Matplotlib – Part 1

Part 1

> The first and last lines of code mostly will be

import matplotlib.pyplot as plt ---> first line

plt.show() ----> code to display plots

> There are two methods to plot in matplotlib - one is functional method and other is object oriented method, we will discuss functional first but will shift to object oriented one since its the better way

## Functional way -

import matplotlib.pyplot as plt

import numpy as np

x = np.linspace(0,5,11)

y = x \*\* 2

# Functional way

plt.plot(x,y) -------> code to draw a plot

plt.xlabel("Numbers") -------> code to specify x label

plt.ylabel("Squares") -------> code to specify y label

plt.title("Numbers & Squares plotted") -------> code to specify title

plt.show()

> Creating sub-plots on the same canvas

import matplotlib.pyplot as plt

import numpy as np

x = np.linspace(0,5,11)

y = x \*\* 2

# Functional way

plt.subplot(1,2,1) #----> (number of rows, number of columns, plot on which we want to work)

plt.plot(x,y,"r") #----> "r" specifies red colour line

plt.subplot(1,2,2)

plt.plot(y,x,"b")

plt.show()

## Object Oriented way –

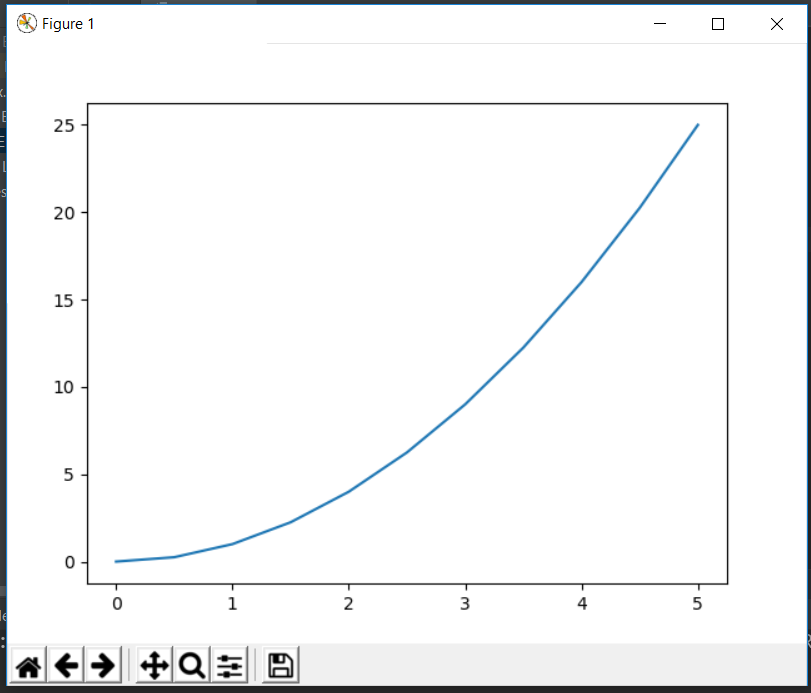
The main idea in using the more formal Object Oriented method is to create figure objects and then just call methods or attributes off of that object. This approach is nicer when dealing with a canvas that has multiple plots on it.

### Basic plot

We instantiate here figure object and add axes to it. The add\_axes method always take a list -dimensions [left, bottom, width, height] . All quantities are in fractions of figure width and height.

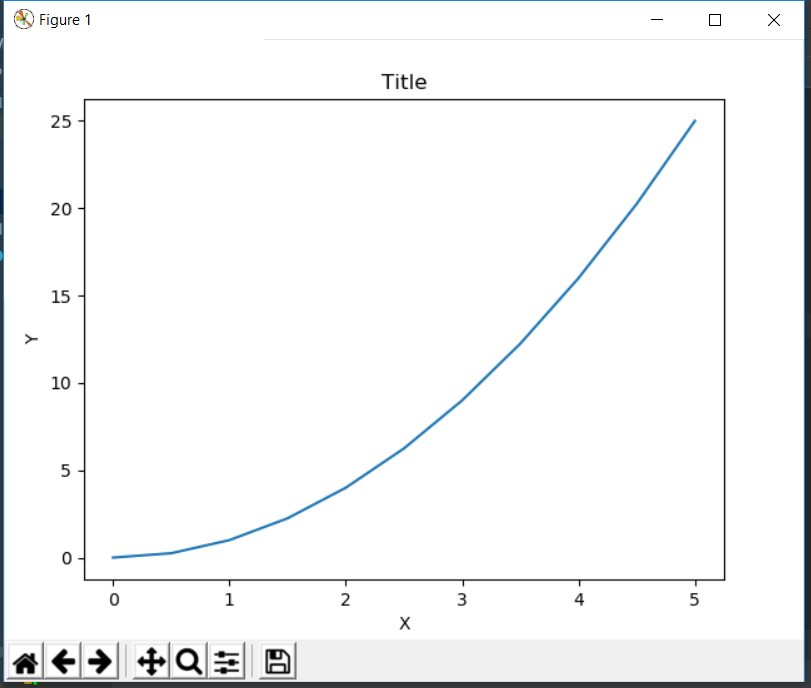
import matplotlib.pyplot as plt  
import numpy as np  
x = np.linspace(0,5,11)  
y = x \*\* 2  
  
fig = plt.figure()  
axes = fig.add\_axes([0.1, 0.1, 0.8, 0.8]) # left, bottom, width, height (range 0 to 1)

axes.plot(x,y)  
plt.show()



