

# PYTHON

Python è un linguaggio di programmazione interpretato. Significa che il codice sorgente che scriviamo viene passato ad un altro programma, detto interprete, che lo esegue direttamente senza la necessita' di compilarlo

# PYTHON

Python è un linguaggio di programmazione interpretato. Significa che il codice sorgente che scriviamo viene passato ad un altro programma, detto interprete, che lo esegue direttamente senza la necessità di compilarlo

i comandi di python rispetto a quelli del C hanno un grado di astrazione maggiore, per questo motivo si dice che Python è un linguaggio di più alto livello rispetto al C

# PYTHON

Python è un linguaggio di programmazione interpretato. Significa che il codice sorgente che scriviamo viene passato ad un altro programma, detto interprete, che lo esegue direttamente senza la necessità di compilarlo

i comandi di python rispetto a quelli del C hanno un grado di astrazione maggiore, per questo motivo si dice che Python è un linguaggio di più alto livello rispetto al C

**Es.**

**C**  
dobbiamo creare un file .c (lo chiameremo ciao\_mondo.c) e scriverci dentro

```
#include <stdio.h>

int main (void){
    printf("Ciao mondo\n");
    return(0);
}
```

il codice creato va poi compilato

```
$ gcc -o ciao_mondo.o ciao_mondo.c
$ ./ciao_mondo.o
Ciao mondo
```

# PYTHON

Python è un linguaggio di programmazione interpretato. Significa che il codice sorgente che scriviamo viene passato ad un altro programma, detto interprete, che lo esegue direttamente senza la necessità di compilarlo

i comandi di python rispetto a quelli del C hanno un grado di astrazione maggiore, per questo motivo si dice che Python è un linguaggio di più alto livello rispetto al C

**Es.**

**C**

dobbiamo creare un file .c (lo chiameremo ciao\_mondo.c) e scriverci dentro

```
#include <stdio.h>

int main (void){
    printf("Ciao mondo\n");
    return(0);
}
```

il codice creato va poi compilato

```
$ gcc -o ciao_mondo.o ciao_mondo.c
$ ./ciao_mondo.o
Ciao mondo
```

**PYTHON**

possiamo scrivere direttamente nel prompt dei comandi

```
$ python3
>>> print("Ciao Mondo")
Ciao Mondo
>>>
```

# PYTHON

Python è un linguaggio di programmazione interpretato. Significa che il codice sorgente che scriviamo viene passato ad un altro programma, detto interprete, che lo esegue direttamente senza la necessità di compilarlo

i comandi di python rispetto a quelli del C hanno un grado di astrazione maggiore, per questo motivo si dice che Python è un linguaggio di più alto livello rispetto al C

**Es.**

**C**

dobbiamo creare un file .c (lo chiameremo ciao\_mondo.c) e scriverci dentro

```
#include <stdio.h>

int main (void){
    printf("Ciao mondo\n");
    return(0);
}
```

il codice creato va poi compilato

```
$ gcc -o ciao_mondo.o ciao_mondo.c
$ ./ciao_mondo.o
Ciao mondo
```

**PYTHON**

possiamo scrivere direttamente nel prompt dei comandi

```
$ python3
>>> print("Ciao Mondo")
Ciao Mondo
>>>
```

oppure creare un file .py (lo chiameremo ciao\_mondo.py) che contiene il comando di stampa. Possiamo poi eseguire il comando

```
$ python3 ciao_mondo.py
Ciao Mondo
```

# PYTHON

Python è un linguaggio di programmazione interpretato. Significa che il codice sorgente che scriviamo viene passato ad un altro programma, detto interprete, che lo esegue direttamente senza la necessità di compilarlo

i comandi di python rispetto a quelli del C hanno un grado di astrazione maggiore, per questo motivo si dice che Python è un linguaggio di più alto livello rispetto al C

**Es.**

notiamo subito che, al contrario del C in python non c'è il `;` a delimitare un'istruzione

In python però è molto importante indentare correttamente le istruzioni (cosa che in C non è necessaria ma lo si fa per una questione di pulizia del codice )

## PYTHON

possiamo scrivere direttamente nel prompt dei comandi

```
$ python3
>>> print("Ciao Mondo")
Ciao Mondo
>>>
```

# PYTHON

Python è un linguaggio di programmazione interpretato. Significa che il codice sorgente che scriviamo viene passato ad un altro programma, detto interprete, che lo esegue direttamente senza la necessità di compilarlo

i comandi di python rispetto a quelli del C hanno un grado di astrazione maggiore, per questo motivo si dice che Python è un linguaggio di più alto livello rispetto al C

**Es.**

notiamo subito che, al contrario del C in python non c'è il ; a delimitare un'istruzione

In python però è molto importante indentare correttamente le istruzioni (cosa che in C non è necessaria ma lo si fa per una questione di pulizia del codice )

tutte le righe precedute dal simbolo **#** sono dei commenti che non vengono letti dall'interprete

## PYTHON

creiamo un file chiamato ex1.py  
scrivendoci dentro

```
#questo programma stampa  
#le lettere contenute  
#nella parola ciao  
for i in "ciao":  
    print(i)
```

# PYTHON

Python è un linguaggio di programmazione interpretato. Significa che il codice sorgente che scriviamo viene passato ad un altro programma, detto interprete, che lo esegue direttamente senza la necessità di compilarlo

i comandi di python rispetto a quelli del C hanno un grado di astrazione maggiore, per questo motivo si dice che Python è un linguaggio di più alto livello rispetto al C

**Es.**

notiamo subito che, al contrario del C in python non c'è il ; a delimitare un'istruzione

In python però è molto importante indentare correttamente le istruzioni (cosa che in C non è necessaria ma lo si fa per una questione di pulizia del codice )

**indentazione:** vengono lasciati 4 spazi vuoti rispetto alla posizione dell'istruzione della riga sopra

## PYTHON

creiamo un file chiamato ex1.py  
scrivendoci dentro

```
#questo programma stampa  
#le lettere contenute  
#nella parola ciao  
for i in "ciao":  
    print(i)
```





# PYTHON

Python è un linguaggio di programmazione interpretato. Significa che il codice sorgente che scriviamo viene passato ad un altro programma, detto interprete, che lo esegue direttamente senza la necessità di compilarlo

i comandi di python rispetto a quelli del C hanno un grado di astrazione maggiore, per questo motivo si dice che Python è un linguaggio di più alto livello rispetto al C

**Es.**

notiamo subito che, al contrario del C in python non c'è il ; a delimitare un'istruzione

In python però è molto importante indentare correttamente le istruzioni (cosa che in C non è necessaria ma lo si fa per una questione di pulizia del codice )

## PYTHON

creiamo un file chiamato ex1.py  
scrivendoci dentro

```
#questo programma stampa  
#le lettere contenute  
#nella parola ciao  
for i in "ciao":  
    print(i)
```

nel prompt digitiamo il comando:

```
$ python3 ex1.py
```

```
c  
i  
a  
o
```

# PYTHON

Python è un linguaggio di programmazione interpretato. Significa che il codice sorgente che scriviamo viene passato ad un altro programma, detto interprete, che lo esegue direttamente senza la necessità di compilarlo

i comandi di python rispetto a quelli del C hanno un grado di astrazione maggiore, per questo motivo si dice che Python è un linguaggio di più alto livello rispetto al C

**Es.**

notiamo subito che, al contrario del C in python non c'è il ; a delimitare un'istruzione

In python però è molto importante indentare correttamente le istruzioni (cosa che in C non è necessaria ma lo si fa per una questione di pulizia del codice )

## PYTHON

facciamo la stessa cosa di prima senza usare l'indentazione

```
#questo programma stampa  
#le lettere contenute  
#nella parola ciao  
for i in "ciao":  
    print(i)
```

# PYTHON

Python è un linguaggio di programmazione interpretato. Significa che il codice sorgente che scriviamo viene passato ad un altro programma, detto interprete, che lo esegue direttamente senza la necessità di compilarlo

i comandi di python rispetto a quelli del C hanno un grado di astrazione maggiore, per questo motivo si dice che Python è un linguaggio di più alto livello rispetto al C

**Es.**

notiamo subito che, al contrario del C in python non c'è il ; a delimitare un'istruzione

In python però è molto importante indentare correttamente le istruzioni (cosa che in C non è necessaria ma lo si fa per una questione di pulizia del codice )

## PYTHON

facciamo la stessa cosa di prima senza usare l'indentazione

```
#questo programma stampa  
#le lettere contenute  
#nella parola ciao  
for i in "ciao":  
print(i)
```

nel prompt digitiamo il comando:

```
$ python3 ex1.py  
File "ex1.py", line 5  
    print(i)  
    ^  
IndentationError: expected an indented block
```

# PYTHON E IL MODULO MATPLOTLIB

**Un modulo è un file che contiene definizioni e istruzioni Python.**

Matplotlib e' un modulo per la generazione di grafici in 2D (e qualcosa in 3D). Esso contiene a sua volta un altro modulo chiamato pyplot progettato per emulare i piu' comuni comandi grafici di Matlab.

per poter sfruttare le istruzioni di matplotlib e pyplot bisogna "importare" nel proprio script python il modulo stesso

**Es.** scriviamo in un file chiamato ex2.py le seguenti righe di codice e salviamo

```
#un primo esempio dell'uso di matplotlib
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.show()
```

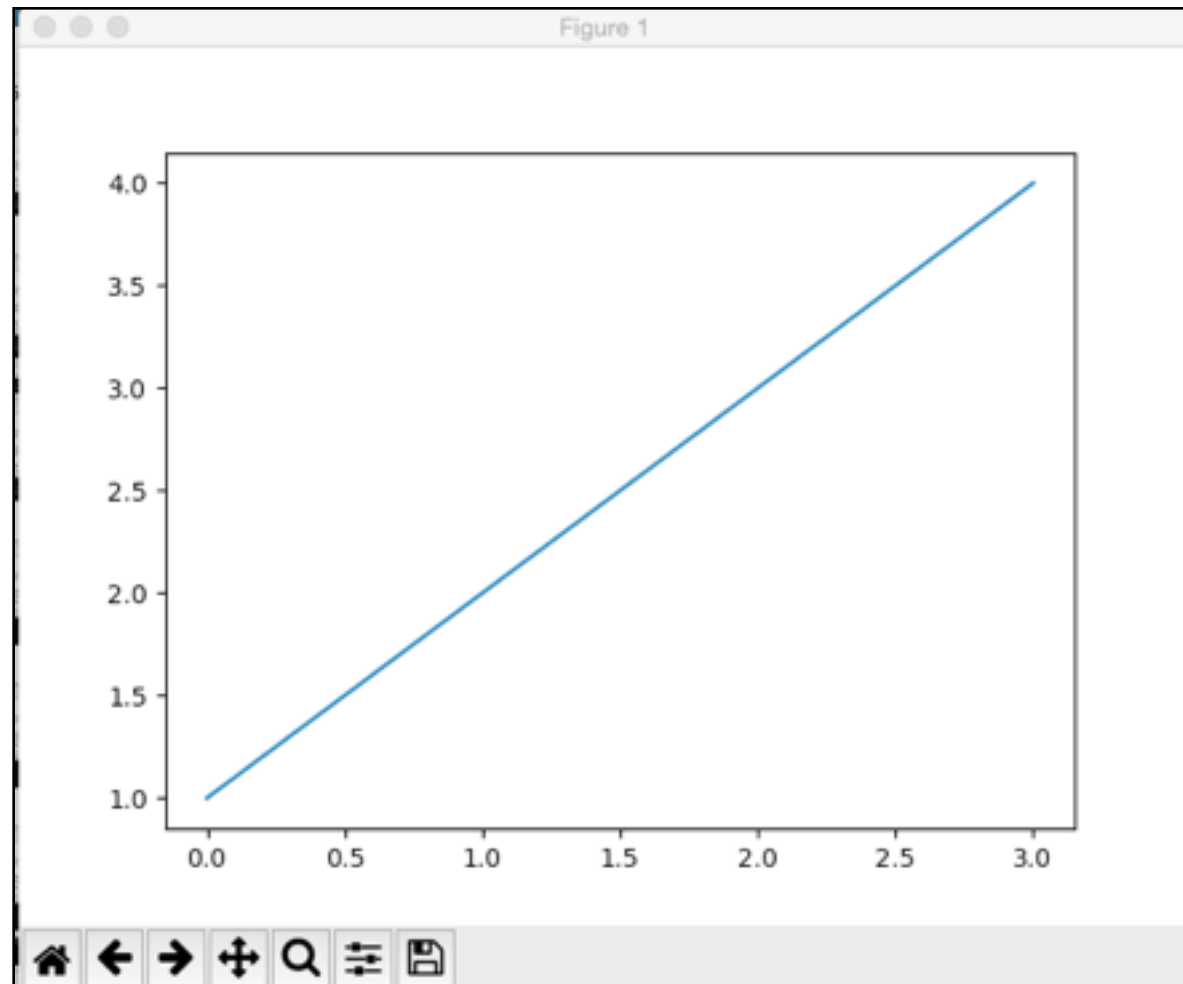
digitiamo nel prompt dei comandi

```
$ python3 ex2.py
```

