Mathplotlib : it is a python library used for data visulization. it is :

- low level graph plotting library that serves as a visualization utility.
- Matplotlib was created by ***John D. Hunter***.
- open source and we can use it freely.
- mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

commans to install it :

- to install module: pip install matplotlib
- to upgrade module: pip install --upgrade matplotlib

to check it/use it in code:

- import matplotlib
- import matplotlib as mtl (after it we can use)
- numpy._ *version* _ (check numpy version)

why do we use it?

- its easy to use visualization
- easy to maintain
- provide better and easier way to handle the data

Combination of other packages with numpy :

- It can be used with other packages like Scipy(ScientificPython), Numpy and pandas.
- it is open–source and can be used as a replacement for matlab.
- install matplotlib :

```
pip install matplotlib
```

- update matplotlib :

```
pip install --upgrade matplotlib
```

- check veersion :

```
import matplotlib
print(matplotlib.__version__)
```

Matplotlib Pyplot :

Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias

- there are two ways to import pyplot from matplotlib:

way 1 :

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

way 2 :

```
from matplotlib import pyplot as plt
```

Matplotlib Plotting -

- plt.plot() - it draw lines between point to point in x or y axis
- first array create plot in x-axis and second in y-axis
- Draw a line in a diagram/graph from position (0, 0) to position (6, 150) :

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

x = np.array([0,6])
y = np.array([0,150])

plt.plot(x,y)

plt.show()
```

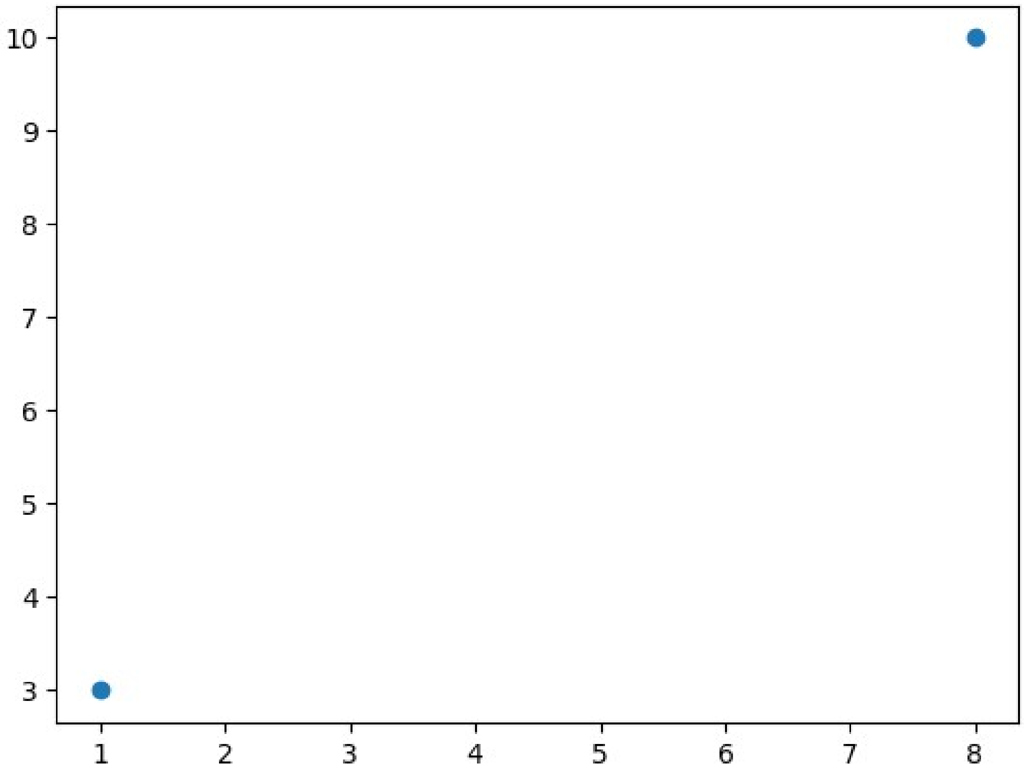
- Draw a line in a diagram from position (1, 3) to position (8, 10) :

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

x = np.array([1,8])
y = np.array([3,10])

plt.plot(x,y)

plt.show()
```

Plotting Without Line : To plot only the markers, you can use shortcut string notation parameter 'o', which means 'rings'

- Draw two points in the diagram, one at position (1, 3) and one in position (8, 10) :

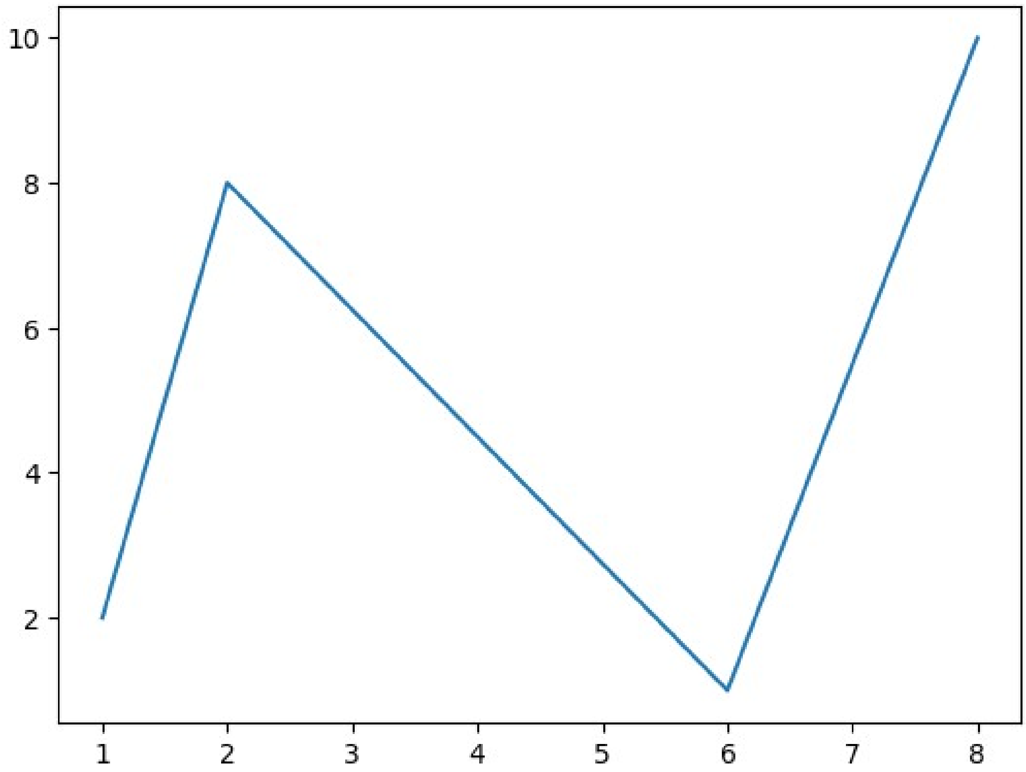
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([1, 8])
y = np.array([3, 10])

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

plt.show()
```



plot with multiple Points : You can plot as many points as you like, just make sure you have the same number of points in both axis.

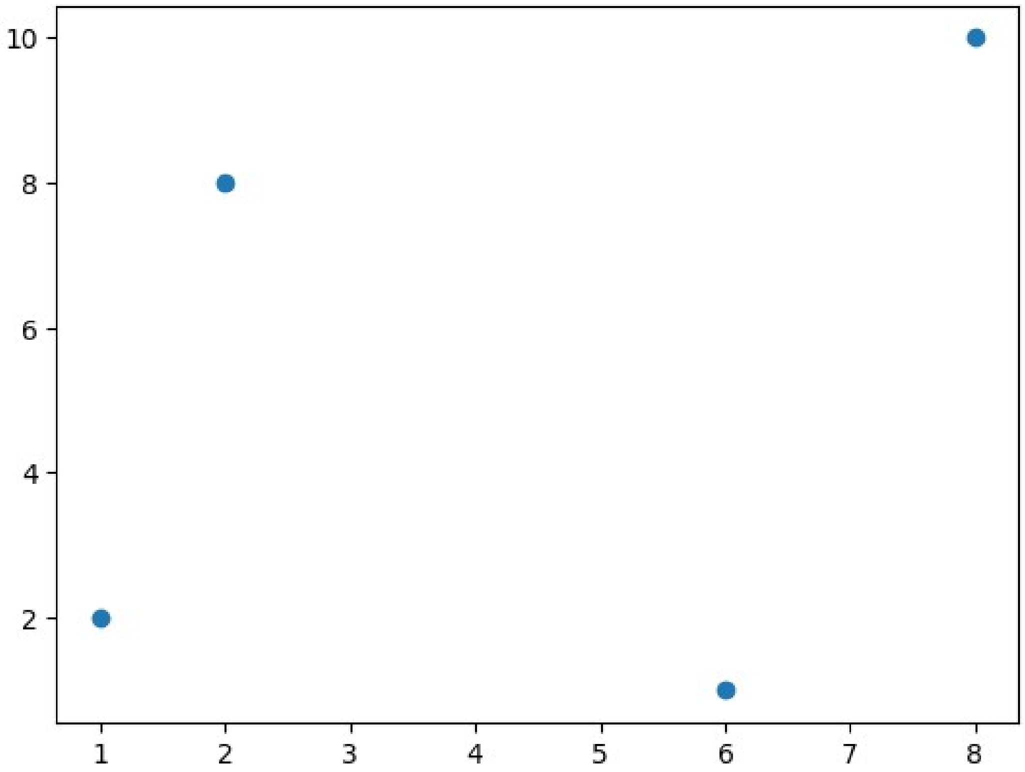
- Draw a line in a diagram from position (1, 3) to (2, 8) then to (6, 1) and finally to position (8, 10) :

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

x = np.array([1, 2, 6, 8])
y = np.array([2, 8, 1, 10])

plt.plot(x,y)

plt.show()
```



- Draw a plot without line in a diagram from position (1, 3) to (2, 8) then to (6, 1) and finally to position (8, 10) :

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

x = np.array([1, 2, 6, 8])
y = np.array([2, 8, 1, 10])

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

plt.show()
```

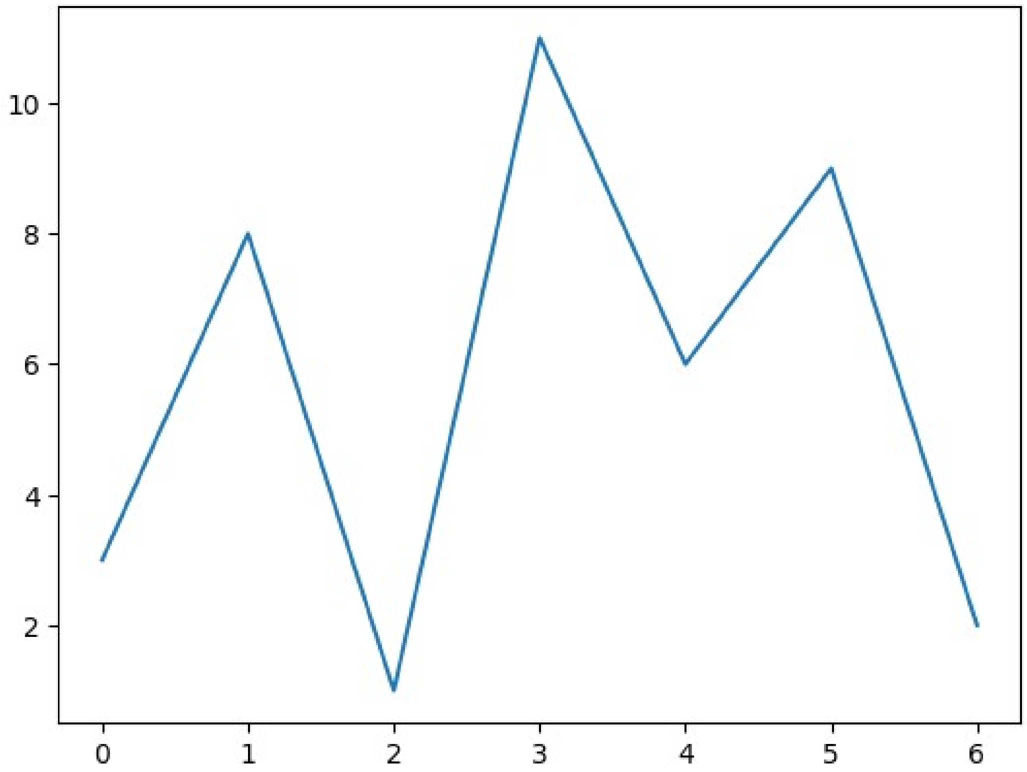


draw plot with Default X-Points : If we do not specify the points on the x-axis, they will get the default values 0, 1, 2, 3 (etc., depending on the length of the y-points.

So, if we take the same example as above, and leave out the x-points, the diagram will look like this:

- Plotting without x-points or default x-points :

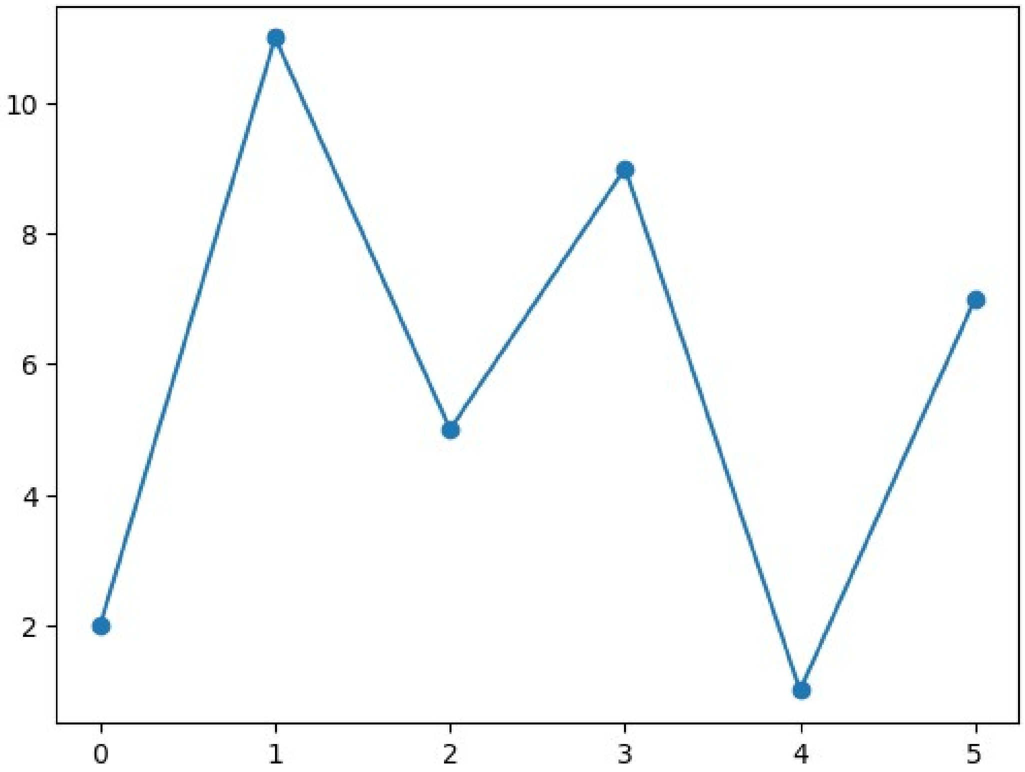
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([3, 8, 1, 11, 6, 9, 2])

plt.plot(y)

plt.show()
```



Matplotlib Markers : You can use the keyword argument ***marker*** to emphasize each point with a specified markers.

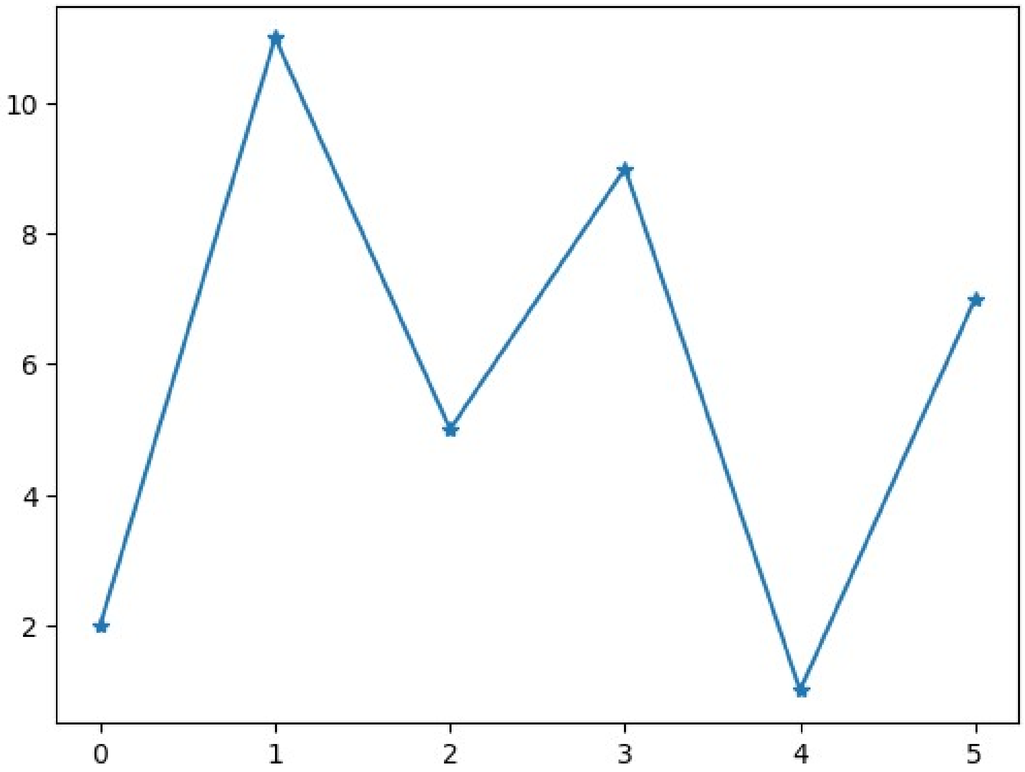
- Mark each point with a circle :

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

y = np.array([2, 11, 5, 9, 1, 7])

plt.plot(y, marker = 'o')   # mark as circle

plt.show()
```



- Mark each point with a star :

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

y = np.array([2, 11, 5, 9, 1, 7])

plt.plot(y, marker = '*')

plt.show()
```



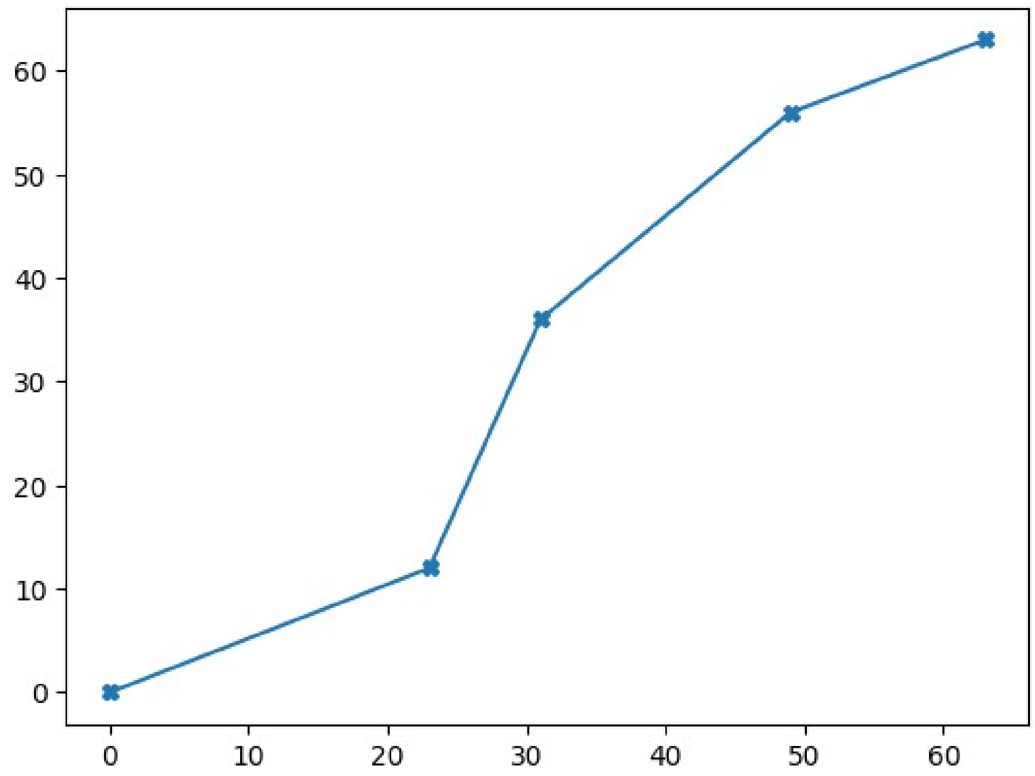
Marker's List :

You can choose any of these markers :

MarkerDescriptionMarkerDescription

- let's try some of them :
- marker = X-filled

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

x = np.array([0, 23, 31, 49, 63])
y = np.array([0, 12, 36, 56, 63])

plt.plot(x,y, marker = 'X')
plt.show()
```

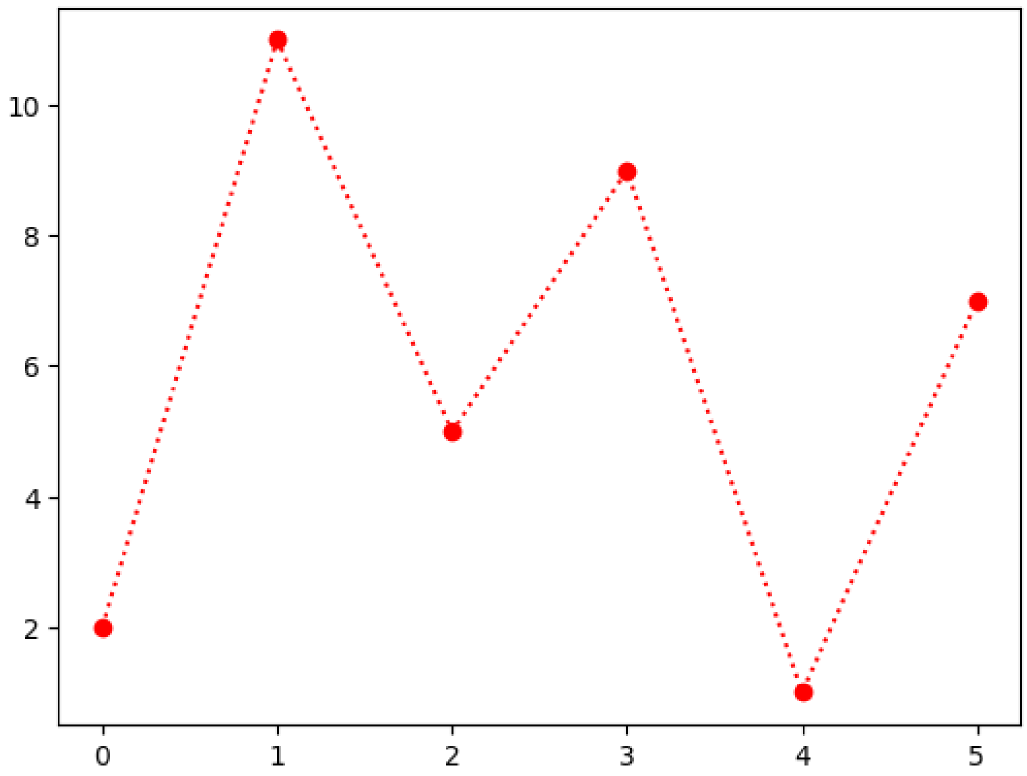


- marker = x

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

y = np.array([2, 11, 5, 9, 1, 7])

plt.plot(y, marker = 'x')

plt.show()
```

- Strings Format : You can use also use the shortcut string notation parameter to specify the marker. This parameter is also called fmt syntax: ***marker|line|color***
- example : Mark each point with a circle,dotted,red:

