

# NUMPY ARRAYS:

1. Numpy stands for "Numeric Python" or "Numerical python."
2. Numpy is a package that contains several classes, functions, variables etc. to deal with scientific calculations in Python.
3. Numpy is useful to create and process single and multi-dimensional arrays.
4. In addition, numpy contains a large library of mathematics like linear algebra functions and Fourier transformations.

NOTE: The arrays which are created using numpy are called n dimensional arrays where n can be any integer. If  $n = 1$  it represents a one dimensional array. If  $n = 2$ , it is a two dimensional array etc.

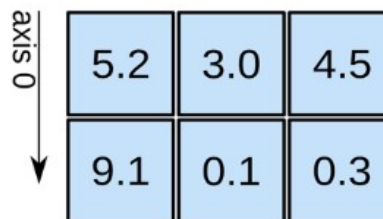
Numpy array can accept only one type of elements. We cannot store different data types into same arrays.

## ARRAY STRUCTURE IN NUMPY

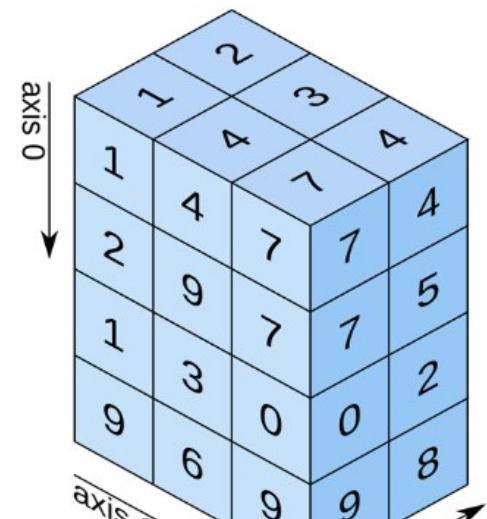
1D array




2D array




3D array

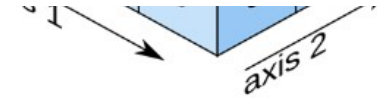


A horizontal arrow pointing to the right, labeled "axis 0" below it.

shape: (4,)

A horizontal arrow pointing to the right, labeled "axis 1" below it.

shape: (2, 3)

A 3D coordinate system with three axes. The vertical axis is labeled "axis 1", the horizontal axis is labeled "axis 2", and the diagonal axis is labeled "axis 0". A blue rectangular prism is shown in the first octant.

shape: (4, 3, 2)

We have 1D array, 2D array and 3D array

## PYTHON AND NUMPY

For working with numpy, we should first import numpy module into our Python program.

Following line of code is use to import numpy in the python programme.

```
import numpy or  
import numpy as <<name>>
```

```
In [2]: import numpy as np  
print(np.__version__)  
1.20.1
```

```
In [3]: #1D array  
import numpy as np
```

```
In [5]: A1=np.array([1,2,3,4])  
print(A1)
```

```
[1 2 3 4]
```

```
In [6]: type(A1)
```

```
Out[6]: numpy.ndarray
```

```
In [7]: A1.shape
```

```
Out[7]: (4,)
```

```
In [8]: A1.size
```

```
Out[8]: 4
```

```
In [9]: #2D array  
A2=np.array([[1,2,3,4],[5,6,7,8]])  
print(A2)
```

```
[[1 2 3 4]  
 [5 6 7 8]]
```

```
In [10]: type(A2)
```

```
Out[10]: numpy.ndarray
```

```
In [11]: A2.shape
```

```
Out[11]: (2, 4)
```

```
In [12]: A2.size
```

```
Out[12]: 8
```

In [13]: `A2.ndim`

Out[13]: 2

In [18]: `#3D array`

```
A3=np.array([[[1,2,3],[4,5,6],[7,8,9]]])  
print(A3)
```

```
[[[1 2 3]  
   [4 5 6]  
   [7 8 9]]]
```

In [19]: `type(A3)`

Out[19]: `numpy.ndarray`

In [20]: `A3.shape`

Out[20]: (1, 3, 3)

In [22]: `A3.size`

Out[22]: 9

In [23]: `A3.ndim`

Out[23]: 3

## Zeros Array-An array in which all values are 0

```
ZA=np.zeros(shape,dtype)
```

1.we can define the shape and data-type of zeros array.

