Plotting in Python: Matplotlib

Matplotlib

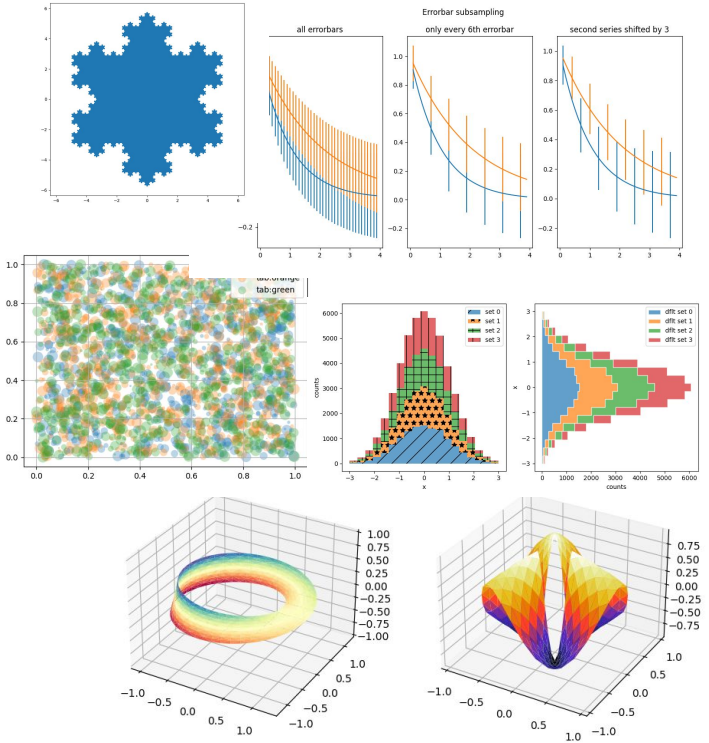
“Matplotlib tries to make easy things easy and hard things possible”
John Hunting

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

Matplotlib can be used in Python scripts, the Python and IPython shell, the jupyter notebook, web application servers and so forth.

You can generate plots, histograms, power spectra, bar charts, error-charts, scatterplots, etc.

For simple plotting the pyplot module provides a **MATLAB-like interface**, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.



Matplotlib.pyplot

matplotlib.pyplot is a collection of command style functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

It is easy to be used in combination with numpy arrays.

It contains several functions for scientific computing and data analysis.

It recognises latex syntax so that you can add formulas to plots.

It is very object oriented! Everything is an object (line, color, axis, figure…)

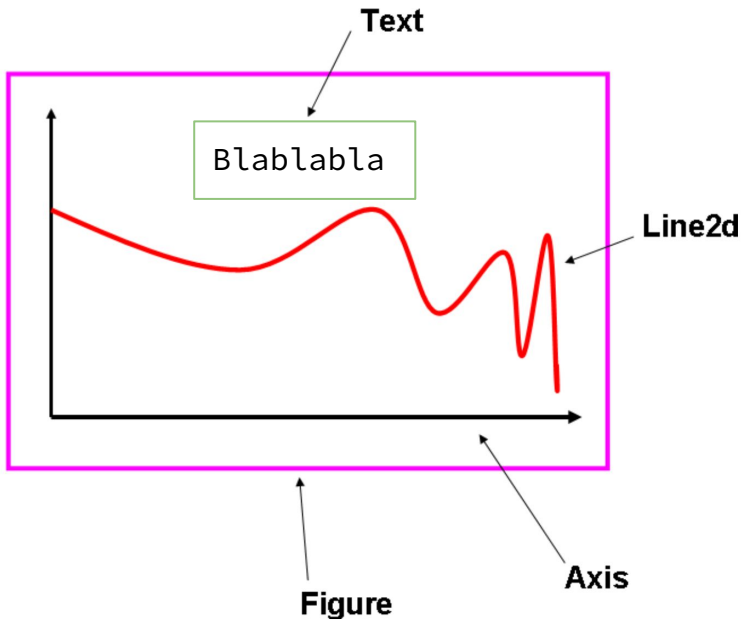
Matplotlib(.pyplot) advantages

- Recognizes latex strings
- Open source
- Cross platform
- Easy to read, simple syntax
- Simple and easy to grasp for beginners.
- Easier to use for people who have had prior experience with Matlab or other graph plotting tools.
- It provides high-quality images and plots in various formats such as png, pdf, pgf, eps, etc.
- Provides control to various elements of a figure such as DPI, figure colour, figure size…

Matplotlib.pyplot How to

The most important instances found in matplotlib are:

- Figures with its own attributes (resolution, dimensions…)
- 2D lines with different markers, line styles, widths, colors…
- Text that can be added both in plain or math mode
- Axis that manages axes properties