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

y = np.array([2, 11, 5, 9, 1, 7])

plt.plot(y, 'o:r')
# point = 'o', line = ':', color = 'r'

plt.show()
```



Line Reference :

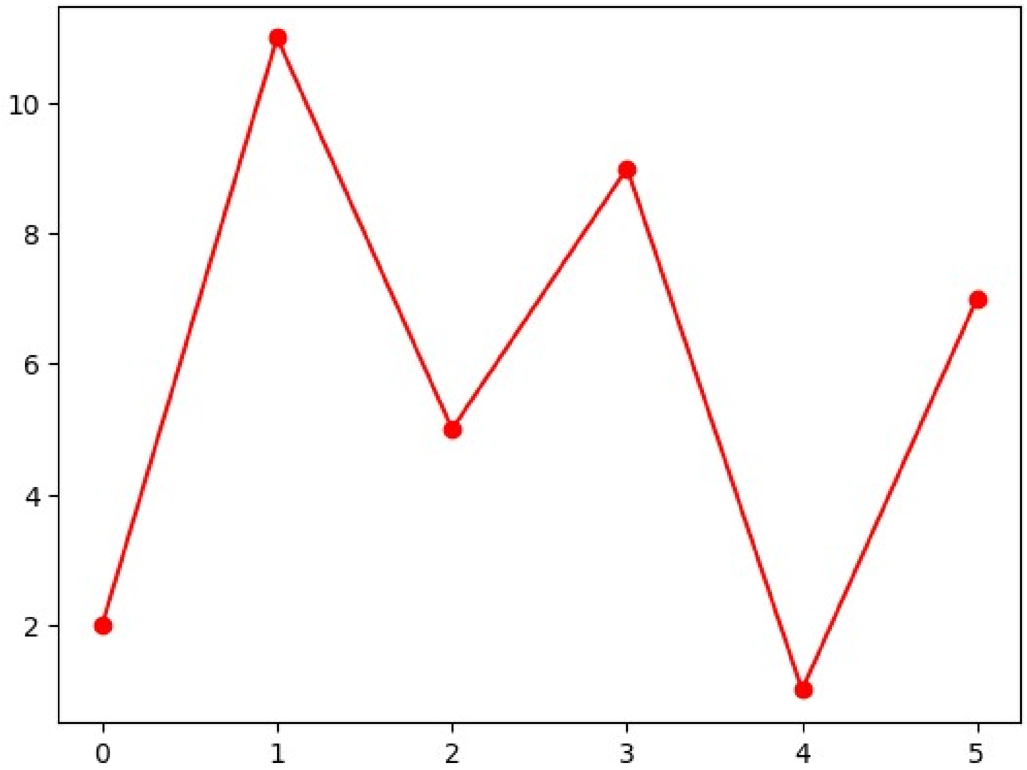
The marker value can be anything from the Marker Reference above.

LineDescription

'-'Solid line ':'Dotted line '--'Dashed line '-.'Dashed dotted line

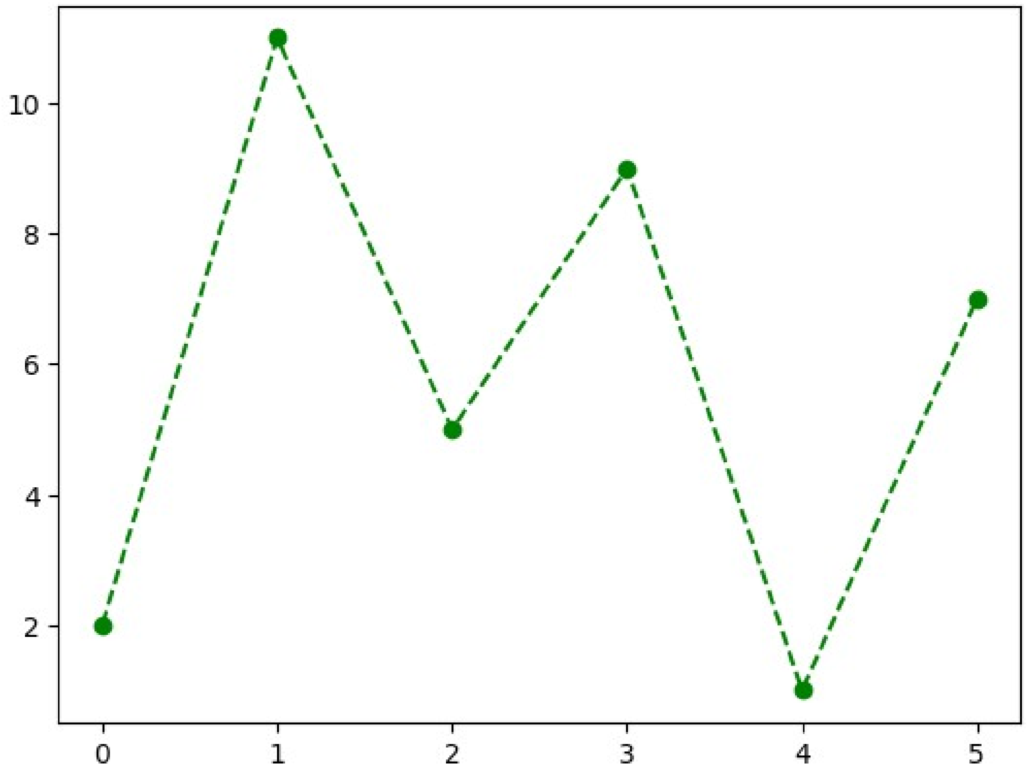
- Let's see some of them :
- Mark each point with a circle,full line,red :

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

y = np.array([2, 11, 5, 9, 1, 7])

plt.plot(y, 'o-r')
# point = 'o', line = '-', color = 'r'

plt.show()
```



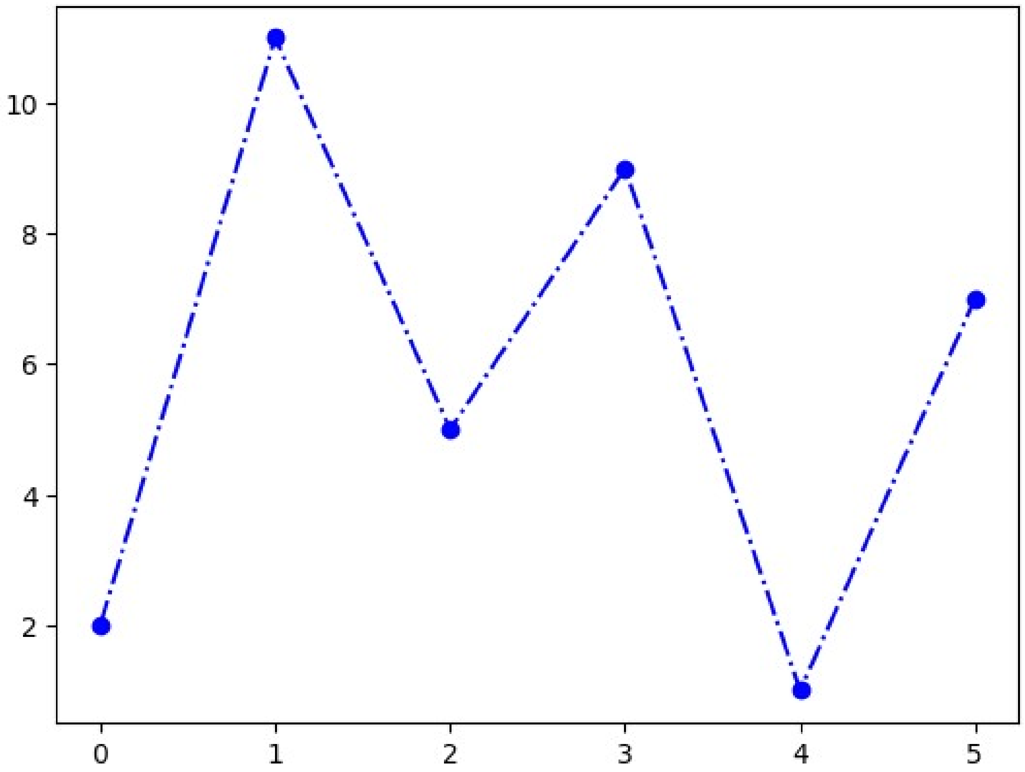
- Mark each point with a circle,dashed,green :

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

y = np.array([2, 11, 5, 9, 1, 7])

plt.plot(y, 'o--g')
# point = 'o', line = '--', color = 'g'

plt.show()
```



- Mark each point with a circle, dashed/dotted, blue :

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

y = np.array([2, 11, 5, 9, 1, 7])

plt.plot(y, 'o-.b')
# point = 'o', line = '-.', color = 'b'

plt.show()
```

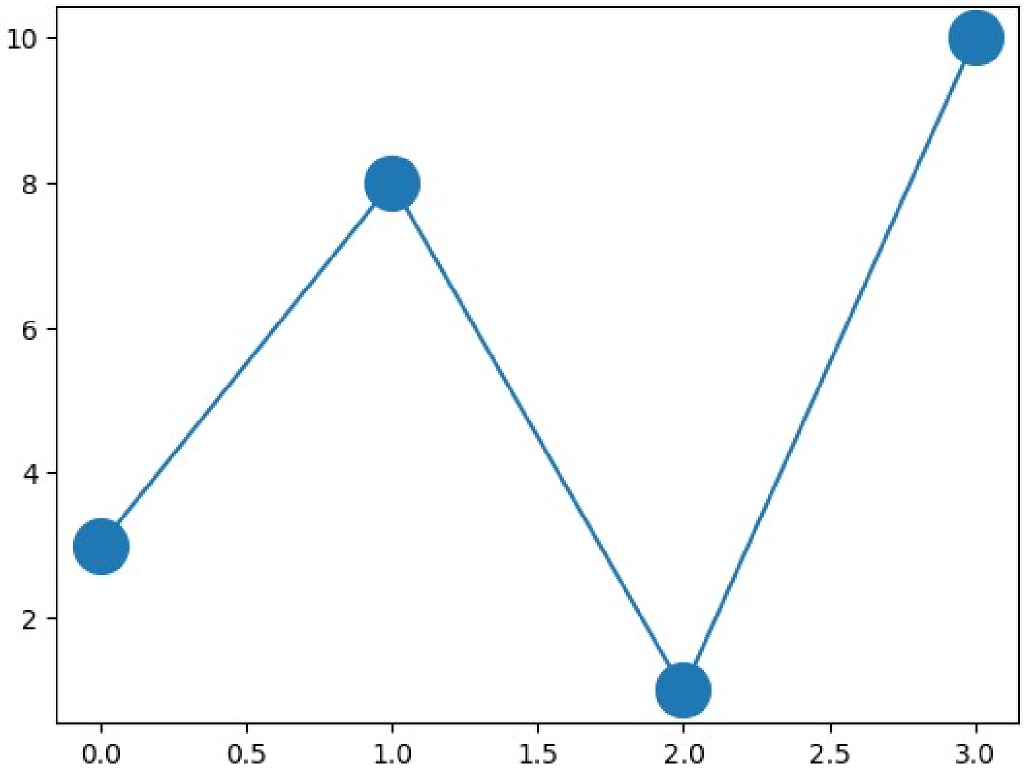


Color Reference:

The short color value can be one of the following : ____ Color SyntaxDescription 'r'Red 'g'Green 'b'Blue 'c'Cyan 'm'Magenta 'y'Yellow 'k'Black 'w'White

- Marker Size : You can use the keyword argument ***markersize*** or in short ***ms*** to set the size of the markers:
- Set the size of the markers to 20 :

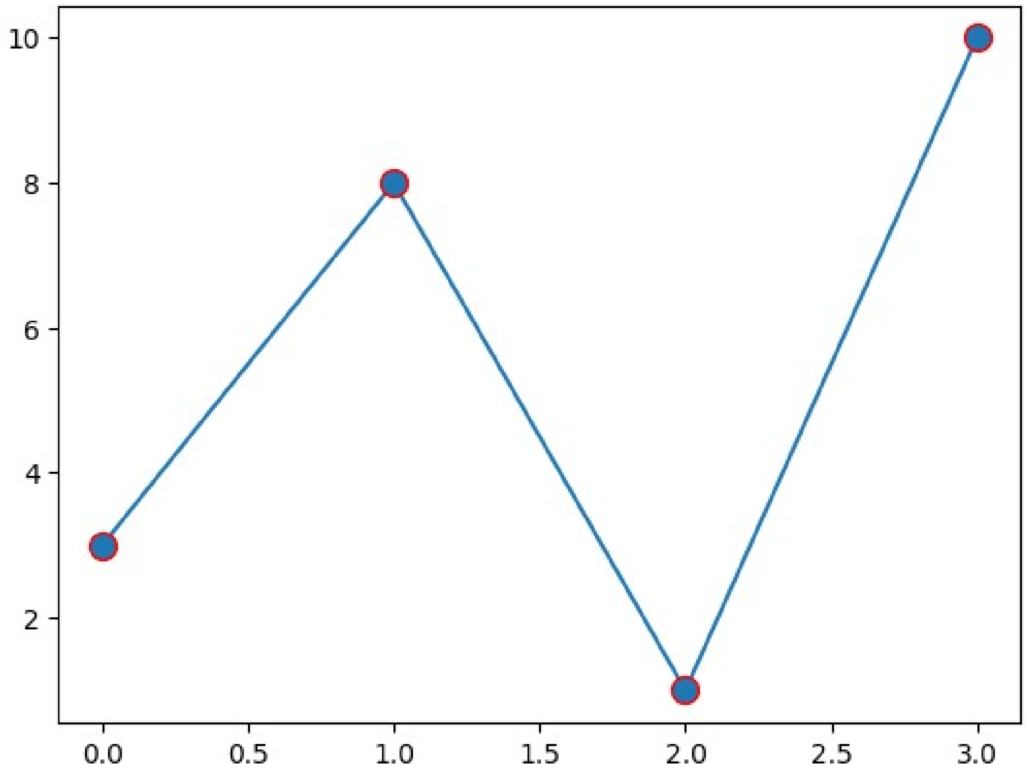
```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o', ms = 20)
plt.show()
```



Marker Edge Color: You can use the keyword argument ***markeredgecolor*** or the shorter ***mec*** to set the color of the edge of the markers :

- Set the EDGE color to red :

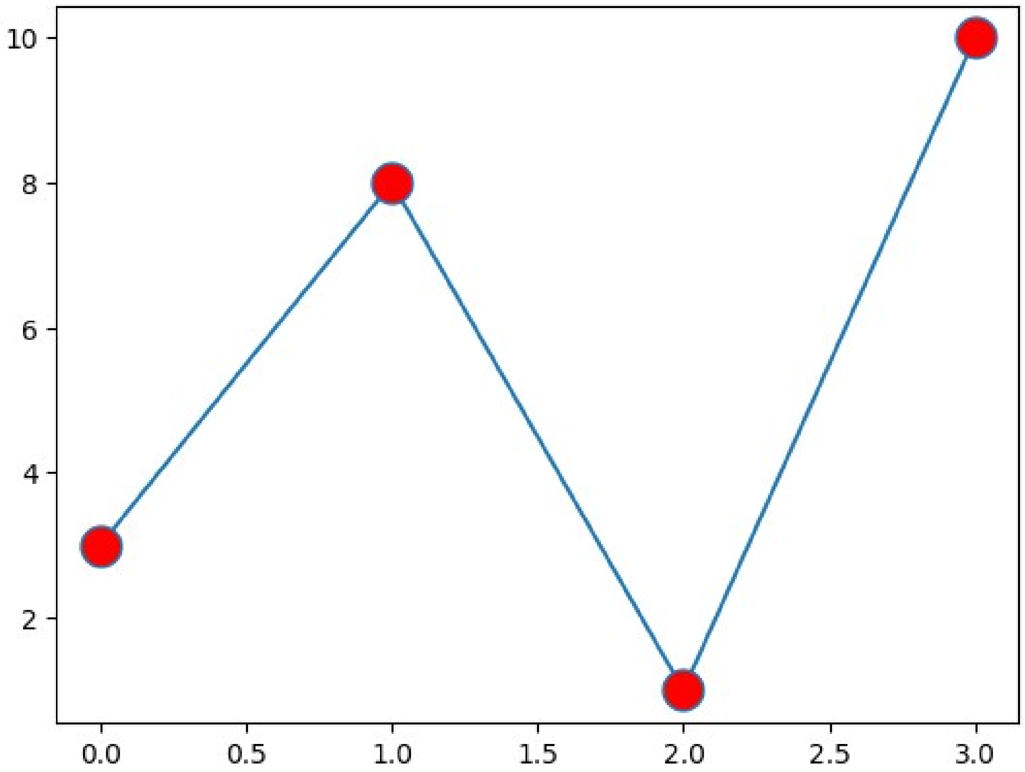
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([3, 8, 1, 10])

plt.plot(y, marker = 'o', ms = 10, mec = 'r')
plt.show()
```



marker face color : You can use the keyword argument ***markerfacecolor*** or the shorter ***mfc*** to set the color inside the edge of the markers :

- Set the FACE color to red :

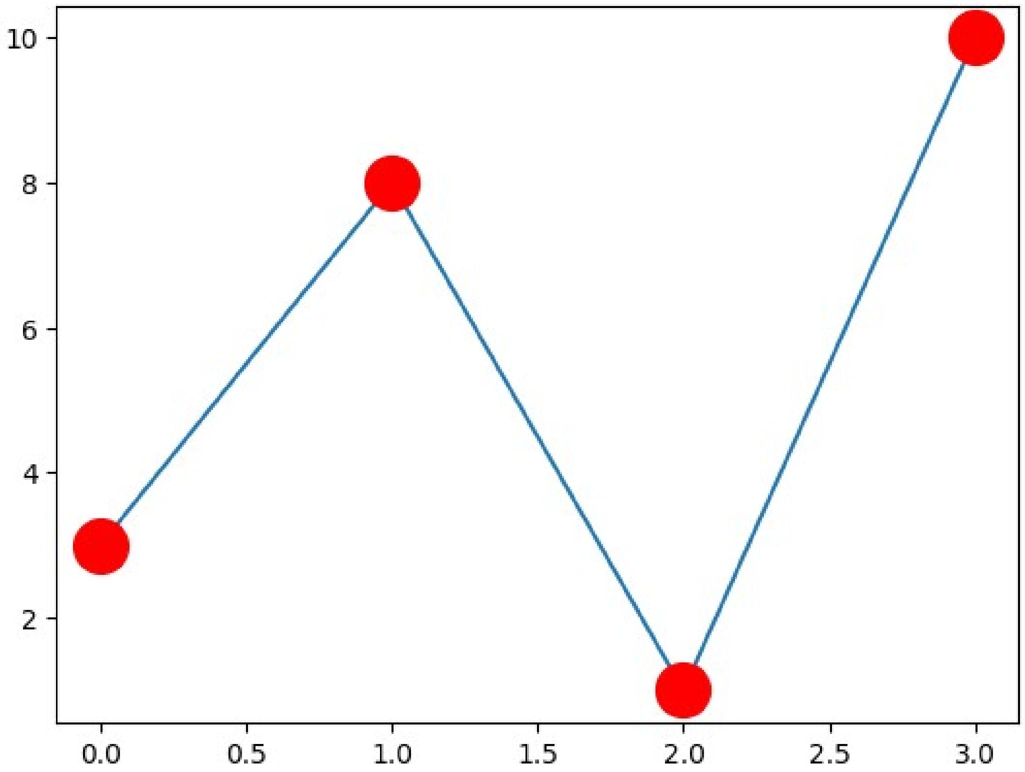
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([3, 8, 1, 10])

plt.plot(y, marker = 'o', ms = 15, mfc = 'r')
plt.show()
```



Use oth the mec and mfc arguments to color the entire marker.

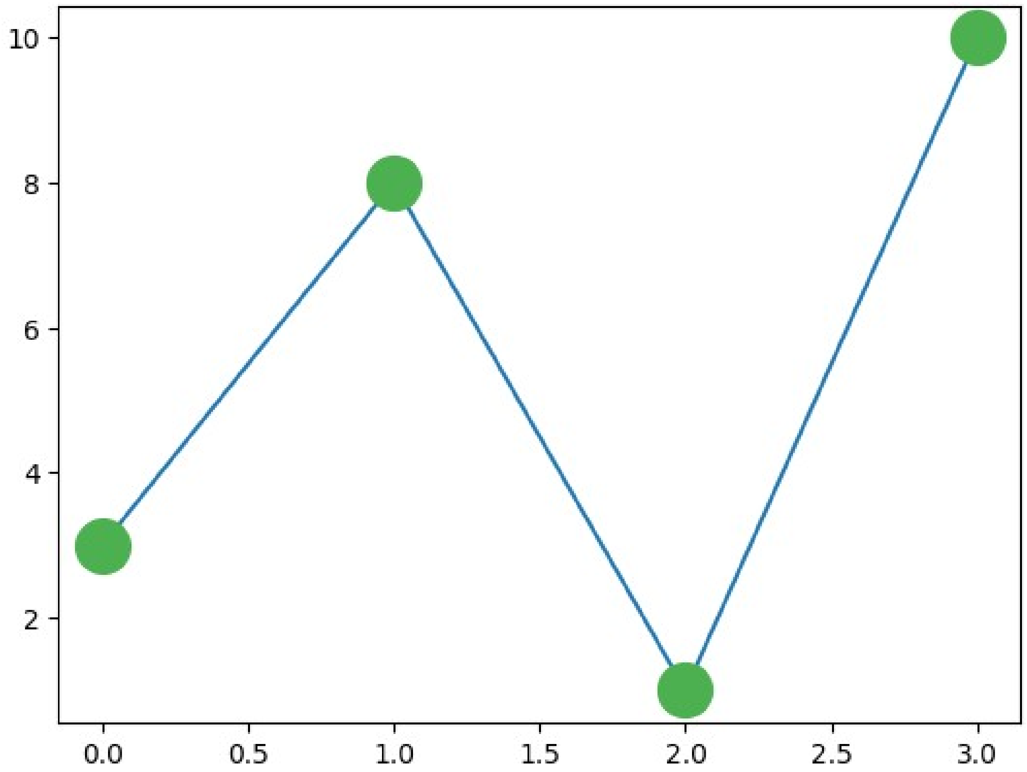
- Set the color of both the edge and the face to red :

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

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o', ms = 20, mec = 'r', mfc = 'r')
plt.show()
```



- NOTE : You can also use Hexadecimal color values for colors :
- Mark each point with a beautiful green color :

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

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o', ms = 20, mec = '#4CAF50', mfc =
'#4CAF50')
plt.show()
```

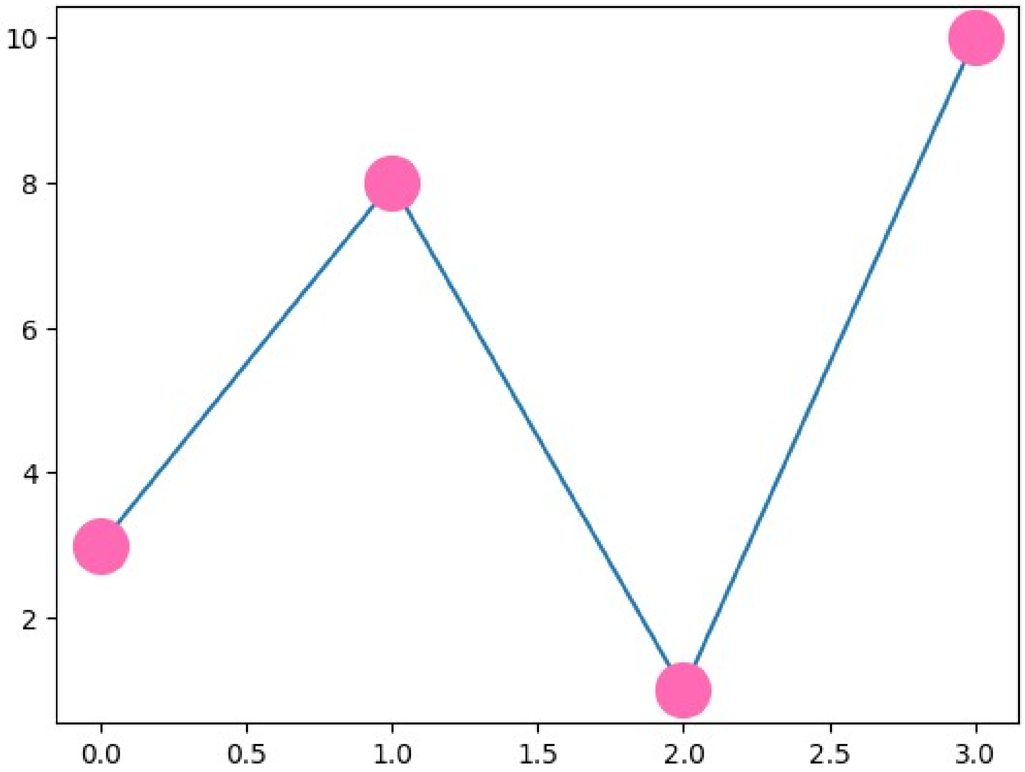


CLICK HERE TO SEE : "colour-code" AND ***"Color-Names"***

example :

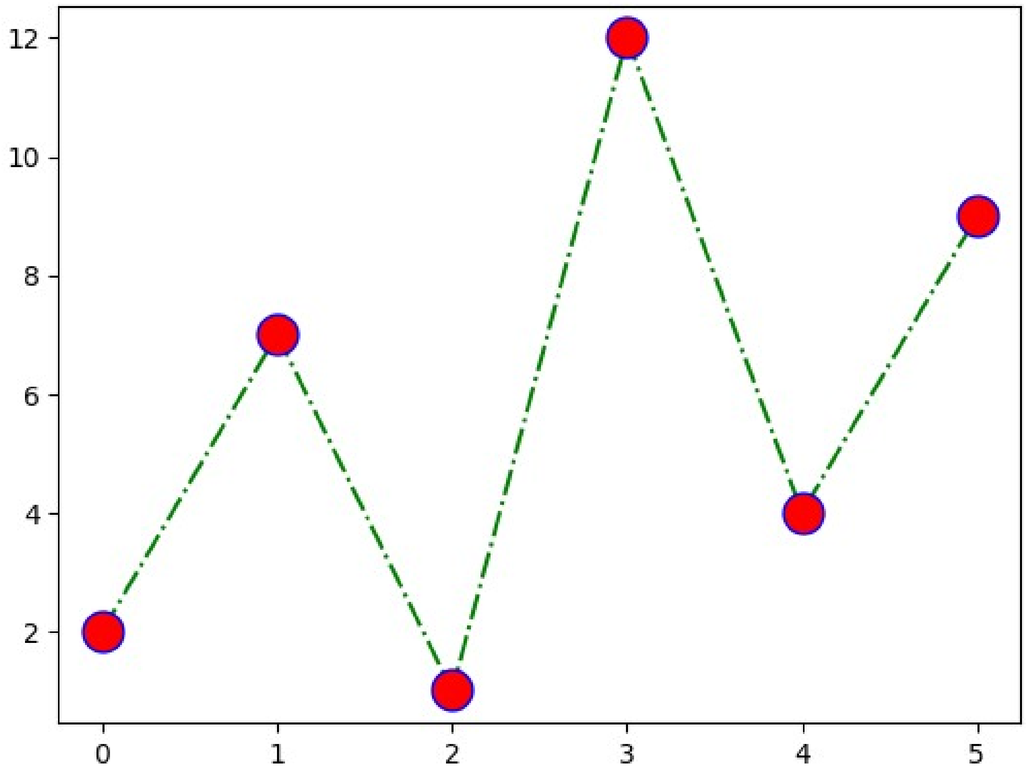
- Mark each point with the color named "hotpink" :