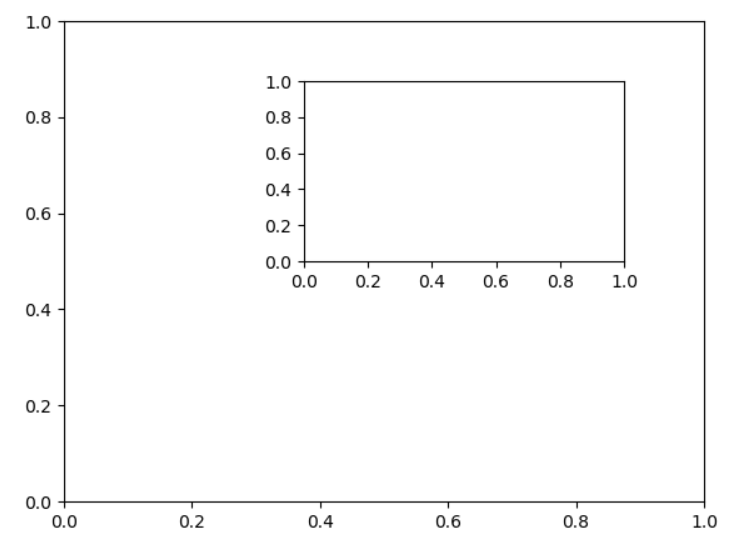
### Specifying labels and title

import matplotlib.pyplot as plt  
import numpy as np  
x = np.linspace(0,5,11)  
y = x \*\* 2  
  
fig = plt.figure()  
axes = fig.add\_axes([0.1,0.1,0.8,0.8])  
axes.plot(x,y)  
axes.set\_xlabel("X") # Notice the use of set\_ to begin methods  
axes.set\_ylabel("Y")  
axes.set\_title("Title")  
plt.show()

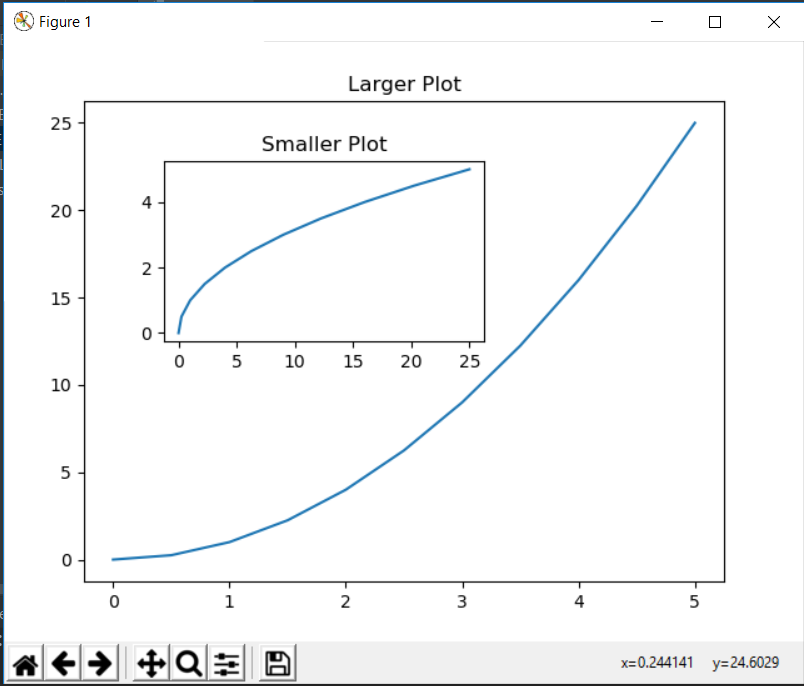


### Creating multi-plots and understanding add\_axes() method

import matplotlib.pyplot as plt  
import numpy as np  
x = np.linspace(0,5,11)  
y = x \*\* 2  
  
fig = plt.figure()  
axes1 = fig.add\_axes([0.1,0.1,0.8,0.8])  
axes2 = fig.add\_axes([0.4,0.5,0.4,0.3])  
plt.show()

