Chapter3: Matplotlib

Matplotlib is a free graph plotting library in python that serves as a visualization utility.

1. Installation of Matplotlib

```
pip install matplotlib
```

2. Import Matplotlib

```
import matplotlib
```

3. Pyplot

Most of the Matplotlib utilities lies under the `pyplot` submodule

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

4. Plotting

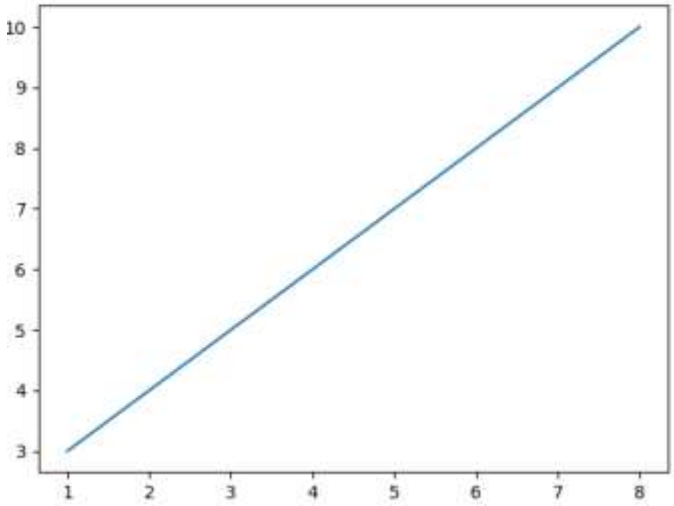
a) Plotting x and y points

The `plot()` function is used to draw points (markers) in a diagram or a line from point to point.

If we need to plot a line, we need two arrays as parameter to the plot function.

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

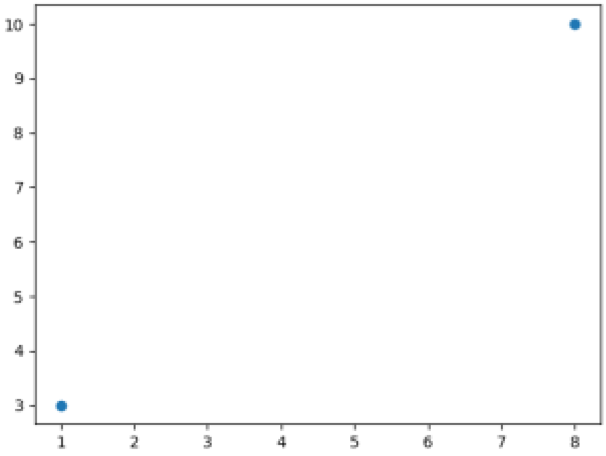
```
import matplotlib.pyplot as plt
import numpy as np

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

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



b) Plotting without line

To plot only the markers, you can use *shortcut string notation* parameter 'o', which means 'rings'.

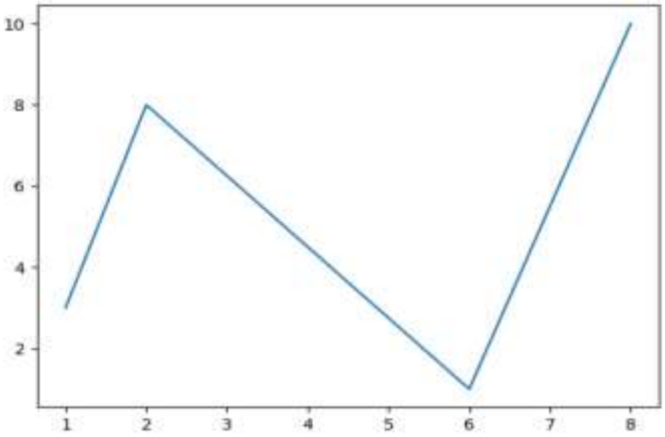
Draw two points in the diagram, one at position (1, 3) and one in position (8, 10)