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

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o', ms = 20, mec = 'hotpink', mfc =
'hotpink')
plt.show()
```

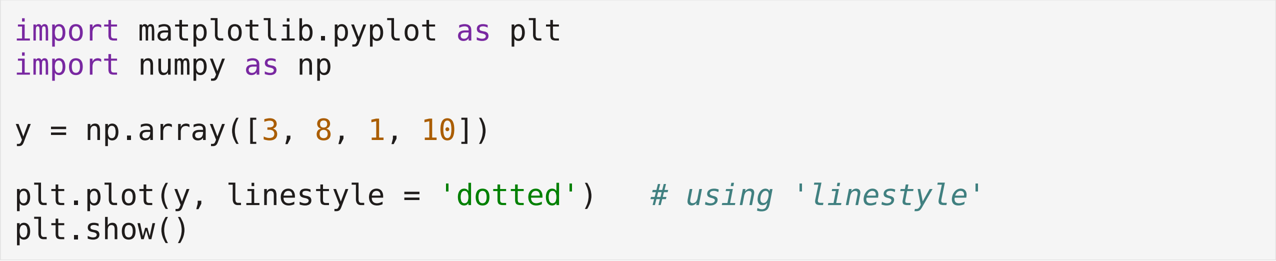


revision : Let's create a plot with using almost every attribute of marker

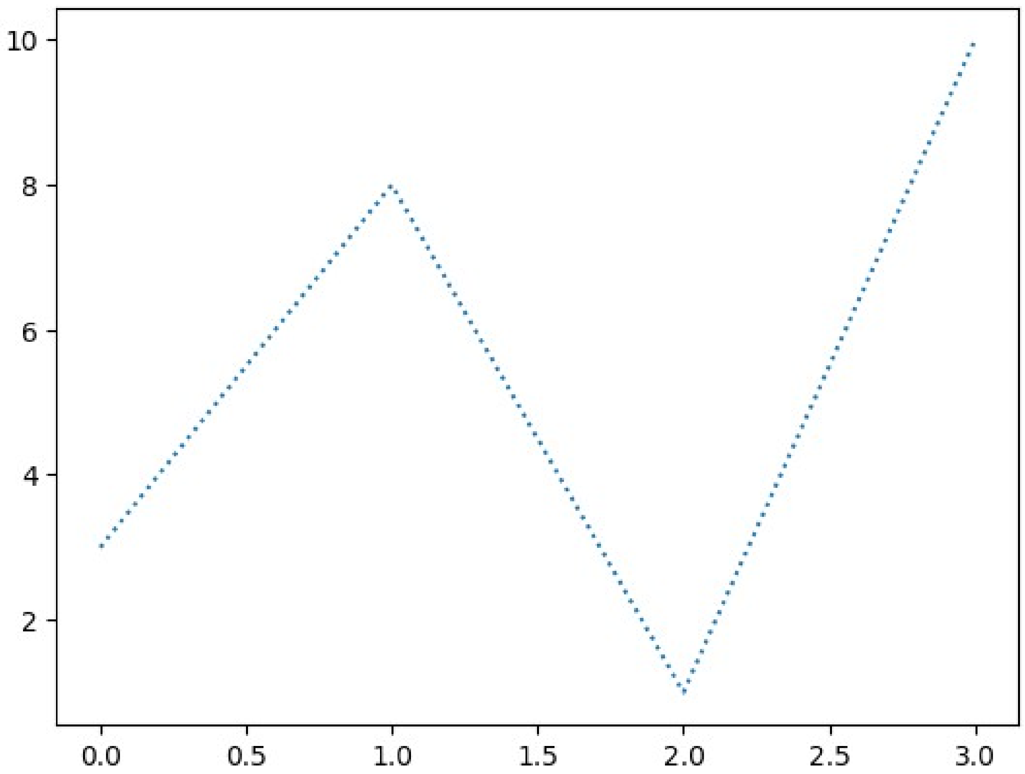
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([2, 7, 1, 12, 4, 9])

plt.plot(y, '-.g', marker='o', mec='b', mfc='r', ms=15)
plt.show()
```



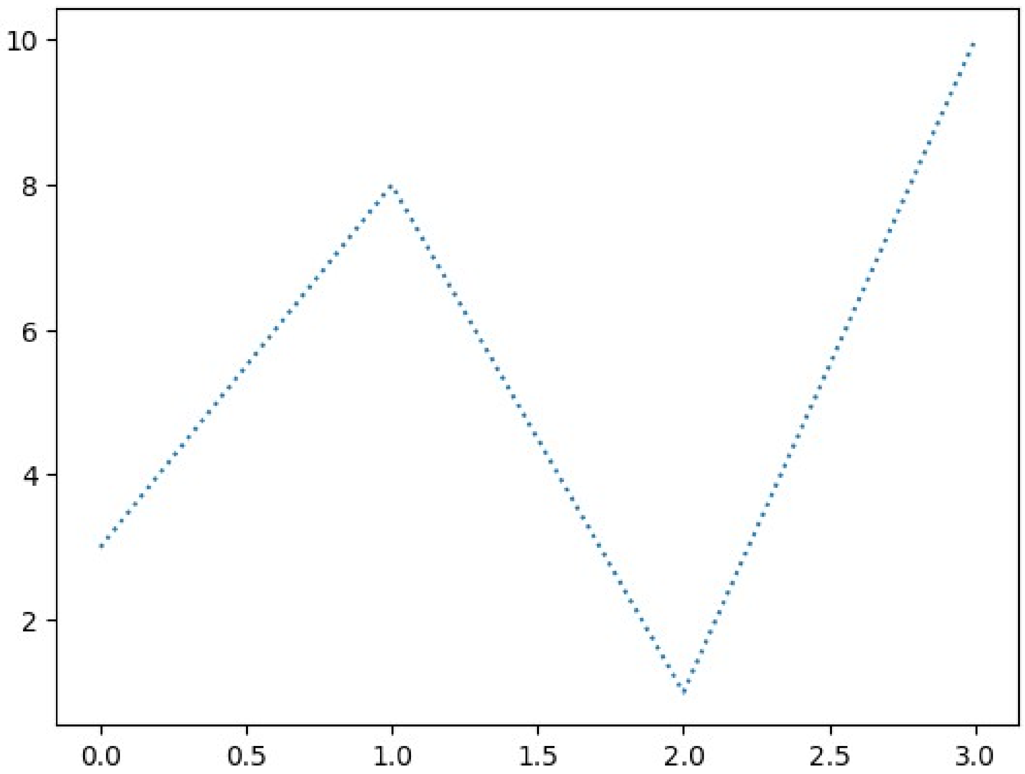
Matplotlib Line : You can use the keyword argument ***linestyle*** or shorter ***ls***, to change the style of the plotted line.



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

y = np.array([3, 8, 1, 10])

plt.plot(y, linestyle = 'dotted')   # using 'linestyle'
plt.show()
```

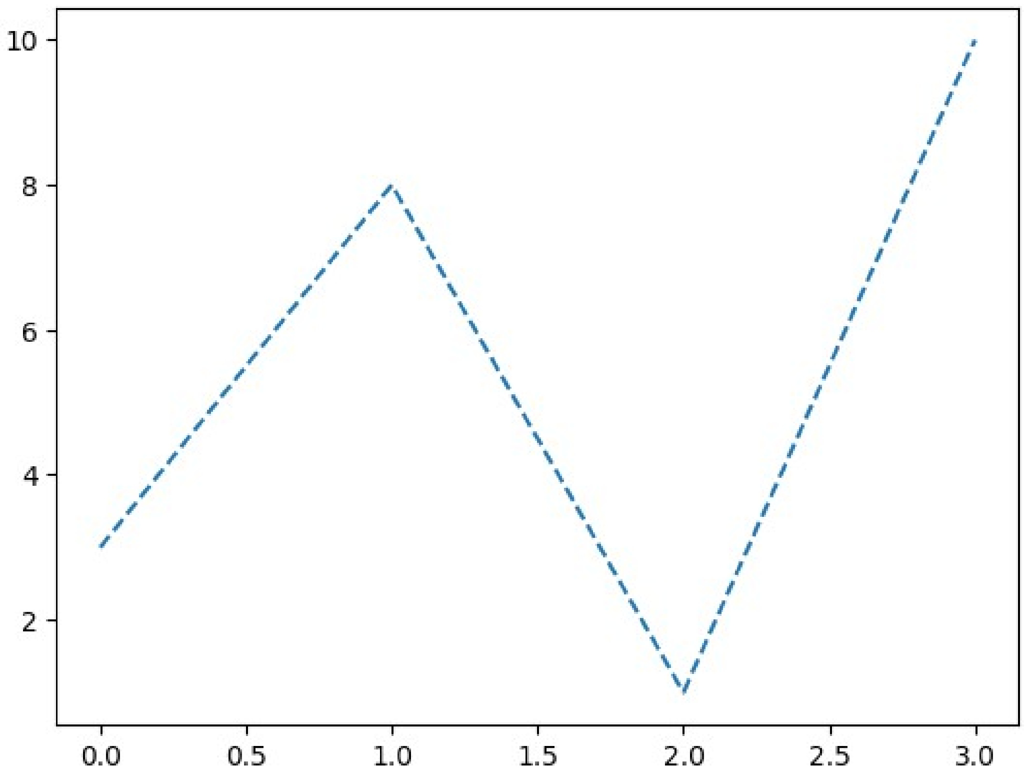


```python
import matplotlib.pyplot as plt
import numpy as np

y = np.array([3, 8, 1, 10])

plt.plot(y, ls = 'dotted')   # using 'ls'
plt.show()
```



- Use a dashed line :

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

y = np.array([3, 8, 1, 10])

plt.plot(y, linestyle = 'dashed')   # using 'ls'
plt.show()
```



Line Styles

You can choose any of these styles:

Line signDescription

'-'solid (default) ':'dotted '--'dashed '-.'dashdot '' or ' 'None

Line Color : You can use the keyword argument color or the shorter ***'c'*** to set the color of the line.

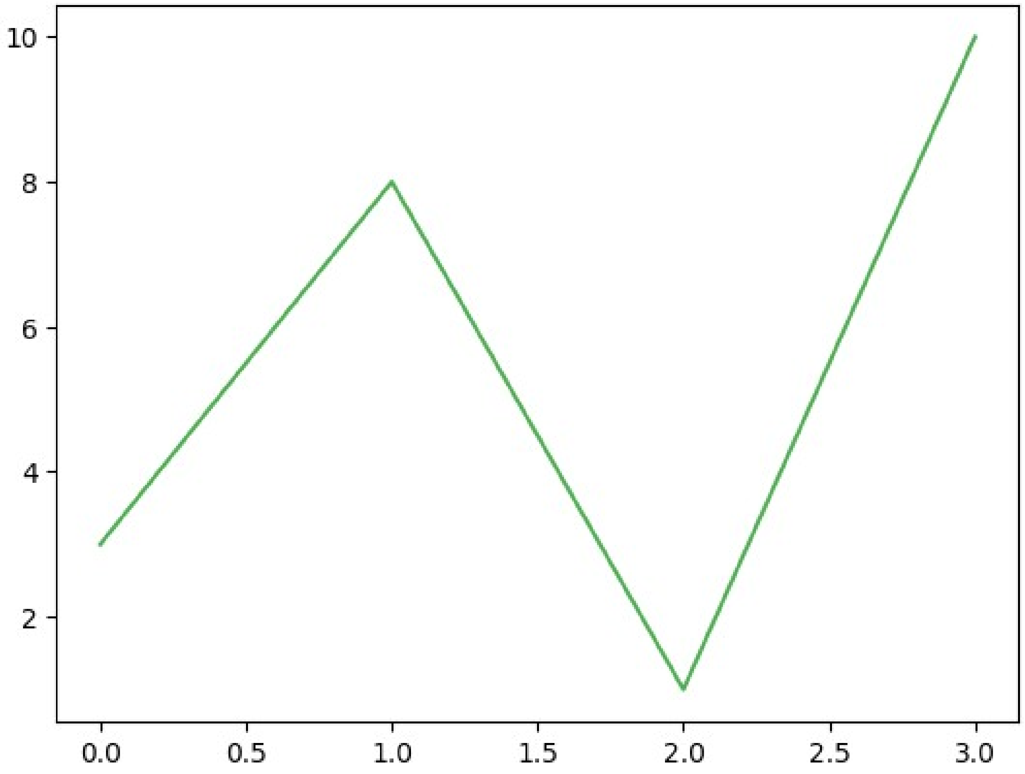
- Set the line color to red :

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

y = np.array([3, 8, 1, 10])

plt.plot(y, color = 'r')
plt.show()
```

- Set the line color-code to green :

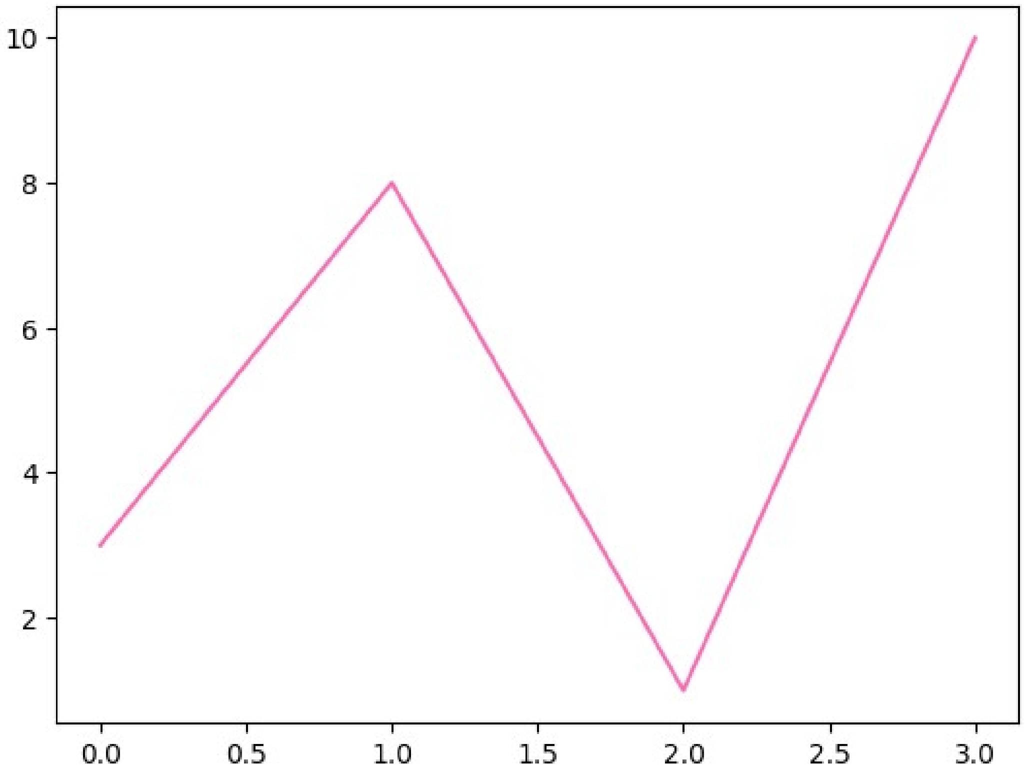
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([3, 8, 1, 10])
```

```
plt.plot(y, c = '#4CAF50')
plt.show()
```



- Set the line color-name hotpink :

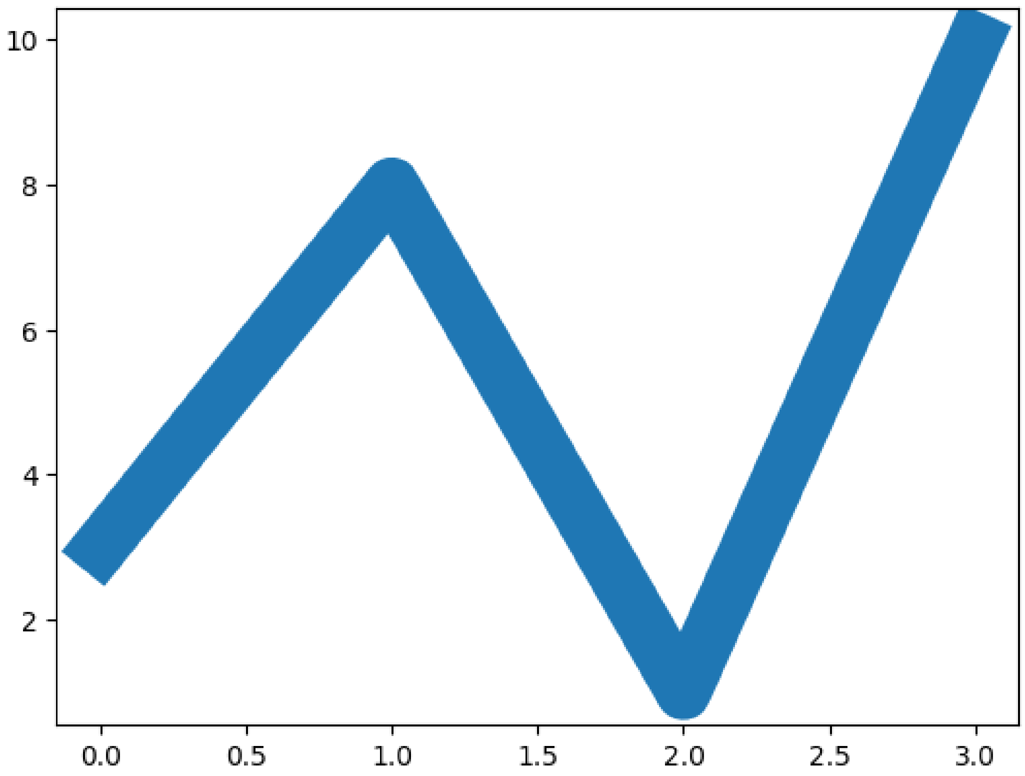
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([3, 8, 1, 10])

plt.plot(y, c = 'hotpink')
plt.show()
```



Line Width : You can use the keyword argument linewidth or the shorter lw to change the width of the line. The value is a floating number, in points:

- Plot with a 20.5pt wide line :

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

y = np.array([3, 8, 1, 10])

plt.plot(y, linewidth = '20.5')
plt.show()
```

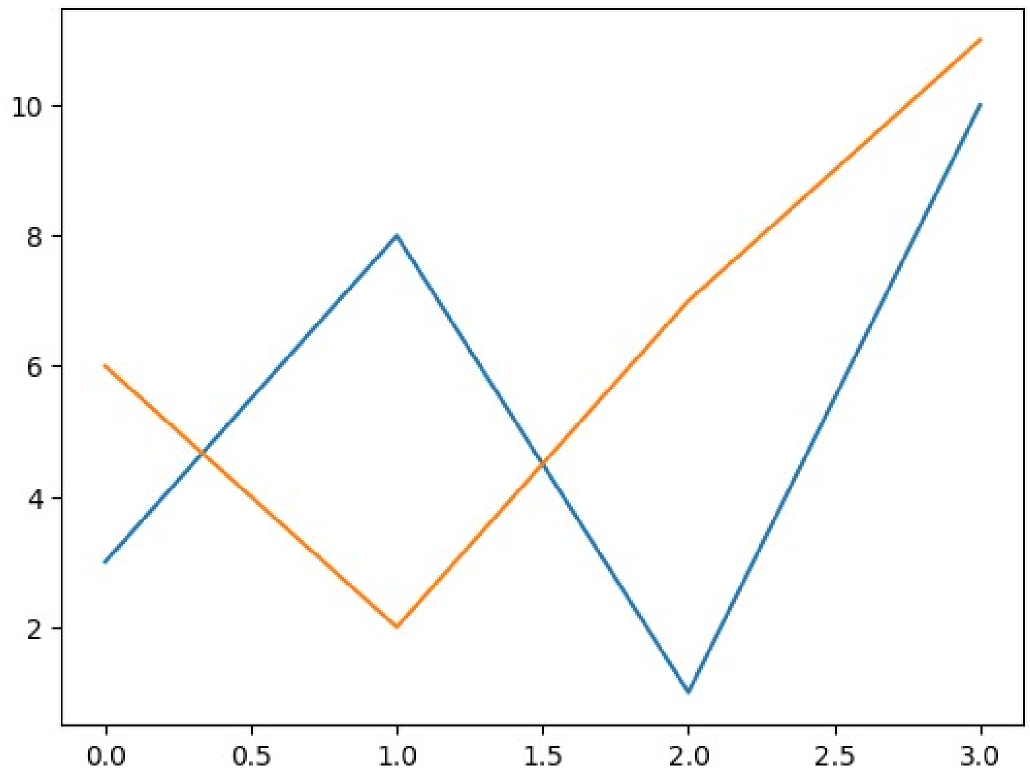


Multiple Lines:

You can plot as many lines as you like by simply adding more plt.plot() functions.

- Draw two lines by specifying a plt.plot() function for each line :

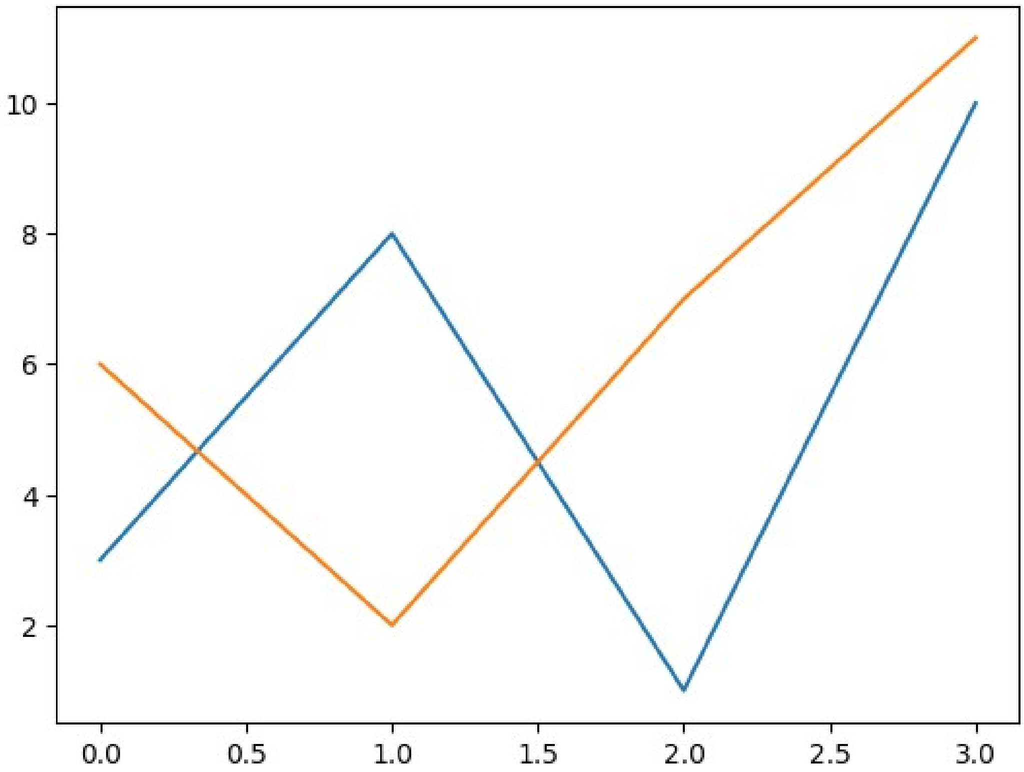
```
import matplotlib.pyplot
import numpy as np

y1 = np.array([3, 8, 1, 10])
y2 = np.array([6, 2, 7, 11])

plt.plot(y1)
plt.plot(y2)

plt.show()
```



The x- and y- values come in pairs:

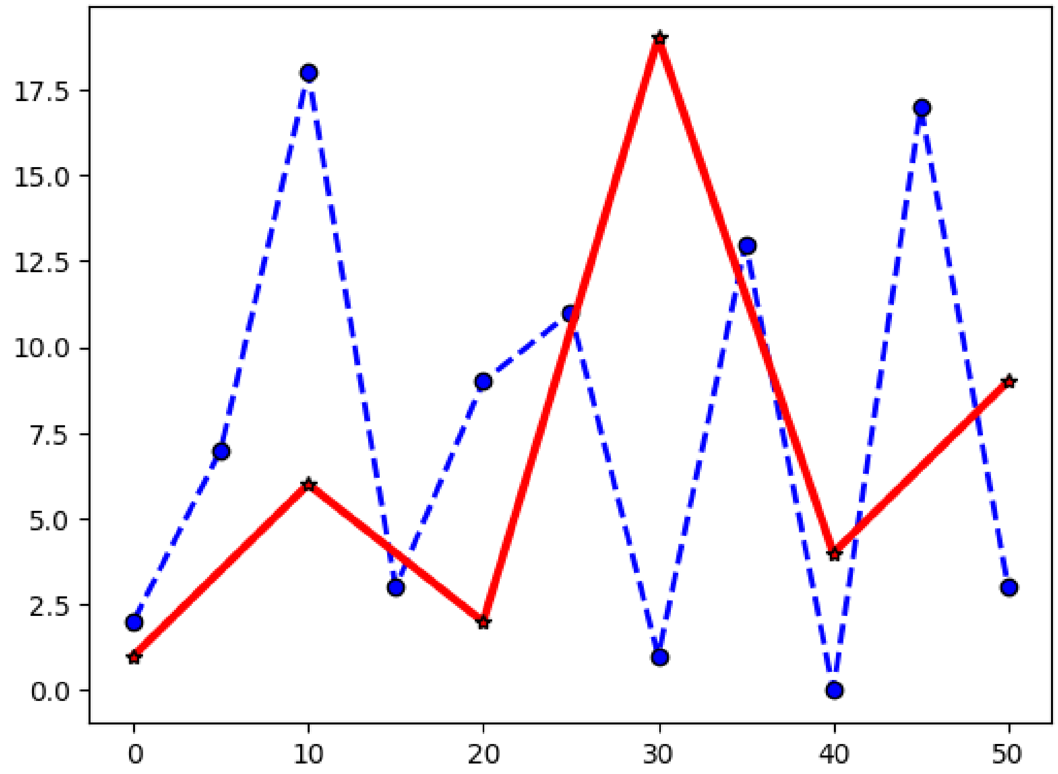
Draw two lines by specifiyng the x- and y-point values for both lines:

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

x1 = np.array([0, 1, 2, 3])
y1 = np.array([3, 8, 1, 10])

x2 = np.array([0, 1, 2, 3])
y2 = np.array([6, 2, 7, 11])

plt.plot(y1)
plt.plot(x2, y2)

plt.show()
```



- Let's make a plot using above topic :

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

x1 = np.array([0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50])
y1 = np.array([2, 7, 18, 3, 9, 11, 1, 13, 0, 17, 3])

x2 = np.array([0, 10, 20, 30, 40, 50])
y2 = np.array([1, 6, 2, 19, 4, 9])


plt.plot(x1,y1, marker = 'o', c = 'b', ls = '--', linewidth = '2', mec
= '#000000')
plt.plot(x2,y2, marker = '*', c = 'r', ls = '-', linewidth = '3', mec
= '#000000')
plt.show()
```



Matplotlib Labels and Title :

let's create labels and title to make our plot more meaningful and usefull.