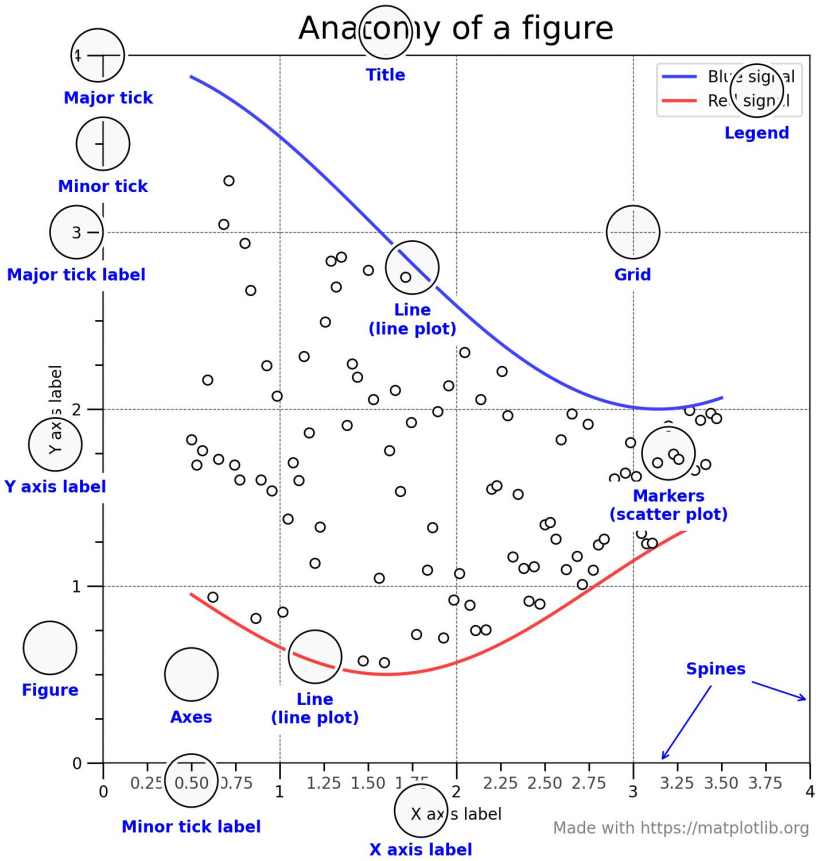


Some basics+nomenclature

Each figure contains one (default) or more axes; in the latter, plots can be drawn in terms of (x,y) - cartesinan 2D or (x,y,z) - cartesinan 3D or (r, theta) - polar coordinates.

There can be more than one axis in a figure, and figure keeps track of all child axes, together with some special **Artists** (titles, figure legends, colorbars, etc), and even nested subfigures.

Basically, everything visible on the Figure is an **Artist** (even Figure, Axes, and Axis objects). This includes Text objects, Line2D objects, collections objects, Patch objects, etc. When the Figure is rendered, all of the Artists are drawn to the canvas. Most Artists are tied to an Axes; such an Artist cannot be shared by multiple Axes, or moved from one to another.

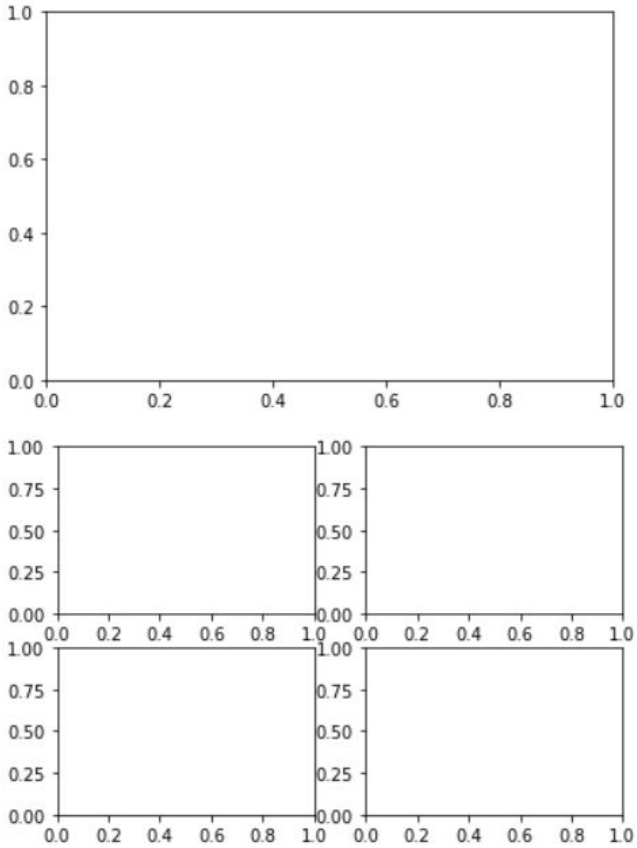


Define figures and Axes (≠ Axis)

```
import matplotlib.pyplot as plt
```

```
fig = plt.figure()  # an empty figure with no Axes
fig, ax = plt.subplots()  # a figure with a single Axes
fig, axs = plt.subplots(2, 2)  # a figure with a 2x2 grid of Axes
```

```
<Figure size 432x288 with 0 Axes>
```



Matplotlib.pyplot BASICS

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
```

Often not needed

```
fig = plt.figure(figsize=(3.0,3.0))
ax = fig.add_subplot(111)

plt.xlim(0, 5)
plt.ylim(0, 3)
plt.xlabel("x")
plt.ylabel("y")