2.we can create 1D,2D and 3D zeros array.

3.the default data-type is float

```
In [25]: import numpy as np

z1=np.zeros(3)
z1
```

```
Out[25]: array([0., 0., 0.])
```

```
In [26]: #changing data type

z1=np.zeros(3,dtype=int)
z1
```

```
Out[26]: array([0, 0, 0])
```

```
In [27]: z1.shape
```

```
Out[27]: (3,)
```

```
In [28]: z1.size
```

```
Out[28]: 3
```

```
In [29]: z1.ndim
```

```
Out[29]: 1
```

```
In [30]: type(z1)
```

```
Out[30]: numpy.ndarray
```

```
In [32]: z2=np.zeros((3,4))
z2
```

```
Out[32]: array([[0., 0., 0., 0.],
               [0., 0., 0., 0.],
               [0., 0., 0., 0.]])
```

```
In [33]: z2=np.zeros((3,4),dtype=int)
z2
```

```
Out[33]: array([[0, 0, 0, 0],
               [0, 0, 0, 0],
               [0, 0, 0, 0]])
```

```
In [34]: type(z2)
```

```
Out[34]: numpy.ndarray
```

```
In [35]: z2.shape
```

```
Out[35]: (3, 4)
```

```
In [36]: z2.size
```

```
Out[36]: 12
```

```
In [37]: z2.ndim
```

```
Out[37]: 2
```

```
In [38]: z3=np.zeros((2,3,4))
z3
```

```
Out[38]: array([[[0., 0., 0., 0.],
                 [0., 0., 0., 0.],
                 [0., 0., 0., 0.]],
               [[0., 0., 0., 0.],
                 [0., 0., 0., 0.],
                 [0., 0., 0., 0.]])
```

```
In [39]: z3=np.zeros((2,3,4),dtype=int)
z3
```

```
Out[39]: array([[[0, 0, 0, 0],
                [0, 0, 0, 0],
                [0, 0, 0, 0]],

               [[0, 0, 0, 0],
                [0, 0, 0, 0],
                [0, 0, 0, 0]]])
```

```
In [40]: type(z3)
```

```
Out[40]: numpy.ndarray
```

```
In [41]: z3.ndim
```

```
Out[41]: 3
```

```
In [42]: z3.size
```

```
Out[42]: 24
```

```
In [43]: z3.shape
```

```
Out[43]: (2, 3, 4)
```

## Ones Array-An array in which all values are 1

```
A=np.ones(shape,dtype)
```

1.we can define the shape and data type of ones array.

2 we can create 1D,2D,& 3D ones array.