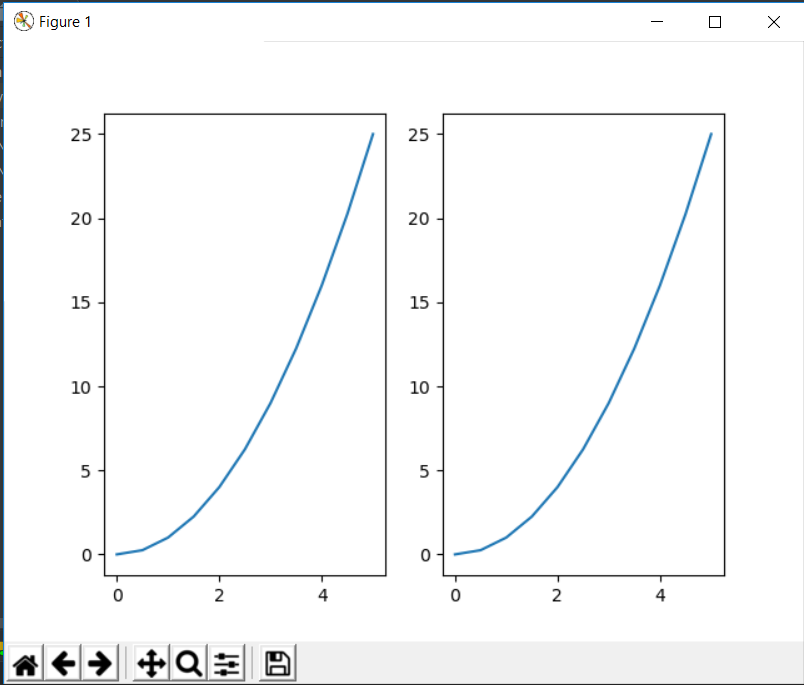


import matplotlib.pyplot as plt  
import numpy as np  
x = np.linspace(0,5,11)  
y = x \*\* 2  
  
fig = plt.figure()  
axes1 = fig.add\_axes([0.1,0.1,0.8,0.8])  
axes2 = fig.add\_axes([0.2,0.5,0.4,0.3])  
  
axes1.plot(x,y)  
axes1.set\_title("Larger Plot")  
axes2.plot(y,x)  
axes2.set\_title("Smaller Plot")  
plt.show()



### Creating subplots using object oriented way

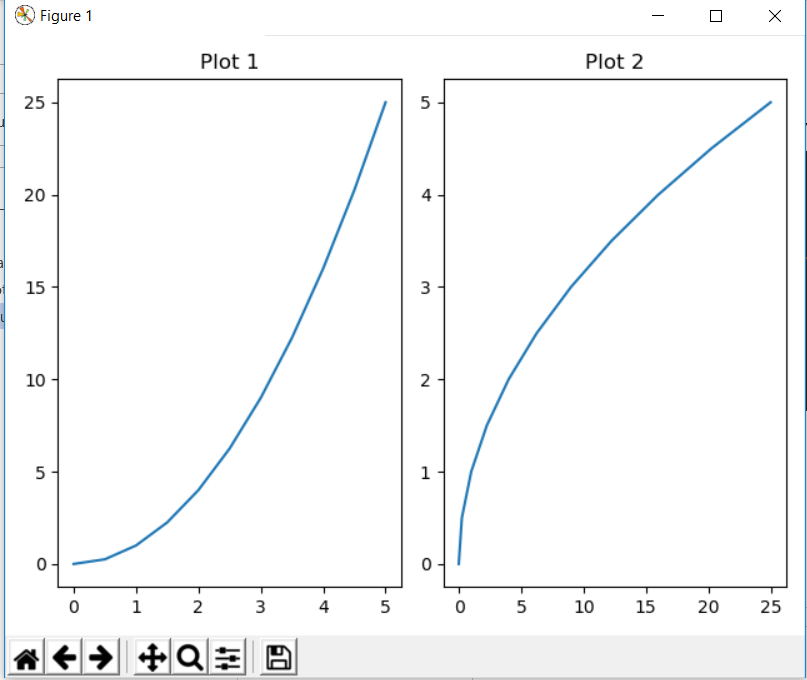
import matplotlib.pyplot as plt  
import numpy as np  
x = np.linspace(0,5,11)  
y = x \*\* 2  
  
fig,axes = plt.subplots(nrows=1,ncols=2)  
for current\_ax in axes:  
 current\_ax.plot(x,y)  
plt.show()



While doing tuple unpacking on the line – fig,axes = …., we can see that axes is an array of axes and we can iterate over it

We can also access this array by indexing as below

import matplotlib.pyplot as plt  
import numpy as np  
x = np.linspace(0,5,11)  
y = x \*\* 2  
  
fig,axes = plt.subplots(nrows=1,ncols=2)  
axes[0].plot(x,y)  
axes[0].set\_title("Plot 1")  
  
axes[1].plot(y,x)  
axes[1].set\_title("Plot 2")  
  
plt.tight\_layout() # this line take care of overlapping plots  
plt.show()

