

Barplot & Piechart





Boxplot

Il `boxplot()`, risulta utile per visualizzare il range dei dati.

```
> x = np.array((3, 3, 3, 2, 1, 4, 6, 2, 4, 1, 6, 4, 2, 1, 5, 4
> plt.boxplot(x)
```