3.the default data-type is float.

```
In [45]: #1D ones array  
import numpy as np  
a1=np.ones(3)  
a1
```

```
Out[45]: array([1., 1., 1.])
```

```
In [46]: a1=np.ones(3,dtype=int)  
a1
```

```
Out[46]: array([1, 1, 1])
```

```
In [49]: type(a1)
```

```
Out[49]: numpy.ndarray
```

```
In [50]: a1.ndim
```

```
Out[50]: 1
```

```
In [51]: a1.shape
```

```
Out[51]: (3,)
```

```
In [52]: a1.size
```

```
Out[52]: 3
```

```
In [48]: #2D ones array  
a2=np.ones([3,4])  
a2
```

```
Out[48]: array([[1., 1., 1., 1.],  
                [1., 1., 1., 1.],  
                [1., 1., 1., 1.]])
```



```
In [53]: a2=np.ones([3,4],dtype=int)
a2
```

```
Out[53]: array([[1, 1, 1, 1],
               [1, 1, 1, 1],
               [1, 1, 1, 1]])
```

```
In [54]: type(a2)
```

```
Out[54]: numpy.ndarray
```

```
In [55]: a2.ndim
```

```
Out[55]: 2
```

```
In [56]: a2.size
```

```
Out[56]: 12
```

```
In [57]: a2.shape
```

```
Out[57]: (3, 4)
```

```
In [60]: #3D ones array
a3=np.ones([4,2,3])
a3
```

```
Out[60]: array([[[1., 1., 1.],
                 [1., 1., 1.]],

                [[1., 1., 1.],
                 [1., 1., 1.]],

                [[1., 1., 1.],
                 [1., 1., 1.]],

                [[1., 1., 1.],
                 [1., 1., 1.]])
```

```
In [61]: type(a3)
```

```
Out[61]: numpy.ndarray
```

```
In [62]: a3.shape
```

```
Out[62]: (4, 2, 3)
```

```
In [63]: a3.size
```

```
Out[63]: 24
```

## Full Array: Am array in which all values are same(constant)

```
F=np.full(shape,fill_value)
```

1.we can define the shape and pass the value to be filled in the "full arrays"

2.we can create 1D,2D,and 3D full arrays.

3.default data type is integer.

```
In [66]: import numpy as np
```

```
#1D full array  
f1=np.full(3,9)  
f1
```

```
Out[66]: array([9, 9, 9])
```

```
In [68]: f1=np.full(3,9,dtype=float)  
f1
```

```
Out[68]: array([9., 9., 9.])
```

```
In [69]: type(f1)
```

```
Out[69]: numpy.ndarray
```

```
In [70]: f1.shape
```

```
Out[70]: (3,)
```

```
In [71]: f1.size
```

```
Out[71]: 3
```

```
In [72]: f1.ndim
```

```
Out[72]: 1
```

```
In [74]: #2D full array  
f2=np.full([2,3],9)  
f2
```

```
Out[74]: array([[9, 9, 9],  
               [9, 9, 9]])
```

```
In [75]: #3D full array  
  
f3=np.full([4,2,3],10)  
f3
```

```
Out[75]: array([[[10, 10, 10],  
                [10, 10, 10]],  
               [[10, 10, 10],  
                [10, 10, 10]],  
               [[10, 10, 10],  
                [10, 10, 10]],  
               [[10, 10, 10],  
                [10, 10, 10]])
```

```
In [76]: f3=np.full([4,2,3],10,dtype='str')  
f3
```

```
Out[76]: array([[['1', '1', '1'],  
                ['1', '1', '1']],  
               [['1', '1', '1'],  
                ['1', '1', '1']],  
               [['1', '1', '1'],  
                ['1', '1', '1']],  
               [['1', '1', '1'],  
                ['1', '1', '1']]), dtype='<U1')
```

```
In [77]: type(f3)
```

```
Out[77]: numpy.ndarray
```

```
In [78]: f3.ndim
```

```
Out[78]: 3
```

```
In [79]: f3.shape
```

```
Out[79]: (4, 2, 3)
```

```
In [80]: f3.size
```

```
Out[80]: 24
```

## Numpy Operations:

**we can perform following operations on the arrays:**

1.addition

2.subtraction

3.multiplication

4.division

5.modulo

Rules for implementing operations in array:

1.shape of two arrays must be equal(rows and columns). \*element to element operation will be done

a=[1,2],[3,4]

b=[5,6],[7,8]

2.second array should have atleast one dimension and the number of elements in that dimensions should be equal to first array.

a=[1,2],[3,4]----(2,2)

b=[5,6]------(2,2)

interpreter will consider:[5,6],[5,6]---(2,2)

a=[1,2],[3,4]

b=[5,6,7]

3.second array having single element

a=[1,2],[3,4]

b=10

b=[10,10],[10,10]