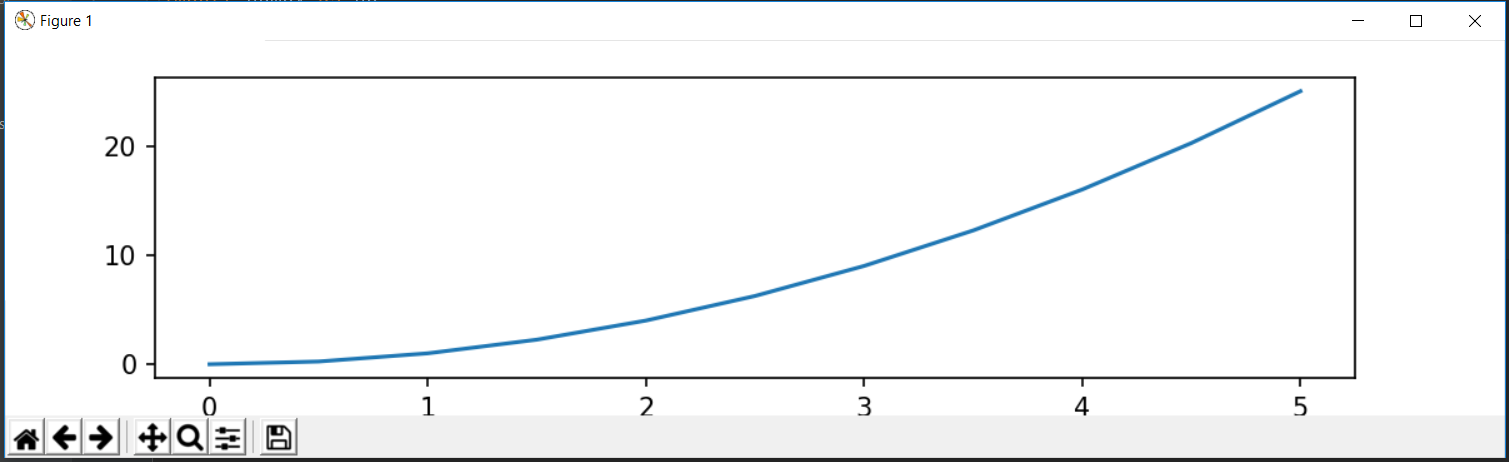


## Figure Size, Aspect Ratio and DPI

* figsize is a tuple of the width and height of the figure in inches
* dpi is the dots-per-inch (pixel per inch).

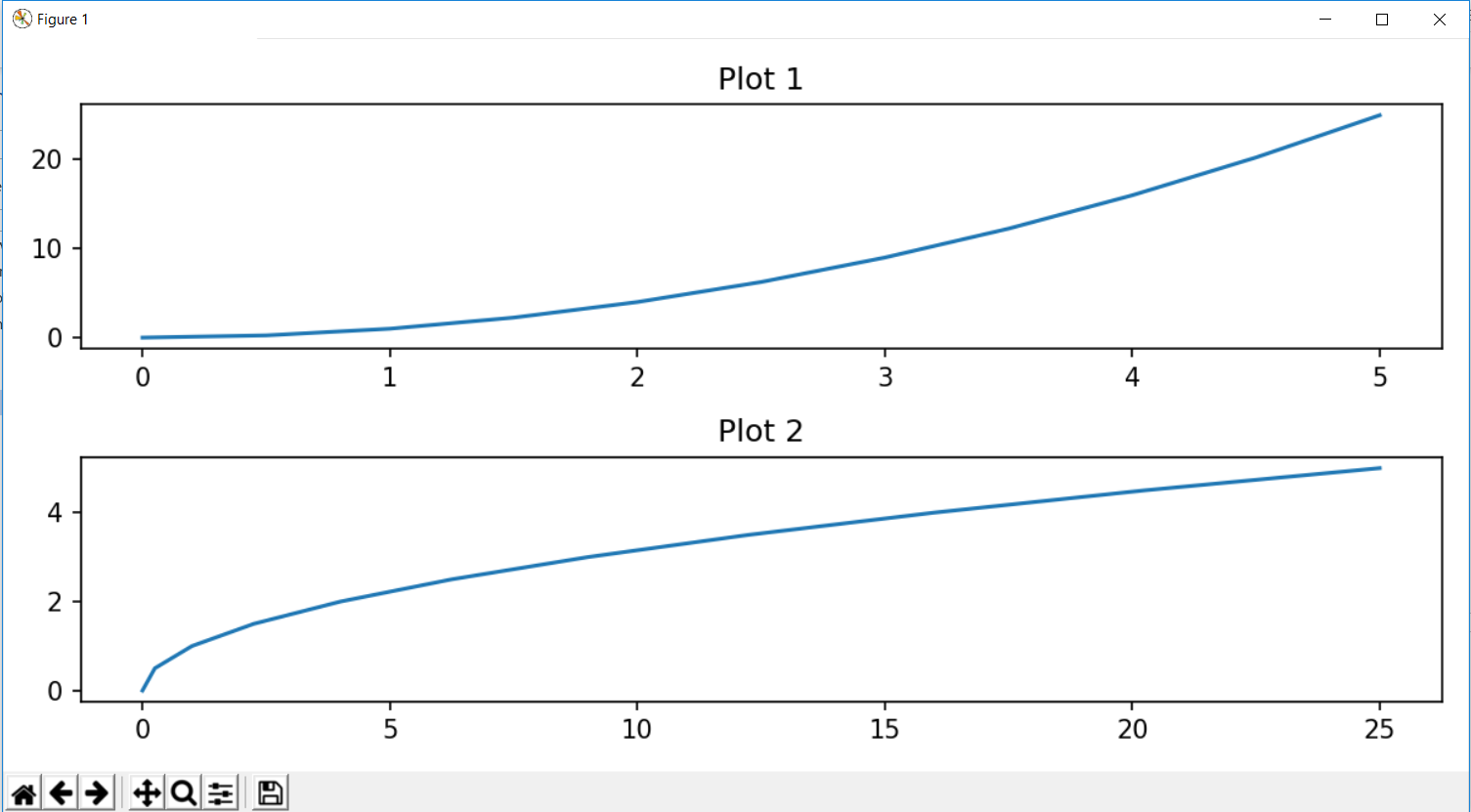
Specifying figure size and dpi-

import matplotlib.pyplot as plt  
import numpy as np  
x = np.linspace(0,5,11)  
y = x \*\* 2  
  
fig = plt.figure(figsize=(8,2),dpi = 150)  
axes = fig.add\_axes([0.1,0.1,0.8,0.8])  
axes.plot(x,y)  
  
plt.tight\_layout()  
plt.show()