**....che cosa succede?**

# PYTHON E IL MODULO MATPLOTLIB

**Abbiamo creato il nostro primo grafico in python!**

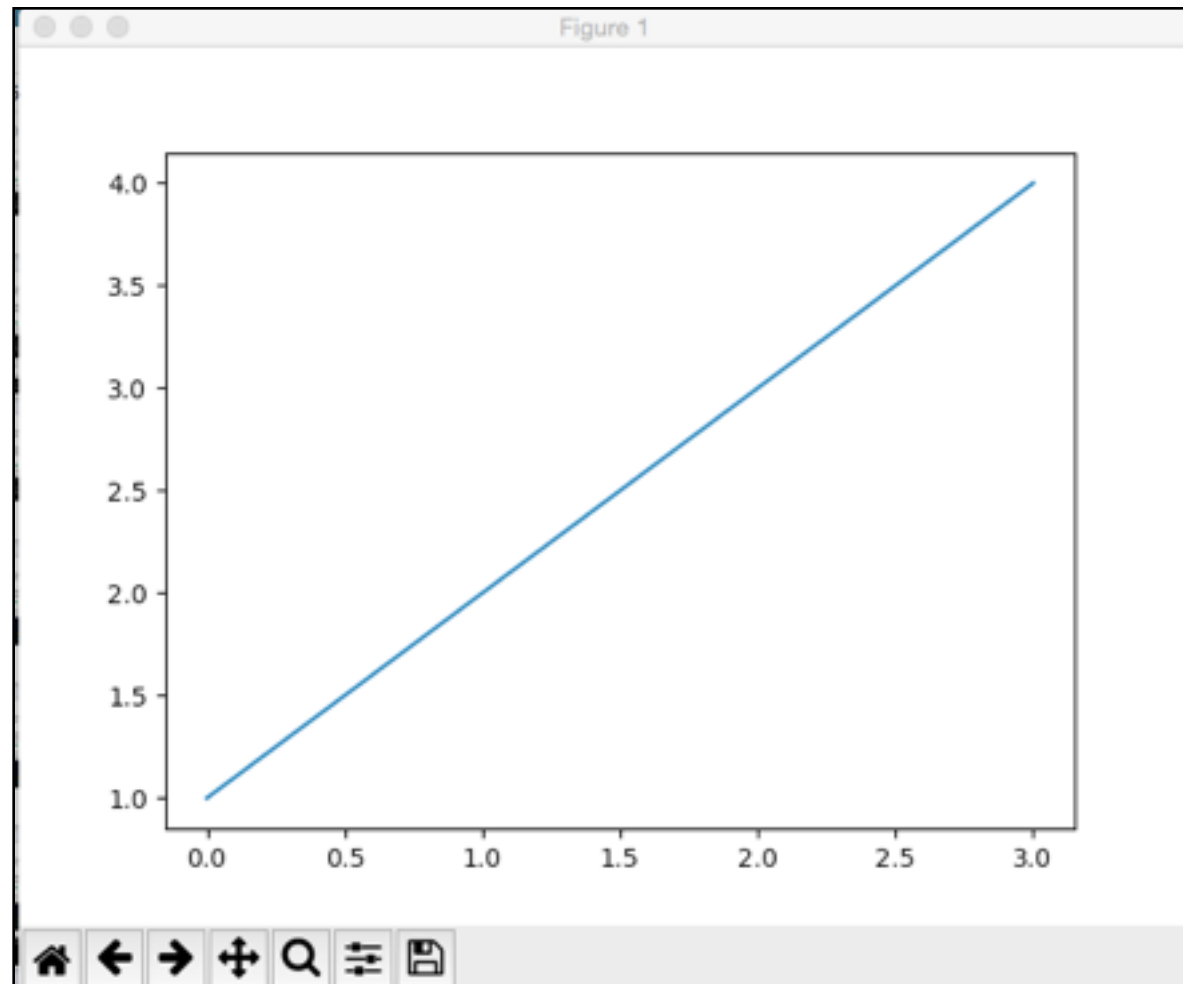


```
#un primo esempio dell'uso di matplotlib
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.show()
```

Lo script importa il modulo `matplotlib.pyplot` e semplicemente lo rinomina (alias) per comodità come `plt`. Per chiamare una qualsiasi istruzione del modulo bisogna precederlo dalla dicitura `plt.` (eq. `plt.plot()`, `plt.ylabel()`, `plt.show()`).

# PYTHON E IL MODULO MATPLOTLIB

**Abbiamo creato il nostro primo grafico in python!**



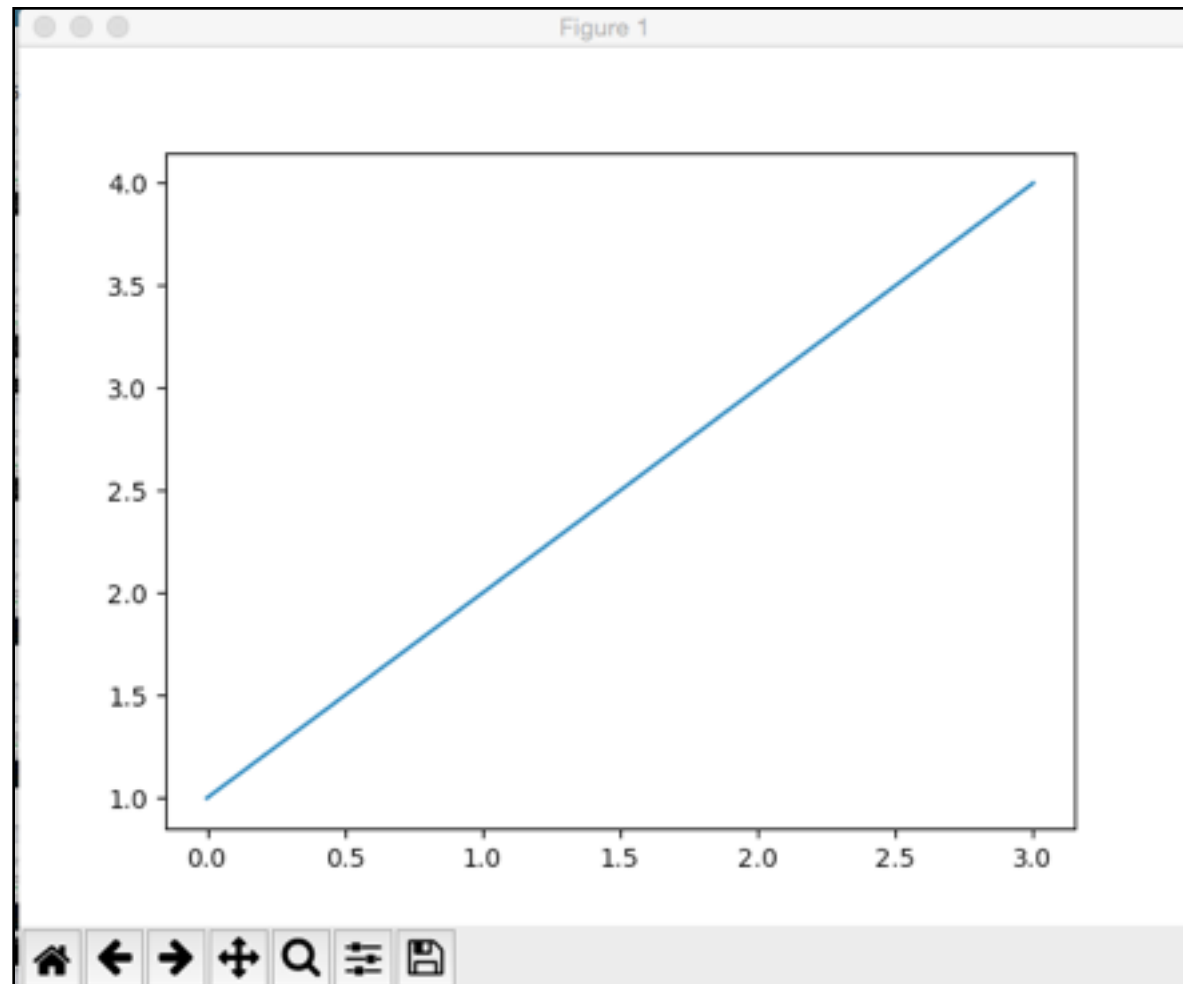
```
#un primo esempio dell'uso di matplotlib
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.show()
```

Lo script importa il modulo `matplotlib.pyplot` e semplicemente lo rinomina (alias) per comodità come `plt`. Per chiamare una qualsiasi istruzione del modulo bisogna precederlo dalla dicitura `plt.` (eq. `plt.plot()`, `plt.ylabel()`, `plt.show()`).

Lo script grafica la lista di numeri (array) `[1, 2, 3, 4]` attraverso la funzione `plt.plot()`. Il modulo `matplotlib` assume che questi valori corrispondano all'asse `y` e automaticamente gli associa una sequenza naturale di valori per l'asse `x`: `0, 1, 2, 3, ...`

# PYTHON E IL MODULO MATPLOTLIB

## Abbiamo creato il nostro primo grafico in python!



```
#un primo esempio dell'uso di matplotlib
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.show()
```

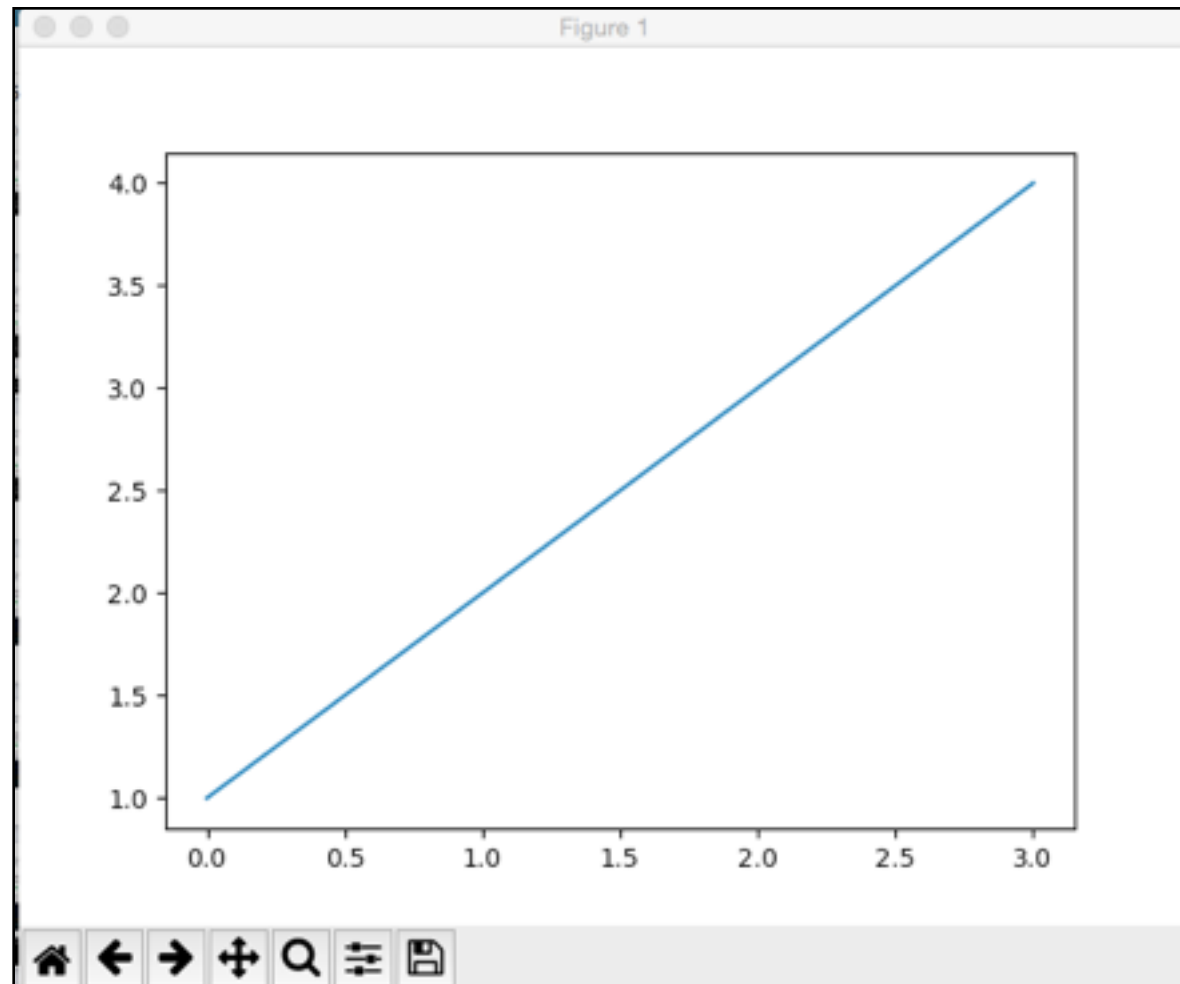
Lo script importa il modulo `matplotlib.pyplot` e semplicemente lo rinomina (alias) per comodità come `plt`. Per chiamare una qualsiasi istruzione del modulo bisogna precederlo dalla dicitura `plt.` (eq. `plt.plot()`, `plt.ylabel()`, `plt.show()`).

Lo script grafica la lista di numeri (array) `[1, 2, 3, 4]` attraverso la funzione `plt.plot()`. Il modulo `matplotlib` assume che questi valori corrispondano all'asse `y` e automaticamente gli associa una sequenza naturale di valori per l'asse `x`: `0, 1, 2, 3, ...`