plt.show()
```

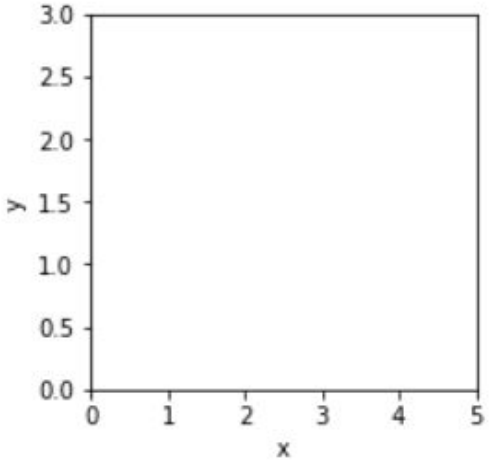
Tuple of two floats in inches

*Three integers (*nrows*, *ncols*, *index*).*

#Limits (default: 0,1)

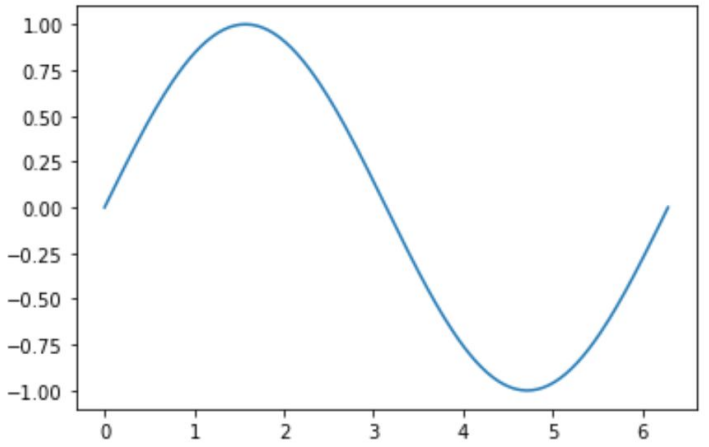
#Axes label

#Displays the plot



Matplotlib.pyplot howto

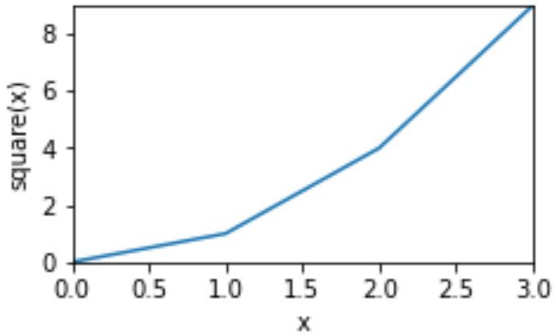
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2 * np.pi, 200)
y = np.sin(x)

fig, ax = plt.subplots()  #analogous to plt.figure()
ax.plot(x, y)             #and fig.add_subplot()
plt.show()
```



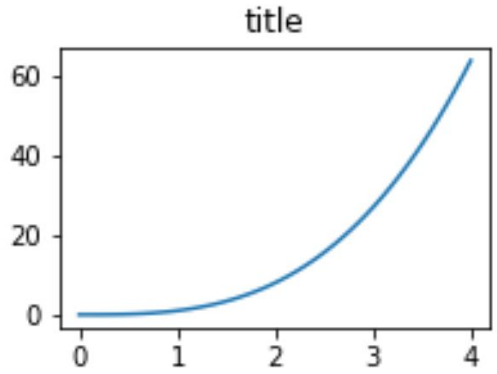

```
fig = plt.figure(figsize=(3.5,2))
ax = fig.add_subplot(111)

ax.set_xlim(0, 3)
ax.set_ylim(0, 9)
ax.set_xlabel("x")
ax.set_ylabel("square(x)")
ax.plot([0,1,2,3],[0,1,4,9])

plt.show()
```

Note:

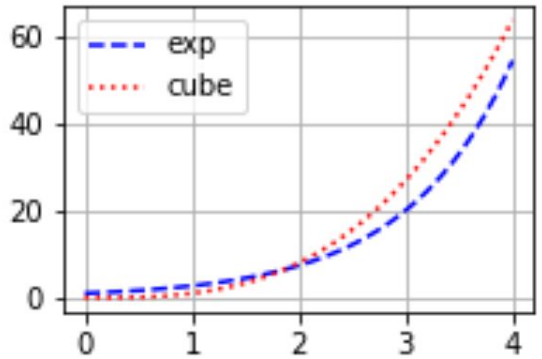
- *ax.set_xlim() behaves very similarly to plt.xlim() [same for y]*
- *ax.set_xlabel() behaves very similarly to plt.xlabel() [same for y]*
- *Same for ax.plot() and plt.plot()*



```
fig,ax = plt.subplots(figsize=(3,2))
x=np.linspace(0,4,50)
ax.plot(x, x**3)
plt.title("title")
plt.show()
```




```
ax.plot(x, np.exp(x), "--b", label="exp")
ax.plot(x, x**3, ":r", label="cube")
plt.legend()
plt.grid()
plt.show()
```



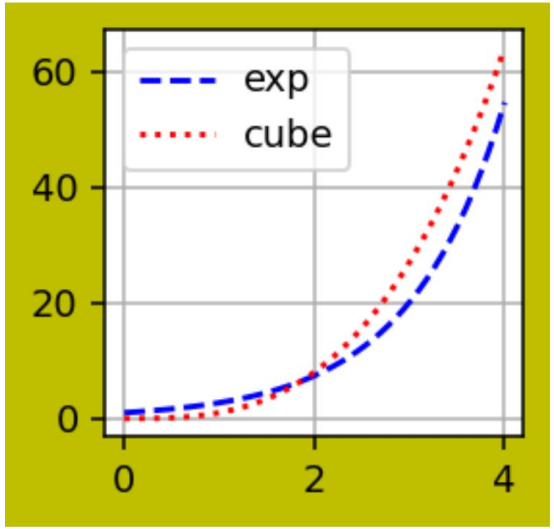
The figure instance

The figure() command allows to create one or multiple figures.