This can also be done with subplots-

import matplotlib.pyplot as plt  
import numpy as np  
x = np.linspace(0,5,11)  
y = x \*\* 2  
  
fig,axes = plt.subplots(figsize=(8,4),dpi = 150,nrows=2,ncols=1)  
axes[0].plot(x,y)  
axes[0].set\_title("Plot 1")  
  
axes[1].plot(y,x)  
axes[1].set\_title("Plot 2")  
  
  
plt.tight\_layout()  
plt.show()



## Saving a figure

Matplotlib can generate high-quality output in a number formats, including PNG, JPG, EPS, SVG, PGF and PDF.

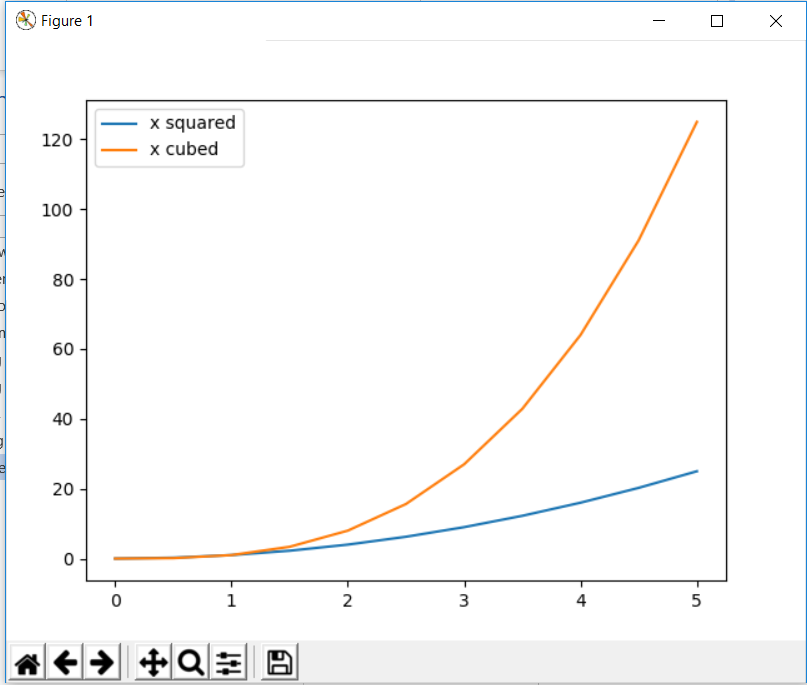
import matplotlib.pyplot as plt  
import numpy as np  
x = np.linspace(0,5,11)  
y = x \*\* 2  
  
fig,axes = plt.subplots(figsize=(8,4),dpi = 150,nrows=2,ncols=1)  
axes[0].plot(x,y)  
axes[0].set\_title("Plot 1")  
  
axes[1].plot(y,x)  
axes[1].set\_title("Plot 2")  
fig.savefig("C:/Users/Vinayak/Desktop/IELTS Writings/test.pdf",dpi = 300)  
  
plt.tight\_layout()  
plt.show()

## Putting legends in the plot

Location code for legends can be seen from the documentation of matplotlib. Also, if we don’t specify label while defining plot, then axes.legend() line will give warning that legends are not present.

The **legend** function takes an optional keyword argument **loc** that can be used to specify where in the figure the legend is to be drawn. The allowed values of **loc** are numerical codes for the various places the legend can be drawn.

import matplotlib.pyplot as plt  
import numpy as np  
x = np.linspace(0,5,11)  
y = x \*\* 2  
  
fig = plt.figure()  
axes = fig.add\_axes([0.1,0.1,0.8,0.8])  
axes.plot(x,x\*\*2,label="x squared")  
axes.plot(x,x\*\*3,label="x cubed")  
  
axes.legend(loc=0)  
#plt.tight\_layout()  
plt.show()



## Setting appearance with Matplotlib

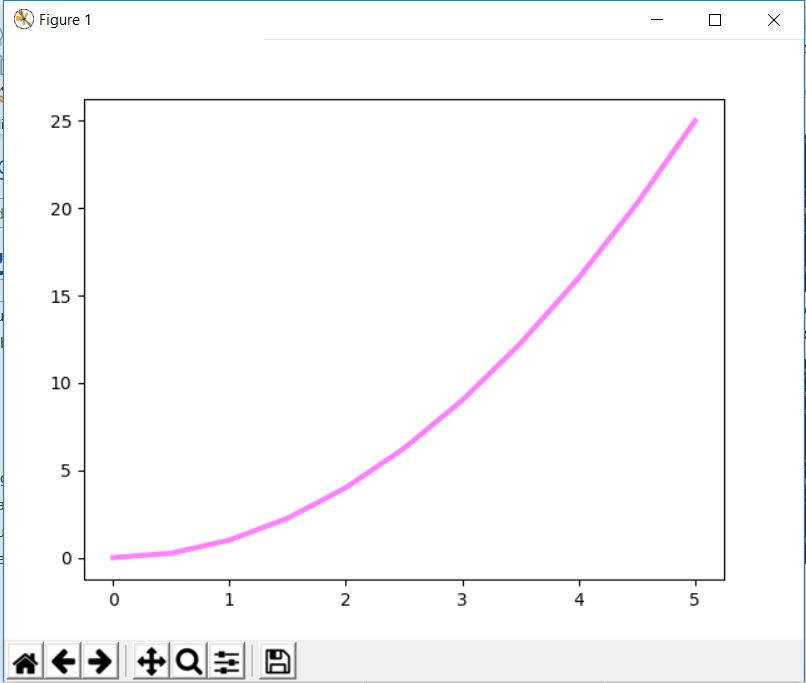
### For colours, we can specify the colour name as string or RGB hex code of the colour as string