Create Labels for a Plot : With Pyplot, you can use the xlabel() and ylabel() functions to set a label for the x- and y-axis.

- Add labels to the x- and y-axis :

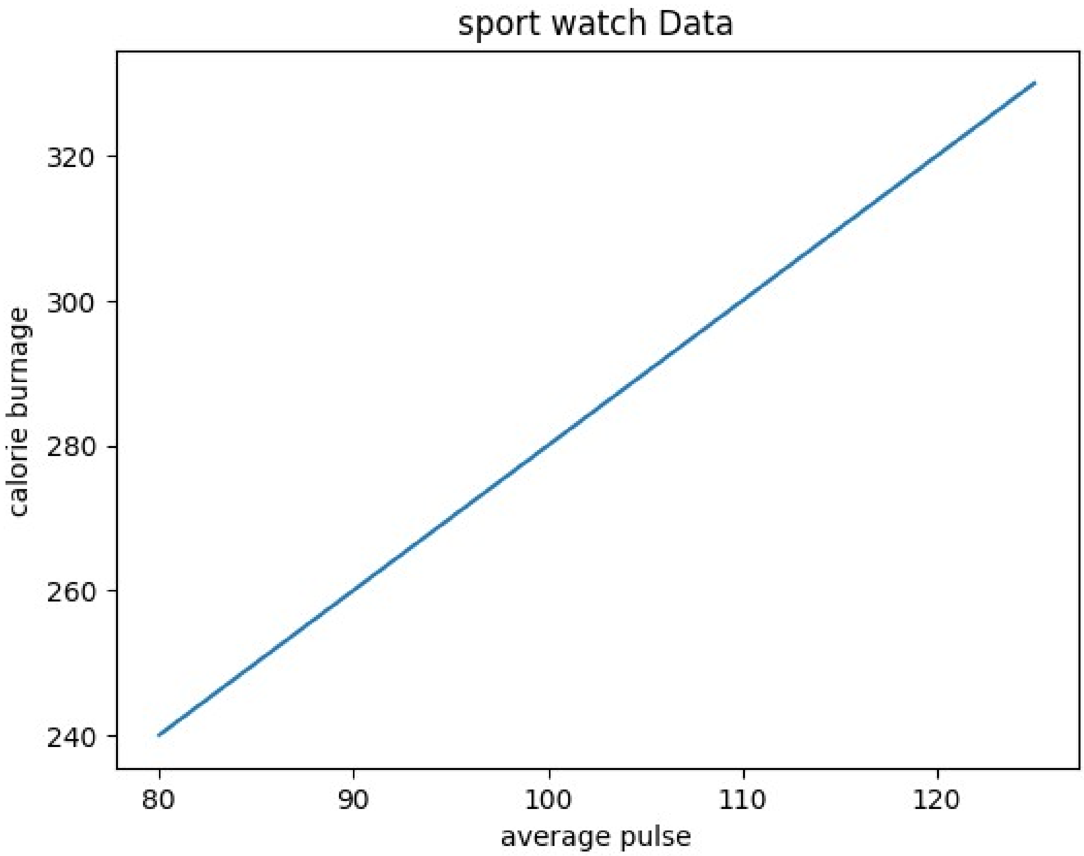
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x,y)

plt.xlabel('average pulse')
plt.ylabel('calorie burnage')

plt.show()
```

Create a Title for a Plot :

With Pyplot, you can use the title() function to set a title for the plot.

- Add a plot title and labels for the x- and y-axis :

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

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x,y)

plt.title('sport watch Data')

plt.xlabel('average pulse')
plt.ylabel('calorie burnage')

plt.show()
```

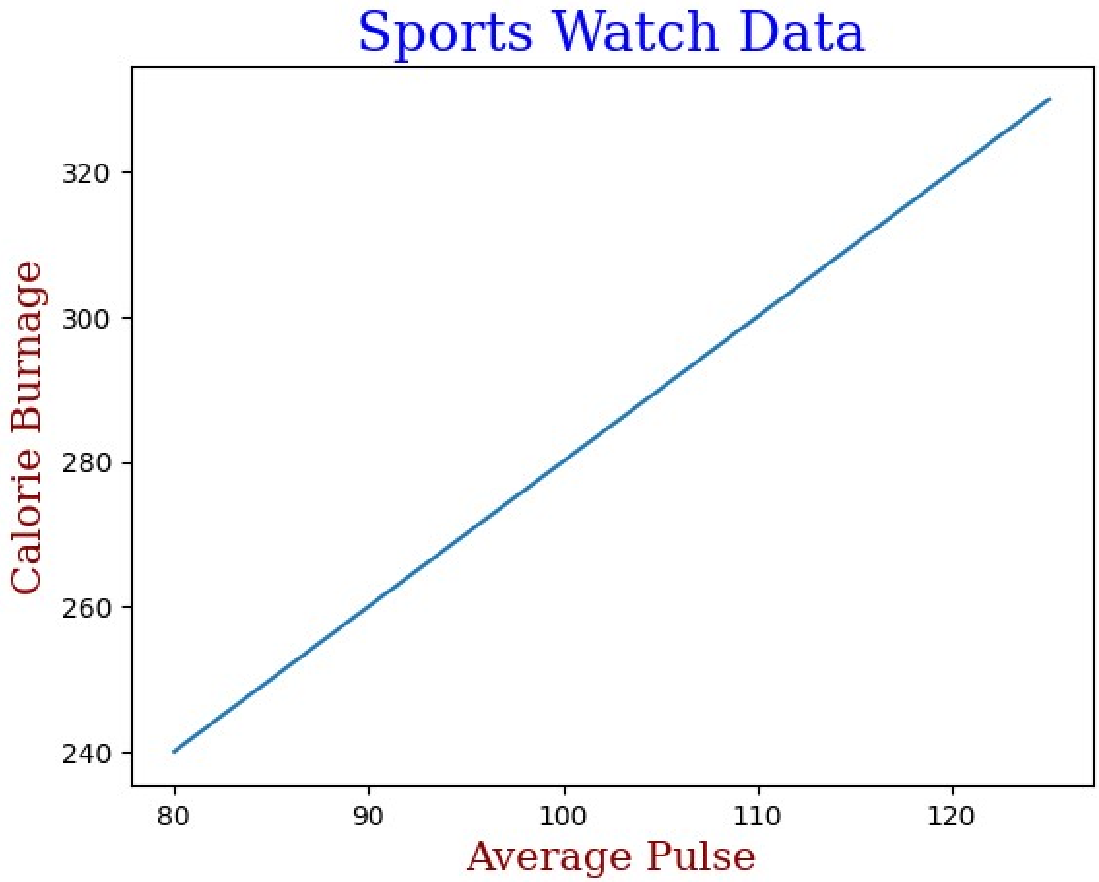


Set Font Properties for Title and Labels : You can use the fontdict parameter in xlabel(), ylabel(), and title() to set font properties for the title and labels.

- Set font properties for the title and labels :

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

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
```

```
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}

plt.title("Sports Watch Data", fontdict = font1)

plt.xlabel("Average Pulse", fontdict = font2)
plt.ylabel("Calorie Burnage", fontdict = font2)

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

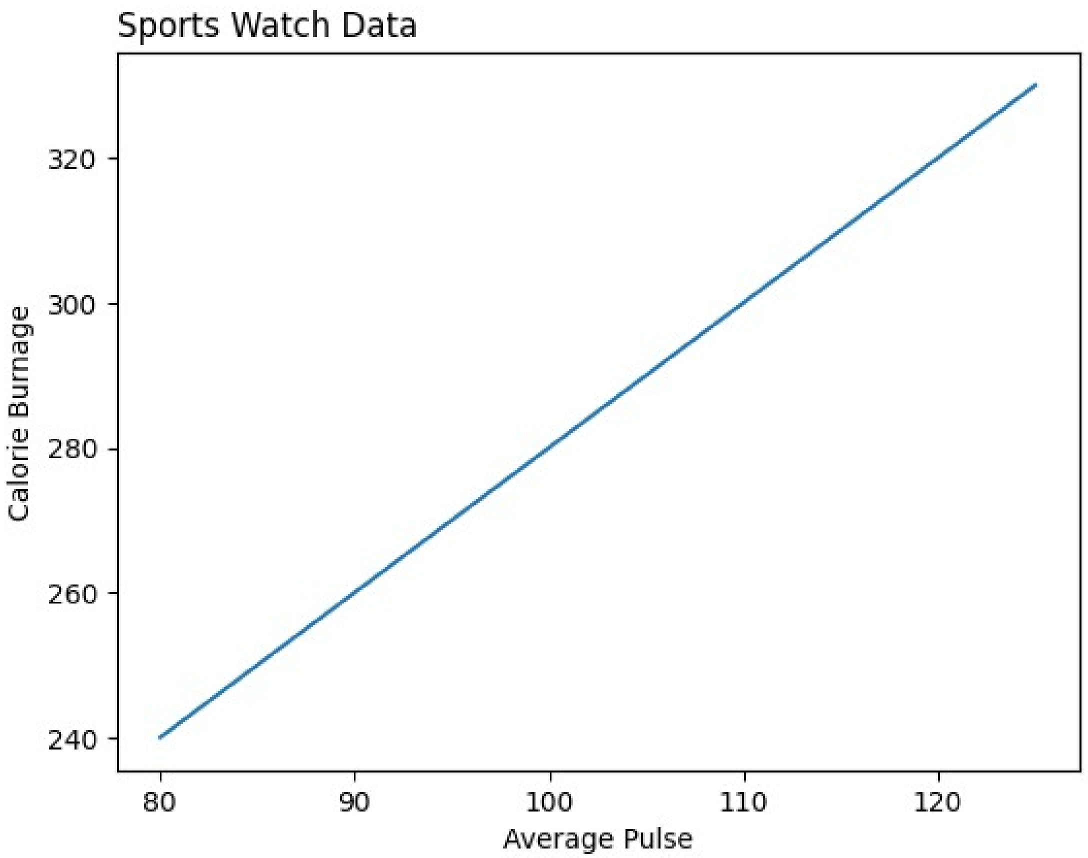


Position the Title : You can use the loc parameter in title() to position the title. Legal values are: 'left', 'right', and 'center'. Default value is 'center'.

- Position the title to the left :

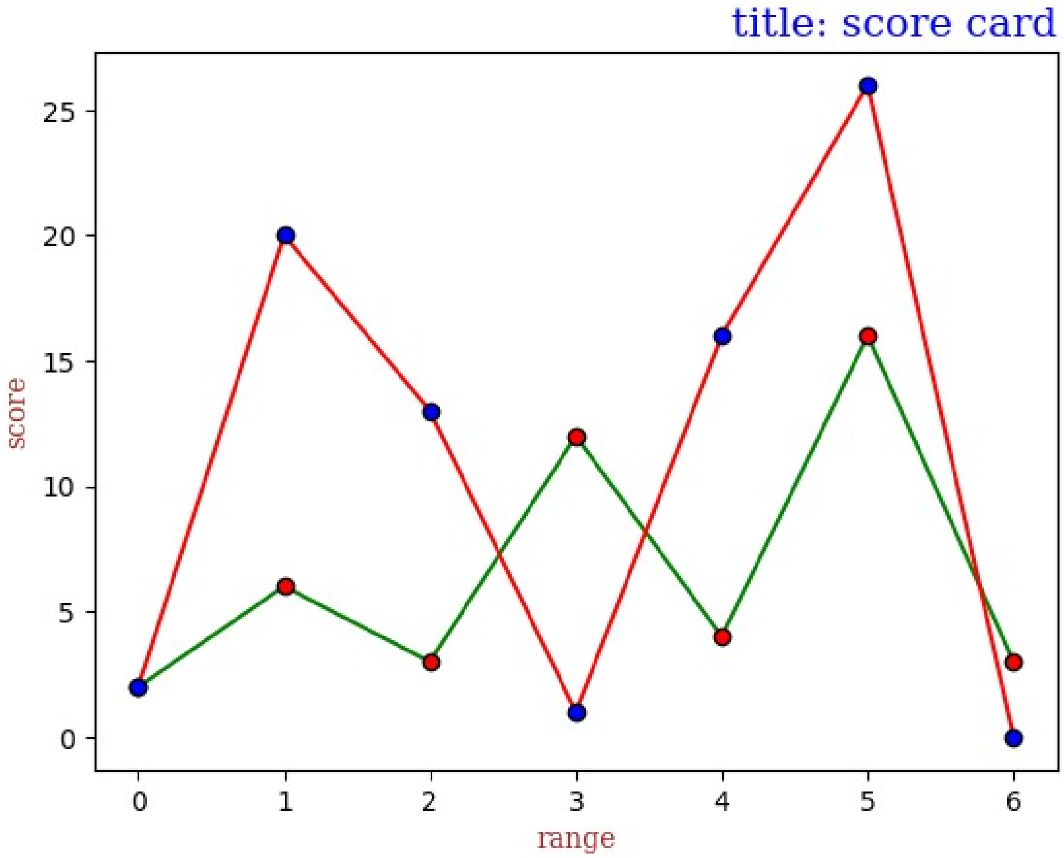
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])
```

```
plt.title("Sports Watch Data", loc = 'left')
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

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



let's revise it :

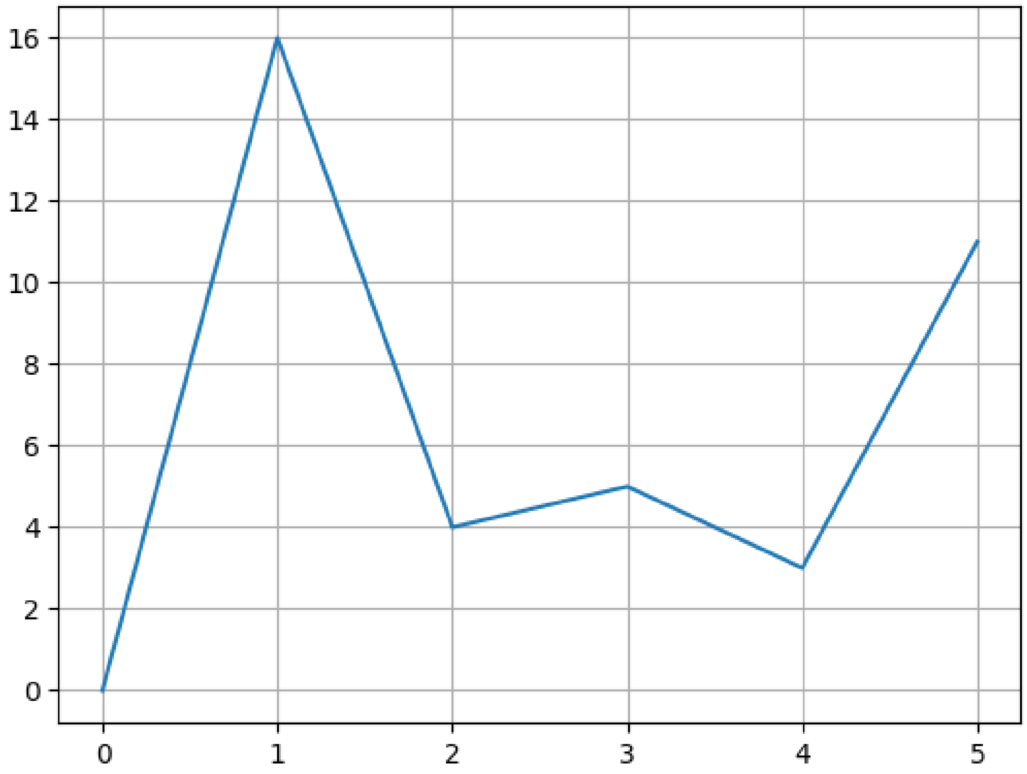
```
import matplotlib.pyplot as plt
import numpy as np

y1 = np.array([2, 6, 3, 12, 4, 16, 3])
y2 = np.array([2, 20, 13, 1, 16, 26, 0])

font1 = {'family':'serif','color':'blue','size':15}
font2 = {'family':'serif','color':'brown','size':10}

plt.plot(y1, marker='o', mec='#000000',mfc='r', c= 'g')
plt.plot(y2, marker='o', mec='#000000',mfc='b',c ='r')

plt.title('title: score card', fontdict = font1, loc = 'right')
plt.xlabel("range", fontdict = font2)
plt.ylabel("score", fontdict = font2)

plt.show()
```



Matplotlib Adding Grid Lines :

plt.grid() :A dd Grid Lines to a Plot With Pyplot

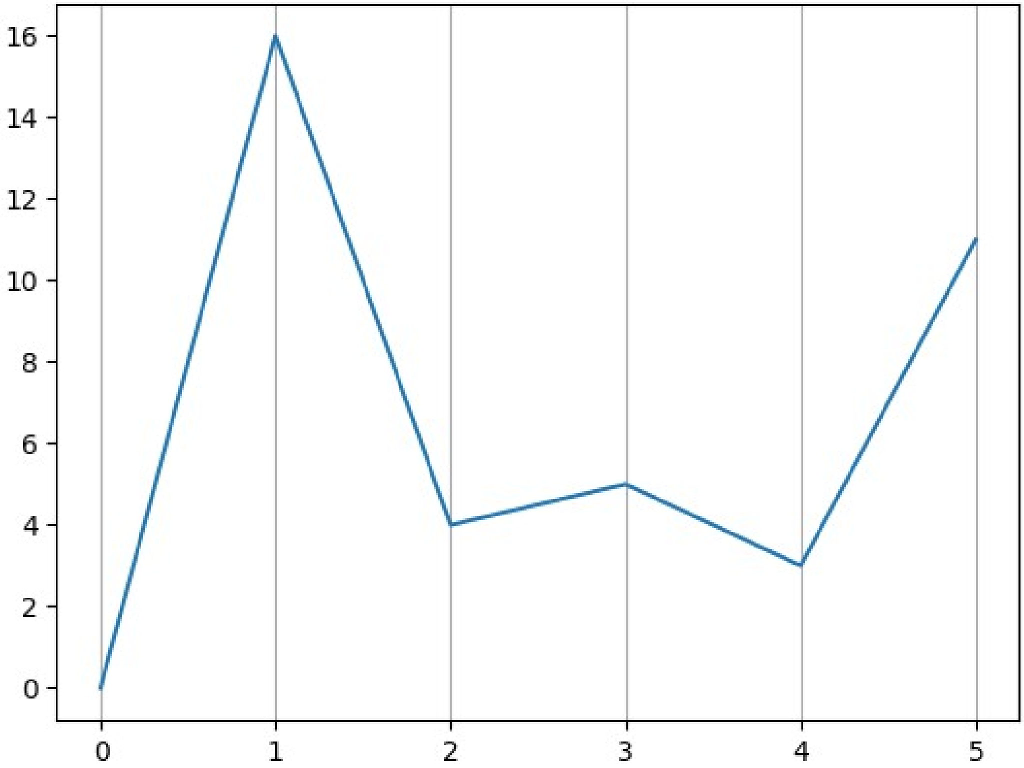
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([0, 16, 4, 5, 3, 11])

plt.grid()
plt.plot(y)
plt.show()
```



Specify Which Grid Lines to Display : you can specify which axis grid lines you want, for it, we use 'axis' parameter in grid().

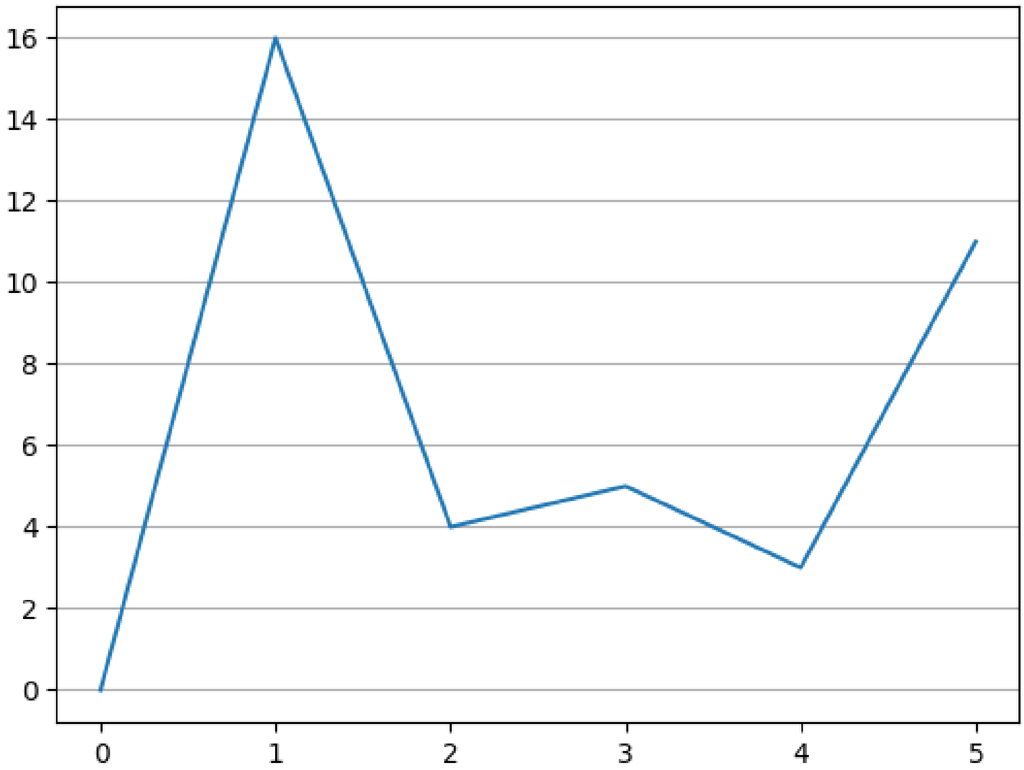
- for x-axis :

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

y = np.array([0, 16, 4, 5, 3, 11])

plt.grid(axis='x')
plt.plot(y)
plt.show()
```



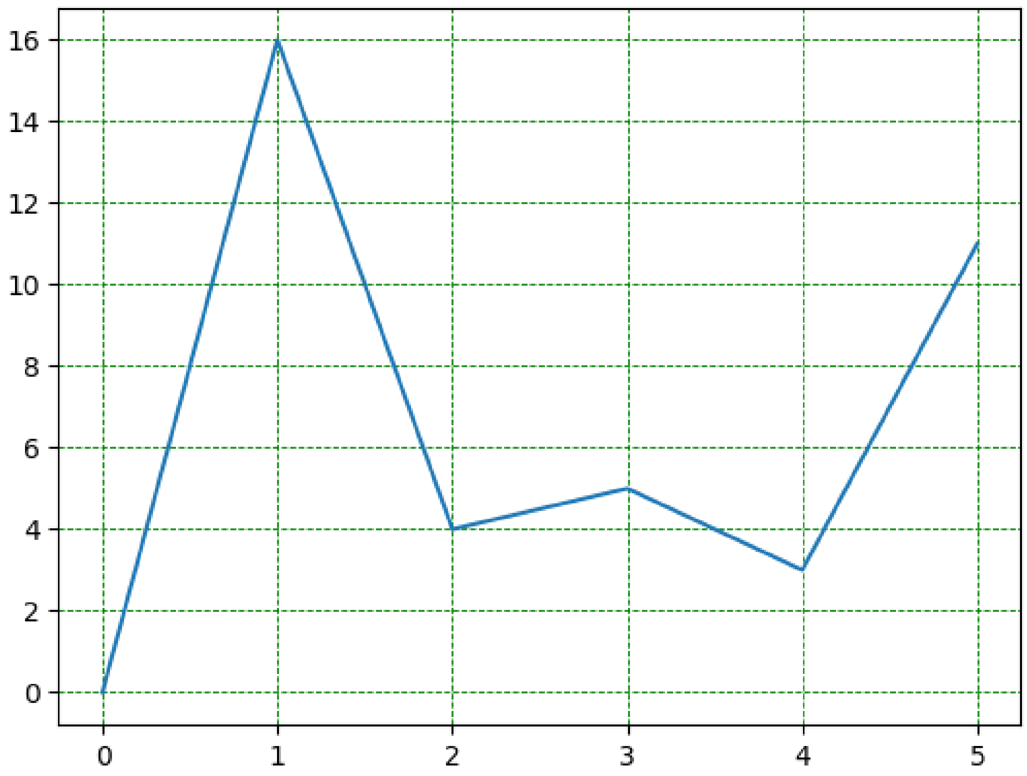
- for y-axis :