The most important attributes of a figure are:

- figsize: dimension in inches
- facecolor: filling color
- edgecolor: border colors
- dpi: The resolution of the figure in dots-per-inch
- frameon: default is true; if false, suppress drawing the figure frame

```
fig= plt.figure(figsize=(2,2), dpi=140, facecolor="y")
x=np.linspace(0,4,50)
plt.plot(x, np.exp(x), "--b", label="exp")
plt.plot(x, x**3, ":r", label="cube")
plt.legend()
plt.grid()
plt.show()
```



Subplots

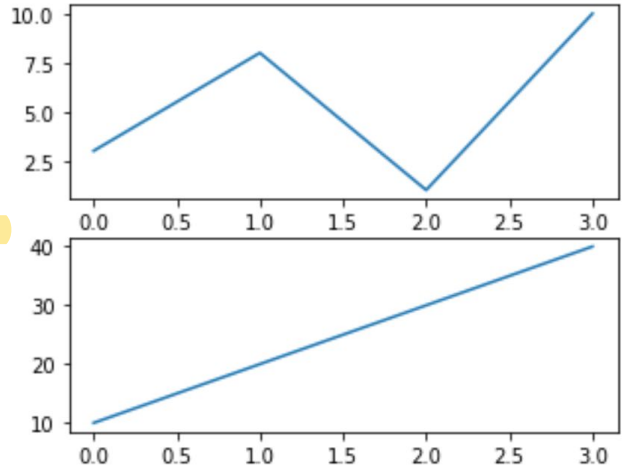
With the subplot() function you can draw multiple plots in one figure:

*Three integers (*nrows*, *ncols*, *index*). *index* starts at 1 in the upper left corner and increases to the right.*

OR

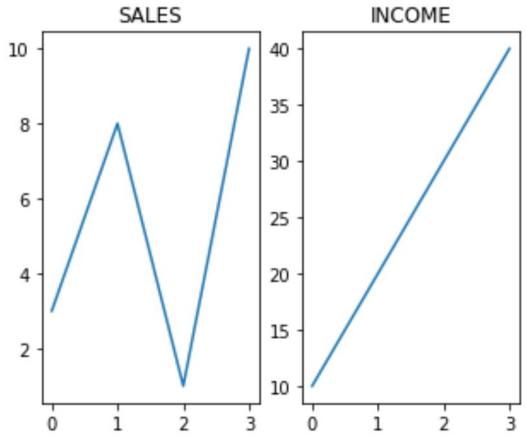
A 3-digit integer. The digits are interpreted as if given separately as three single-digit integers, i.e. ``fig.add_subplot(235)`` is the same as ``fig.add_subplot(2, 3, 5)``. Only for <9 subplots.

```
fig = plt.figure(figsize=(5,4))
#1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(211) #or (2,1,1)
plt.plot(x,y)

#2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(2, 1, 2) #or 212
plt.plot(x,y)

plt.show()
```



Subplots with titles

```
fig = plt.figure(figsize=(5,4))

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")
plt.show()
```

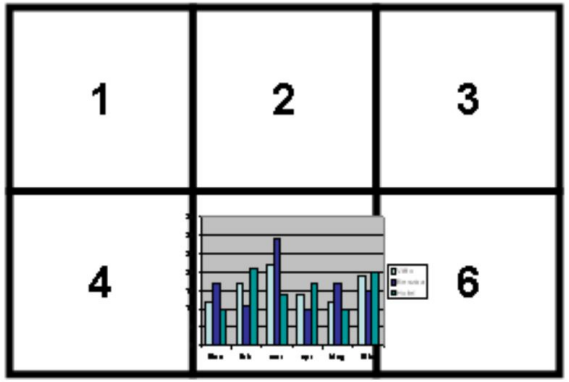


Subplots

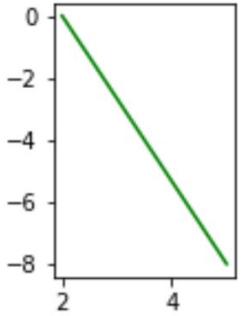
The subplot command plt.subplot(nrows,ncol,index) allows to add multiple graphs on a figure with a grid having a specific number of rows and columns:

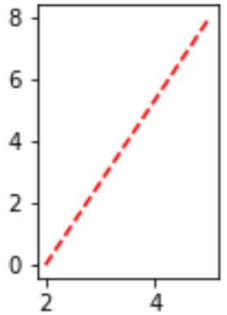
Es: plt.subplot(2,3,5)

→



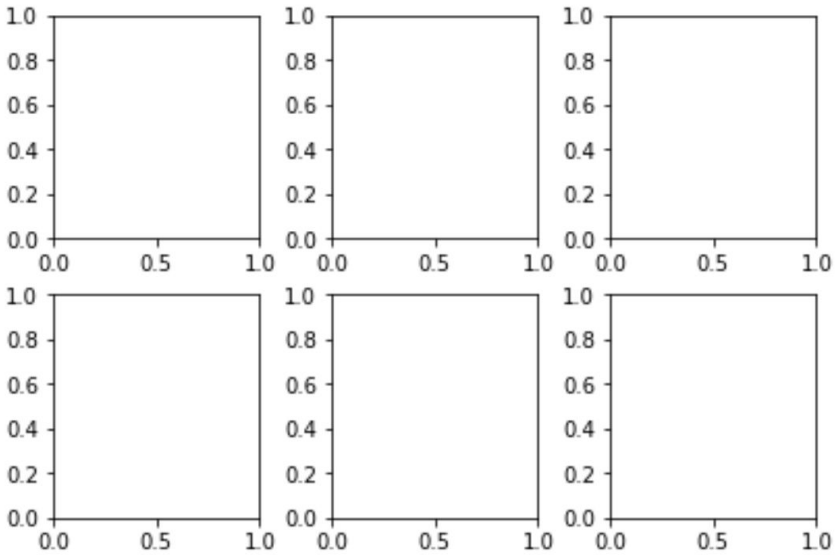

```
fig = plt.figure(figsize=(5,5))
plt.subplot(2,3,1)
plt.plot([2,5],[0,-8], "g-")
plt.subplot(2,3,6)
plt.plot([2,5],[0,8], "r--")
plt.show()
```