import matplotlib.pyplot as plt  
import numpy as np  
x = np.linspace(0,5,11)  
y = x \*\* 2  
  
fig = plt.figure()  
axes = fig.add\_axes([0.1,0.1,0.8,0.8])  
axes.plot(x,x\*\*2,color="#FF00FF")  
  
#plt.tight\_layout()  
plt.show()

### For line width and line transparency

Linewidth parameter can also be written as lw and alpha is the parameter for transparency

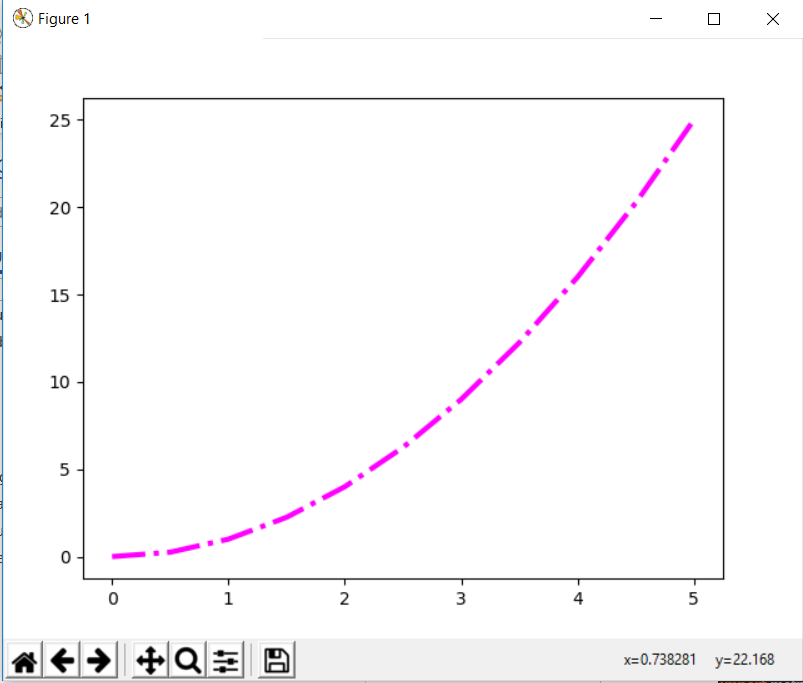
import matplotlib.pyplot as plt  
import numpy as np  
x = np.linspace(0,5,11)  
y = x \*\* 2  
  
fig = plt.figure()  
axes = fig.add\_axes([0.1,0.1,0.8,0.8])  
axes.plot(x,x\*\*2,color="#FF00FF",linewidth=3,alpha=0.5)  
  
#plt.tight\_layout()  
plt.show()



### For linestyle

It can also be written as ls=””. The options for ls can be seen from the documentation.

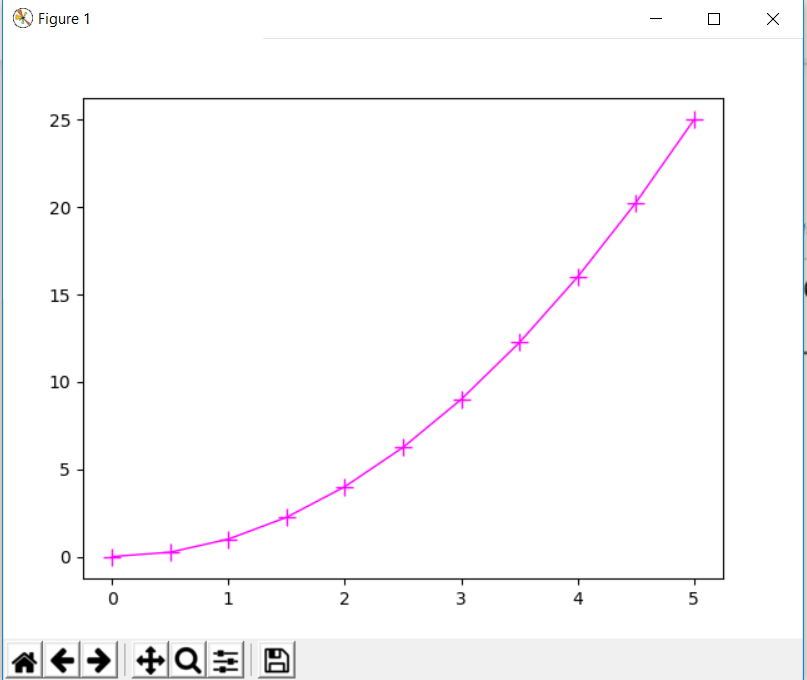
import matplotlib.pyplot as plt  
import numpy as np  
x = np.linspace(0,5,11)  
y = x \*\* 2  
  
fig = plt.figure()  
axes = fig.add\_axes([0.1,0.1,0.8,0.8])  
axes.plot(x,x\*\*2,color="#FF00FF",linewidth=3,linestyle="-.")  
  