```
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()
```

c) Multiple Points

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



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

x = np.array([1, 2, 6, 8])
y = np.array([3, 8, 1, 10])

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

d) Default X-Points

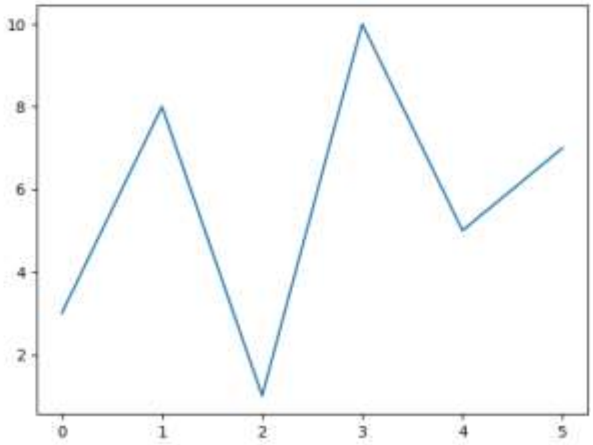
If we do not specify the points on the x-axis, they will get the default values 0, 1, 2, 3 etc., depending on the length of the y-points.

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

```
y = np.array([3, 8, 1, 10, 5, 7])

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

The **x-points** in the example above are

[0, 1, 2, 3, 4, 5].

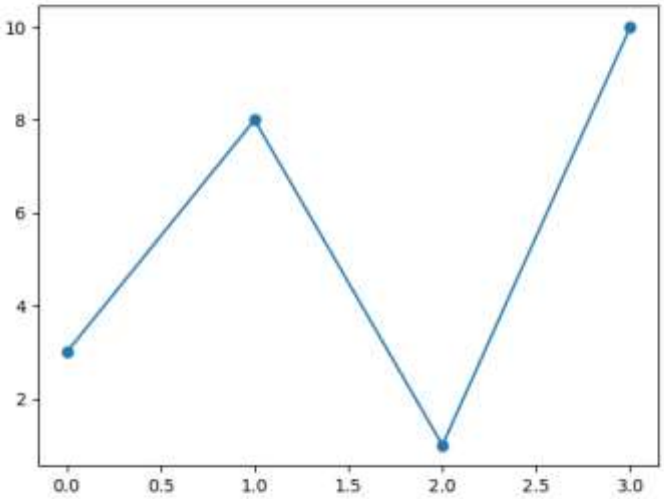


5. Markers

You can use the keyword argument `marker` to emphasize each point with a specified marker:

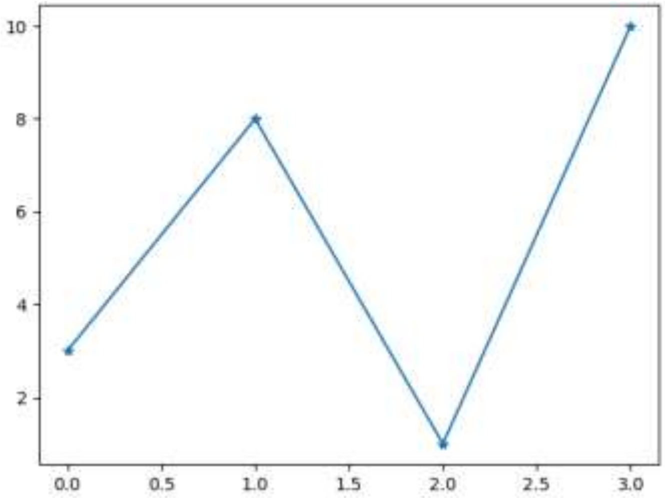
Mark each point with a circle:

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

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

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



Mark each point with a star:

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

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

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



a) Marker Reference

You can choose any of these markers:

Marker	Description
'o'	Circle
'*'	Star
'.'	Point
','	Pixel
'x'	X
'X'	X (filled)
'+'	Plus
'P'	Plus (filled)
's'	Square
'D'	Diamond
'd'	Diamond (thin)
'p'	Pentagon
'H'	Hexagon
'h'	Hexagon
'v'	Triangle Down
'^'	Triangle Up
'<'	Triangle Left
'>'	Triangle Right

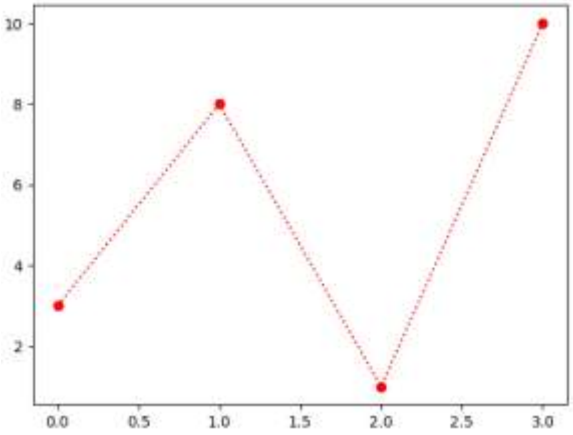
b) Format Strings

You can also use the *shortcut string notation* parameter to specify the marker.