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

y = np.array([0, 16, 4, 5, 3, 11])

plt.grid(axis = 'y')
plt.plot(y)
plt.show()
```



grid line Properties : You can also set the line properties of the grid, like this: grid(color = 'color', linestyle = 'linestyle', linewidth = number).

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

y = np.array([0, 16, 4, 5, 3, 11])

plt.grid(color='green', linestyle='--',linewidth=0.6)
plt.plot(y)
plt.show()
```

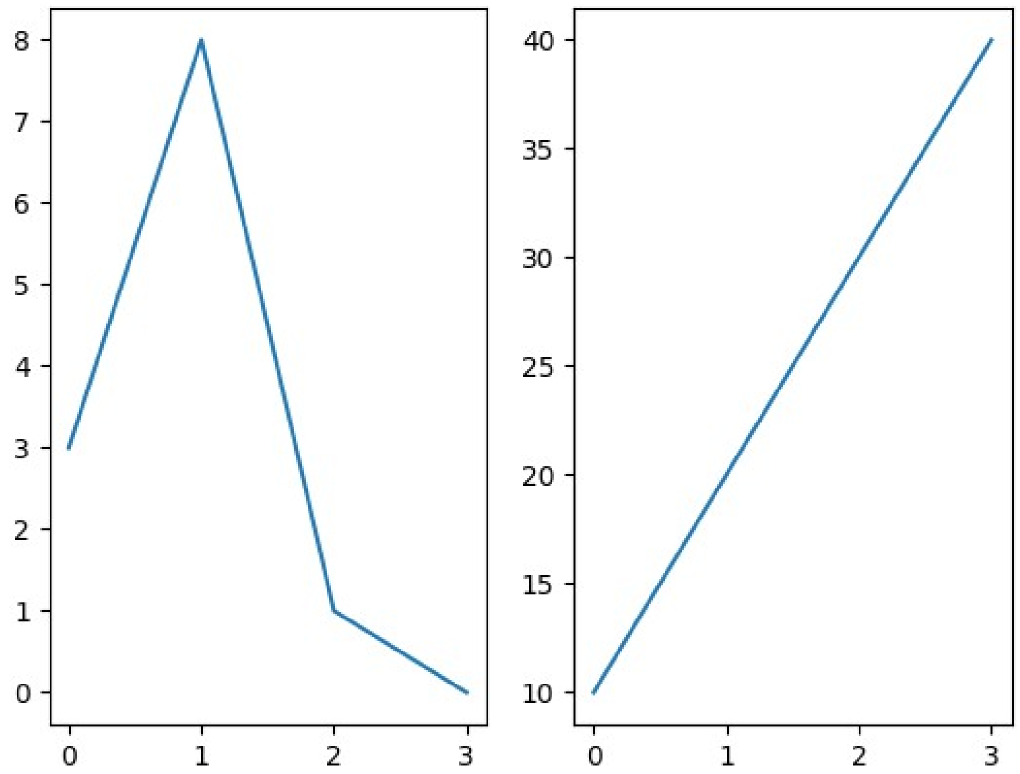


Matplotlib Subplot:

matplotlib allows you to draw more then one plots in single box using ***subplot()***

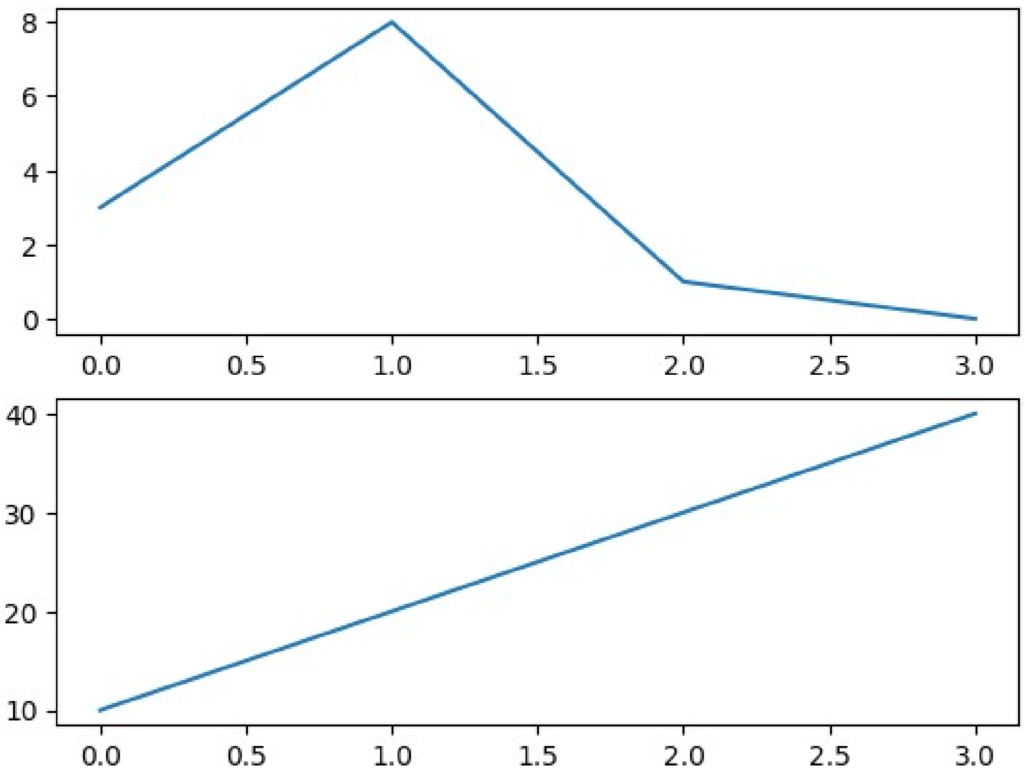
- Draw 2 plots in 1 rows with 2 columns :

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

# plot 1
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 0])
plt.subplot(1, 2, 1)               # subplot(1row,1cols,1pos)
plt.plot(x,y)

# plot 2
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(1, 2, 2)             # subplot(1row,1cols,2pos
plt.plot(x,y)

plt.show()
```



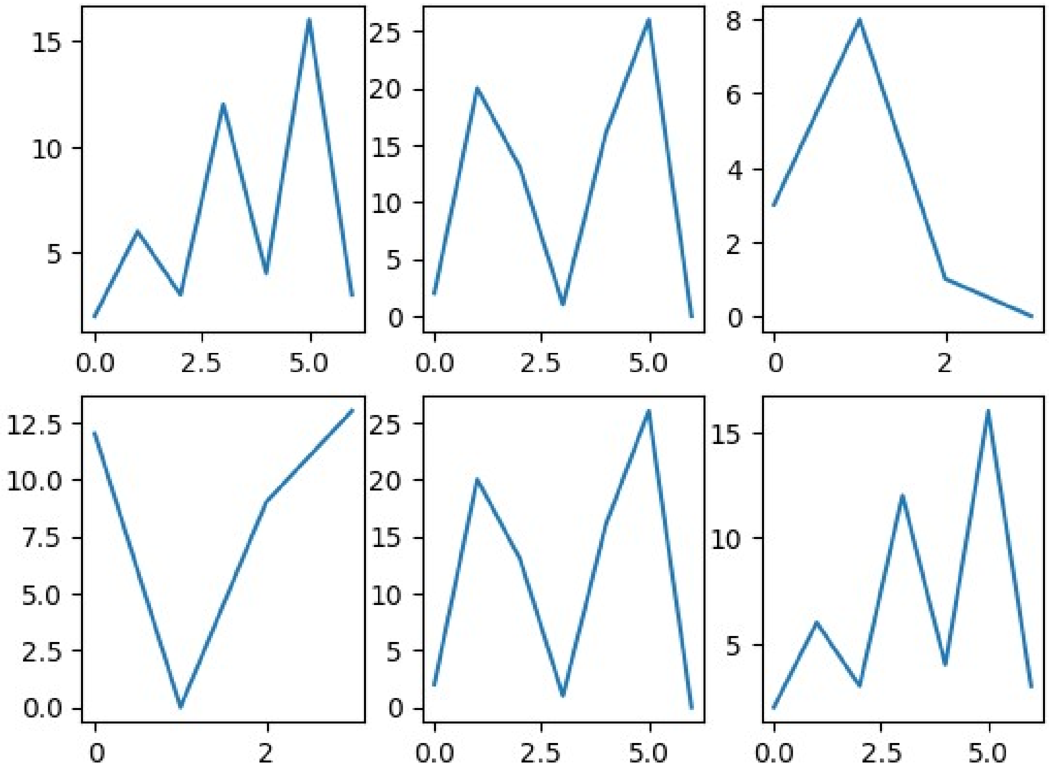
- Draw 2 plots in 1 rows with 2 columns :

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

# plot 1
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 0])
plt.subplot(2, 1, 1)               # subplot(2row,1cols,1pos)
plt.plot(x,y)

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

plt.subplot(2, 1, 2) # subplot(2row,1cols,2pos
plt.plot(x,y)

plt.show()
```



- 6 plot on same page :

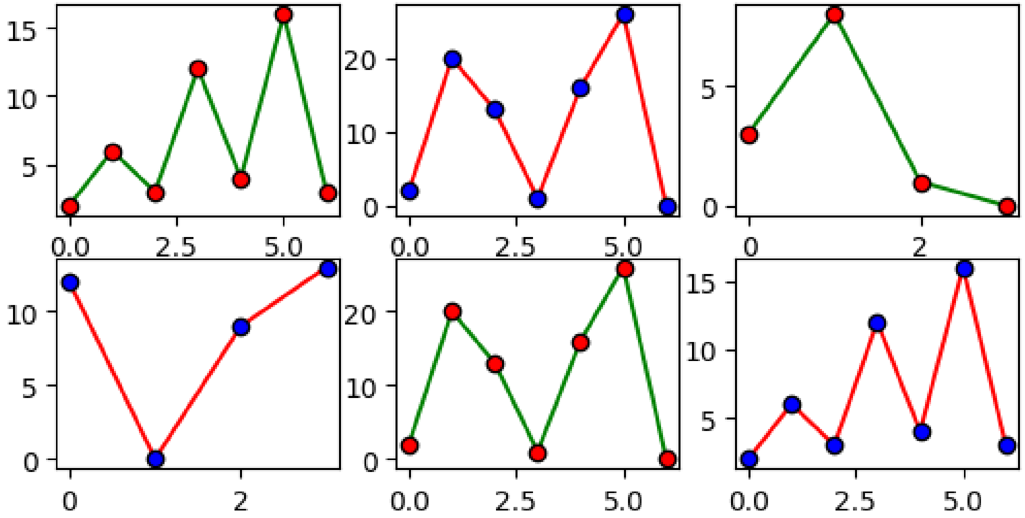
```
import matplotlib.pyplot as plt
import numpy as np

y1 = np.array([2, 6, 3, 12, 4, 16, 3])
plt.subplot(2, 3, 1)
plt.plot(y1)

y2 = np.array([2, 20, 13, 1, 16, 26, 0])
plt.subplot(2, 3, 2)
plt.plot(y2)

y3 = np.array([3, 8, 1, 0])
plt.subplot(2, 3, 3)
plt.plot(y3)

y4 = np.array([12, 0, 9, 13])
plt.subplot(2, 3, 4)
plt.plot(y4)

y5 = np.array([2, 20, 13, 1, 16, 26, 0])
plt.subplot(2, 3, 5)
plt.plot(y5)

y6 = np.array([2, 6, 3, 12, 4, 16, 3])
plt.subplot(2, 3, 6)
plt.plot(y6)

plt.show()
```



- add title to each plotfun with plot :

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

y1 = np.array([2, 6, 3, 12, 4, 16, 3])
plt.subplot(3, 3, 1)
plt.plot(y1, marker='o', mec='#000000',mfc='r', c= 'g')

y2 = np.array([2, 20, 13, 1, 16, 26, 0])
plt.subplot(3, 3, 2)
plt.plot(y2, marker='o', mec='#000000',mfc='b',c ='r')

y3 = np.array([3, 8, 1, 0])
plt.subplot(3, 3, 3)
plt.plot(y3, marker='o', mec='#000000',mfc='r', c= 'g')

y4 = np.array([12, 0, 9, 13])
plt.subplot(3, 3, 4)
plt.plot(y4, marker='o', mec='#000000',mfc='b',c ='r')

y5 = np.array([2, 20, 13, 1, 16, 26, 0])
```

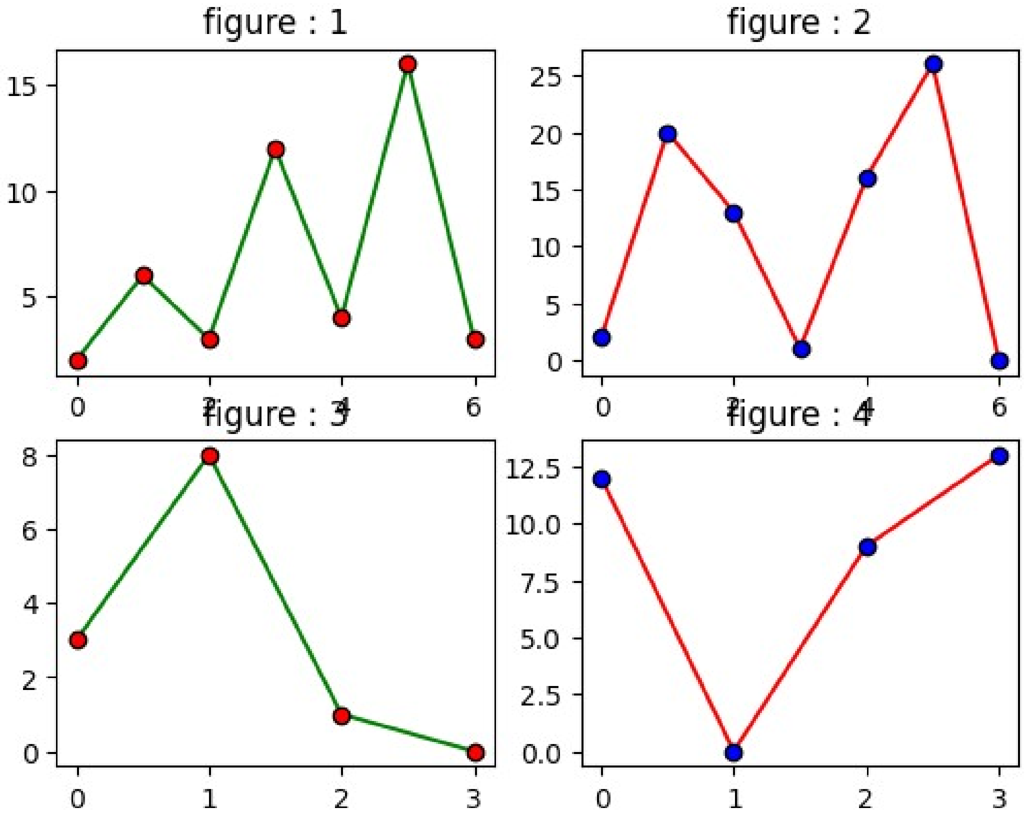
```
plt.subplot(3, 3, 5)
plt.plot(y5, marker='o', mec='#000000',mfc='r', c= 'g')

y6 = np.array([2, 6, 3, 12, 4, 16, 3])
plt.subplot(3, 3, 6)
plt.plot(y6, marker='o', mec='#000000',mfc='b',c ='r')

plt.show()
```



Add Title on each plot in multiplotb : You can add a title to each plot with the title() function:

- add title to each plot :

```
import matplotlib.pyplot as plt
import numpy as np
y1 = np.array([2, 6, 3, 12, 4, 16, 3])
plt.subplot(2, 2, 1)
plt.plot(y1, marker='o', mec='#000000',mfc='r', c= 'g')
plt.title('figure : 1')

y2 = np.array([2, 20, 13, 1, 16, 26, 0])
plt.subplot(2, 2, 2)
plt.plot(y2, marker='o', mec='#000000',mfc='b',c ='r')
plt.title('figure : 2')

y3 = np.array([3, 8, 1, 0])
plt.subplot(2, 2, 3)
plt.plot(y3, marker='o', mec='#000000',mfc='r', c= 'g')
plt.title('figure : 3')

y4 = np.array([12, 0, 9, 13])
plt.subplot(2, 2, 4)
plt.plot(y4, marker='o', mec='#000000',mfc='b',c ='r')
plt.title('figure : 4')
plt.show()
```

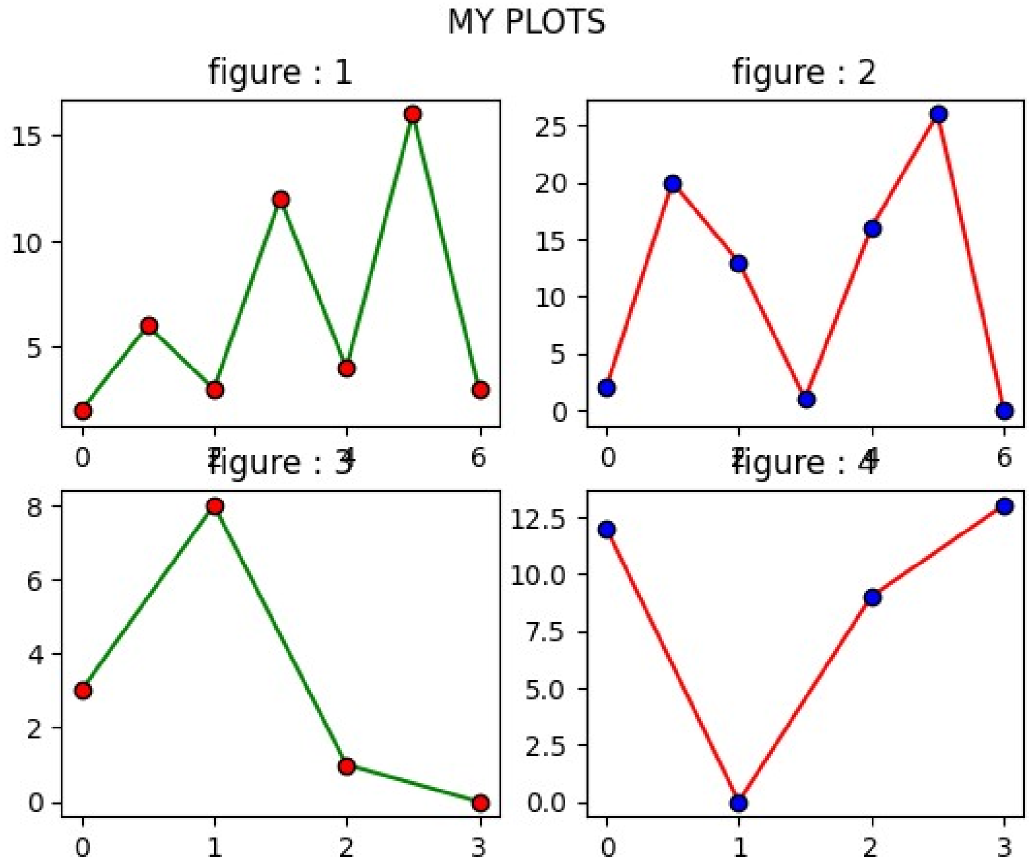


Super Title

You can add a title to the entire figure with the suptitle() function as heading to collection of plots.

- add super-title/heading ofeach plot :

```
import matplotlib.pyplot as plt
import numpy as np
y1 = np.array([2, 6, 3, 12, 4, 16, 3])
plt.subplot(2, 2, 1)
plt.plot(y1, marker='o', mec='#000000',mfc='r', c= 'g')
plt.title('figure : 1')

y2 = np.array([2, 20, 13, 1, 16, 26, 0])
plt.subplot(2, 2, 2)
plt.plot(y2, marker='o', mec='#000000',mfc='b',c ='r')
plt.title('figure : 2')

y3 = np.array([3, 8, 1, 0])
plt.subplot(2, 2, 3)
plt.plot(y3, marker='o', mec='#000000',mfc='r', c= 'g')
plt.title('figure : 3')

y4 = np.array([12, 0, 9, 13])
plt.subplot(2, 2, 4)
plt.plot(y4, marker='o', mec='#000000',mfc='b',c ='r')
plt.title('figure : 4')

plt.suptitle('MY PLOTS')
plt.show()
```



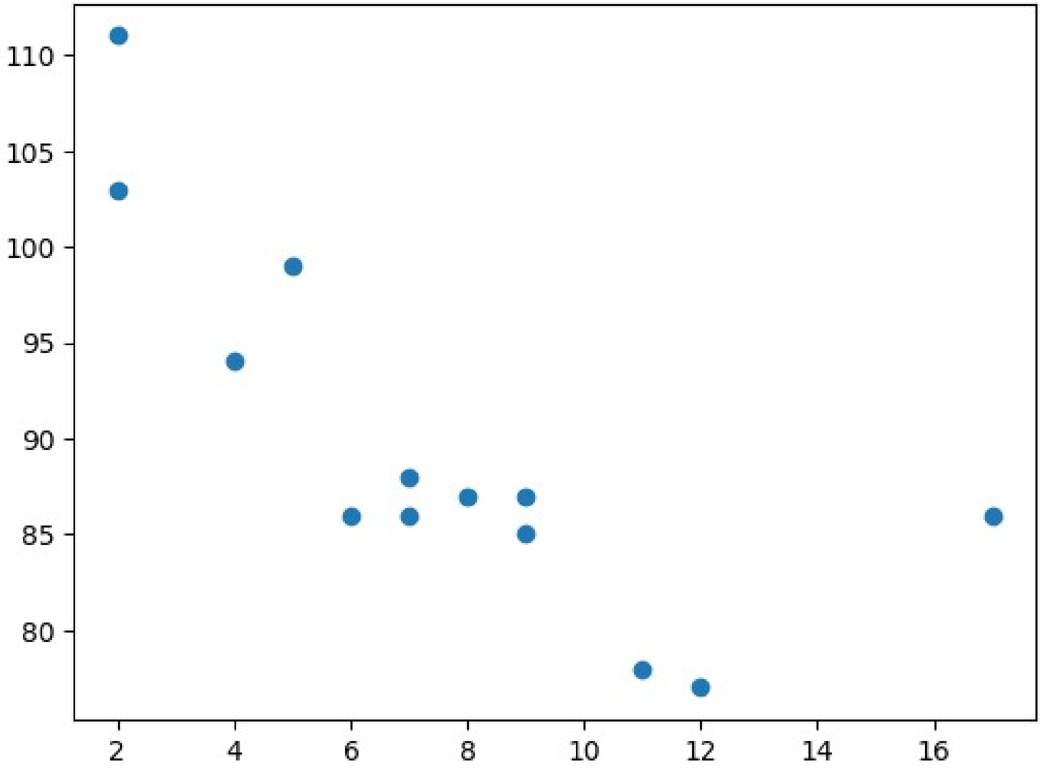
Matplotlib Scatter:

pyplot.scatter(): With Pyplot, you can use the scatter() function to draw a scatter plot. The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis

- A simple scatter plot :

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

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
```

```
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

plt.scatter(x,y)
plt.show()
```



EXPLAINATION :

- The observation in the example above is the result of 13 cars passing by.
- The X-axis shows how old the car is.
- The Y-axis shows the speed of the car when it passes. Are there any relationships between the observations?
- It seems that the newer the car, the faster it drives, but that could be a coincidence, after all we only registered 13 cars.

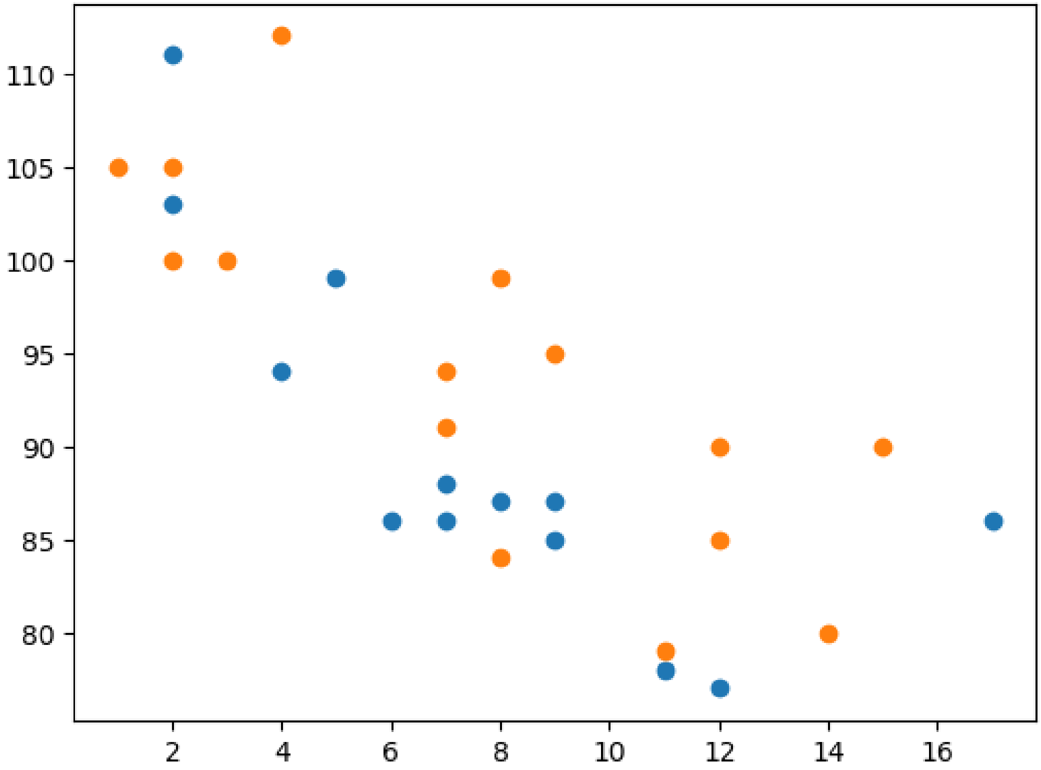
Compare Plots :

In the example above, there seems to be a relationship between speed and age, but what if we plot the observations from another day as well? Will the scatter plot tell us something else?