#plt.tight\_layout()  
plt.show()



### For putting markers

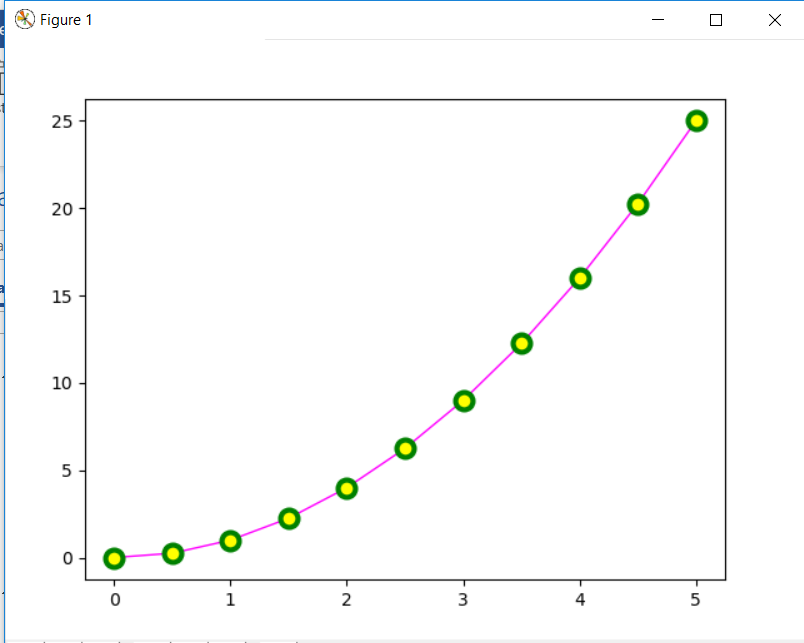
Complete list of available marker which can be applied to a plot can be seen from the documentation

import matplotlib.pyplot as plt  
import numpy as np  
x = np.linspace(0,5,11)  
y = x \*\* 2  
  
fig = plt.figure()  
axes = fig.add\_axes([0.1,0.1,0.8,0.8])  
axes.plot(x,x\*\*2,color="#FF00FF",linewidth=1,linestyle="-",marker="+",markersize=10)  
  
#plt.tight\_layout()  
plt.show()



We can also specify marker face colour, marker edge width and marker edge colour.

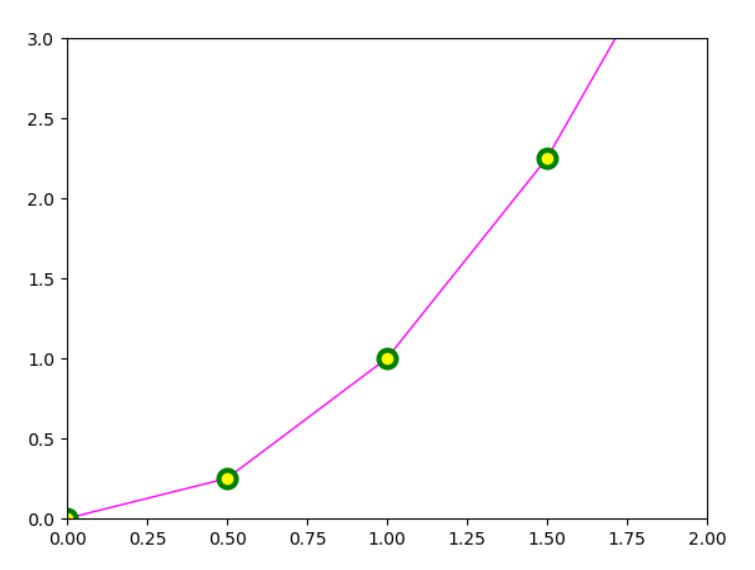
import matplotlib.pyplot as plt  
import numpy as np  
x = np.linspace(0,5,11)  
y = x \*\* 2  
  
fig = plt.figure()  
axes = fig.add\_axes([0.1,0.1,0.8,0.8])  
axes.plot(x,x\*\*2,color="#FF00FF",linewidth=1,linestyle="-",marker="o",markersize=10,  
 markerfacecolor="yellow",markeredgewidth=3,markeredgecolor="green")  
  
#plt.tight\_layout()  
plt.show()



### Axes appearance and plot range

We can configure the ranges of the axes using the set\_ylim and set\_xlim methods in the axis object, or axis('tight') for automatically getting "tightly fitted" axes ranges:

import matplotlib.pyplot as plt  
import numpy as np  
x = np.linspace(0,5,11)  
y = x \*\* 2  
  
fig = plt.figure()  
axes = fig.add\_axes([0.1,0.1,0.8,0.8])  
axes.plot(x,x\*\*2,color="#FF00FF",linewidth=1,linestyle="-",marker="o",markersize=10,  
 markerfacecolor="yellow",markeredgewidth=3,markeredgecolor="green")  
axes.set\_xlim([0,2])  
axes.set\_ylim([0,3])  
  