In [ ]:

```
In [91]: a=np.array([1,2,3])  
         b=np.array([1,2,3])  
  
         add=np.add(a,b)  
         add
```

```
Out[91]: array([2, 4, 6])
```

```
In [92]: a=np.array([5,10,20])  
         b=np.array([4,8,10])  
  
         sub=np.subtract(a,b)  
         sub
```

```
Out[92]: array([ 1,  2, 10])
```

```
In [93]: a=np.array([5,10,20])  
         b=np.array([4,8,10])  
  
         sub=np.multiply(a,b)  
         sub
```

```
Out[93]: array([ 20,  80, 200])
```

```
In [94]: a=np.array([5,10,20])  
         b=np.array([4,8,10])  
  
         sub=np.divide(a,b)  
         sub
```

```
Out[94]: array([1.25, 1.25, 2.  ])
```

```
In [95]: a=np.array([5,10,20])  
         b=np.array([4,8,10])  
  
         sub=np.mod(a,b)  
         sub
```

```
Out[95]: array([1, 2, 0], dtype=int32)
```

```
In [96]: a=np.array([5,10,20])
         b=np.array([4,8,10])

         sub=np.power(a,b)
         sub
```

```
Out[96]: array([      625, 100000000, 797966336], dtype=int32)
```

## DATA VISUALIZATION:

- 1.Data visualization is graphical representation of data.
- 2.It involves producing images that communicate relationships among the represented data to viewers.
- 3.Visualizing data is an essential part of data analysis and ML.

## Installing matplotlib:

```
In [97]: !pip install matplotlib
```

```
Requirement already satisfied: matplotlib in c:\users\admin\anaconda3\lib\site-packages (3.3.4)
Requirement already satisfied: cyclor>=0.10 in c:\users\admin\anaconda3\lib\site-packages (from matplotlib) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\admin\anaconda3\lib\site-packages (from matplotlib) (2.8.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\users\admin\anaconda3\lib\site-packages (from matplotlib) (2.4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\admin\anaconda3\lib\site-packages (from matplotlib) (1.3.1)
Requirement already satisfied: numpy>=1.15 in c:\users\admin\anaconda3\lib\site-packages (from matplotlib) (1.20.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\admin\anaconda3\lib\site-packages (from matplotlib) (8.2.0)
Requirement already satisfied: six in c:\users\admin\anaconda3\lib\site-packages (from cyclor>=0.10->matplotlib) (1.15.0)
```

## Importing matplotlib:

```
In [98]: import matplotlib.pyplot as plt
```

### 1.Line Chart:

\*Line charts are one of the simplest and most widely used data visualization technique.

\*A line chart displays information as a series of data points or markers connected by straight line.

\*We can customize the shape,size,color and other aesthetic elements of the markers and lines for better visualization.

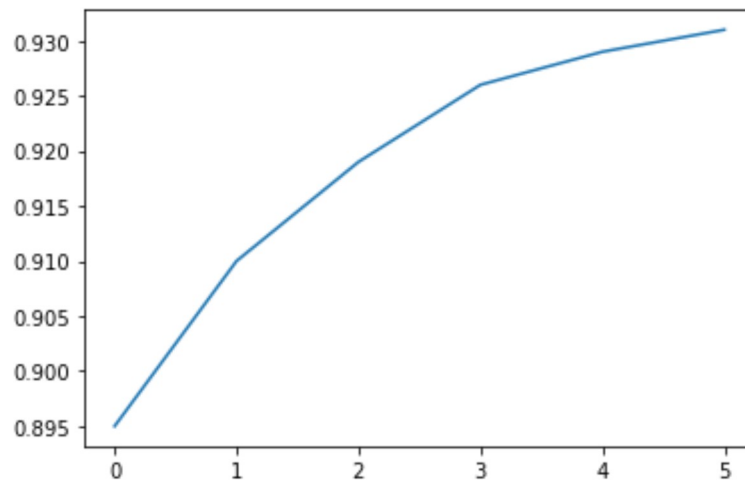
```
In [99]: """
here is a python list showing the yield of apples (tons per hectare) over 6 years in an imaginary country called Appleland
"""
yield_Apples=[0.895,0.91,0.919,0.926,0.929,0.931]
```



```
In [100]: plt.plot(yield_Apples)

# can visualize how the apples changes over time using line chart.
# draw a line chart we can use plt.plot() function.
```

```
Out[100]: [<matplotlib.lines.Line2D at 0x2ab7d969370>]
```