La funzione `plt.show()` permette di visualizzare il grafico

# PYTHON E IL MODULO MATPLOTLIB

**Abbiamo creato il nostro primo grafico in python!**



## Toolbar della finestra (plotting window)



mostra l'area del grafico originale



Undo/Redo della visualizzazione



navigazione all'interno del grafico



zoom di una porzione rettangolare del grafico



Personalizzazione del grafico



Salvataggio/Esportazione della figura

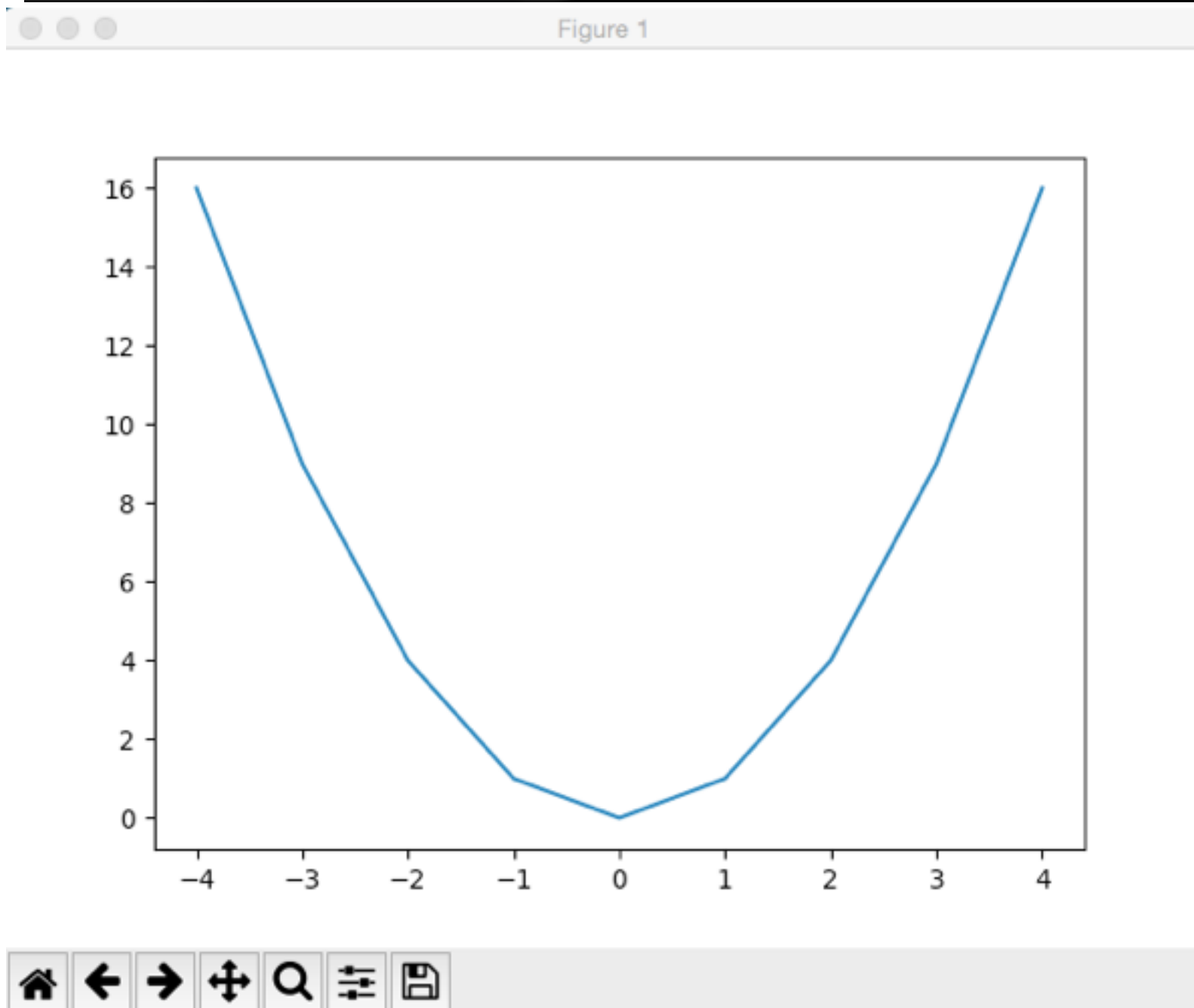
**Provate a giocare un po' con la toolbar e a salvare il grafico**



# PYTHON E IL MODULO MATPLOTLIB

**Es.** graficare una parabola con  $x=[-4, 4]$

```
import matplotlib.pyplot as plt  
plt.plot([-4, -3, -2, -1, 0, 1, 2, 3, 4], [16, 9, 4, 1, 0, 1, 4, 9, 16])  
plt.show()
```

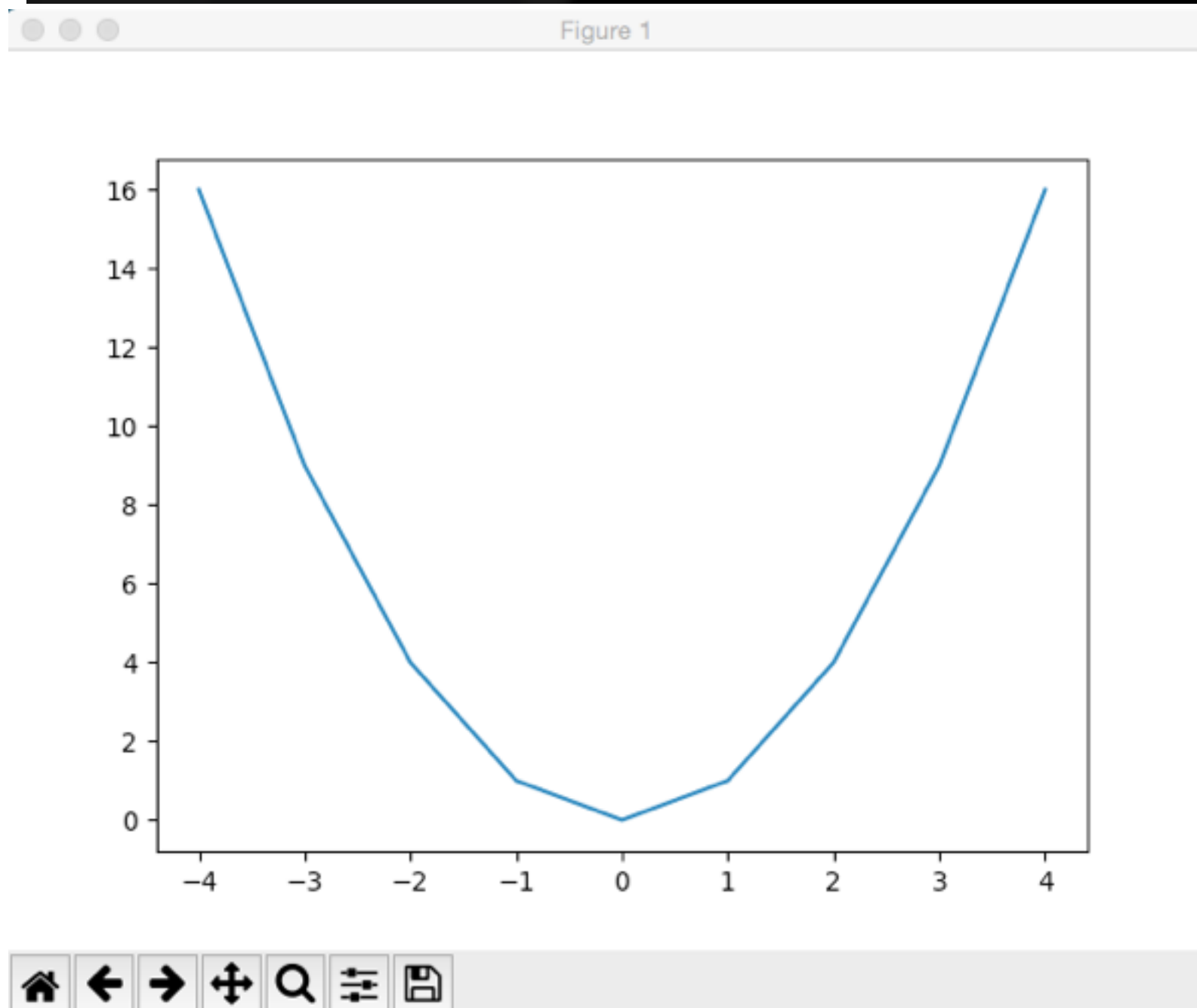


# PYTHON E IL MODULO MATPLOTLIB

**Es.** graficare una parabola con  $x=[-4, 4]$

```
import matplotlib.pyplot as plt
plt.plot([-4, -3, -2, -1, 0, 1, 2, 3, 4], [16, 9, 4, 1, 0, 1, 4, 9, 16])
plt.show()
```

**asse x** **asse y**



in questo caso ho specificato la lista dei valori dell'asse x e dell'asse y.

**Ma se volessi graficare funzioni matematiche piu' complesse e su intervalli spaziali piu' grandi?**

# PYTHON E IL MODULO NUMPY

Le liste viste nell'esempio precedente non supportano operazioni matematiche. il modulo numpy (di solito indicato con alias *np* quando viene importato) permette di leggere, manipolare e stampare facilmente vettori, matrici e, più in generale, tensori di qualunque ordine. Nonostante non faccia parte della libreria standard di Python, numpy è de facto lo standard per lavorare con oggetti di questo tipo.

Usando numpy, possiamo convertire una sequenza di numeri come quella dell'esempio precedente in un array a manipolarlo usando funzioni matematiche

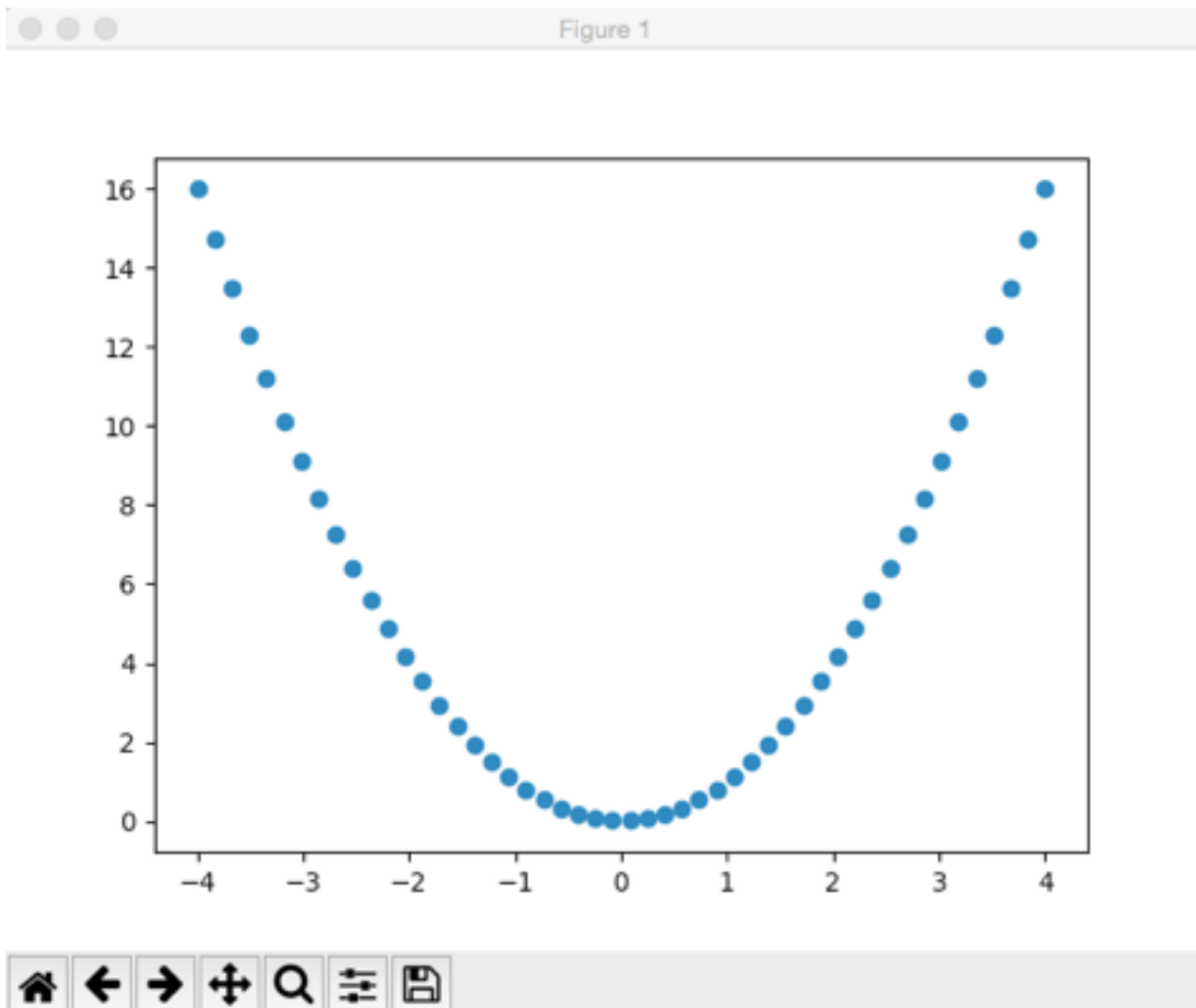
**Es.** graficare una parabola con  $x = [-4, 4]$

# PYTHON E IL MODULO NUMPY

Le liste viste nell'esempio precedente non supportano operazioni matematiche. il modulo numpy (di solito indicato con alias *np* quando viene importato) permette di leggere, manipolare e stampare facilmente vettori, matrici e, più in generale, tensori di qualunque ordine. Nonostante non faccia parte della libreria standard di Python, numpy è de facto lo standard per lavorare con oggetti di questo tipo.

Usando numpy, possiamo convertire una sequenza di numeri come quella dell'esempio precedente in un array a manipolarlo usando funzioni matematiche

**Es.** graficare una parabola con  $x=[-4, 4]$



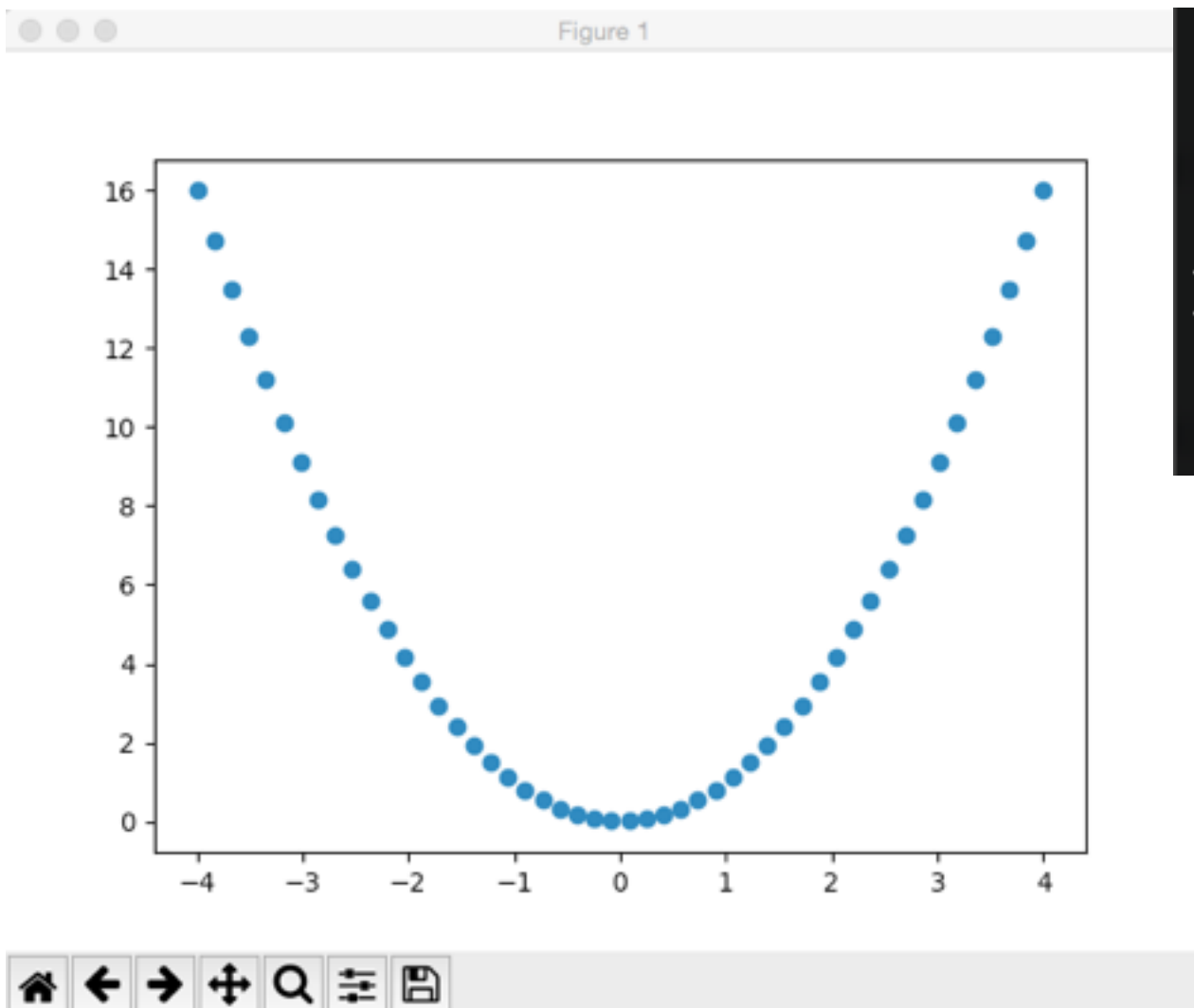
```
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(-4,4,50,endpoint=True)
y=x**2
plt.plot(x,y, 'o')
plt.show()
```

# PYTHON E IL MODULO NUMPY

Le liste viste nell'esempio precedente non supportano operazioni matematiche. il modulo numpy (di solito indicato con alias *np* quando viene importato) permette di leggere, manipolare e stampare facilmente vettori, matrici e, più in generale, tensori di qualunque ordine. Nonostante non faccia parte della libreria standard di Python, numpy è de facto lo standard per lavorare con oggetti di questo tipo.