- Draw two plots on the same figure :

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

#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
```

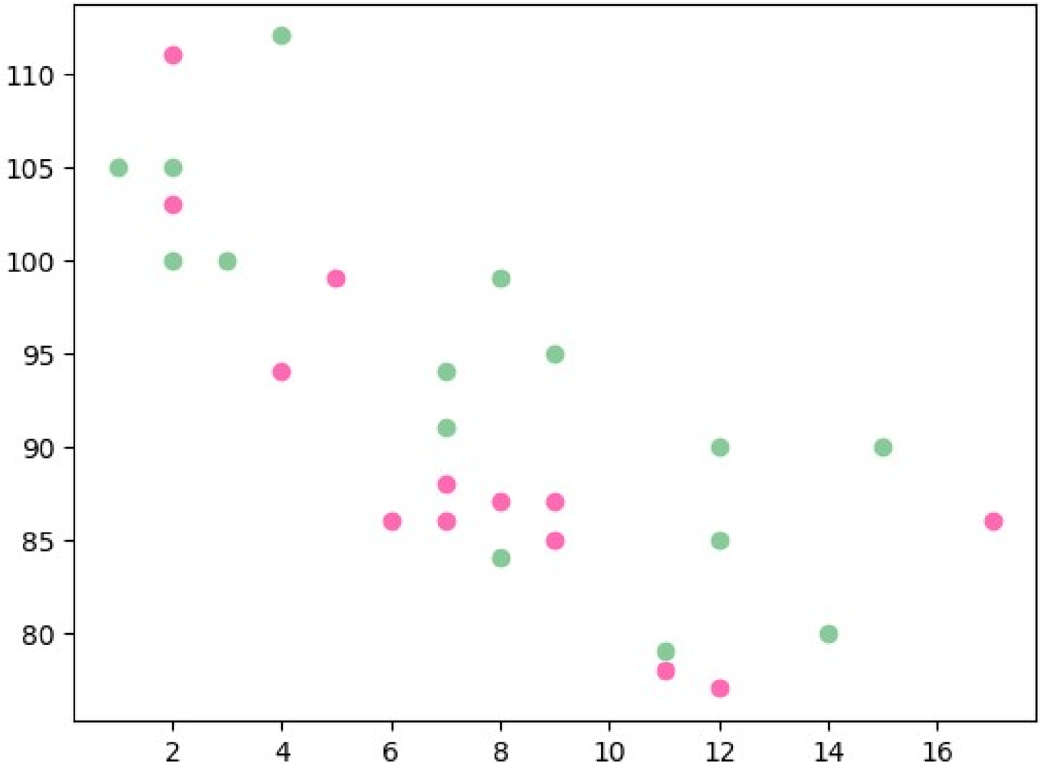
```
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)

#day two, the age and speed of 15 cars:
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y)

plt.show()
```



Colors : You can set your own color for each scatter plot with the color or the ***"c"*** argument.

- Set your own color of the markers :

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

#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y, c='hotpink')

#day two, the age and speed of 15 cars:
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y, color ='#88c999')
plt.show()
```



Color Each Dot : You can even set a specific color for each dot by using an array of colors as value for the c argument: ***Note: You cannot use the color argument for this, only the c argument.***

- Set your own color of the markers :

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

#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors =
(["red","green","blue","yellow","pink","black","orange","purple","beig
e","brown","gray","cyan","magenta"])

plt.scatter(x, y, c=colors)
plt.show()
```

colormap:

- ***print list of colormap values :***

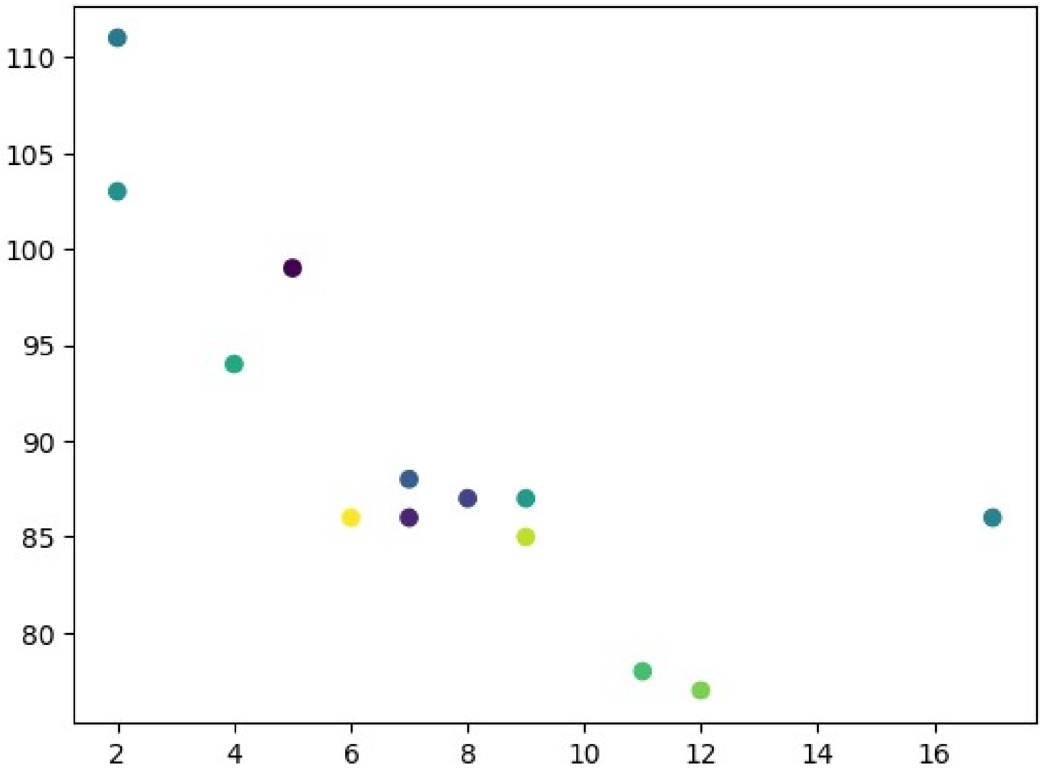
```
from matplotlib import colormaps
list(colormaps)

['magma',
 'inferno',
 'plasma',
 'viridis',
 'cividis',
 'twilight',
 'twilight_shifted',
 'turbo',
 'Blues',
 'BrBG',
 'BuGn',
 'BuPu',
 'CMRmap',
 'GnBu',
 'Greens',
 'Greys',
 'OrRd',
 'Oranges',
 'PRGn',
 'PiYG',
 'PuBu',
 'PuBuGn',
 'PuOr',
 'PuRd',
 'Purples',
 'RdBu',
 'RdGy',
 'RdPu',
 'RdYlBu',
 'RdYlGn',
 'Reds',
 'Spectral',
 'Wistia',
 'YlGn',
 'YlGnBu',
 'YlOrBr',
 'YlOrRd',
 'afmhot',
 'autumn',
 'binary',
 'bone',
 'brg',
 'bwr',
 'cool',
 'coolwarm',
 'copper',
 'cubehelix',
 'flag',
 'gist_earth',
 'gist_gray',
 'gist_heat',
 'gist_ncar',
 'gist_rainbow',
 'gist_stern',
 'gist_yarg',
 'gnuplot',
 'gnuplot2',
 'gray',
 'hot',
 'hsv',
 'jet',
 'nipy_spectral',
 'ocean',
 'pink',
 'prism',
 'rainbow',
 'seismic',
 'spring',
 'summer',
 'terrain',
 'winter',
 'Accent',
 'Dark2',
 'Paired',
 'Pastel1',
 'Pastel2',
 'Set1',
 'Set2',
 'Set3',
 'tab10',
 'tab20',
 'tab20b',
 'tab20c',
 'grey',
 'gist_grey',
 'gist_yerg',
 'Grays',
 'magma_r',
 'inferno_r',
 'plasma_r',
 'viridis_r',
 'cividis_r',
 'twilight_r',
 'twilight_shifted_r',
 'turbo_r',
 'Blues_r',
 'BrBG_r',
 'BuGn_r',
 'BuPu_r',
 'CMRmap_r',
 'GnBu_r',
 'Greens_r',
 'Greys_r',
 'OrRd_r',
 'Oranges_r',
 'PRGn_r',
 'PiYG_r',
 'PuBu_r',
 'PuBuGn_r',
 'PuOr_r',
 'PuRd_r',
 'Purples_r',
 'RdBu_r',
 'RdGy_r',
 'RdPu_r',
 'RdYlBu_r',
 'RdYlGn_r',
 'Reds_r',
 'Spectral_r',
 'Wistia_r',
 'YlGn_r',
 'YlGnBu_r',
 'YlOrBr_r',
 'YlOrRd_r',
 'afmhot_r',
 'autumn_r',
 'binary_r',
 'bone_r',
 'brg_r',
 'bwr_r',
 'cool_r',
 'coolwarm_r',
 'copper_r',
 'cubehelix_r',
 'flag_r',
 'gist_earth_r',
 'gist_gray_r',
 'gist_heat_r',
 'gist_ncar_r',
 'gist_rainbow_r',
 'gist_stern_r',
 'gist_yarg_r',
 'gnuplot_r',
 'gnuplot2_r',
 'gray_r',
 'hot_r',
 'hsv_r',
 'jet_r',
 'nipy_spectral_r',
 'ocean_r',
 'pink_r',
 'prism_r',
 'rainbow_r',
 'seismic_r',
 'spring_r',
 'summer_r',
 'terrain_r',
 'winter_r',
 'Accent_r',
 'Dark2_r',
 'Paired_r',
 'Pastel1_r',
 'Pastel2_r',
 'Set1_r',
 'Set2_r',
 'Set3_r',
 'tab10_r',
 'tab20_r',
 'tab20b_r',
 'tab20c_r']
```

How to Use the ColorMap : You can specify the colormap with the keyword argument cmap with the value of the colormap, in this case 'viridis' which is one of the built-in colormaps available in Matplotlib. In addition you have to create an array with values (from 0 to 100), one value for each point in the scatter plot.

in details you can read here,

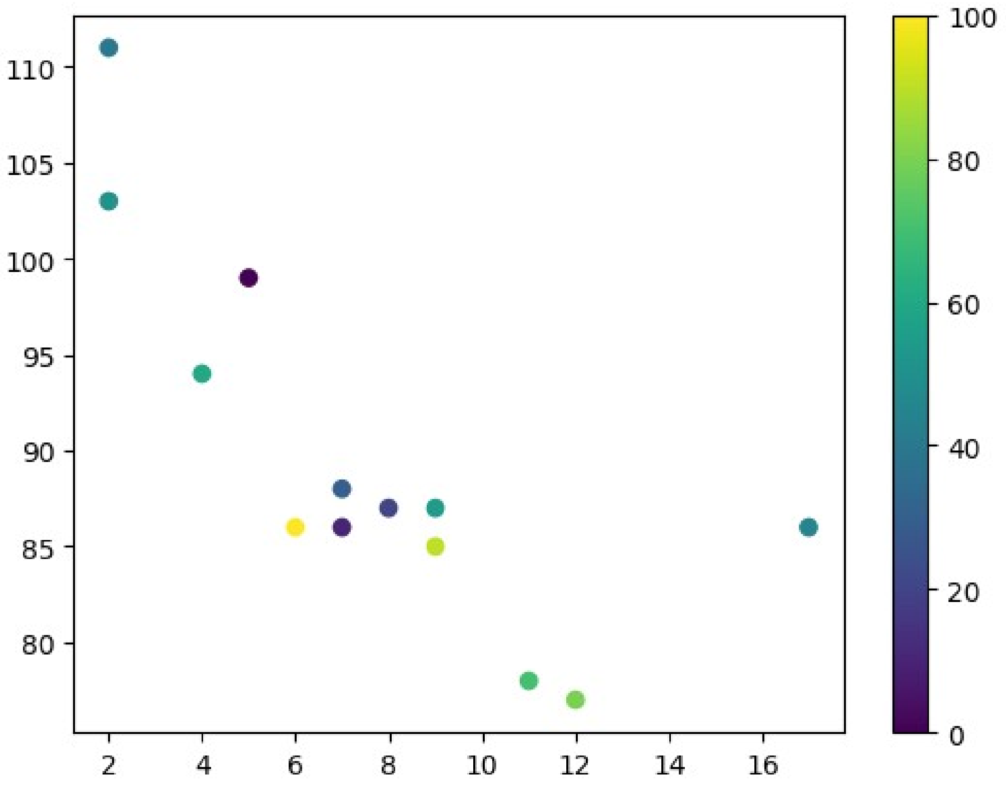
- Create a color array, and specify a colormap in the scatter plot :

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

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90,
100])

plt.scatter(x, y, c=colors, cmap='viridis')
plt.show()
```



NOTE- You can include the colormap in the drawing by including the ***plt.colorbar()*** statement.

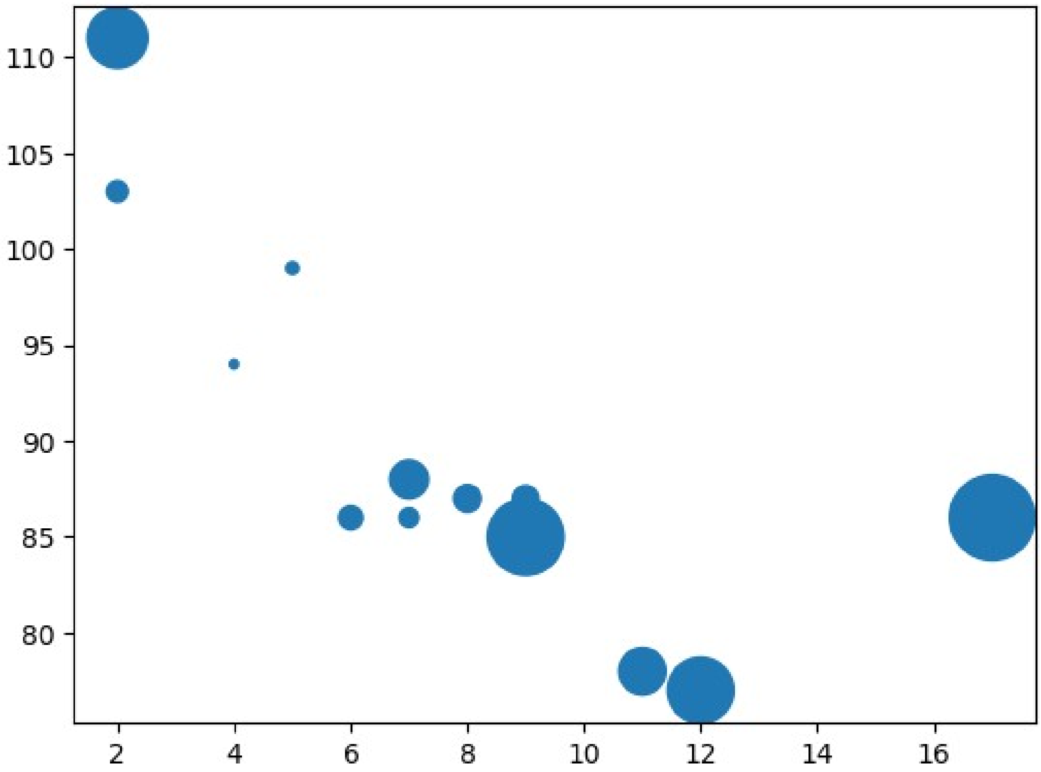
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90,
100])

plt.scatter(x, y, c=colors, cmap='viridis')

plt.colorbar()

plt.show()
```



Size:

You can change the size of the dots with the 's' argument. Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis.

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

x = np.array([5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6])
y = np.array([99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86])
sizes = ([20, 50, 100, 200, 500, 1000, 60, 90, 10, 300, 600, 800, 75])

plt.scatter(x, y, s = sizes)

plt.show()
```

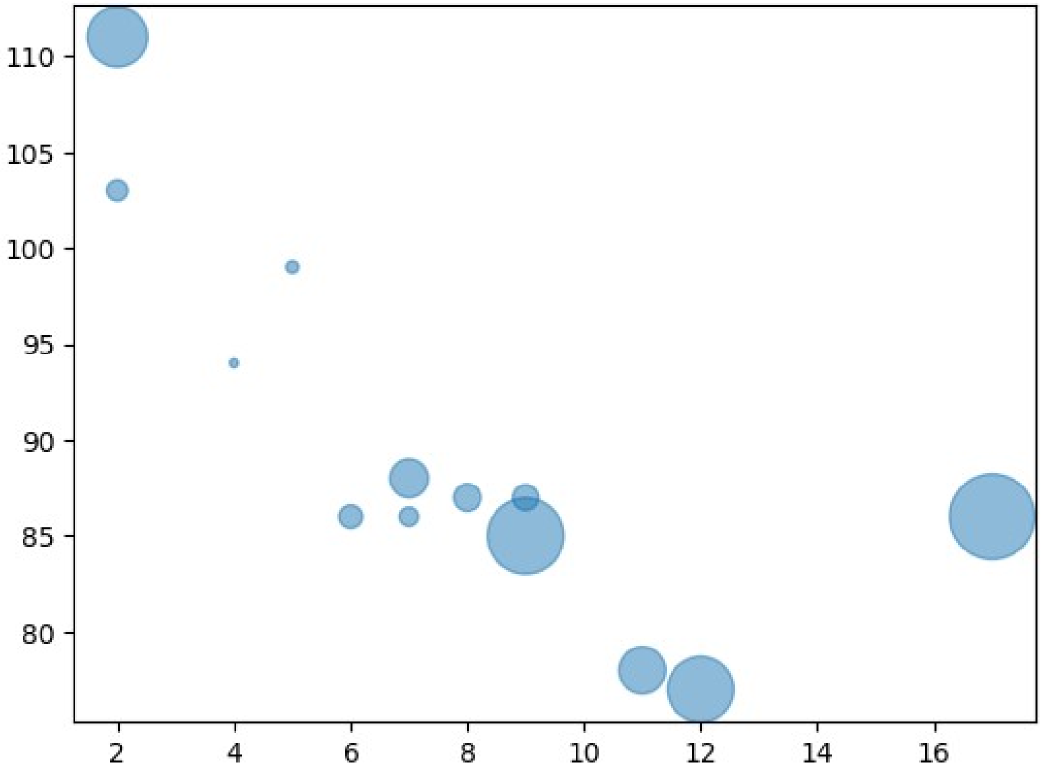


Alpha :

You can adjust the ***"transparency of the dots"*** with the ***alpha*** argument. Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis.

- Set your own size for the markers :

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

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])

plt.scatter(x,y, s = sizes,alpha=0.5)
plt.show()
```

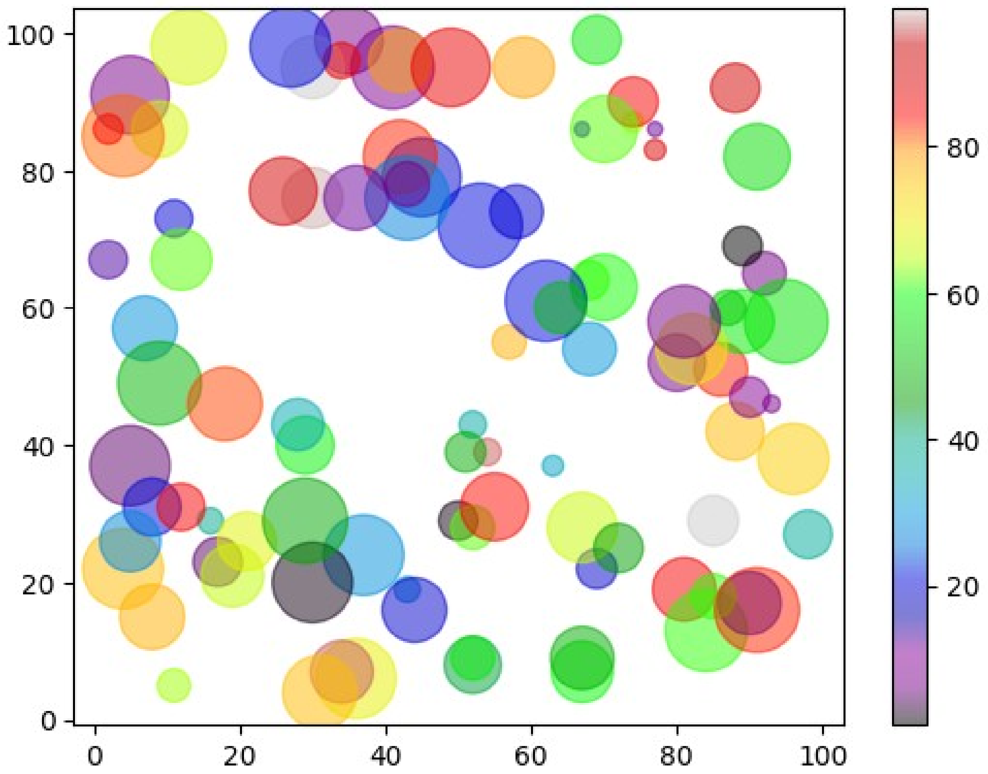


Combine Color Size and Alpha:

You can combine a colormap with different sizes of the dots. This is best visualized if the dots are transparent.

- Create random arrays with 100 values for x-points, y-points, colors and sizes :

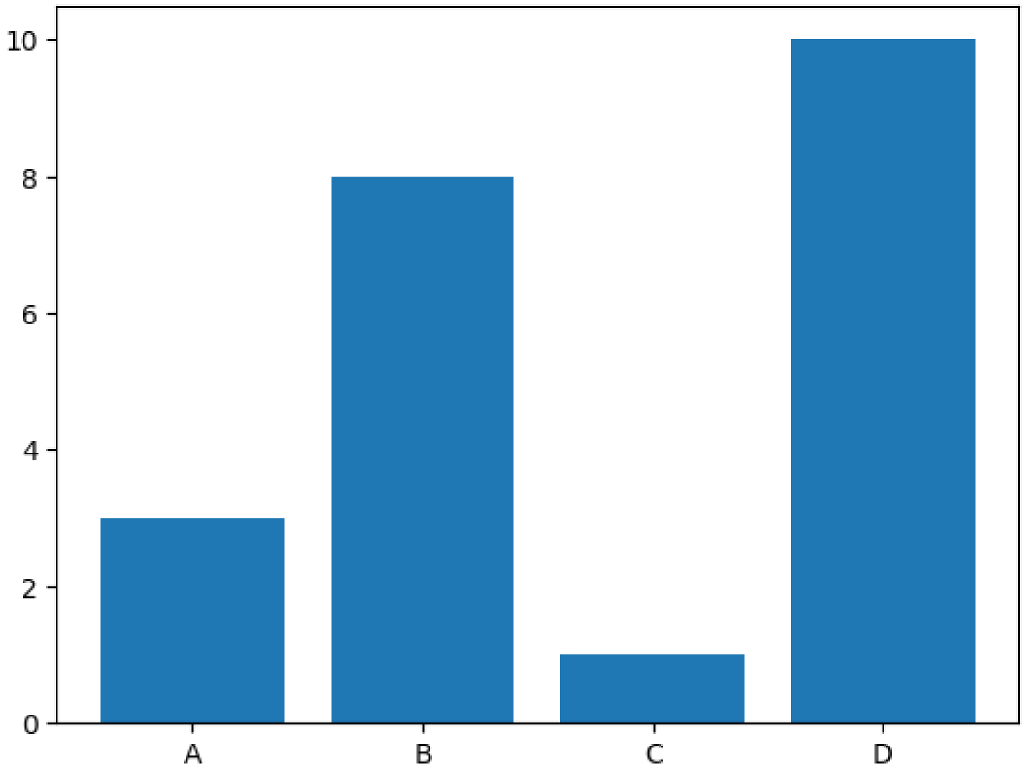
```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.randint(100, size=(100))
y = np.random.randint(100, size=(100))
colors = np.random.randint(100, size=(100))
sizes = 10 * np.random.randint(100, size=(100))

plt.scatter(x, y, c=colors, s=sizes, alpha=0.5, cmap='nipy_spectral')

plt.colorbar()

plt.show()
```



Bar Chart :

Creating Bars With Pyplot, you can use the bar() function to draw bar graphs:

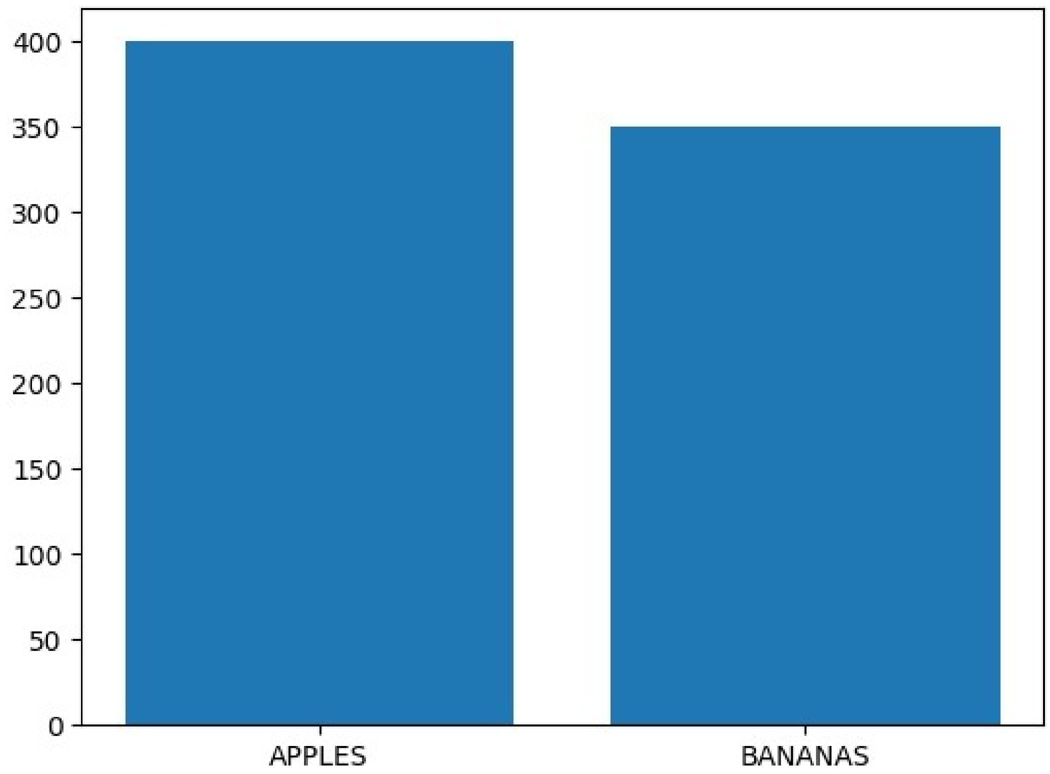
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(['A', 'B', 'C', 'D'])
y = np.array([3, 8, 1, 10])

plt.bar(x, y)
plt.show()
```



- The bar() function takes arguments that describes the layout of the bars.
- The categories and their values represented by the first and second argument as arrays.