This parameter is also called `fmt`, and is written with this syntax:
marker|line|color

Mark each point with a circle:

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

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

plt.plot(y, 'o:r')
plt.show()
```



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

The line value can be one of the following:

c) Line Reference

Line Syntax	Description
'-'	Solid line
':'	Dotted line
'--'	Dashed line
'-.'	Dashed/dotted line

Note: If you leave out the *line* value in the fmt parameter, no line will be plotted.

The short color value can be one of the following:

d) Color Reference

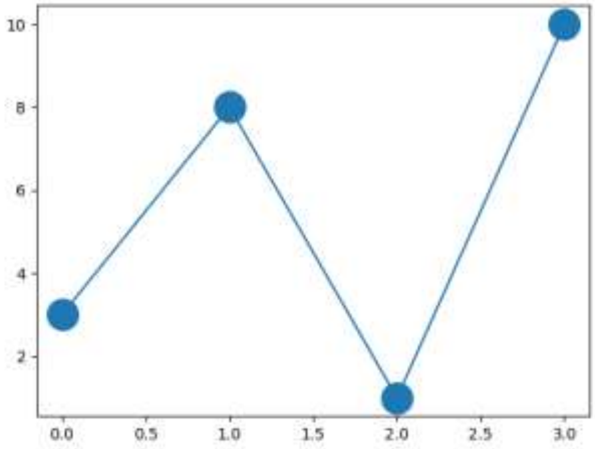
Color Syntax	Description
'r'	Red
'g'	Green
'b'	Blue
'c'	Cyan
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

e) Marker Size

You can use the keyword argument `markersize` or the shorter version, `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

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

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

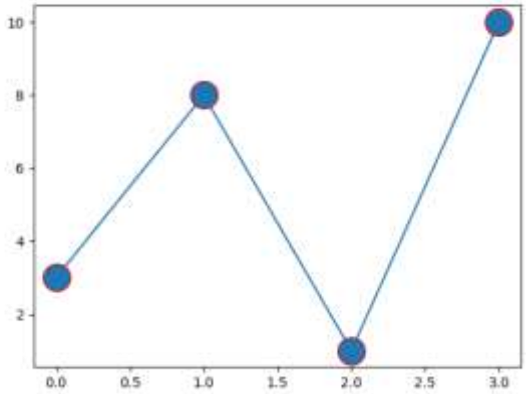


f) Marker edge Color

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:

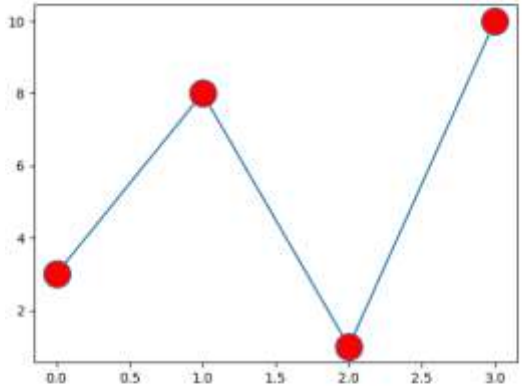
```
import matplotlib.pyplot as plt
import numpy as np

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

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



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:

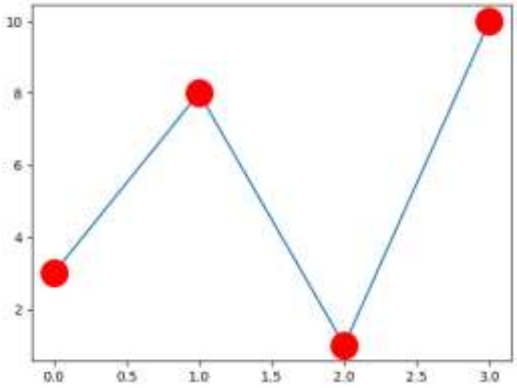
```
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(y, 'o-b', ms = 20, mfc = 'r')
plt.show()
```