Usando numpy, possiamo convertire una sequenza di numeri come quella dell'esempio precedente in un array a manipolarlo usando funzioni matematiche

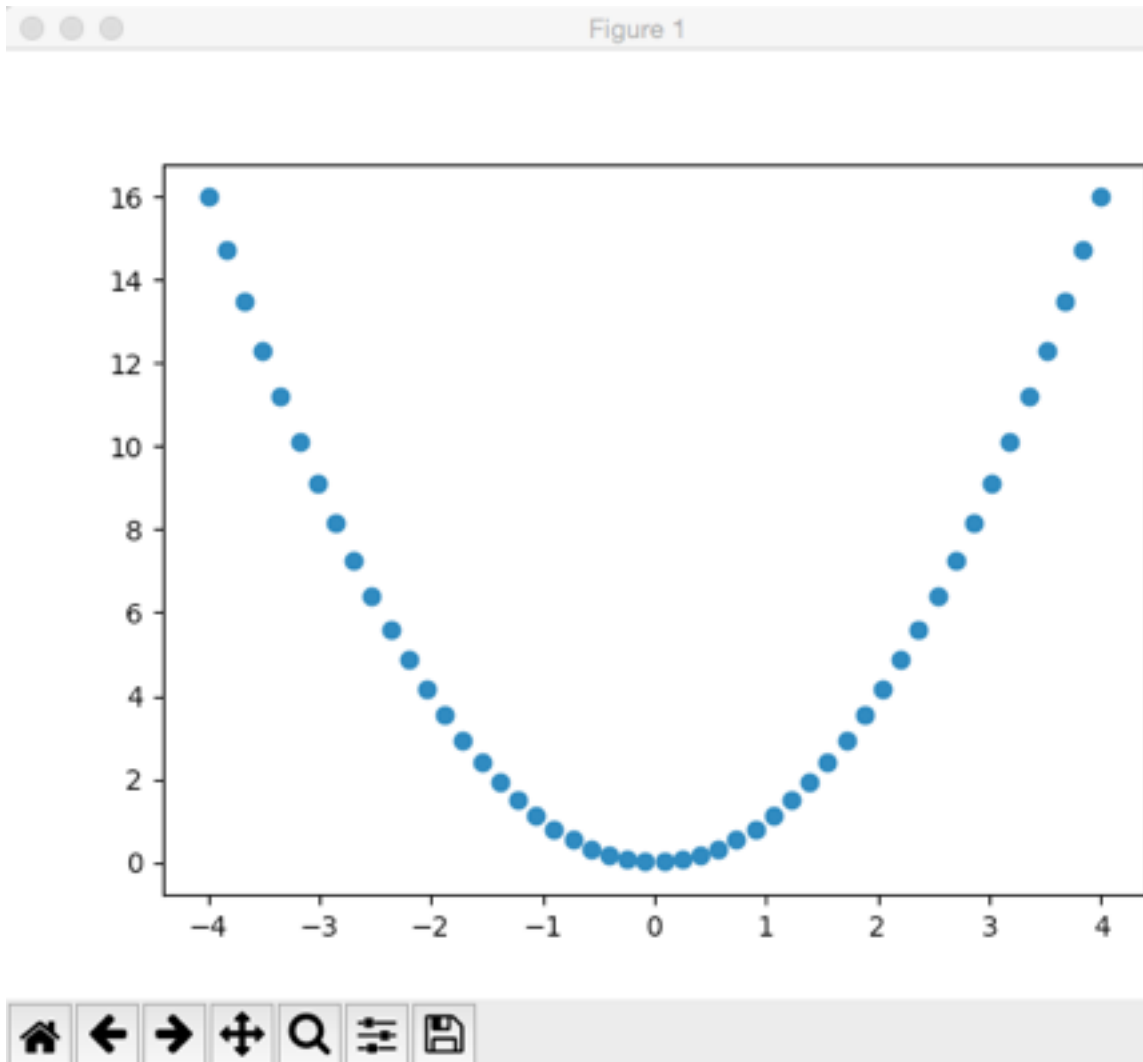
**Es.** graficare una parabola con  $x=[-4, 4]$



```
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(-4,4,50,endpoint=True)
y=x**2
plt.plot(x,y,'o')
plt.show()
```

# PYTHON E IL MODULO NUMPY

**Es.** graficare una parabola con  $x=[-4, 4]$



**provate a giocare con il codice, modificando il numero dei punti da graficare, e gli estremi del grafico**

```
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(-4,4,50,endpoint=True)
y=x**2
plt.plot(x,y,'o')
plt.show()
```

abbiamo usato la funzione `np.linspace()` di numpy per creare un array (vettore di numeri) chiamato `x` che contiene 50 valori compresi tra `[-4, 4]` (4 incluso perche' perche' abbiamo specificato `endpoint=True`.)

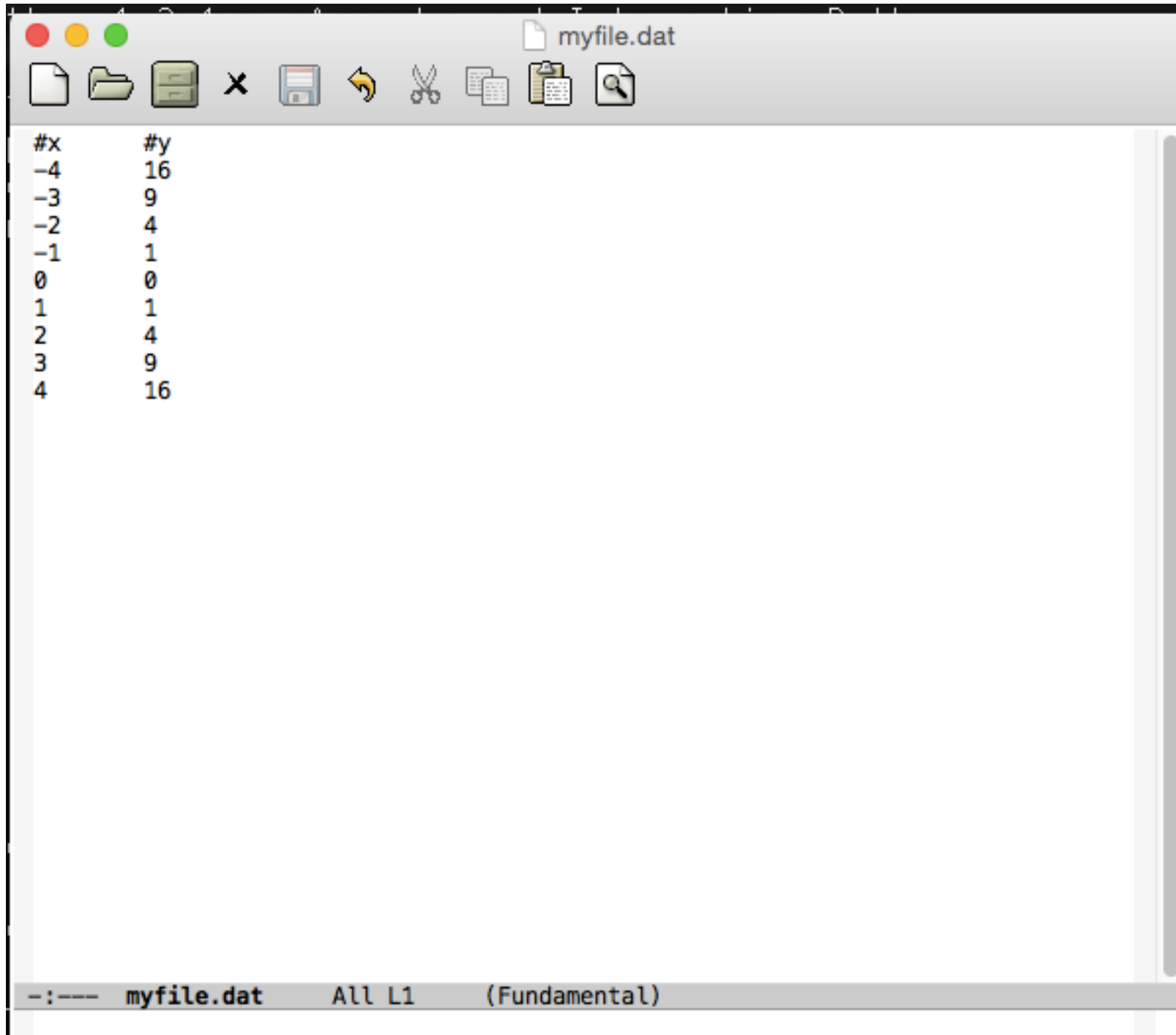
in questo modo si puo' creare un secondo array chiamato `y` della stessa lunghezza di `x`, che contiene valori che sono il quadrato dei valori contenuti in `x`.

Con `plt.plot(x,y)` grafichiamo il set di valori per l'asse `x` e `y`. Il decrittore `'o'` in `plt.plot()` dice a python di graficare dei cerchietti.

# PYTHON E IL MODULO NUMPY

Un'altra possibilita' e' quella di inizializzare un array numpy con valori letti da un file di testo che contiene i dati che vogliamo graficare

**Es.** creiamo un file di testo myfile.dat dove metteremo i seguenti numeri



```
#x      #y
-4      16
-3      9
-2      4
-1      1
0       0
1       1
2       4
3       9
4      16
```

La prima riga del file e' un commento quindi verra' ignorata da python. Le due colonne rappresentano l'asse x e y di un parabola che vogliamo graficare

# PYTHON E IL MODULO NUMPY

Un'altra possibilit  e' quella di inizializzare un array numpy con valori letti da un file di testo che contiene i dati che vogliamo graficare

**Es.** scriviamo su un file chiamato ex4.py il seguente codice e lanciamolo come visto prima

```
import matplotlib.pyplot as plt
import numpy as np
x, y = np.loadtxt('myfile.dat', unpack=True, usecols=(0, 1))
plt.plot(x, y, '-o')
plt.savefig('grafico.png')
plt.show()
```

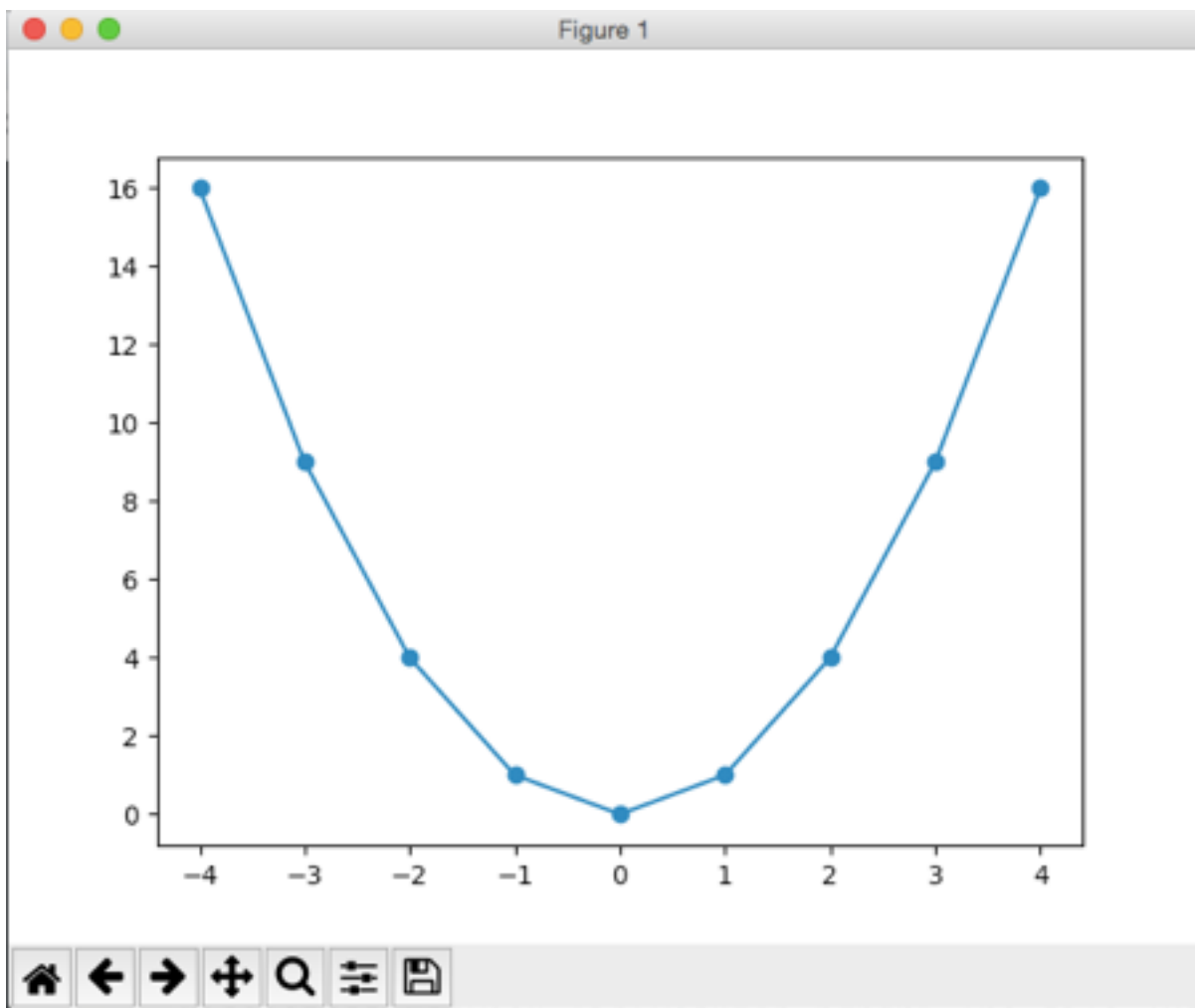


# PYTHON E IL MODULO NUMPY

Un'altra possibilit  e' quella di inizializzare un array numpy con valori letti da un file di testo che contiene i dati che vogliamo graficare