#plt.tight\_layout()  
plt.show()



### Colors with MATLAB type syntax

With matplotlib, we can define the colors of lines and other graphical elements in a number of ways. First of all, we can use the MATLAB-like syntax where 'b' means blue, 'g' means green, etc. The MATLAB API for selecting line styles are also supported: where, for example, 'b.-' means a blue line with dots:

**# MATLAB style line color and style**

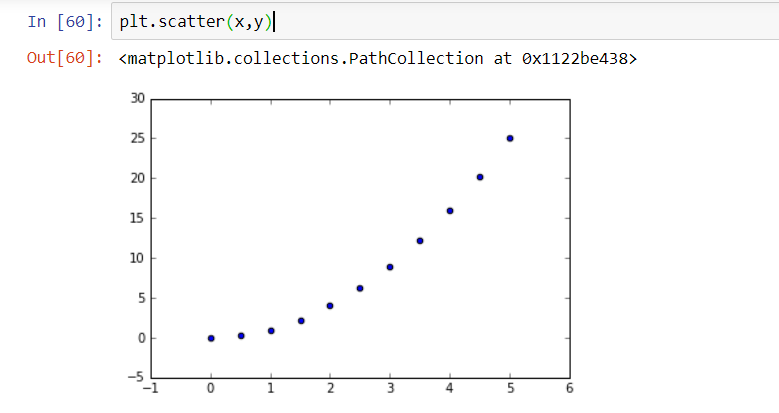
**fig, ax = plt.subplots()**

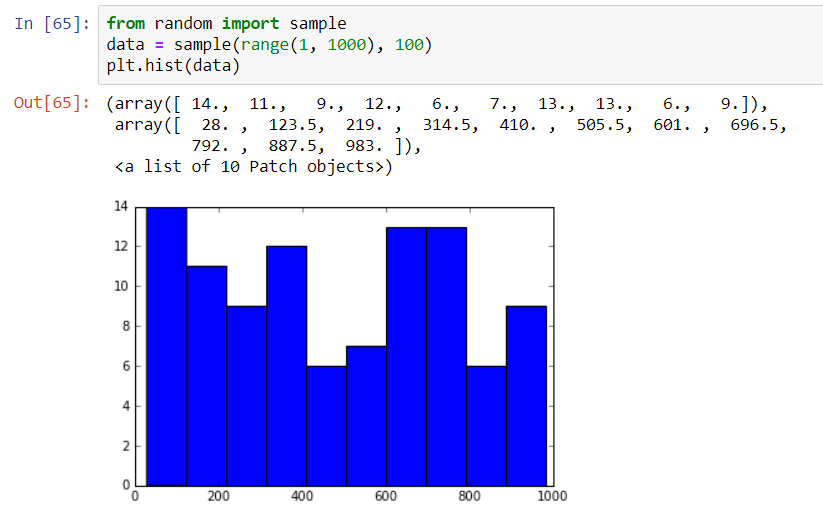
**ax.plot(x, x\*\*2, 'b.-') # blue line with dots**

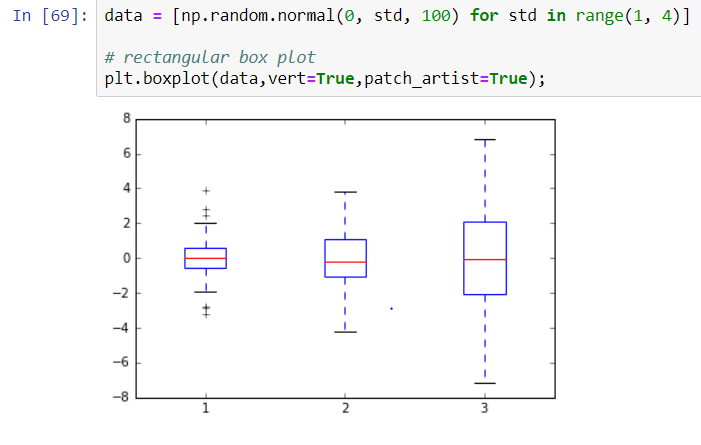
**ax.plot(x, x\*\*3, 'g--') # green dashed line**

## Plot Types

There are many specialized plots we can create, such as barplots, histograms, scatter plots, and much more. Most of these type of plots we will actually create using seaborn, a statistical plotting library for Python. But here are a few examples of these type of plots:







# For Further Reading –

* [http://www.matplotlib.org](http://www.matplotlib.org/) - The project web page for matplotlib.
* <https://github.com/matplotlib/matplotlib> - The source code for matplotlib.
* <http://matplotlib.org/gallery.html> - A large gallery showcaseing various types of plots matplotlib can create. Highly recommended!
* <http://www.loria.fr/~rougier/teaching/matplotlib> - A good matplotlib tutorial.
* <http://scipy-lectures.github.io/matplotlib/matplotlib.html> - Another good matplotlib reference.