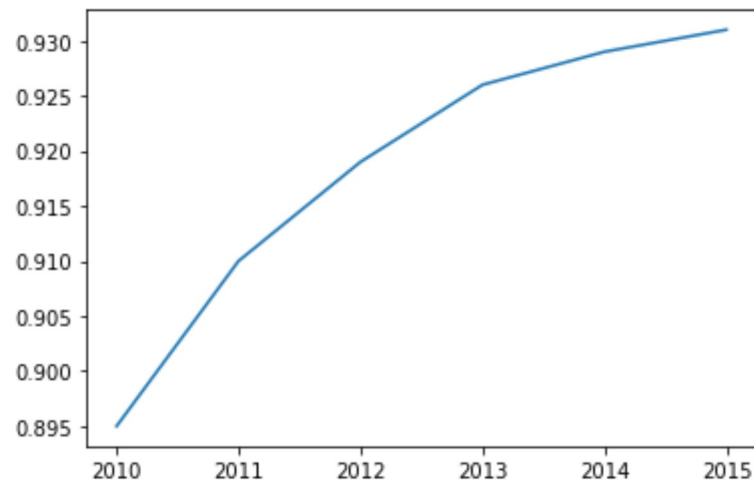
## Customizing x-axis:

- 1.The x-axis of the plot currently shows list elements indexes from 0 to 5.
- 2.The would be more informative if we could show the year for whcih the data being plotted.
- 3.We can do this by using plt.plot.

```
In [101]: years=[2010,2011,2012,2013,2014,2015]
yield_apples=[0.895,0.91,0.919,0.926,0.929,0.931]

plt.plot(years,yield_apples)
```

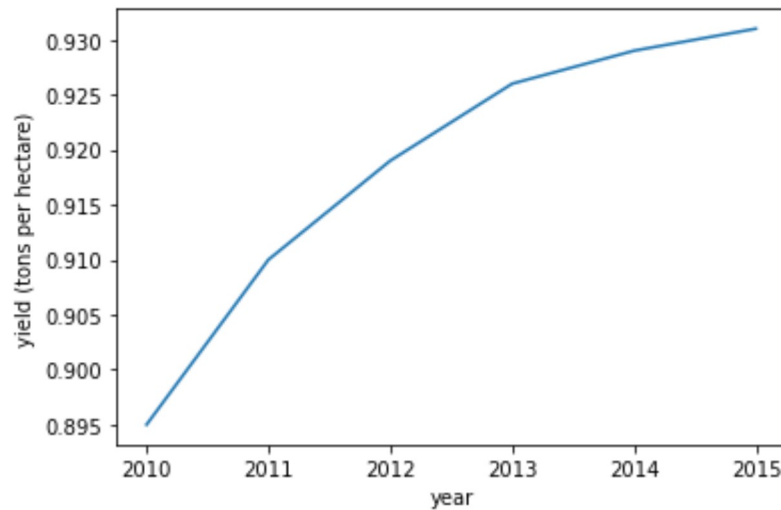
```
Out[101]: [<matplotlib.lines.Line2D at 0x2ab7d92f190>]
```



## Axis Labels:

we can add the labels to the axes to show what each axis represents using `plt.xlabel` and `plt.ylabel` methods:

```
In [102]: plt.plot(years,yield_apples)
plt.xlabel('year')
plt.ylabel('yield (tons per hectare)');
```