**Es.** scriviamo su un file chiamato ex4.py il seguente codice ed eseguiamolo

```
import matplotlib.pyplot as plt
import numpy as np
x, y = np.loadtxt('myfile.dat', unpack=True, usecols=(0, 1))
plt.plot(x, y, '-o')
plt.savefig('grafico.png')
plt.show()
```



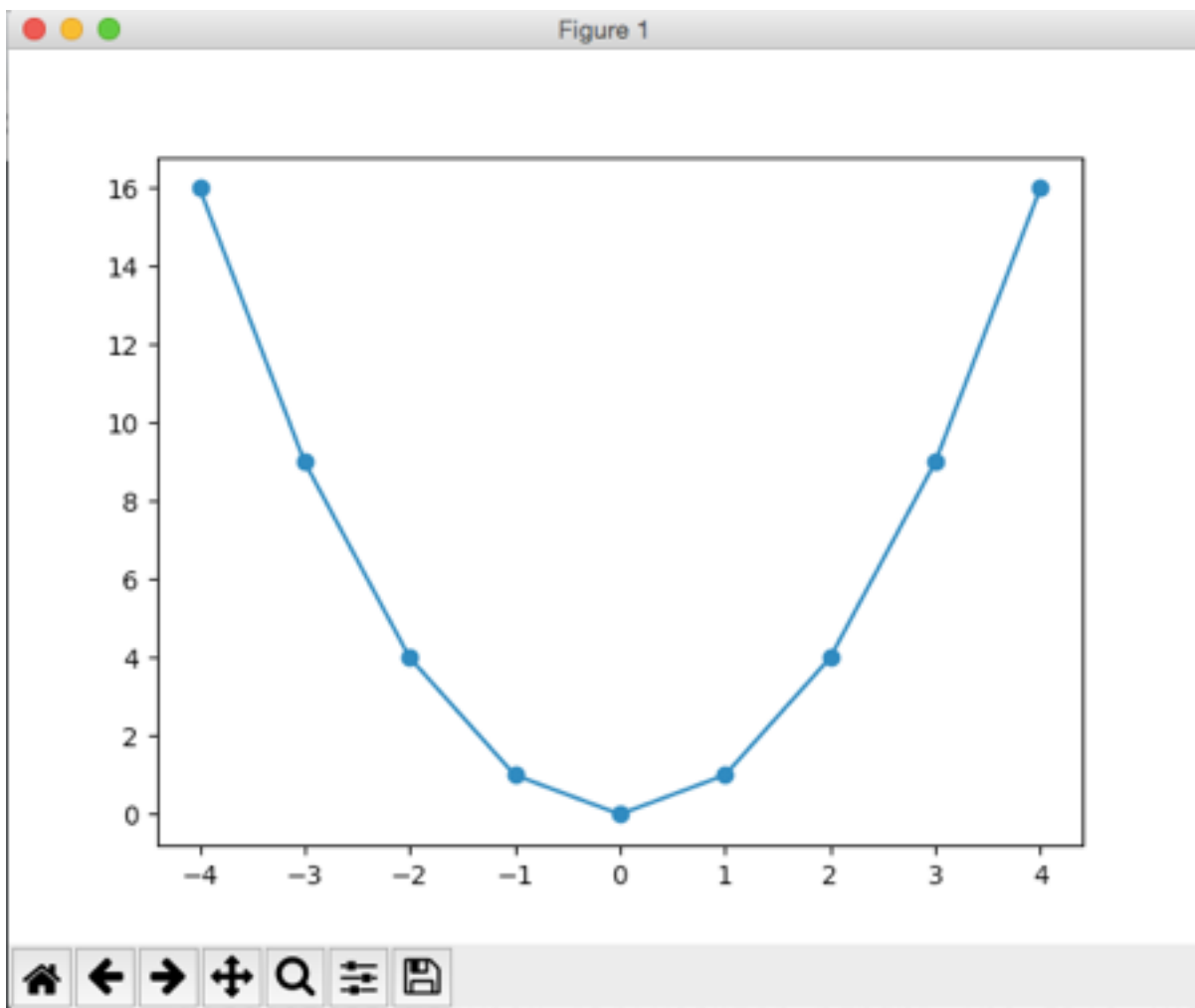
- `np.loadtxt('myfile.dat')` permette di inizializzare un array numpy con valori letti dal file `myfile.dat`. In questo caso col comando `x, y = np.loadtxt('myfile.dat', unpack=True, usecols=(0, 1))`, inizializzo `x` ed `y` che conterranno i dati della prima e seconda colonna del file.

# PYTHON E IL MODULO NUMPY

Un'altra possibilit  e' quella di inizializzare un array numpy con valori letti da un file di testo che contiene i dati che vogliamo graficare

**Es.** scriviamo su un file chiamato ex4.py il seguente codice ed eseguiamolo

```
import matplotlib.pyplot as plt
import numpy as np
x, y = np.loadtxt('myfile.dat', unpack=True, usecols=(0, 1))
plt.plot(x, y, '-o')
plt.savefig('grafico.png')
plt.show()
```



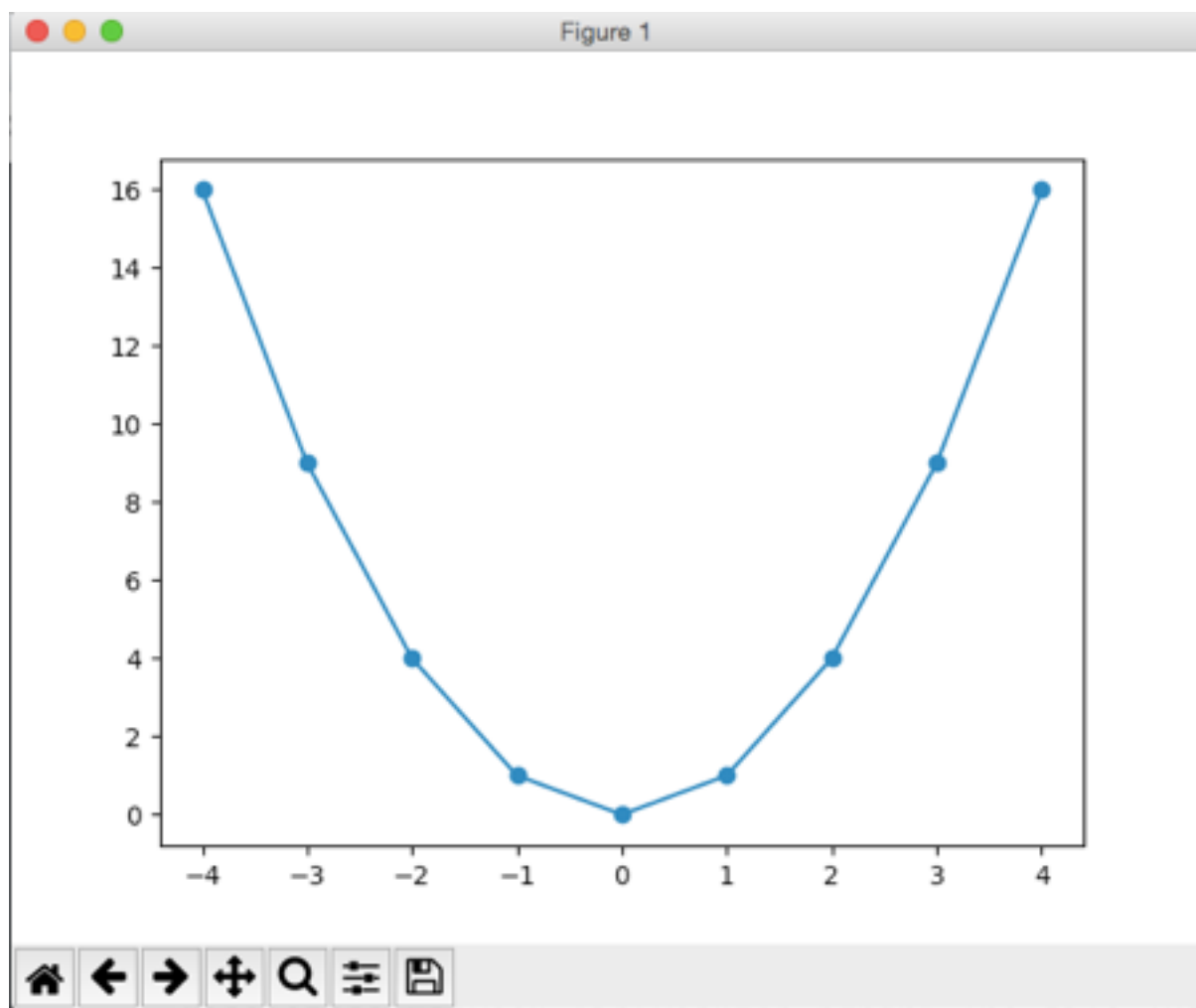
- `np.loadtxt('myfile.dat')` permette di inizializzare un array numpy con valori letti dal file `myfile.dat`. In questo caso col comando `x, y = np.loadtxt('myfile.dat', unpack=True, usecols=(0, 1))`, inizializzo `x` ed `y` che conterranno i dati della prima e seconda colonna del file.
- `unpack=True` serve a “spacchettare” i dati in modo da associare a `x` e `y` ciascuno la propria colonna.

# PYTHON E IL MODULO NUMPY

Un'altra possibilit  e' quella di inizializzare un array numpy con valori letti da un file di testo che contiene i dati che vogliamo graficare

**Es.** scriviamo su un file chiamato ex4.py il seguente codice ed eseguiamolo

```
import matplotlib.pyplot as plt
import numpy as np
x, y = np.loadtxt('myfile.dat', unpack=True, usecols=(0, 1))
plt.plot(x, y, '-o')
plt.savefig('grafico.png')
plt.show()
```



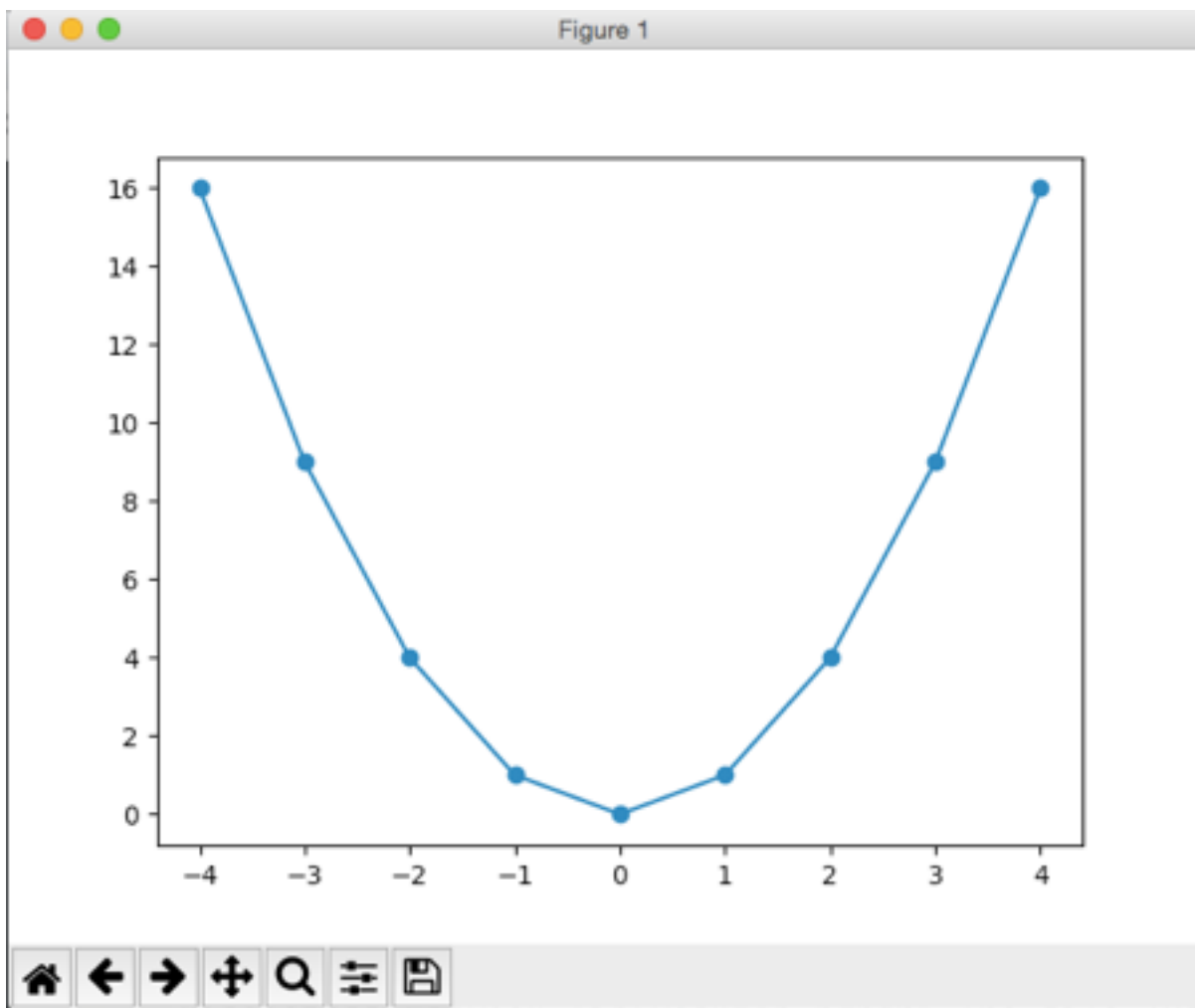
- `np.loadtxt('myfile.dat')` permette di inizializzare un array numpy con valori letti dal file `myfile.dat`. In questo caso col comando `x, y = np.loadtxt('myfile.dat', unpack=True, usecols=(0, 1))`, inizializzo `x` ed `y` che conterranno i dati della prima e seconda colonna del file.
- `unpack=True` serve a “spacchettare” i dati in modo da associare a `x` e `y` ciascuno la propria colonna.
- `usecols=(0,1)` serve ad associare a `x` e `y` le colonne 0 e 1 del file. In questo caso e' ridondante e non serve ma nel caso ci fosse una terza colonna, dovremmo specificare a quale colonne corrispondono i vettori `x` e `y`