Use *both* 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

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

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

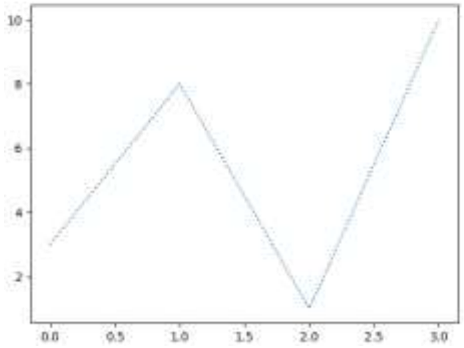


6. Matplotlib Line

LineStyle: You can use the keyword argument linestyle, or shorter ls, to change the style of the plotted line:

Use a dotted line:

```
import matplotlib.pyplot as plt

import numpy as np

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

plt.plot(y, linestyle = 'dotted')

plt.show()
```



The line style can be written in a shorter syntax: linestyle can be written as ls

```
plt.plot(y, ls = ':')
```

a. Styles: You can choose any of these styles:

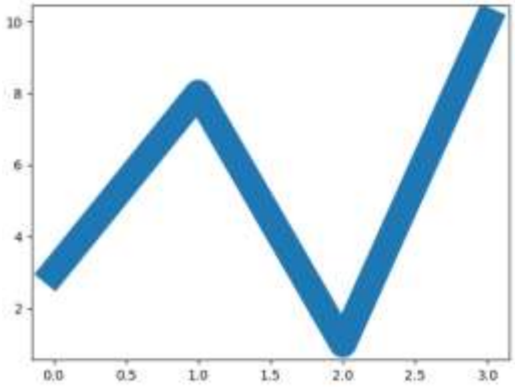
Style	Or
'solid' (default)	'-'
'dotted'	':'
'dashed'	'--'
'dashdot'	'-.'
'None'	'' or ' '

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

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

c. Line Width: You can use the keyword argument linewidth or the shorter lw to change the width of the line.

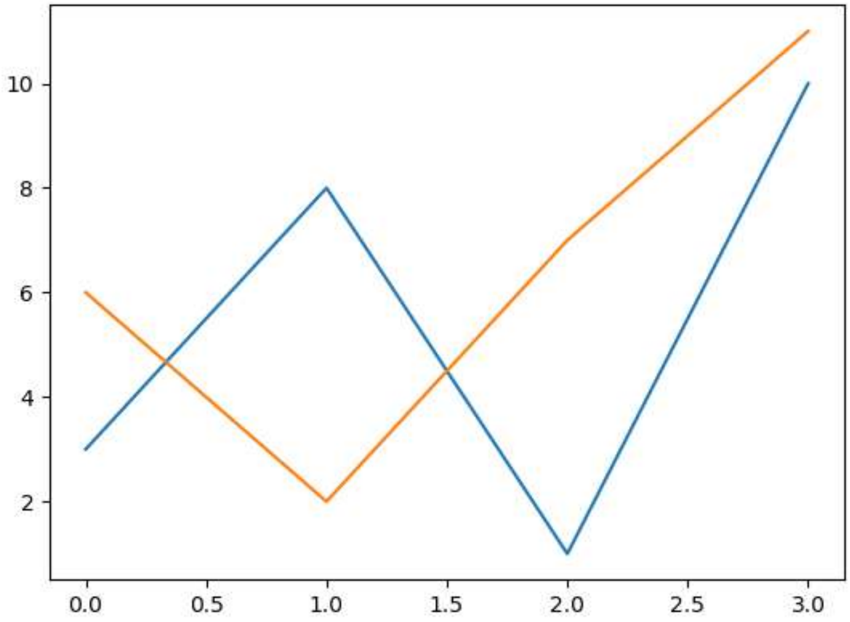
```
import matplotlib.pyplot as plt
import numpy as np

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

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



d. Multiple Lines: You can plot as many lines as you like by simply adding more `plt.plot()` functions:

Draw two lines by specifying 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(x1, y1)

plt.plot(x2, y2)

#or

plt.plot(x1, y1, x2, y2)


plt.show()
```