Example :

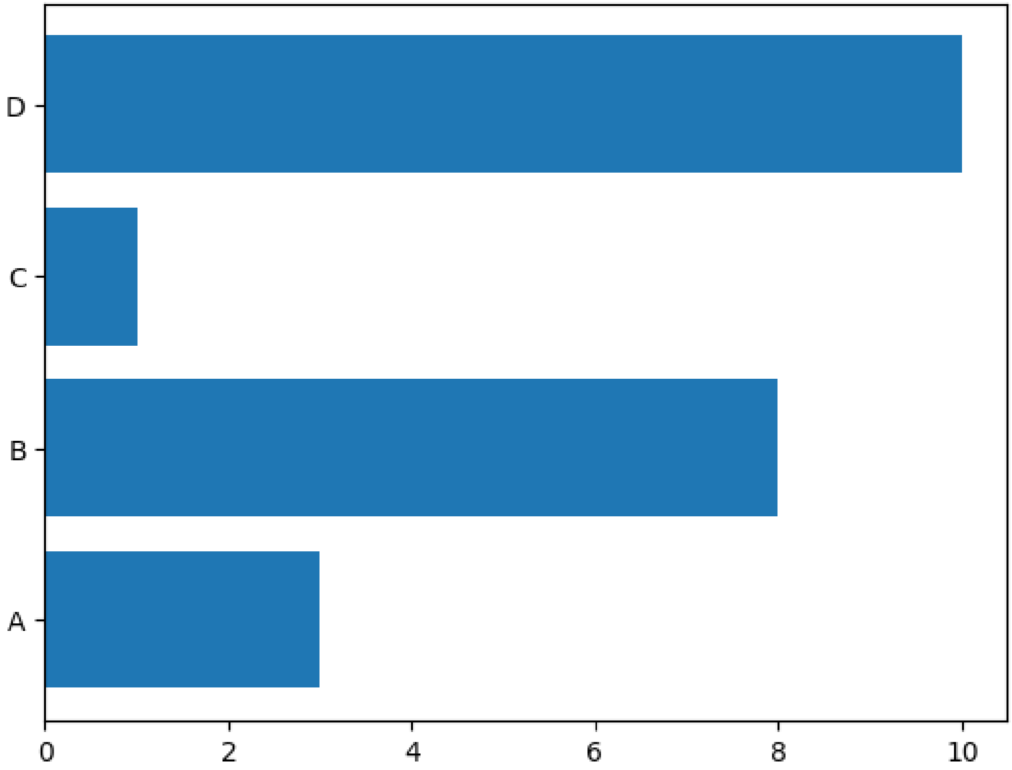
```
import matplotlib.pyplot as plt

x = ["APPLES", "BANANAS"]
y = [400, 350]
plt.bar(x, y)

<BarContainer object of 2 artists>
```



Horizontal Bars :

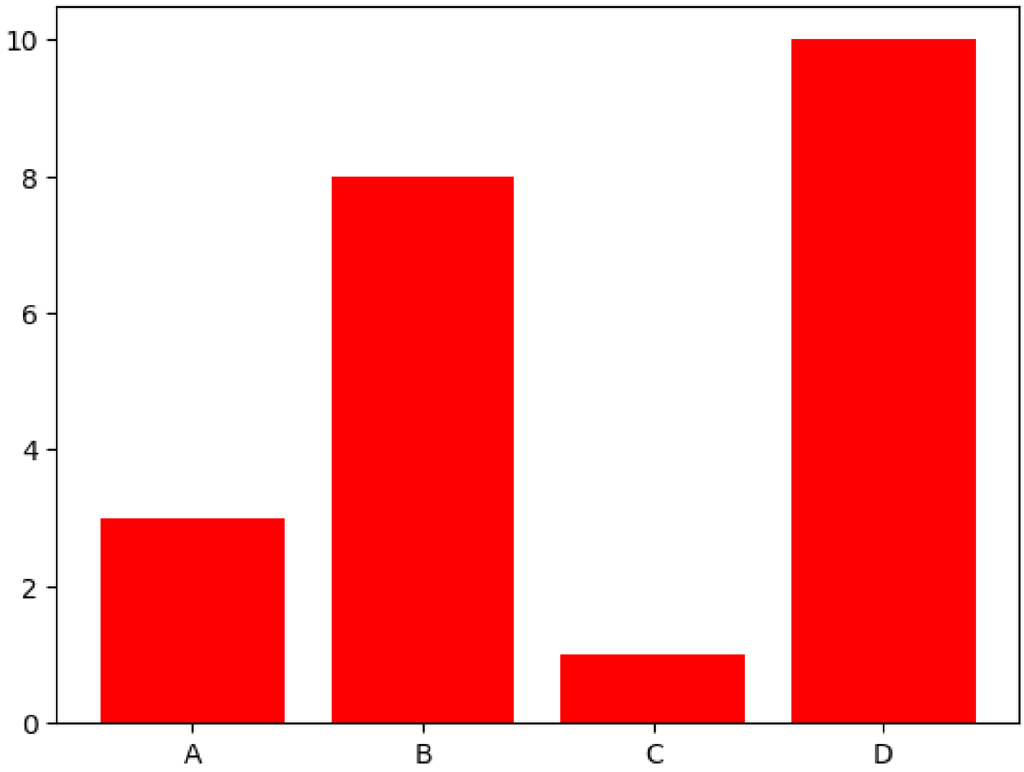
If you want the bars to be displayed horizontally instead of vertically, use the barh() function.

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

x = np.array(['A', 'B', 'C', 'D'])
y = np.array([3, 8, 1, 10])

plt.barh(x, y)
plt.show()
```



Bar Color : The bar() and barh() take the keyword argument color to set the color of the bars.

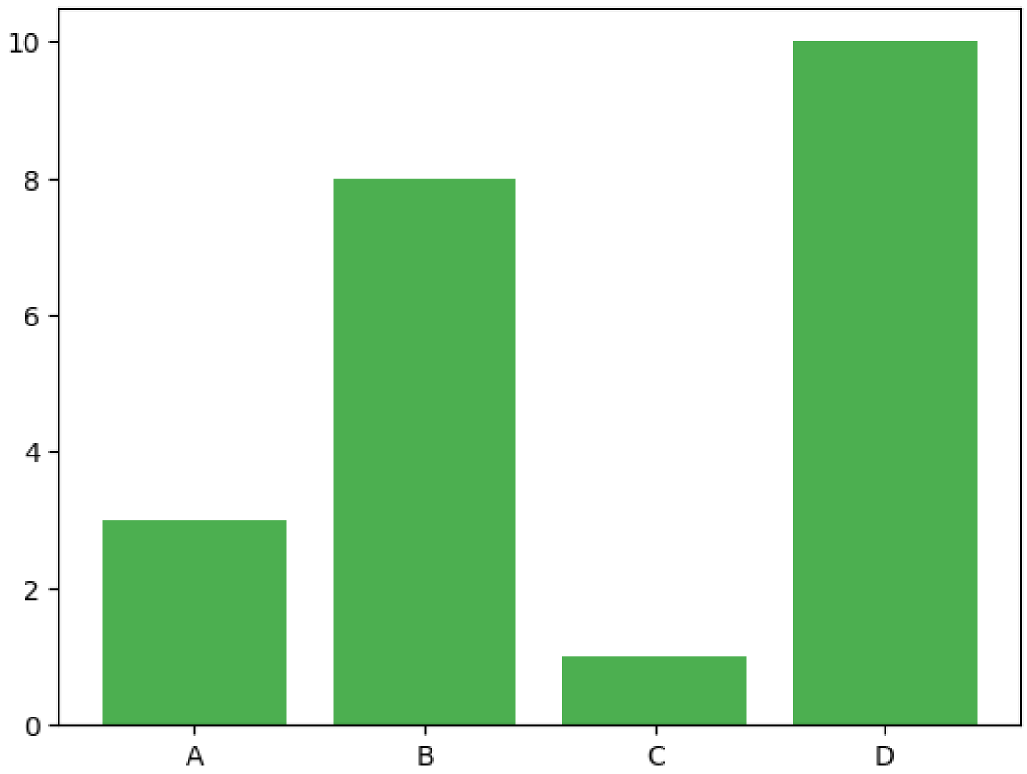
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(['A', 'B', 'C', 'D'])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = 'red')
plt.show()
```



NOTE : don't use 'r' as name or don't use 'c' as args ,it doesn't work.

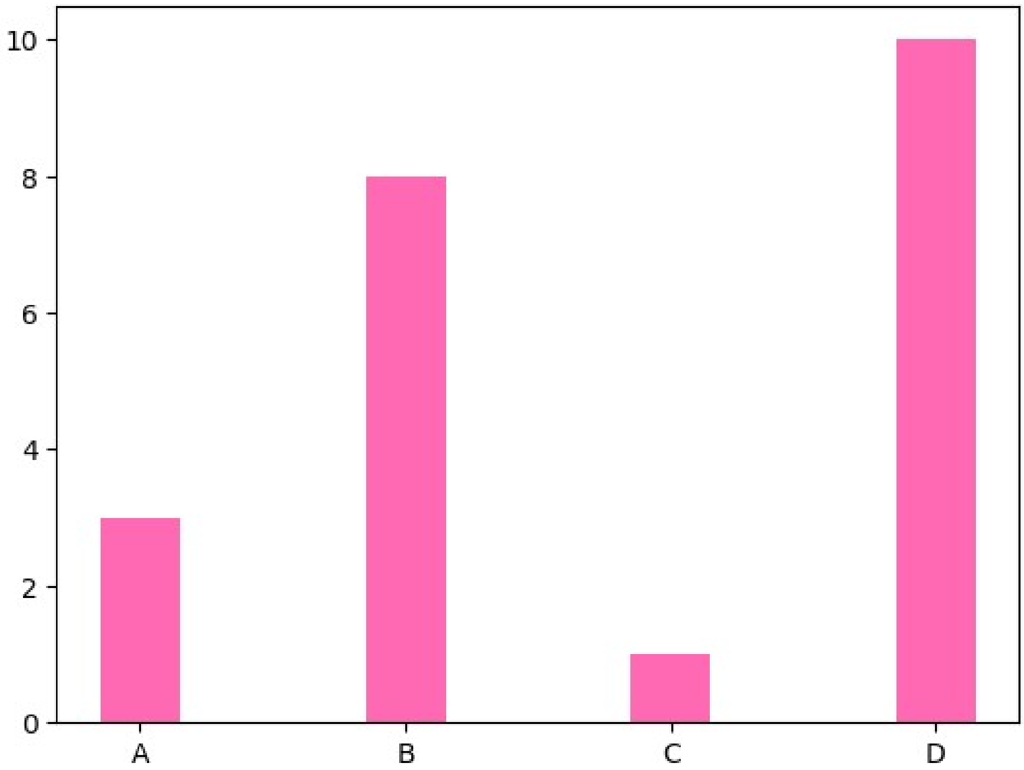
- you can use Hexadecimal color values :
- Draw 4 bars with a beautiful green color :

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = "#4CAF50")
plt.show()
```



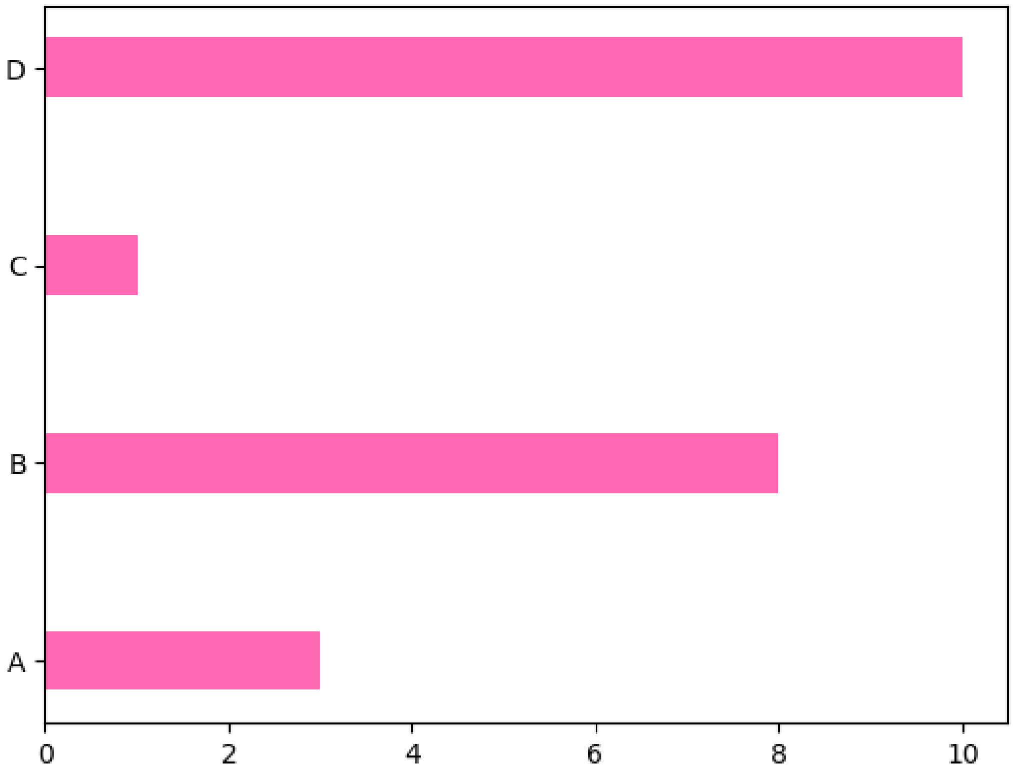
Bar Width : The bar() takes the keyword argument width to set the width of the bars

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

x = np.array(['A', 'B', 'C', 'D'])
y = np.array([3, 8, 1, 10])

plt.bar(x,y, color= 'hotpink', width = 0.3)
plt.show()
```



Bar Height : The barh() takes the keyword argument height to set the height of the bars.

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

x = np.array(['A', 'B', 'C', 'D'])
y = np.array([3, 8, 1, 10])

plt.barh(x,y, color= 'hotpink', height = 0.3)
plt.show()
```



note - The default height/width value is 0.8

Histogram :

A histogram is a graph showing frequency distributions. It is a graph showing the number of observations within each given interval. it is similar to bar chart but there is no space between bars in histogram.

Example : say you ask for the hieght of 250 people, you might end up with a histogram like this: you can read from the histogram that there are approximately :

- 2 people from 140 to 145cm
- 5 people from 145 to 150cm
- 15 people from 151 to 156cm
- 31 people from 157 to 162cm
- 46 people from 163 to 168cm
- 53 people from 168 to 173cm
- 45 people from 173 to 178cm
- 28 people from 179 to 184cm
- 21 people from 185 to 190cm
- 4 people from 190 to 195cm

Create Histogram : In Matplotlib, we use the hist() function to create histograms. The hist() function will use an array of numbers to create a histogram, the array is sent into the function as an argument.

- A Normal Data Distribution by NumPy:

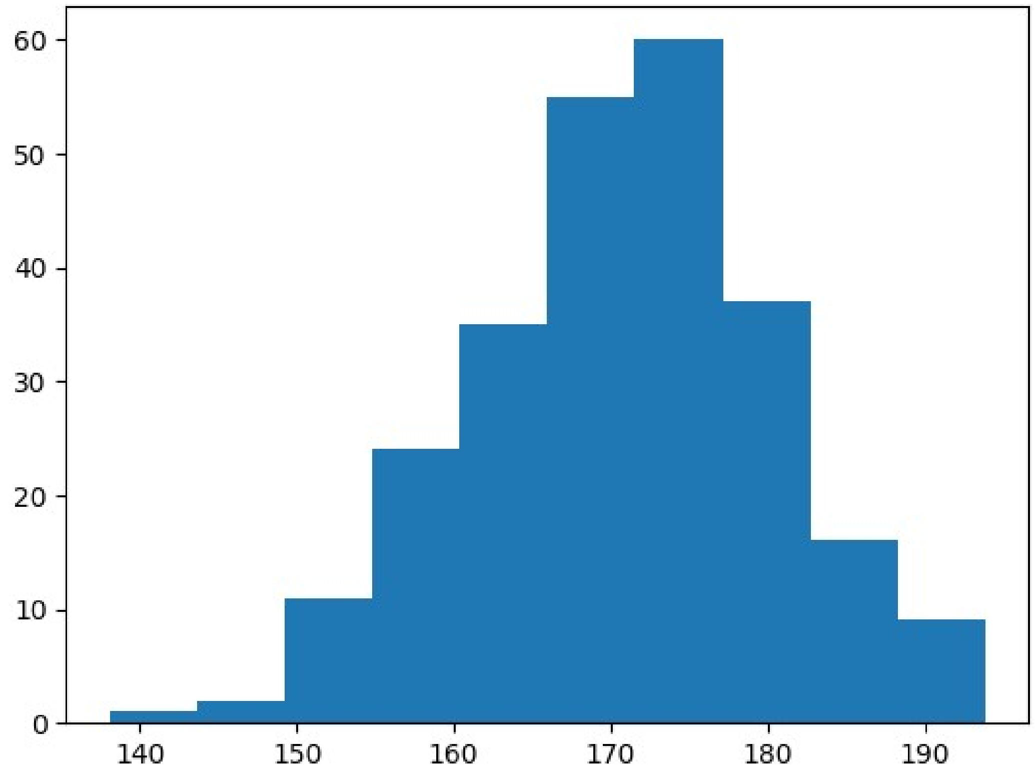
For simplicity we use NumPy to randomly generate an array with 250 values, where the values will concentrate around 170, and the standard deviation is 10. Learn more about Normal Data Distribution in our Machine Learning Tutorial.

```
import numpy as np
x = np.random.normal(170, 10, 250)
print(x)
```

```
[176.82926532 176.58214098 178.29512375 179.26801879 162.05167929
 186.03196429 164.00160168 170.74271765 170.02145339 170.18212699
 182.47526149 172.28743918 162.65039972 164.07824977 165.84227817
 179.93178137 162.09972346 151.77189768 155.5837261  184.58238877
 184.77032333 170.5016743  175.50434797 168.91479635 155.27342862
 146.8926331  194.81755883 186.49715046 176.21819487 177.01468954
 161.70820022 155.0625373  168.45882304 178.0722238  177.18866653
 175.48194287 174.09047752 168.79494565 165.72979789 167.83864432
 166.39097816 162.30734121 180.96654975 143.30630177 169.77180504
 184.02644003 161.55653488 176.23991062 183.83656946 168.04767471
 183.73674067 164.4978119  176.86713937 186.89334456 189.23806723
 172.09304815 182.91071588 157.3908887  168.67351987 170.98932612
 176.23142847 167.02218586 170.69161482 173.77322753 173.49590358
 177.61225352 166.52454628 178.49104171 171.55390301 177.06417553
 182.26339402 187.41444468 154.15185422 184.84473194 167.34948396
 168.18328147 193.71243305 185.53095437 160.64382542 150.88226585
 163.69524905 151.99088982 172.13376974 170.19085235 175.08065087
 168.61077853 172.50324557 187.95558234 157.01375714 180.67450307
 168.93274429 167.87722262 180.16064811 170.75484054 174.19648712
 163.4014213  172.89630167 165.25811198 174.92282502 158.57512108
 176.37636632 175.41164614 171.20705629 173.90762541 179.53550496
 174.21631415 151.92549228 168.48412233 165.62001979 169.73031017
 170.65089828 155.80584415 155.6321723  165.20462749 145.49159239
 177.50630699 182.40999462 159.9730368  173.42166202 170.9266816
 187.93838359 176.31499639 175.16955959 152.77631016 170.5473093
 168.08930118 174.39199145 173.90441818 172.05992056 182.14550948
 171.50359761 189.80462615 165.68394381 178.96188623 180.56869321
 169.13362212 163.52936949 180.06232434 155.87941637 171.49088121
 170.89440051 174.56227081 165.82246391 148.4627608  179.05201673
 178.52234442 193.79751952 189.47349908 174.46303004 164.34760031
 169.60144973 170.01584143 172.50837908 170.82601049 178.50466915
 168.38926219 171.22520684 169.88410498 171.70801062 165.64769394
 185.87465055 171.09663102 176.74356023 174.45448345 172.44267921
 162.68961886 167.11335936 163.5712442  191.14591638 181.3136938
 164.90221452 165.960195   158.66000613 182.00027889 178.16170999
 171.13962892 165.62773627 167.64646125 171.37612137 168.76040179
 171.85230768 162.01583215 176.3858144  169.17171664 158.55986794
 179.11487506 170.85420057 168.70911362 167.84972644 172.20849022
 167.15614461 170.99052457 180.54302781 179.72915294 171.90422759
 174.91323285 163.84079284 171.33616533 174.60139025 171.89602241
 175.71541767 158.19135734 178.23638098 179.53587779 168.47212566
 151.87784807 153.99997392 164.34471286 162.97743416 164.33886517
 152.08091428 170.82953718 167.09823891 167.37822587 178.19905754
 166.91292884 166.53018192 159.43261197 162.65293463 157.08995883
 160.74214934 194.90962256 164.10621534 177.4994362  174.66629916
 155.0038514  165.4239202  163.4869046  171.30089989 172.71799669
 182.10598703 180.00490476 169.50620543 165.67159223 173.96286628
 172.00877968 161.79116875 159.41975154 160.1859403  156.82284307
 174.30648518 189.17115864 178.07123075 167.10400076 162.09212022
 184.43338487 159.71474173 171.64632538 171.63907809 165.28198441]
```

- A simple histogram :

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

x = np.random.normal(170, 10, 250)

plt.hist(x)
plt.show()
```

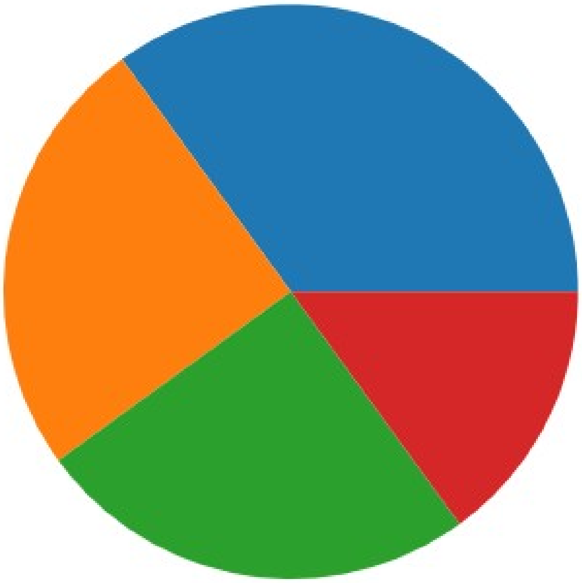


Pie Charts :

Creating Pie Charts : With Pyplot, you can use the pie() function to draw pie charts:

- A simple pie chart:

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

y = np.array([35, 25, 25, 15])

plt.pie(y)
plt.show()
```



NOTE : As you can see the pie chart draws one piece (called a wedge) for each value in the array (in this case [35, 25, 25, 15]).

By default the plotting of the first wedge starts from the x-axis and moves counterclockwise:

NOTE : The size of each wedge is determined by comparing the value with all the other values, by using this formula:

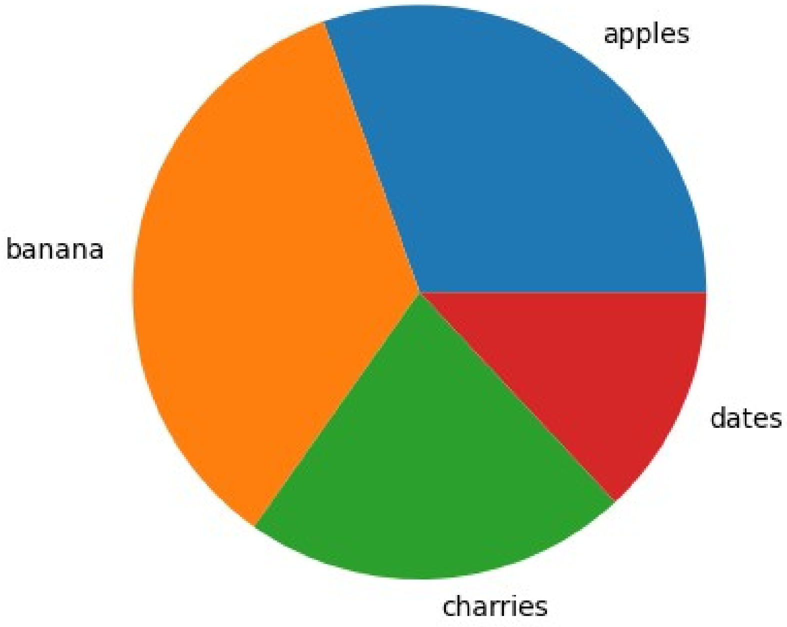
The value divided by the sum of all values: x/sum(x)

Labels :

- Add labels to the pie chart with the label parameter.
- The label parameter must be an array with one label for each wedge.
- a simple pie chart :

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

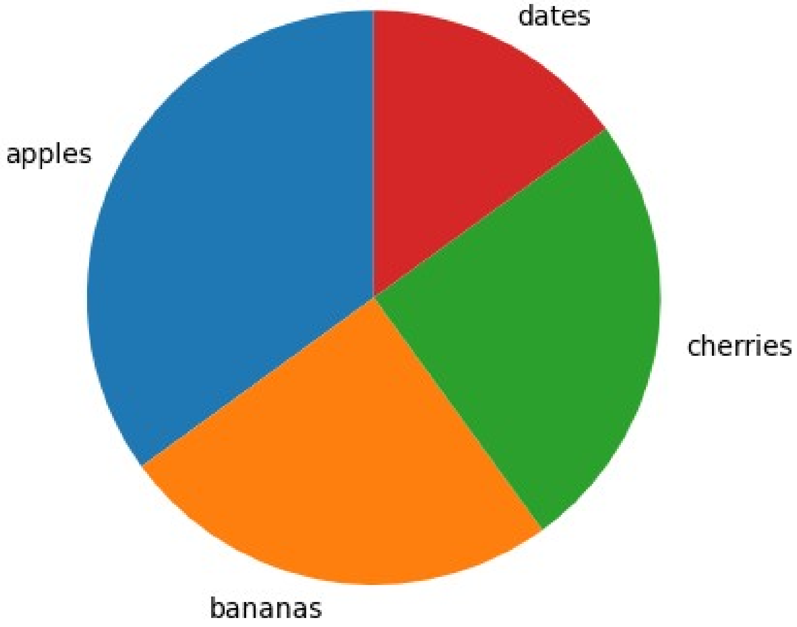
```
y =np.array([35, 40, 25, 15])
mylabels = np.array(['apples', 'banana', 'charries', 'dates'])

plt.pie(y, labels = mylabels)
plt.show()
```



Start Angle : As mentioned the default start angle is at the x-axis, but you can change the start angle by specifying a startangle parameter. The startangle parameter is defined with an angle in degrees, default angle is 0 :

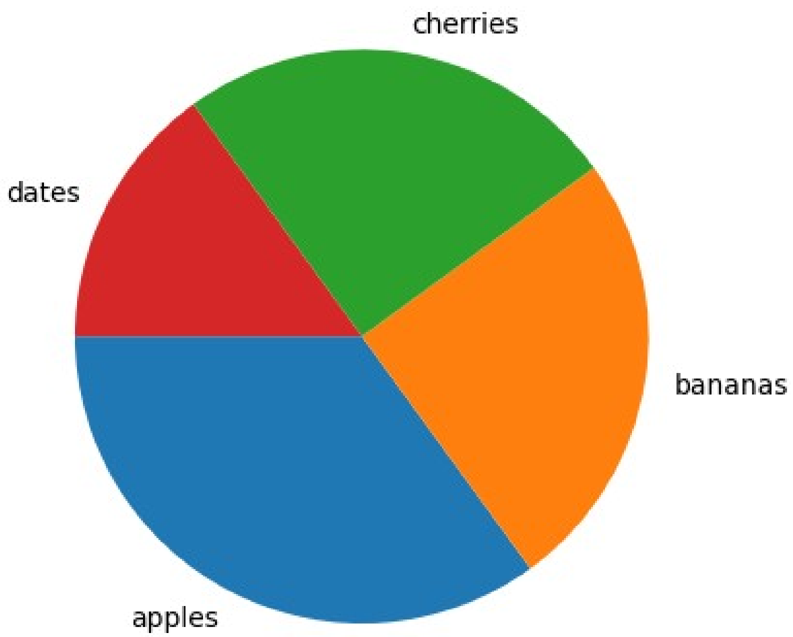
- Start the first wedge at 90 degrees :

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

y = np.array([35, 25, 25, 15])
label = np.array(['apples', 'bananas', 'cherries', 'dates'])

plt.pie(y, labels = label, startangle = 90)
plt.show()
```



- one more exaple for Start the first wedge at 180 degrees :

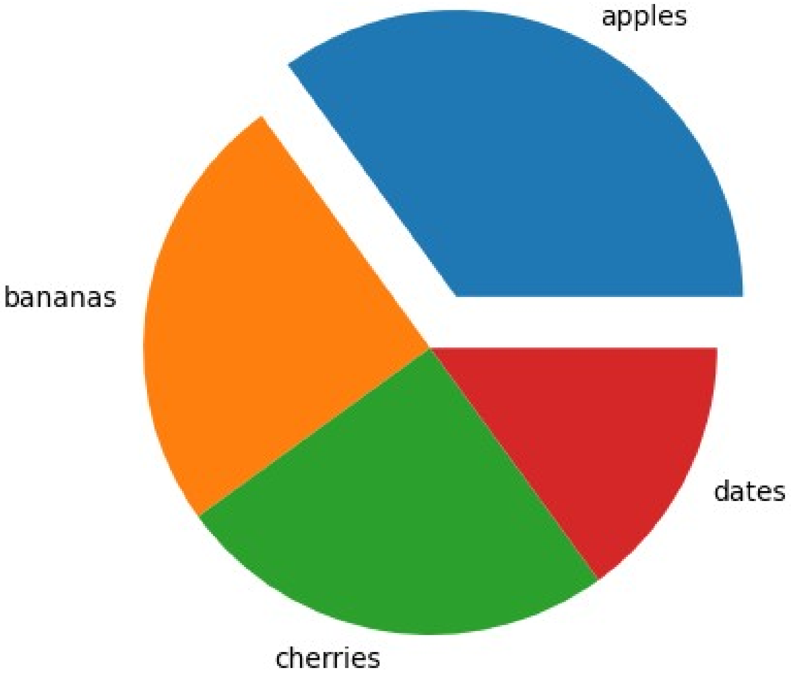
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
label = np.array(['apples', 'bananas', 'cherries', 'dates'])

plt.pie(y, labels = label, startangle = 180)
plt.show()
```



Explode:

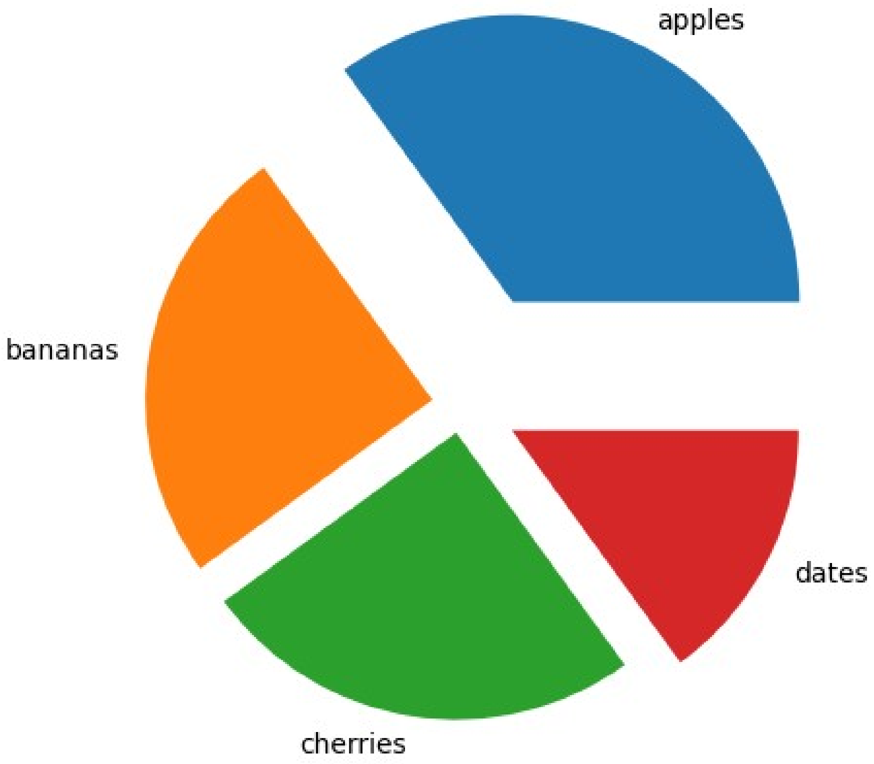
- Maybe you want one of the wedges to stand out? The explode parameter allows you to do that.
- The explode parameter, if specified, and not None, must be an array with one value for each wedge.
- Each value represents how far from the center each wedge is displayed.
- Pull the "Apples" wedge 0.2 from the center of the pie
- one more exaple for Start the first wedge at 90 degrees:

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

y = np.array([35, 25, 25, 15])
label = np.array(['apples', 'bananas', 'cherries', 'dates'])
myexplodes = np.array([0.2, 0, 0, 0])

plt.pie(y, labels = label, explode = myexplodes)
plt.show()
```



- let's try to explode all :

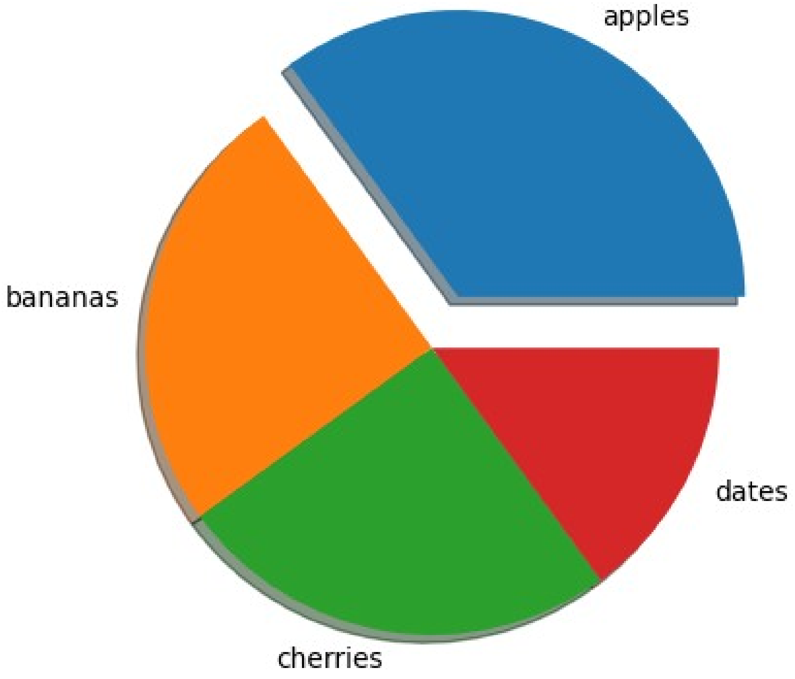
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
label = np.array(['apples', 'bananas', 'cherries', 'dates'])
myexplodes = np.array([0.4, 0.1, 0.1, 0.2])

plt.pie(y, labels = label, explode = myexplodes)
plt.show()
```



Shadow : Add a shadow to the pie chart by setting the shadows parameter to True.

- Pull the "Apples" wedge 0.2 from the center of the pie :
- one more exaple for Start the first wedge at 90 degrees :

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

y = np.array([35, 25, 25, 15])
label = np.array(['apples', 'bananas', 'cherries', 'dates'])
myexplodes = np.array([0.2, 0, 0, 0])

plt.pie(y, labels = label, explode = myexplodes, shadow =True)
plt.show()
```

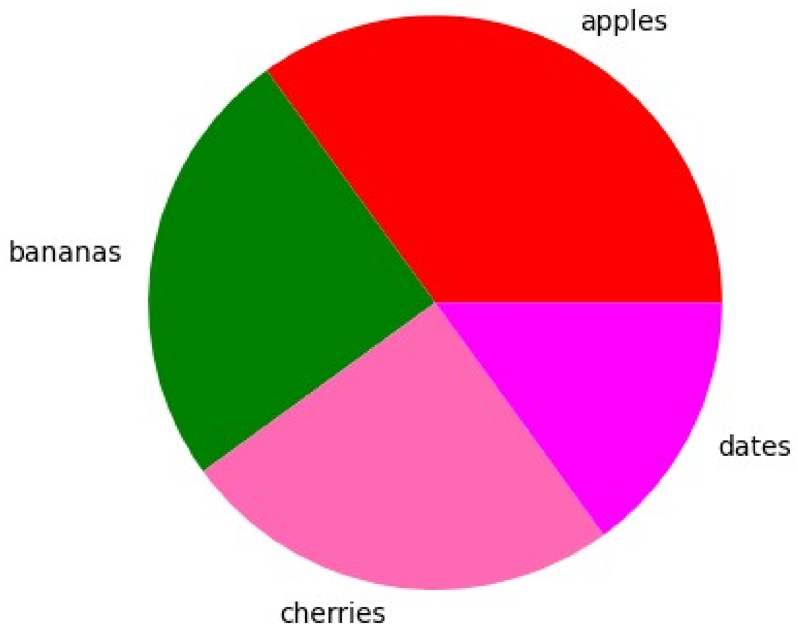


Colors : You can set the color of each wedge with the colors parameter. The colors parameter, if specified, must be an array with one value for each wedge.

****here almost parameters are in plurals so keep mind parameter's name.****

- Pull the "Apples" wedge 0.2 from the center of the pie :
- one more exaple for Start the first wedge at 90 degrees :

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

y = np.array([35, 25, 25, 15])
label = np.array(['apples', 'bananas', 'cherries', 'dates'])
mycolors = np.array(['red', 'green', 'hotpink', 'magenta'])

plt.pie(y, labels = label, colors = mycolors)
plt.show()
```



You can use Hexadecimal color values, any of the 140 supported color names, or one of these shortcuts:

- 'r' - Red
- 'g' - Green
- 'b' - Blue
- 'c' - Cyan
- 'm' - Magenta
- 'y' - Yellow
- 'k' - Black
- 'w' - White

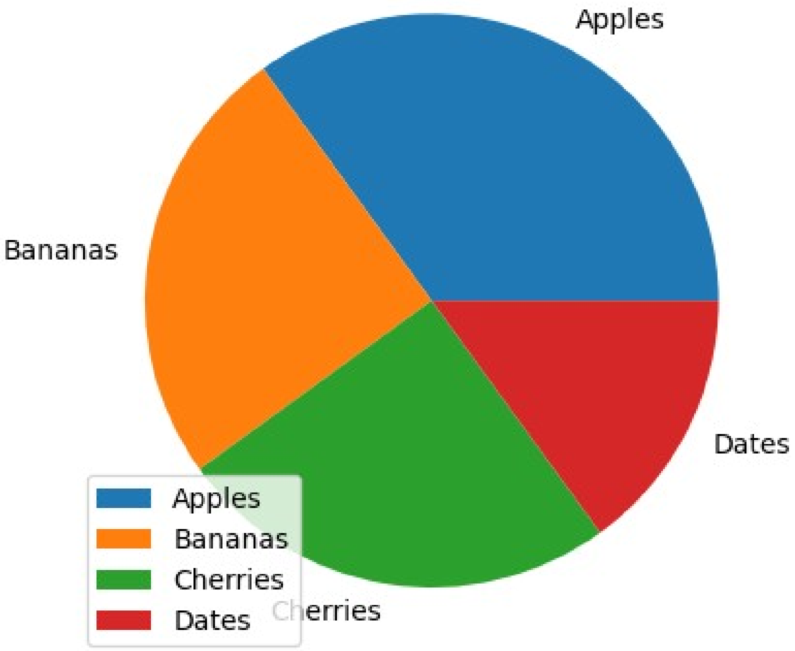
Matplotlib.pyplot.legend() :

A legend is an area describing the elements of the graph. In the matplotlib library, there’s a function called legend() which is used to Place a legend on the axes. The attribute Loc in legend() is used to specify the location of the legend.Default value of loc is loc=”best” (upper left). The strings ‘upper left’, ‘upper right’, ‘lower left’, ‘lower right’ place the legend at the corresponding corner of the axes/figure. The attribute bbox_to_anchor=(x, y) of legend() function is used to specify the coordinates of the legend, and the attribute ncol represents the number of columns that the legend has.It’s default value is 1.

To add a list of explanation for each wedge, use the legend() function

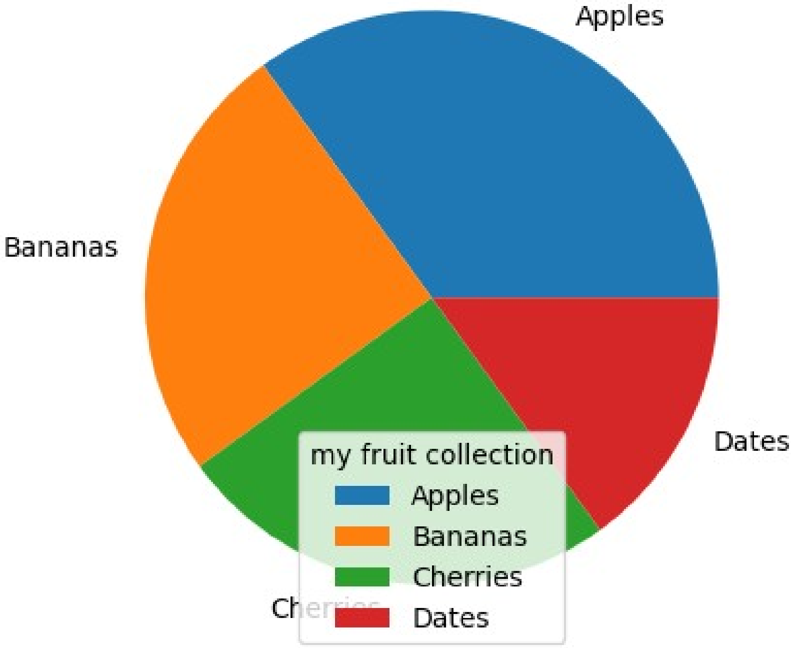
- piechart with a legend panel :

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

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.legend()
plt.show()
```



title of Legend : To add title on the legend, add the title parameter to the legend function.

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

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.legend(title = 'my fruit collection')
plt.show()
```

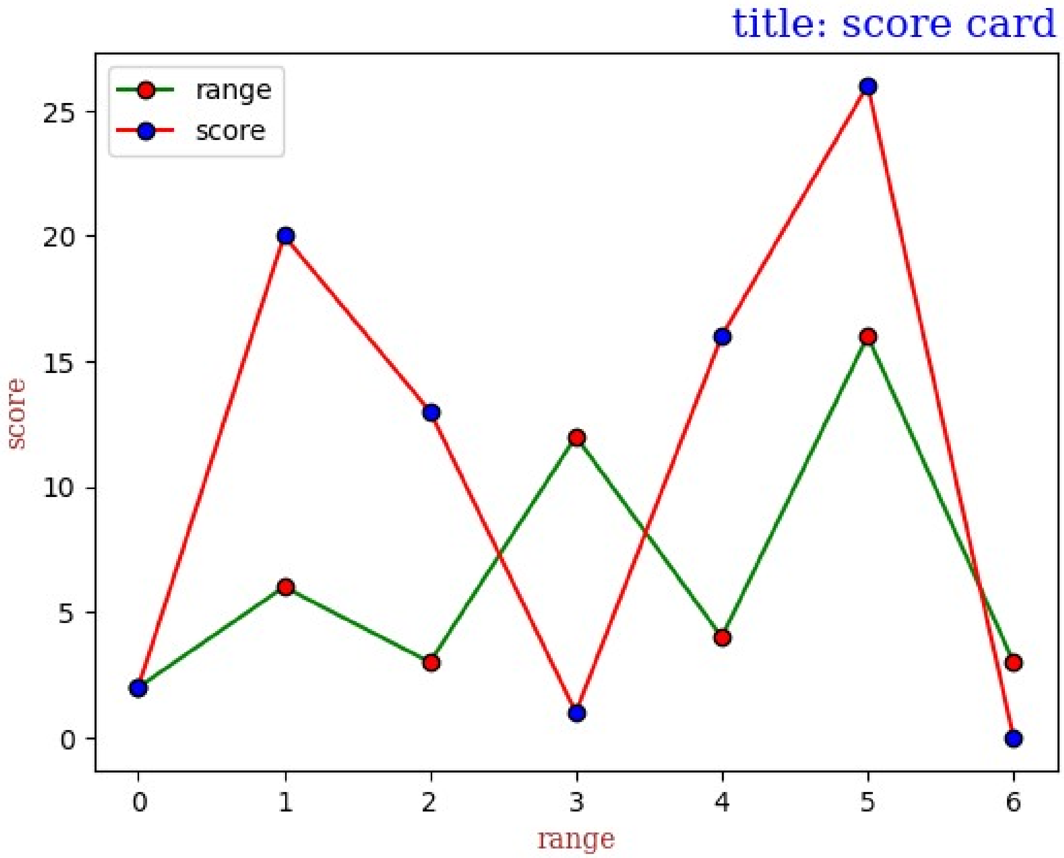


change the location of legend : you can use 'loc' parameter to change the position of legend.

there are position values used loc parameter in matplotlib.pyplot legend in piechart.

for fun:

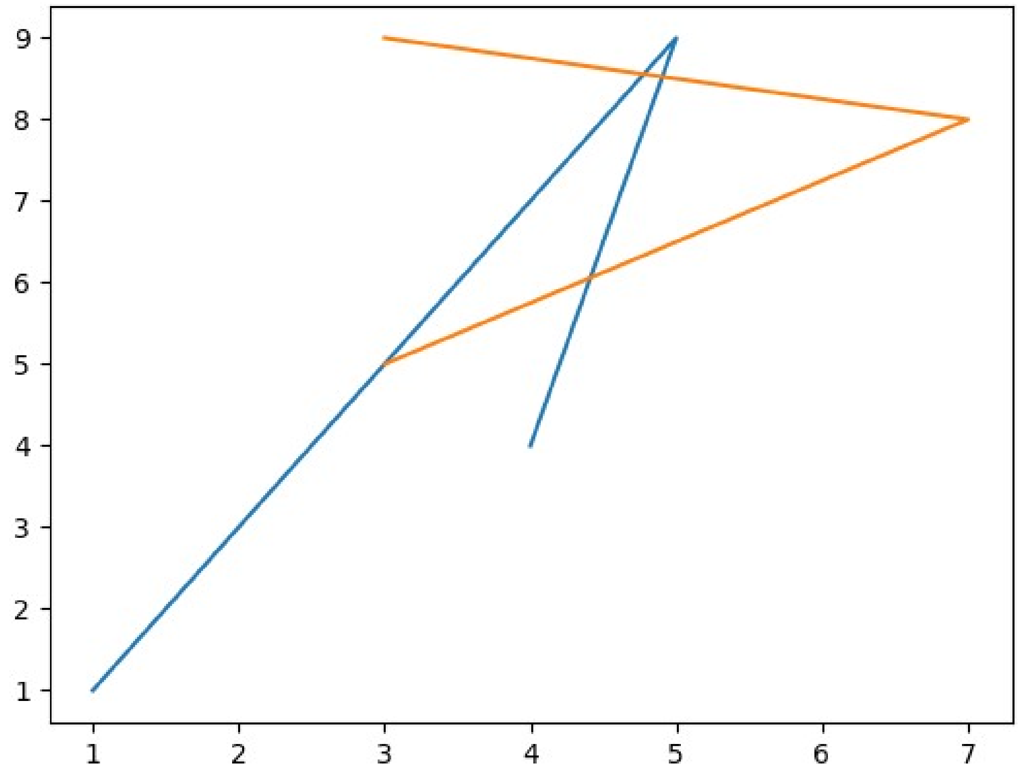
```
import matplotlib.pyplot as plt
import numpy as np

y1 = np.array([2, 6, 3, 12, 4, 16, 3])
y2 = np.array([2, 20, 13, 1, 16, 26, 0])

font1 = {'family':'serif','color':'blue','size':15}
font2 = {'family':'serif','color':'brown','size':10}

plt.plot(y1, marker='o', mec='#000000',mfc='r', c= 'g')
plt.plot(y2, marker='o', mec='#000000',mfc='b',c ='r')

plt.title('title: score card', fontdict = font1, loc = 'right')
plt.xlabel("range", fontdict = font2)
plt.ylabel("score", fontdict = font2)

plt.legend(['range','score'],loc='best')
plt.show()
```



matplotlib style:

you can import from matplotlib to give style to your graphs.

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

x = np.array([1, 5, 4])
y = np.array([1, 9, 4])

x1 = np.array([3, 7, 3])
y1 = np.array([5, 8, 9])

plt.plot(x, y)
plt.plot(x1, y1)

style.use('ggplot')

plt.show()
```



Reference

- documantation
- w3school
- Github
- colorcode hexadesimal values
- color name values
- Style sheets reference