# PYTHON E IL MODULO NUMPY

Un'altra possibilit  e' quella di inizializzare un array numpy con valori letti da un file di testo che contiene i dati che vogliamo graficare

**Es.** scriviamo su un file chiamato ex4.py il seguente codice ed eseguiamolo

```
import matplotlib.pyplot as plt
import numpy as np
x, y = np.loadtxt('myfile.dat', unpack=True, usecols=(0, 1))
plt.plot(x, y, '-o')
plt.savefig('grafico.png')
plt.show()
```



- `np.loadtxt('myfile.dat')` permette di inizializzare un array numpy con valori letti dal file `myfile.dat`. In questo caso col comando `x, y = np.loadtxt('myfile.dat', unpack=True, usecols=(0, 1))`, inizializzo `x` ed `y` che conterranno i dati della prima e seconda colonna del file.
- `unpack=True` serve a “spacchettare” i dati in modo da associare a `x` e `y` ciascuno la propria colonna.
- `usecols=(0,1)` serve ad associare a `x` e `y` le colonne 0 e 1 del file. In questo caso e' ridondante e non serve ma nel caso ci fosse una terza colonna, dovremmo specificare a quale colonne corrispondono i vettori `x` e `y`
- `plt.savefig('grafico.png')` salva la figura graficata con il nome `grafico.png`

# PYTHON E IL MODULO NUMPY

Un'altra possibilit  e' quella di inizializzare un array numpy con valori letti da un file di testo che contiene i dati che vogliamo graficare

**Es.** scriviamo su un file chiamato ex4.py il seguente codice ed eseguiamolo

```
import matplotlib.pyplot as plt
import numpy as np
x, y = np.loadtxt('myfile.dat', unpack=True, usecols=(0, 1))
plt.plot(x, y, '-o')
plt.savefig('grafico.png')
plt.show()
```

**Provate ad aggiungere una terza colonna nel file myfile.dat (ad esempio uguale alla prima colonna) e provate ad importarla al posto della prima o della seconda colonna.**

**Provate anche ad importare tutte e tre le colonne usando un terzo vettore z nello script.**

**Infine provate a graficare x,y,z assieme per scoprire cosa succede**

- np.loadtxt('myfile.dat') permette di inizializzare un array numpy con valori letti dal file myfile.dat. In questo caso col comando x, y = np.loadtxt('myfile.dat', unpack=True, usecols=(0, 1)), inizializzo x ed y che conterranno i dati della prima e seconda colonna del file.
- unpack=True serve a “spacchettare” i dati in modo da associare a x e y ciascuno la propria colonna.
- usecols=(0,1) serve ad associare a x e y le colonne 0 e 1 del file. In questo caso e' ridondante e non serve ma nel caso ci fosse una terza colonna, dovremmo specificare a quale colonne corrispondono i vettori x e y
- plt.savefig('grafico.png') salva la figura grafica con il nome grafico.png

# PYTHON E IL MODULO NUMPY

Un'altra possibilit  e' quella di inizializzare un array numpy con valori letti da un file di testo che contiene i dati che vogliamo graficare

**Es.** scriviamo su un file chiamato ex4.py il seguente codice ed eseguiamolo

```
import matplotlib.pyplot as plt
import numpy as np
x, y = np.loadtxt('myfile.dat', unpack=True, usecols=(0, 1))
plt.plot(x,y, '-o')
plt.savefig('grafico.png')
plt.show()
```

**Ricordatevi di cambiare nome al file .png prima di rilanciare lo script altrimenti sovrascrivete le immagini**

**Provate ad aggiungere una terza colonna nel file myfile.dat (ad esempio uguale alla prima colonna) e provate ad importarla al posto della prima o della seconda colonna.**

**Provate anche ad importare tutte e tre le colonne usando un terzo vettore z nello script.**

**Infine provate a graficare x,y,z assieme per scoprire cosa succede**

- np.loadtxt('myfile.dat') permette di inizializzare un array numpy con valori letti dal file myfile.dat. In questo caso col comando x, y = np.loadtxt('myfile.dat',unpack=True, usecols=(0, 1)), inizializzo x ed y che conterranno i dati della prima e seconda colonna del file.
- unpack=True serve a “spacchettare” i dati in modo da associare a x e y ciascuno la propria colonna.
- usecols=(0,1) serve ad associare a x e y le colonne 0 e 1 del file. In questo caso e' ridondante e non serve ma nel caso ci fosse una terza colonna, dovremmo specificare a quale colonne corrispondono i vettori x e y
- plt.savefig('grafico.png') salva la figura grafica con il nome grafico.png

# QUALCHE INFORMAZIONE IN PIU' SU PYPLOT

pyplot con le sue funzioni e definizioni emula l'ambiente MATLAB in maniera interattiva

Ogni funzione di pyplot agisce su una singola figura

- creando una figura ✓
- definendo un'area all'interno della figura dove comparira' il grafico ✓
- disegnando il dei grafici all'interno di questa area ✓
- decorando il grafico con elementi grafici ed etichette





# PERSONALIZZAZIONE DEI GRAFICI

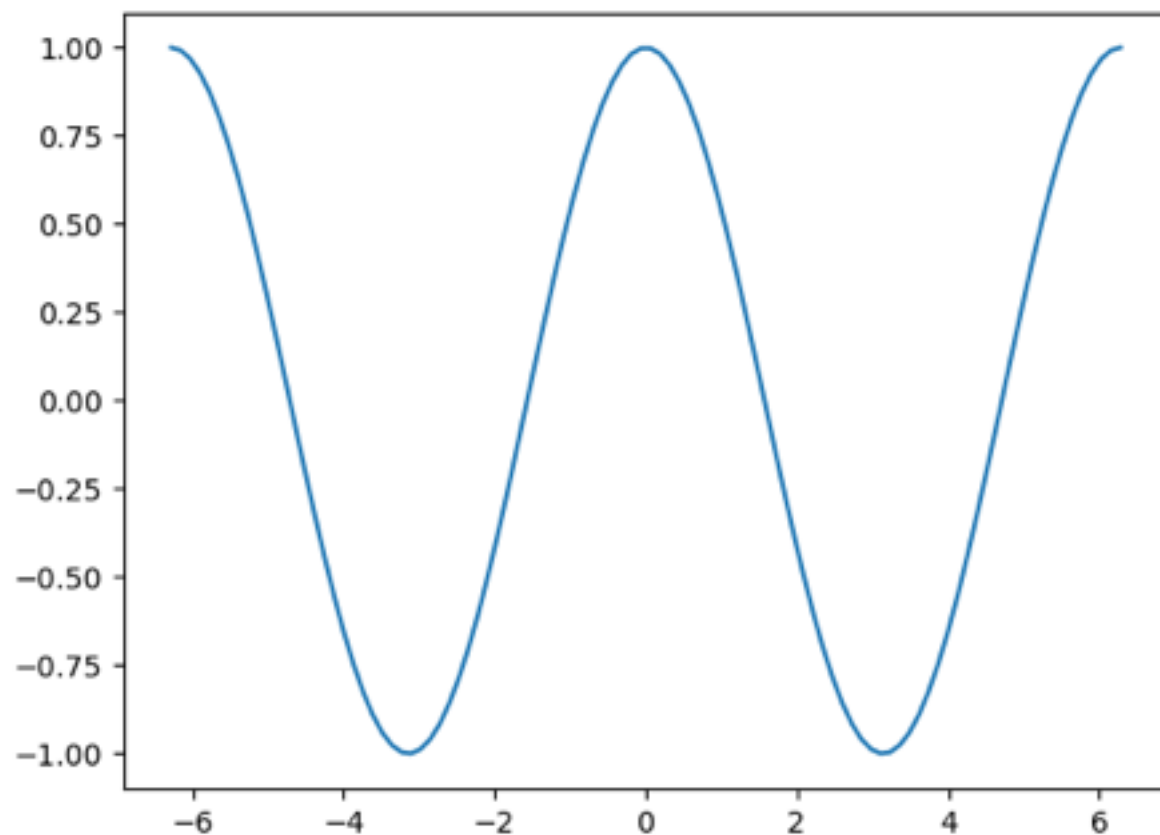
## STILE DELLA LINEA



Es. copiare il seguente script in un file chiamato ex5.py e lanciare lo script

```
#lo script grafica la funzione cos tra -2pi a 2pi
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(-2*np.pi,2*np.pi,100,endpoint=True)
y=np.cos(x)
plt.plot(x,y,linestyle='-') ←
plt.savefig('coseno.png')
plt.show()
```

per personalizzare lo stile della linea possiamo modificare l'argomento passaro a linestyle usando, al posto di '-' le seguenti opzioni. Il comando plt.plot si puo' anche accorciare nel seguente modo

plt.plot(x,y, '-')



	solid line style
	dashed line style
	dash-dot line style
	dotted line style



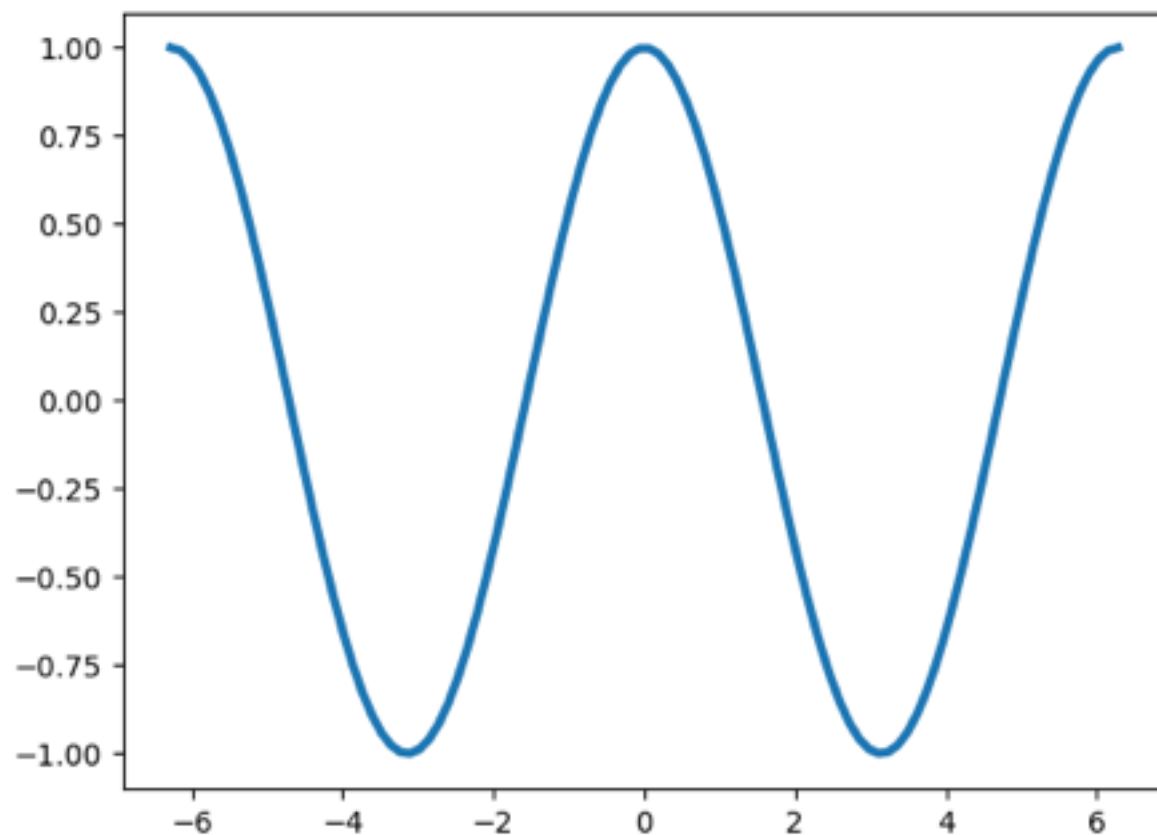
# PERSONALIZZAZIONE DEI GRAFICI

## STILE DELLA LINEA

Es. copiare il seguente script in un file chiamato ex5.py e lanciare lo script

```
#lo script grafica la funzione cos tra -2pi a 2pi
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(-2*np.pi,2*np.pi,100,endpoint=True)
y=np.cos(x)
plt.plot(x,y,linewidth=3) ←
plt.savefig('coseno.png')
plt.show()
```

Lo spessore della linea si può modificare attraverso il campo `linewidth=` seguito da un numero intero positivo in `plt.plot`. Il valore 0 corrisponde ad una linea invisibile