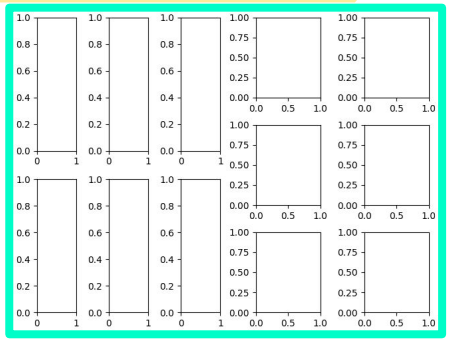


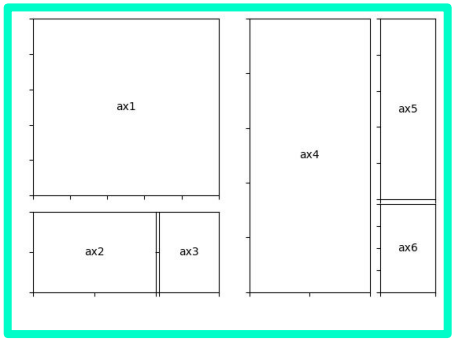
Subplots

```
fig,axes = plt.subplots(2,3)
plt.subplots_adjust(top=0.92, bottom=0.08,
                    left=0.10, right=0.95,
                    hspace=0.25, wspace=0.35)
plt.show()
```



Note: if you need to do more elaborate things with subplots, I encourage you to use the gridspec package which is part of matplotlib!
E.g.:





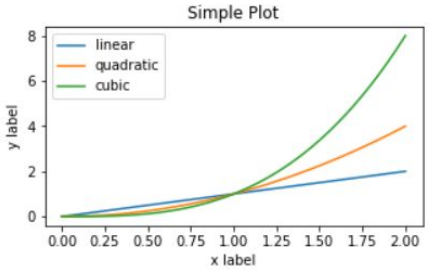
Object oriented (OO) vs pyplot interfaces

As hinted in the previous slides, there are essentially two ways to use Matplotlib:

- Explicitly create Figures and Axes, and call methods on them (the "object-oriented (OO) style").
- Rely on pyplot to automatically create and manage the Figures and Axes, and use pyplot functions for plotting.

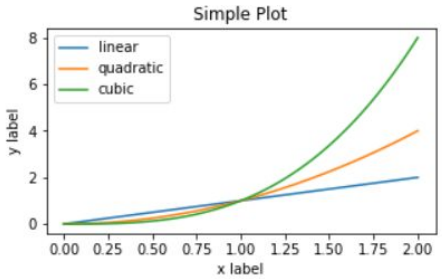
OO Style

```
x = np.linspace(0, 2, 100)  # Sample data.
# Note that even in the OO-style, we use `.pyplot.figure` to create the Figure.
fig, ax = plt.subplots(figsize=(5, 2.7))
ax.plot(x, x, label='linear')  # Plot some data on the axes.
ax.plot(x, x**2, label='quadratic')  # Plot more data on the axes...
ax.plot(x, x**3, label='cubic')  # ... and some more.
ax.set_xlabel('x label')  # Add an x-label to the axes.
ax.set_ylabel('y label')  # Add a y-label to the axes.
ax.set_title("Simple Plot")  # Add a title to the axes.
ax.legend();  # Add a legend.
```



Pyplot interface

```
x = np.linspace(0, 2, 100)  # Sample data.

plt.figure(figsize=(5, 2.7))
plt.plot(x, x, label='linear')  # Plot some data on the (implicit) axes.
plt.plot(x, x**2, label='quadratic')  # etc.
plt.plot(x, x**3, label='cubic')
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend();
```



*Note: use **plt.gca()** to get current plotting axes, **plt.sca()** to set a different one*

AXES vs AXIS (!!!)

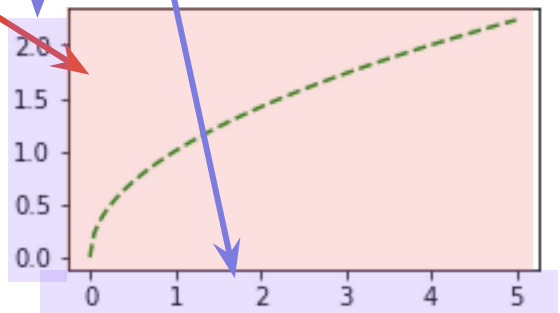
Axes

An Axes is an Artist attached to a Figure that contains a region for plotting data, and usually includes two (or three in the case of 3D) **`Axis`** objects (be aware of the difference between Axes and Axis) that provide ticks and tick labels to provide scales for the data in the Axes. Each **`Axes`** also has a title (set via **`set_title()`**), an x-label (set via **`set_xlabel()`**), and a y-label set via **`set_ylabel()`**.