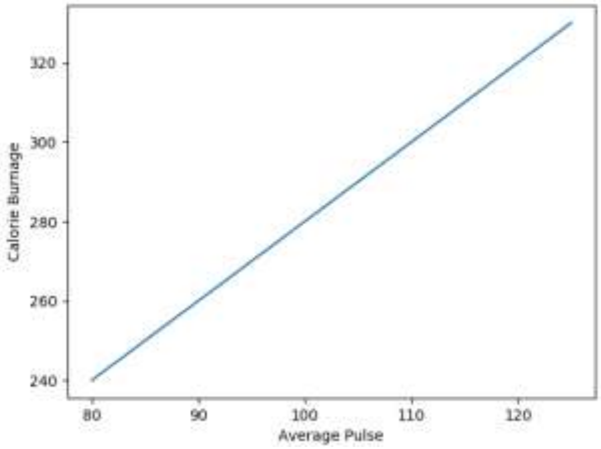
7. Matplotlib Labels and Title

a. Create Labels for a Plot:

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 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.plot(x, y)

plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```

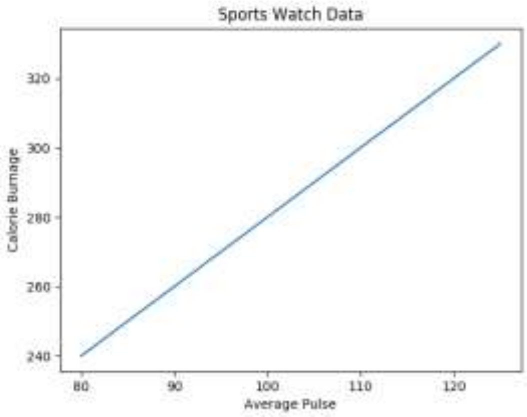


b. Create a Title for a Plot

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 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.plot(x, y)

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```



We can use the `grid()` function to add grid lines to the plot: `plt.grid()`

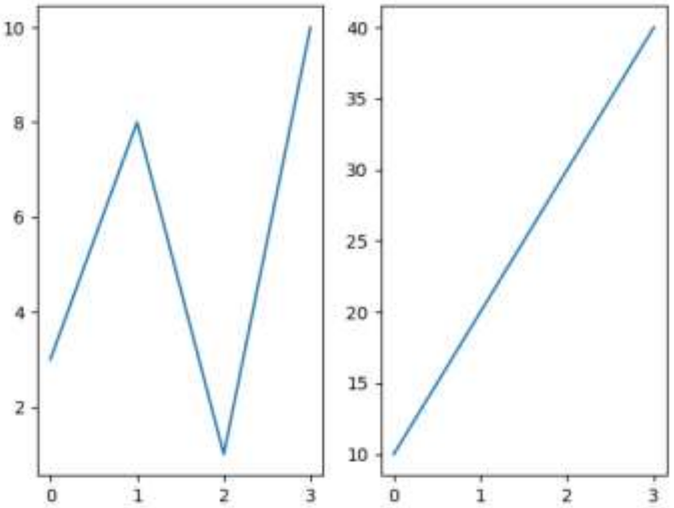
8. Matplotlib Subplot

a. Display Multiple Plots:

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

Draw 2 plots:

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

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

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

plt.show()
```



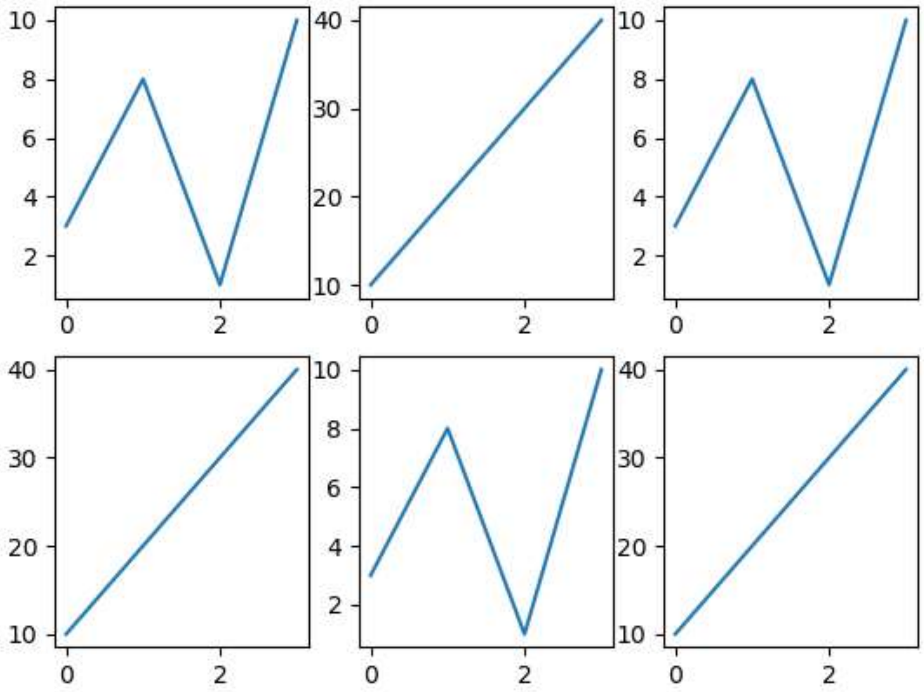
The subplot() is organized in rows and columns, which are represented by the *first* and *second* argument. The third argument represents the index of the current plot.