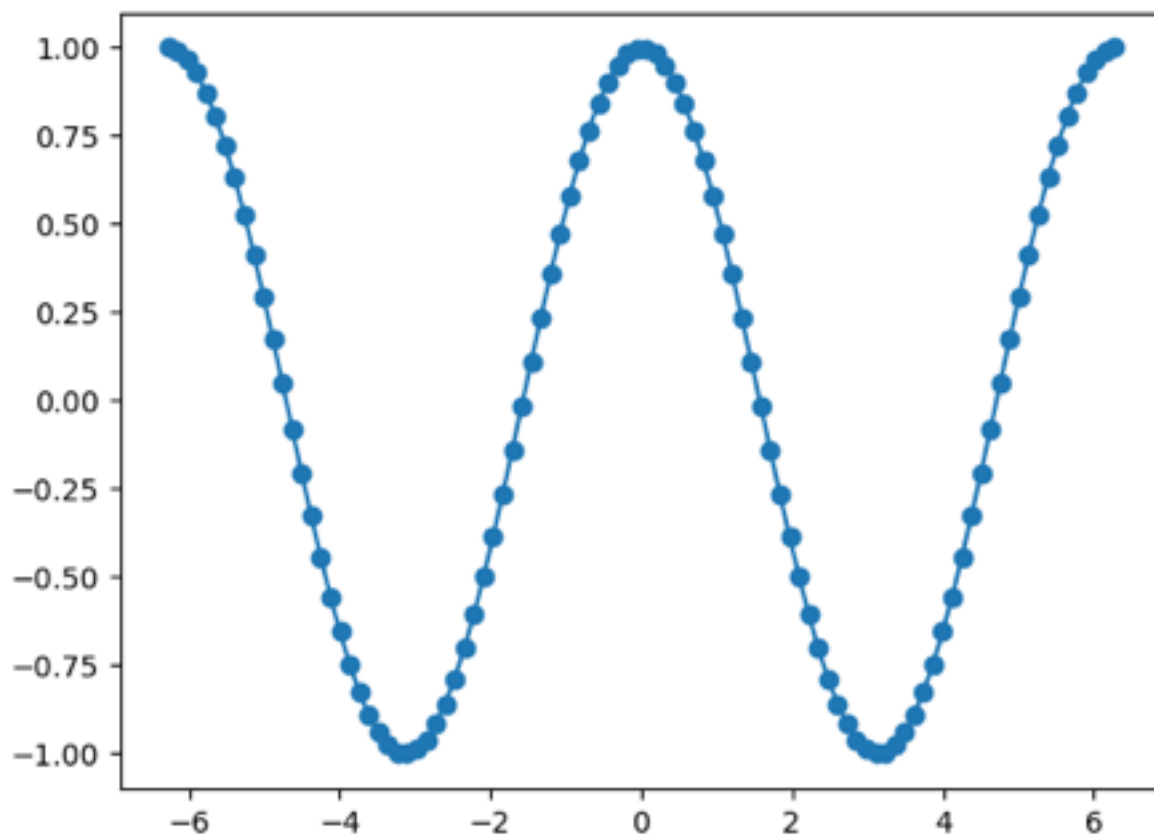
# PERSONALIZZAZIONE DEI GRAFICI

## SIMBOLI E LINEE

Es. copiare il seguente script in un file chiamato ex5.py e lanciare lo script

```
#lo script grafica la funzione cos tra -2pi a 2pi
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(-2*np.pi,2*np.pi,100,endpoint=True)
y=np.cos(x)
plt.plot(x,y,marker='o') ←
plt.savefig('coseno.png')
plt.show()
```

per personalizzare lo stile del grafico con simboli o linee si può usare il campo `marker=` all'interno di `plt.plot()`. Le opzioni per marker sono



'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker

's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond marker
' '	vline marker
'_'	hline marker

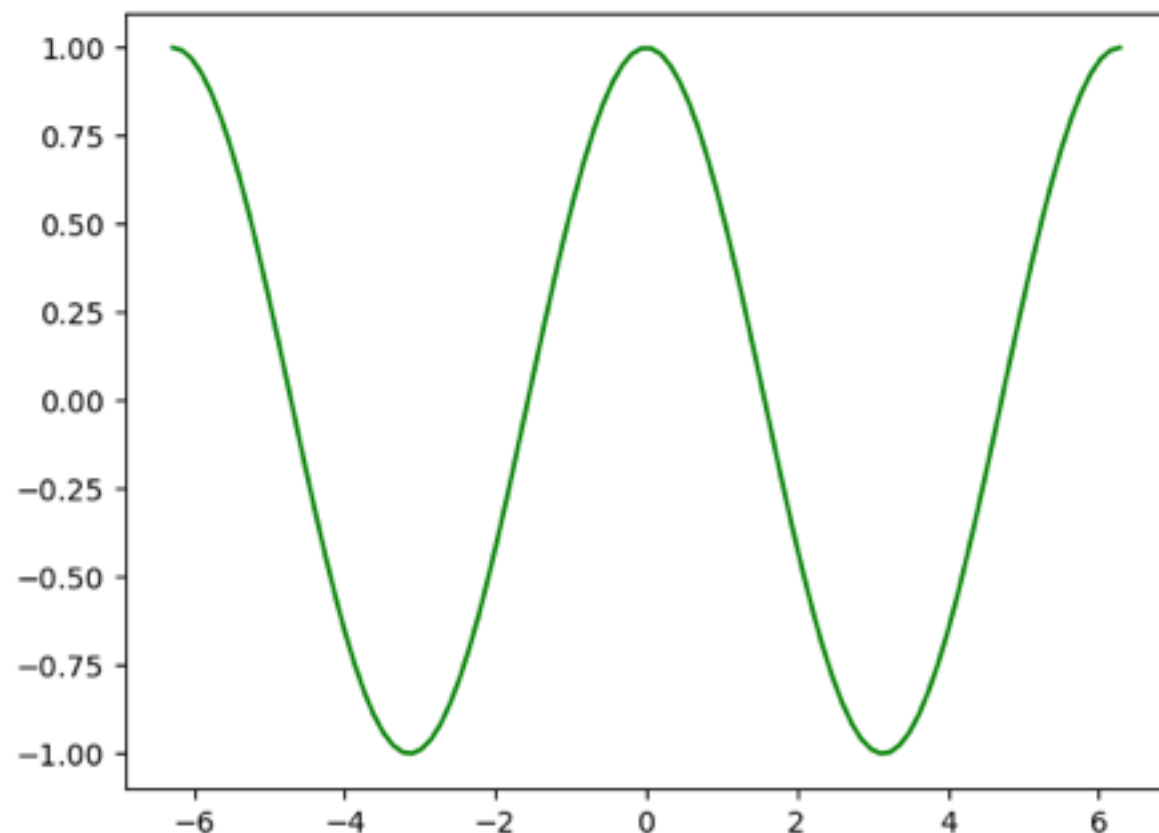
- Tips: aggiungendo anche il campo `linestyle=''` in `plt.plot()` si ottengono solo i simboli

# PERSONALIZZAZIONE DEI GRAFICI

## COLORI

Es. copiare il seguente script in un file chiamato ex5.py e lanciare lo script

```
#lo script grafica la funzione cos tra -2pi a 2pi
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(-2*np.pi,2*np.pi,100,endpoint=True)
y=np.cos(x)
plt.plot(x,y,color='g') ←
plt.savefig('coseno.png')
plt.show()
```



- per personalizzare il colore del grafico si può usare il campo `color=` all'interno di `plt.plot()`. Le opzioni per `color` sono

'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

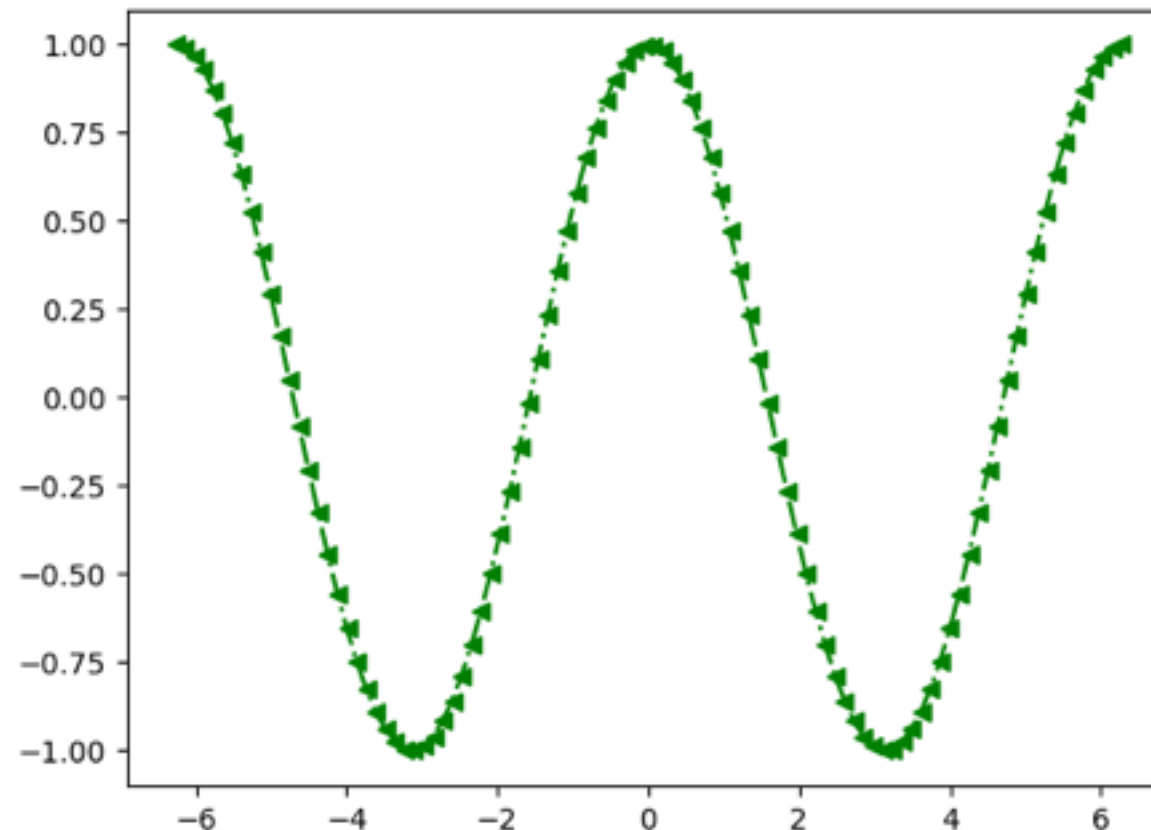
- in alternativa tra apici si può direttamente inserire il codice esadecimale del numero es. rosso='FF0000'

# PERSONALIZZAZIONE DEI GRAFICI

## SCRITTURE A CONFRONTO

Invece di ricordarsi i nomi di tutti i campi (linestyle, marker e color) una notazione piu' compatta e' quella di scrivere tra apici, tutto insieme, colore, stile del marker e stile della linea. Si puo' anche solamente mettere il colore della linea e lo stile, oppure lo stile del simbolo con il colore (in modo da avere solo simboli)

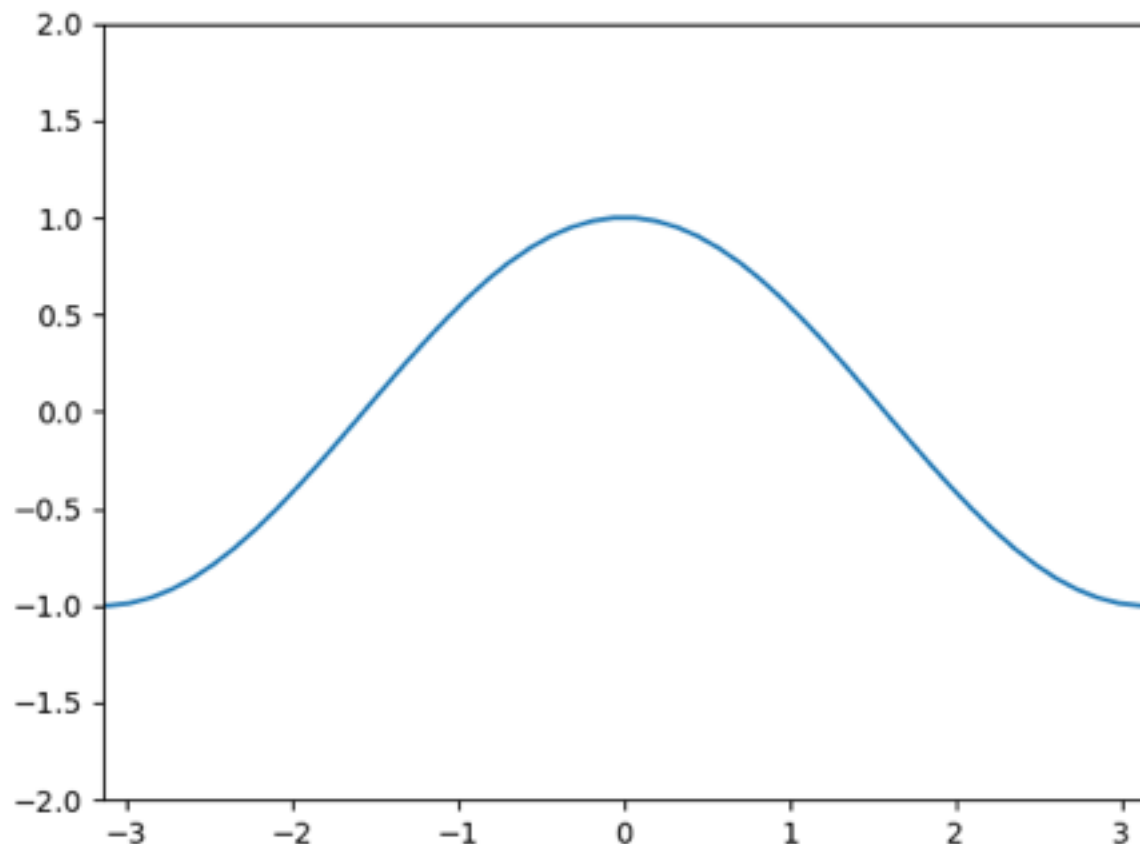
```
#lo script grafica la funzione cos tra -2pi a 2pi
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(-2*np.pi,2*np.pi,100,endpoint=True)
y=np.cos(x)
plt.plot(x,y, 'g<-.') ←
plt.savefig('coseno.png')
plt.show()
```