The **`Axes`** class and its member functions are the primary entry point to working with the OOP interface, and have most of the plotting methods defined on them (e.g. `ax.plot()`, shown above, uses the **`plot`** method)

Axis

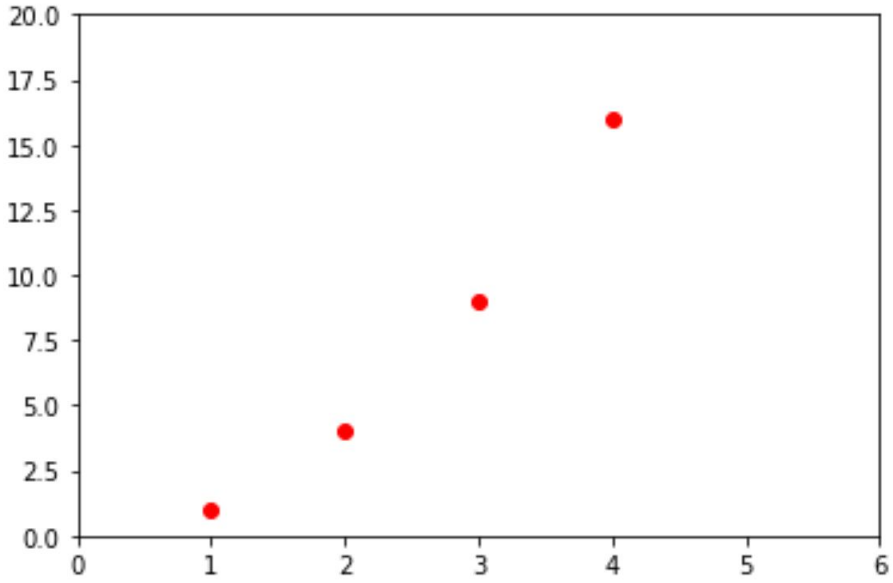
These objects set the scale and limits and generate ticks (the marks on the Axis) and ticklabels (strings labeling the ticks). The location of the ticks is determined by a Locator object and the ticklabel strings are formatted by a Formatter. The combination of the correct Locator and Formatter gives very fine control over the tick locations and labels.



PLT.PLOT()

For every x, y pair of arguments, there is an optional third argument which is the format string that indicates the color and line type of the plot. The letters and symbols of the format string are from MATLAB, and you concatenate a color string with a line style string. The default format string is 'b-', which is a solid blue line. Lines and/or markers can appear on your figure.

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro') #r=red, o=circle
plt.axis([0, 6, 0, 20]) #alternative way to set limits
plt.show()
```



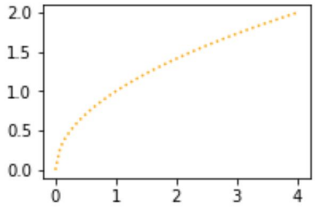
x and y can be lists, np.arrays, pandas dataframes and so forth

Linestyles (ls), linewidth (lw), Color (c)

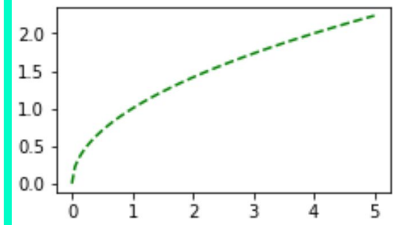
lw: Line widths are specific in points

ls: More refined control can be achieved by providing a dash tuple (offset, (on_off_seq)). For example, (0, (3, 10, 1, 15)) means (3pt line, 10pt space, 1pt line, 15pt space) with no offset.

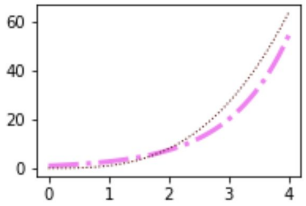
```
import matplotlib.pyplot as plt
import numpy as np
fig,ax = plt.subplots(figsize=(3,2))
x=np.linspace(0,4,50)
ax.plot(x, np.sqrt(x), linestyle="dotted", color="orange")
plt.show()
```

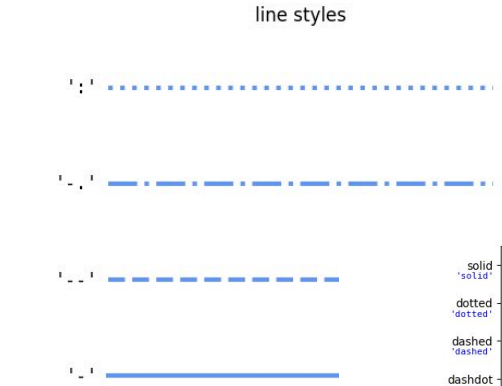



```
fig,ax = plt.subplots(figsize=(3.5,2))
x=np.linspace(0,5,100)
ax.plot(x, np.sqrt(x), "g--")
plt.show()
```



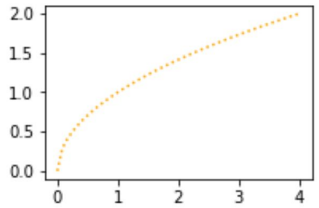

```
fig,ax = plt.subplots(figsize=(3,2))
x=np.linspace(0,4,50)
ax.plot(x, np.exp(x), \
        linestyle="-.", \
        color="violet", linewidth="3")
ax.plot(x, x**3, \
        ls="dotted", \
        c="#4d0000", lw="1")
plt.show()
```



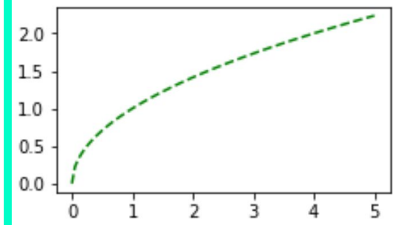


Linestyles (ls), linewidth (lw), Color (c)

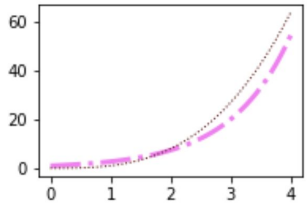
```
import matplotlib.pyplot as plt
import numpy as np
fig,ax = plt.subplots(figsize=(3,2))
x=np.linspace(0,4,50)
ax.plot(x, np.sqrt(x), linestyle="dotted", color="orange")
plt.show()
```