```
plt.subplot(1, 2, 1)
#the figure has 1 row, 2 columns, and this plot is the first plot.
```

Draw 6 plots:

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

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 1)
plt.plot(x,y)

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

plt.subplot(2, 3, 2)
plt.plot(x,y)
```

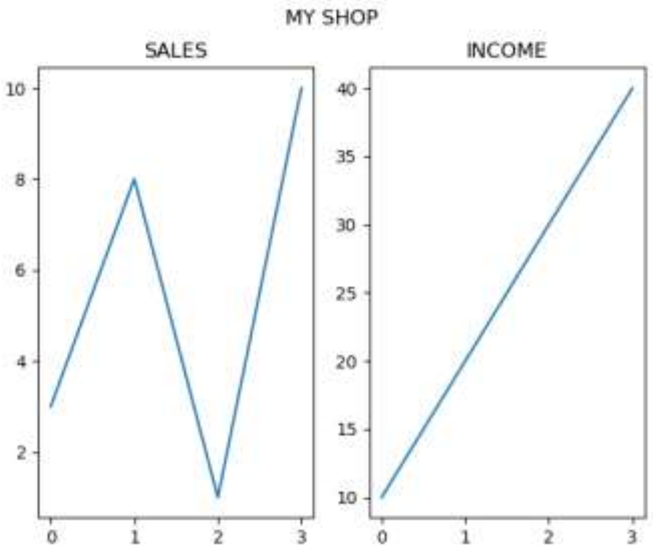
```
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 3)
plt.plot(x,y)

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

plt.subplot(2, 3, 4)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 5)
plt.plot(x,y)

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

plt.subplot(2, 3, 6)
plt.plot(x,y)

plt.show()
```



b. Title and super title

We can add a title to each plot with the title() function and we can add a title to the entire figure with the suptitle() function

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

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

plt.subplot(1, 2, 1)
plt.plot(x1, y1)
plt.title("SALES")

#plot 2:
x2 = np.array([0, 1, 2, 3])
y2 = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x2, y2)
plt.title("INCOME")

plt.suptitle("MY SHOP")
plt.show()
```



9. Matplotlib Scatter

a. Creating Scatter Plots

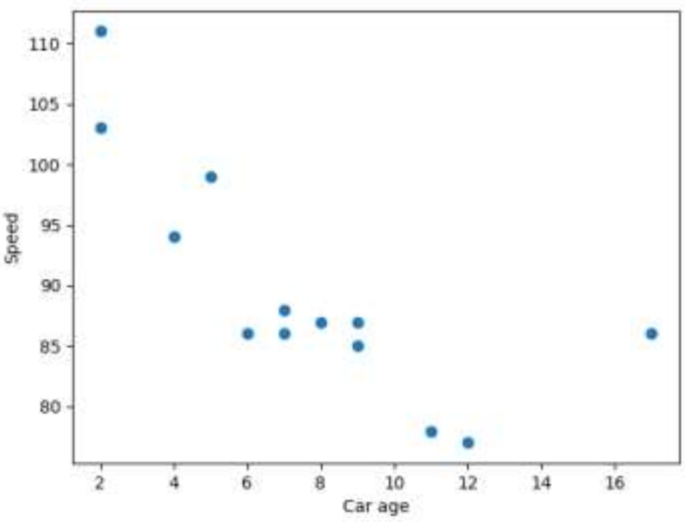
The scatter() function plots one dot for each observation. It needs two arrays of the same length.