# PERSONALIZZAZIONE DEI GRAFICI

## IMPOSTARE LA DIMENSIONE DEGLI ASSI

```
#lo script grafica la funzione cos
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(-2*np.pi,2*np.pi,100,endpoint=True)
y=np.cos(x)
plt.plot(x,y)
plt.xlim(-np.pi,np.pi)
plt.ylim(-2.,2.)
plt.savefig('coseno.png')
plt.show()
```



dopo il comando **plt.plot()** si puo' usare:

`plt.xlim(xmin, xmax)`

e

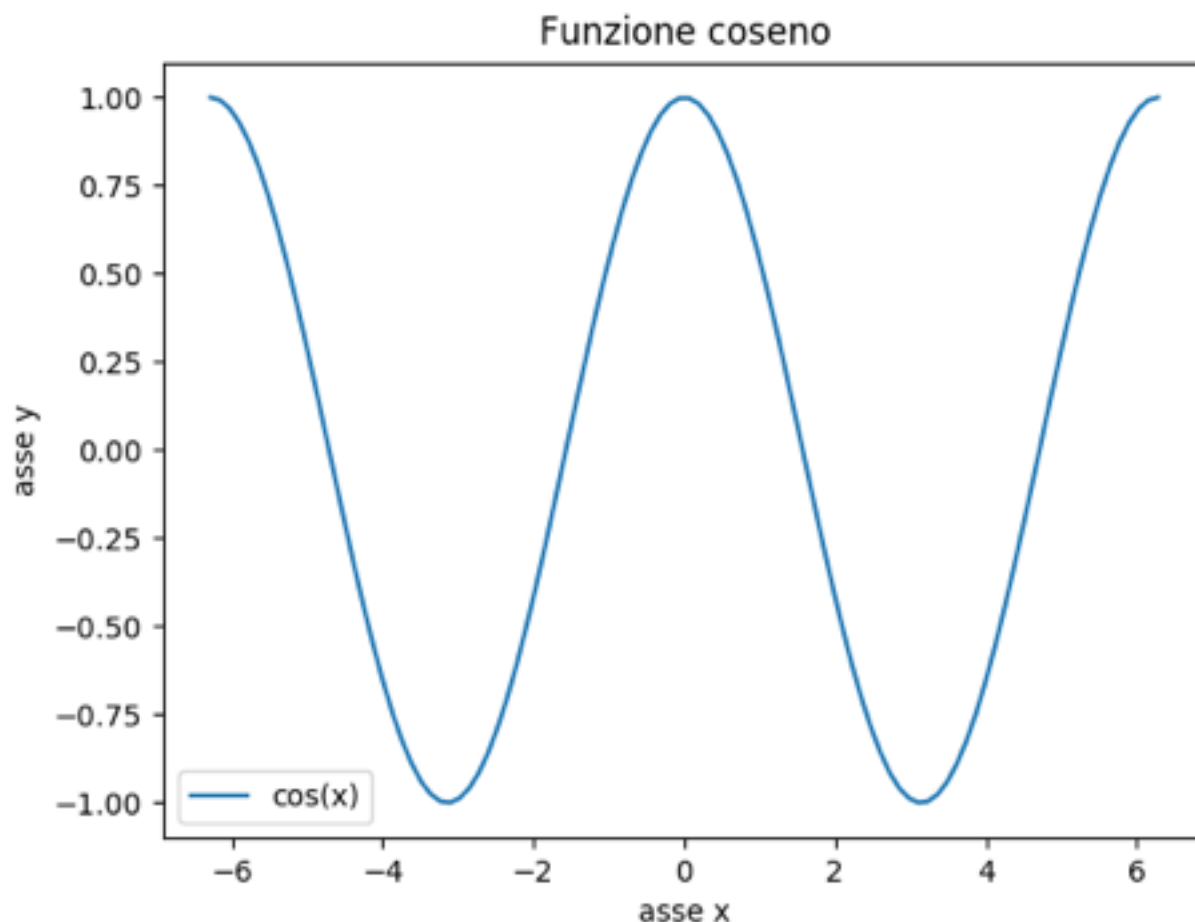
`plt.ylim(ymin, ymax)`

per ridimensionare il range dell'asse x e dell'asse y a piacimento. In questo esempio abbiamo scelto x tra  $[-\pi, \pi]$  e y tra  $[-2, 2]$

# PERSONALIZZAZIONE DEI GRAFICI

## TITOLO, ETICHETTE DEGLI ASSI E LEGENDA

```
#lo script grafica la funzione cos
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(-2*np.pi,2*np.pi,100,endpoint=True)
y=np.cos(x)
plt.title('Funzione coseno')
plt.xlabel('asse x')
plt.ylabel('asse y')
plt.plot(x,y,label='cos(x)')
plt.legend()
plt.savefig('coseno.png')
plt.show()
```



**prima del comando `plt.plot()` si può usare:**

`plt.title('...')` per far apparire un titolo in alto nel grafico

`plt.xlabel('...')` per far apparire il nome dell'asse x

`plt.ylabel('...')` per far apparire il nome dell'asse y

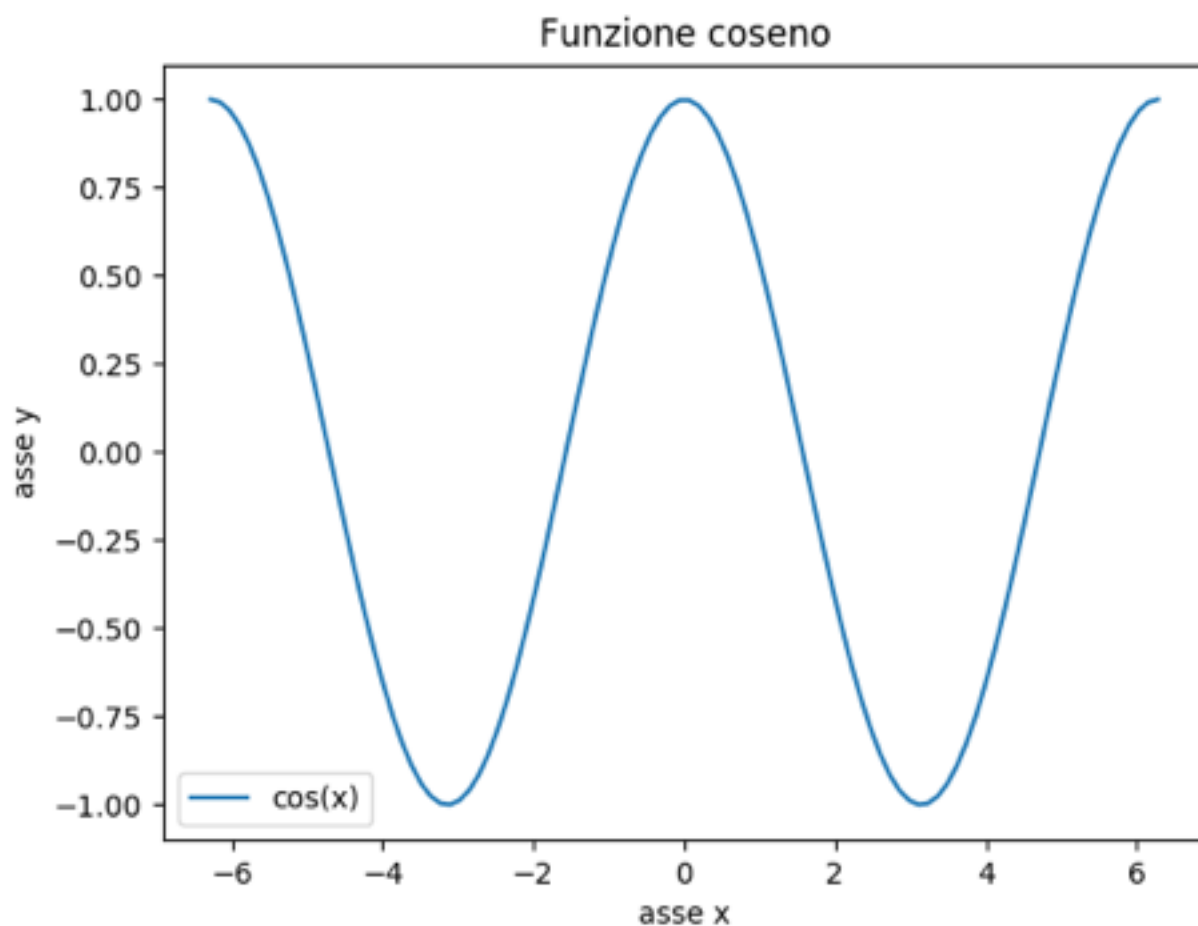
il comando `label` in `plt.plot` permette di creare la legenda mentre `plt.legend()` la fa apparire nel grafico



# PERSONALIZZAZIONE DEI GRAFICI

## TITOLO, ETICHETTE DEGLI ASSI E LEGENDA

```
#lo script grafica la funzione cos
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(-2*np.pi,2*np.pi,100,endpoint=True)
y=np.cos(x)
plt.title('Funzione coseno')
plt.xlabel('asse x')
plt.ylabel('asse y')
plt.plot(x,y,label='cos(x)')
plt.legend()
plt.savefig('coseno.png')
plt.show()
```



**prima del comando `plt.plot()` si può usare:**

`plt.title('...')` per far apparire un titolo in alto nel grafico

`plt.xlabel('...')` per far apparire il nome dell'asse x

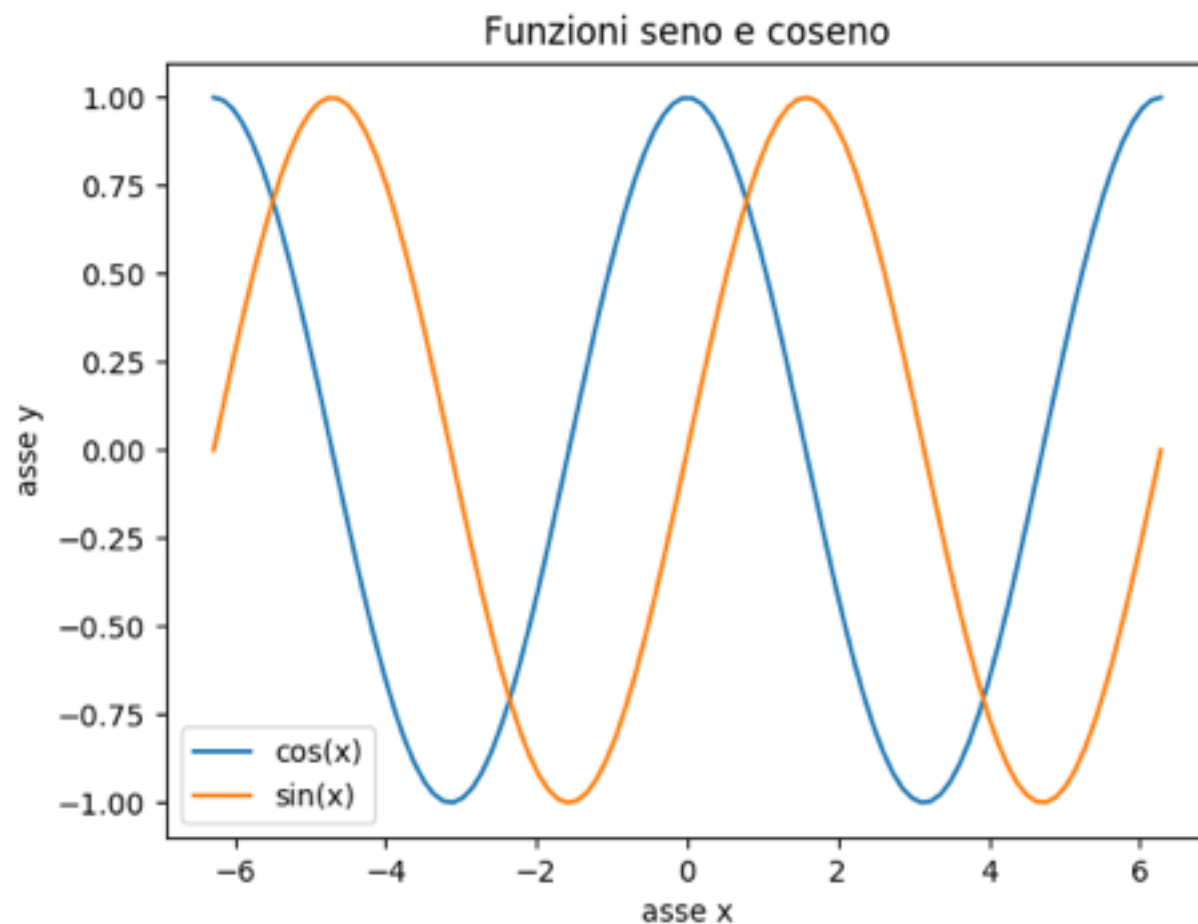
`plt.ylabel('...')` per far apparire il nome dell'asse y

il comando `label` in `plt.plot` permette di creare la legenda mentre `plt.legend()` la fa apparire nel grafico

# PERSONALIZZAZIONE DEI GRAFICI

## GRAFICI CON PIU' CURVE

```
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(-2*np.pi,2*np.pi,100,endpoint=True)
y=np.cos(x)
z=np.sin(x)
plt.title('Funzioni seno e coseno')
plt.xlabel('asse x')
plt.ylabel('asse y')
plt.plot(x,y,label='cos(x)')
plt.plot(x,z,label='sin(x)')
plt.legend()
plt.savefig('sincos.png')
plt.show()
```



in questo esempio grafico le funzioni seno e coseno usando due volte l'istruzione `plt.plot()` per i set di dati relativi alle due funzioni.



# PERSONALIZZAZIONE DEI GRAFICI

## ESERCIZIO

Si grafichi funzione  $y(x)=x^3$  con  $x$   $[-5.25, 5.25]$  (con l'estremo destro incluso). Il grafico deve contenere una legenda, un titolo e delle etichette sull'asse  $x$  e  $y$ . Il numero di punti con cui la curva deve essere rappresentata è 100. Il range lungo l'asse  $x$  del grafico deve essere  $[-5.5, 5.5]$  mentre sull'asse  $y$   $[-200.0, 200.0]$ . Giocare con gli stili, colori, simboli ecc....e salvare il grafico come un file .png dal nome cubic.png