```
fig,ax = plt.subplots(figsize=(3.5,2))
x=np.linspace(0,5,100)
ax.plot(x, np.sqrt(x), "g--")
plt.show()
```



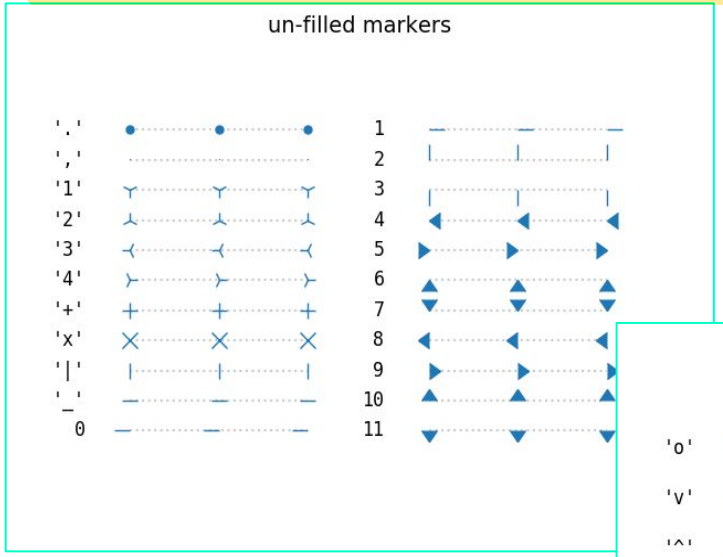

```
fig,ax = plt.subplots(figsize=(3,2))
x=np.linspace(0,4,50)
ax.plot(x, np.exp(x), \
        linestyle="-.", \
        color="violet", linewidth="3")
ax.plot(x, x**3, \
        ls="dotted", \
        c="#4d0000", lw="1")
plt.show()
```

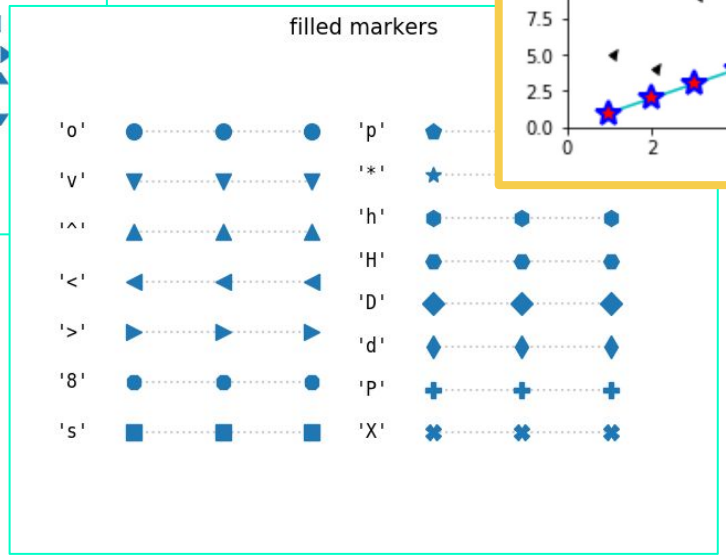


Some ways to specify colors(color=...)

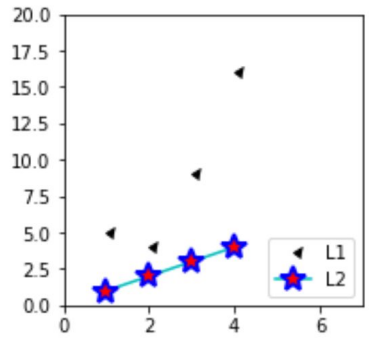
- RGB: `c = (red, green, blue, alpha)` where alpha is transparency
- RGBA string (html) as a string; es: `c = '#0f0f0f'`
- String representation of float value between 0,1 for greyscale as `c='0.54'`
- Default color names `c="red"` or `c="cyan"` or colors in the default palette, as `c="C0"` … `c="C9"`
- Letter to shorten color name `c="b"`
  - `'b'` as blue
  - `'g'` as green
  - `'r'` as red
  - `'c'` as cyan
  - `'m'` as magenta
  - `'y'` as yellow
  - `'k'` as black
  - `'w'` as white

Marker styles,sizes, colors (plt.plot)





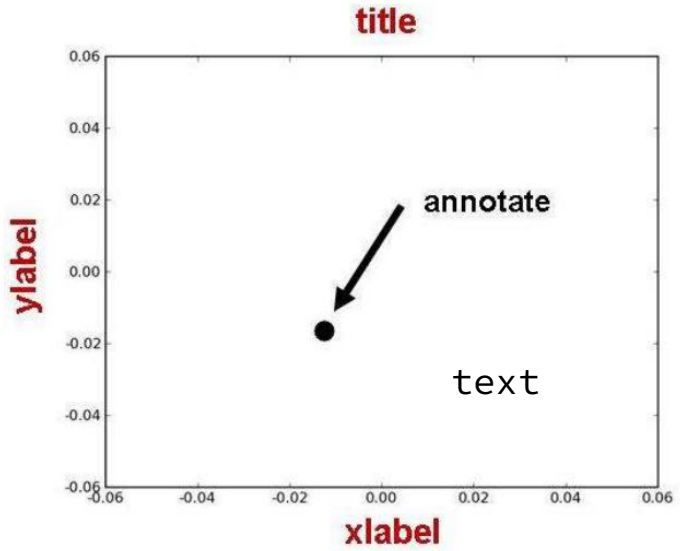