```
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.xlabel("Car age")
plt.ylabel("Speed")

plt.show()
```

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.

It seems that the **newer the car, the faster it drives, but that could be a coincidence, after all we only registered 13 cars**.



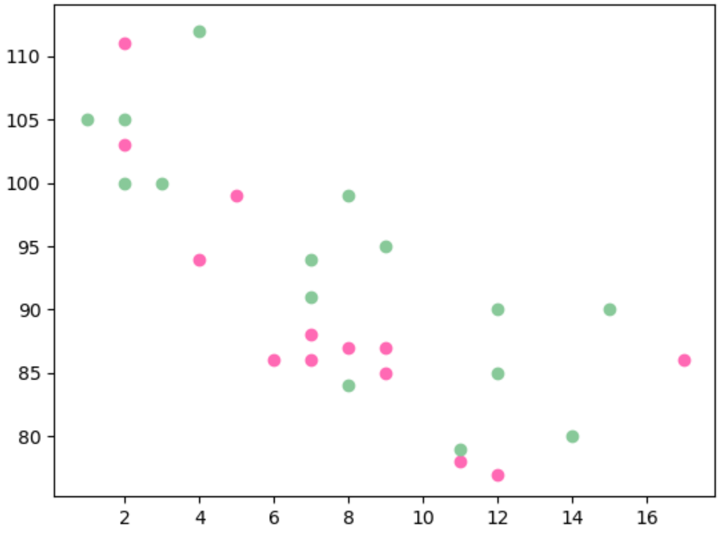
b. <u>Compare Plots</u>

In the example above, there seems to be a relationship between speed and age in one day, but in other day the observation would be the same?

```
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, color = 'pink')

#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 = 'lightgreen')

plt.show()
```

Note: The default color is blue and orange.

By comparing the two plots, I think it is safe to say that they both gives us the **same conclusion: the newer the car, the faster it drives**.

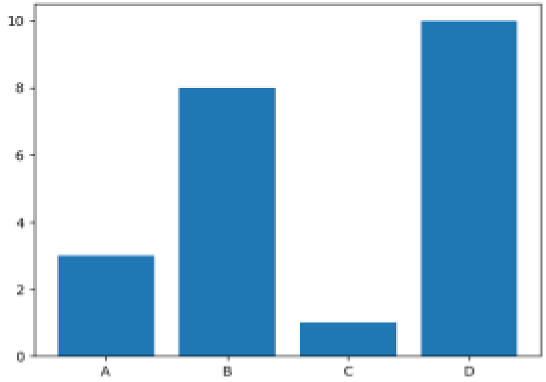


10. Matplotlib Bars

a. Creating Bars

Draw 4 bars:

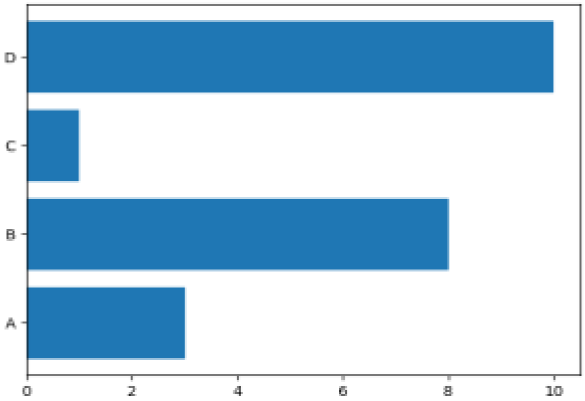
```
import matplotlib.pyplot as plt
import numpy as np

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

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



b. Horizontal Bars

Draw 4 horizontal bars:

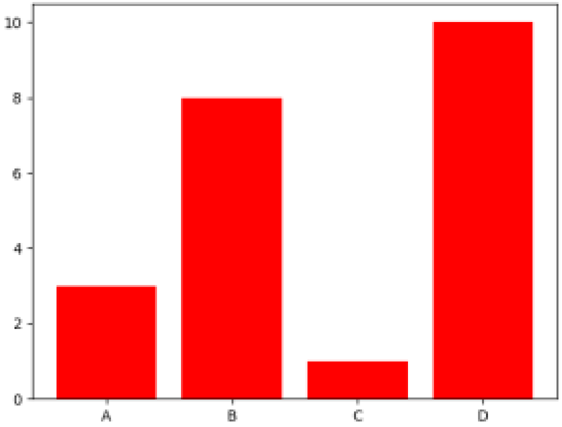
```
import matplotlib.pyplot as plt
import numpy as np

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

plt.barh(x, y, color=”red”)
plt.show()
```



c. Bar Color

Draw 4 red bars:

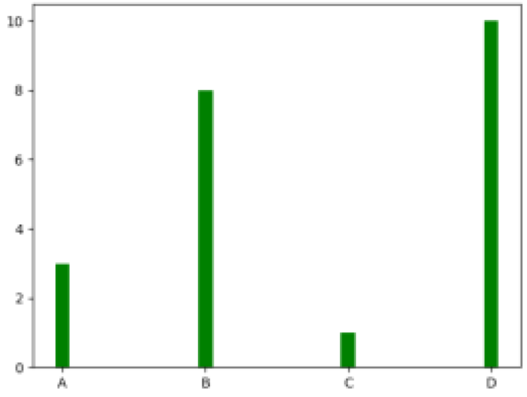
```
import matplotlib.pyplot as plt
import numpy as np

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

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



d. Bar Width (used in “bar” only)

Draw 4 very thin 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, width = 0.1, color="green")
plt.show()
```



The default width value is **0.8**

Note: For horizontal bars, use `height` instead of `width`.

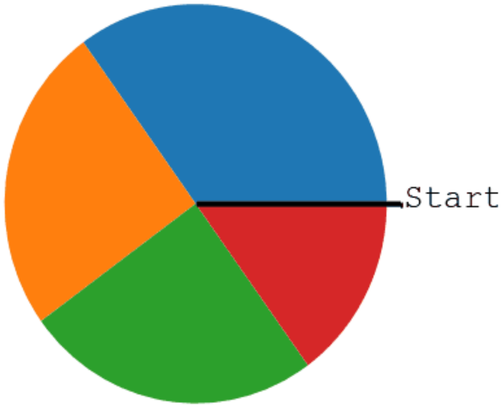
11. Matplotlib Pie Charts

a. Creating Pie Charts

A simple one piece pie chart: (wedge)

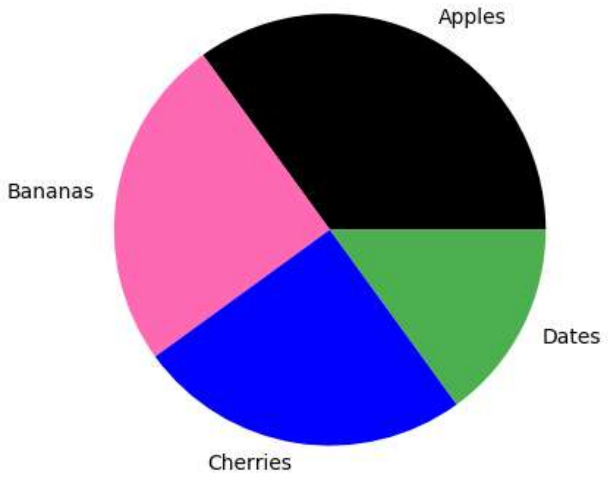
```
import matplotlib.pyplot as plt
import numpy as np

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

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

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



b. Labels and colors

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

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
mycolors = ["black", "pink", "blue", "green"]

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



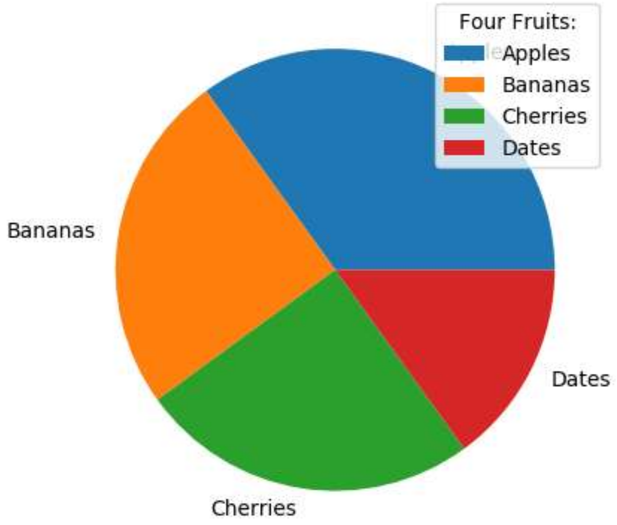
c. Legend with header

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

To add a header to the legend, add the `title` parameter to the `legend` function.

Add a legend with a header:

```
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 = "Four Fruits:")
plt.show()
```


Four Fruits:
Apples
Bananas
Cherries
Dates
Bananas
Dates
Cherries