```
fig,ax = plt.subplots(figsize=(3,3))
plt.plot([1, 2, 3, 4], [5, 4, 9, 16], marker=4,
         ls = "", c= "k",label="L1")
plt.plot([1, 2, 3, 4], [1, 2, 3, 4], marker="*", c="c",
         markerfacecolor="r", markeredgecolor="b",
         markeredgewidth=2, markersize=15,label="L2")
plt.axis([0, 7, 0, 20])
plt.legend(loc = "lower right")
plt.show()
```



TEXT HANDLING

There are several functions that allow to handle text in matplotlib.pyplot:

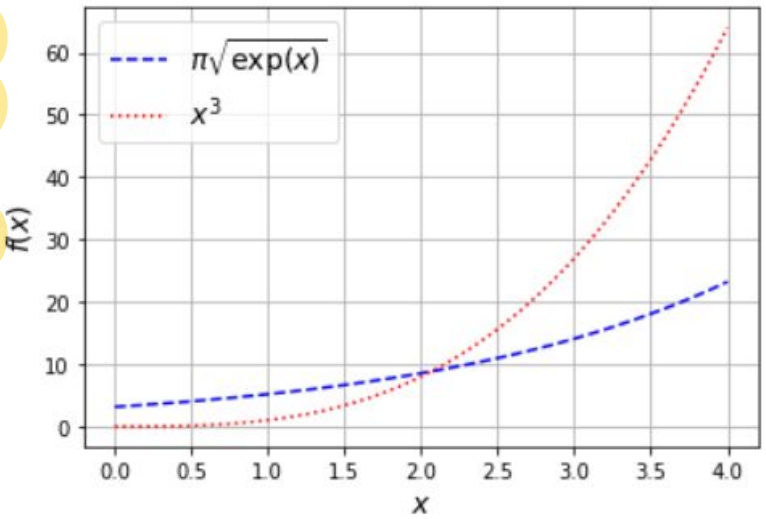
- `xlabel (s, *args, **kwargs)`
- `ylabel (s, *args, **kwargs)`
- `title (s, *args, **kwargs)`
- `annotate (s, xy, xytext=None, xycoords='data', textcoords='data', arrowprops=None,**props)`
- `text (x, y, s, fontdict=None, **kwargs)`



Text handling

The most important argument of plt methods dealing with text is the string **s** in which the text to be written is contained. It can be formatted in latex!

```
fig= plt.figure()
x=np.linspace(0,4,50)
plt.plot(x, np.pi*np.sqrt(np.exp(x)), "--b", label=r"$\pi\sqrt{\exp(x)}$")
plt.plot(x, x**3, ":r", label=r"$x^3$")
plt.legend(fontsize=14)
plt.xlabel(r"$x$", fontsize=14)
plt.ylabel(r"$f(x)$", fontsize=14, )
plt.grid()
plt.show()
```



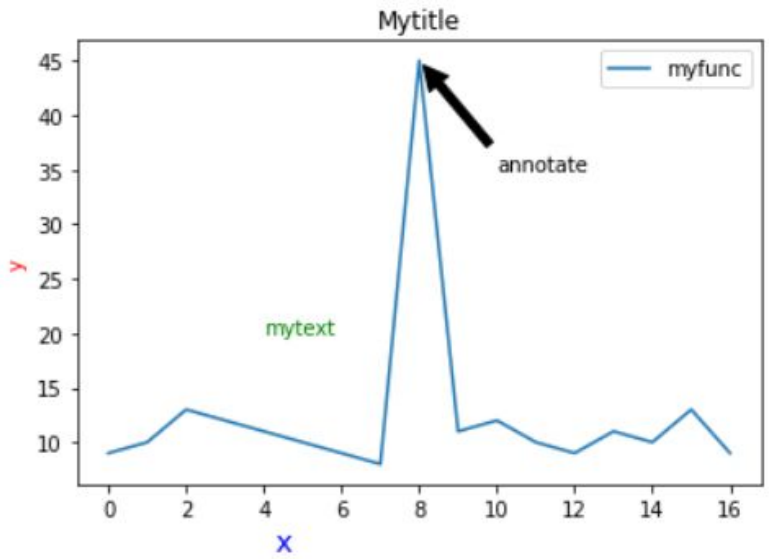
Text handling

The most important properties of text are:

- **fontsize**: xx-small, x-small, small, medium, large, x-large, xx-large
- **fontstyle**: normal, italic, oblique
- **color**
- **rotation**: degree , ‘vertical’, ’horizontal’
- **verticalalignment**: ‘top’, ‘center’, ‘bottom’
- **horizontalalagnment**: ‘left’, ‘center’, right’

Text handling: an example

```
fig= plt.figure()
x=[9,10,13,12,11,10,9,8,45,11,12,10,9,11,10,13,9]
plt.plot(x,label='myfunc')
plt.legend()
plt.title('Mytitle')
plt.ylabel('y',fontsize='medium',color='r')
plt.xlabel('x',fontsize='x-large',color='b',position=(0.3,1))
plt.text(4,20,'mytext', color='g',fontsize='medium')
plt.annotate('annotate',xy=(8,45),xytext=(10,
35),arrowprops=dict(facecolor='black',shrink=0.05))
plt.show()
```



Save a figure

To save a figure you have just plotted, you can use

```
savefig(fname, dpi=None, facecolor='w', edgecolor='w', orientation='portrait', papertype=None, format=None, transparent=False, bbox_inches=None, pad_inches=0.1, frameon=None, metadata=None)
```

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2, 100)  # Sample data.

fig = plt.figure(figsize=(5, 2.7))
plt.plot(x, x, label='linear')  # Plot some data on the (implicit) axes.
plt.plot(x, x**2, label='quadratic')  # etc.
plt.plot(x, x**3, label='cubic')
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend()
fig.savefig("myplot.png")
```