## Plotting multiple lines:

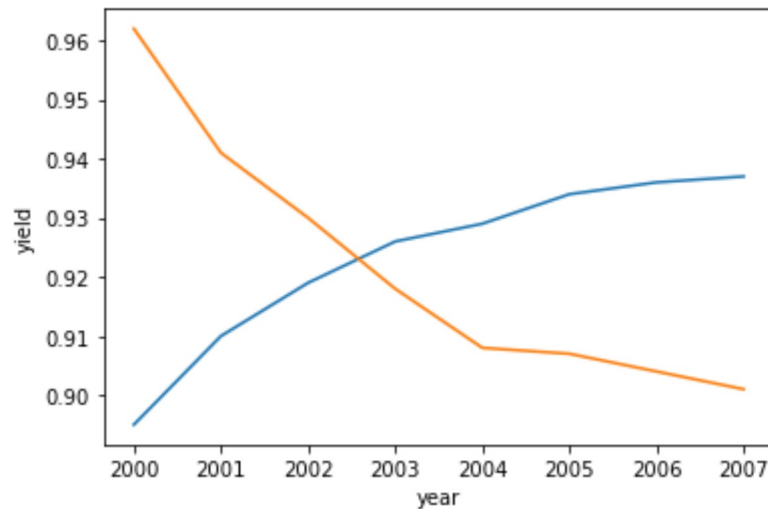
- 1.it's really easy to plot multiple lines in the same graph.
- 2.Just invoke the plt.plot function multiple times.
- 3.Let's compare the yields of apples vs oranges in kanto.

```
In [103]: year=range(2000,2008)

apples=[0.895,0.91,0.919,0.926,0.929,0.934,0.936,0.937]
oranges=[0.962,0.941,0.930,0.918,0.908,0.907,0.904,0.901]
```

```
In [104]: plt.plot(year,apples)
plt.plot(year,oranges)
plt.xlabel('year')
plt.ylabel('yield')
```

```
Out[104]: Text(0, 0.5, 'yield')
```



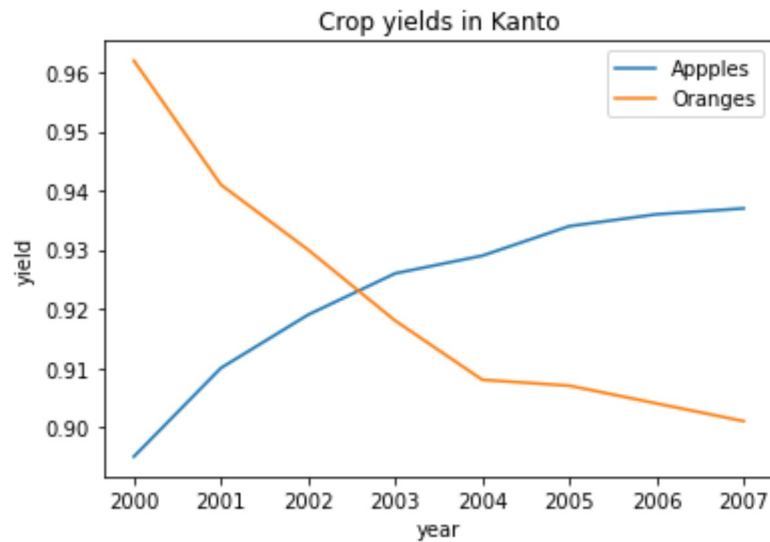
## Chart title and legend:

To differentiate between multiple lines ,we can include a legend within the graph using plt.legend function.

We also give the entire chart a title using the plt.title function

```
In [105]: year=range(2000,2008)

apples=[0.895,0.91,0.919,0.926,0.929,0.934,0.936,0.937]
oranges=[0.962,0.941,0.930,0.918,0.908,0.907,0.904,0.901]
plt.plot(year,apples)
plt.plot(year,oranges)
plt.xlabel('year')
plt.ylabel('yield')
plt.title('Crop yields in Kanto')
plt.legend(['Apples', 'Oranges']);
```

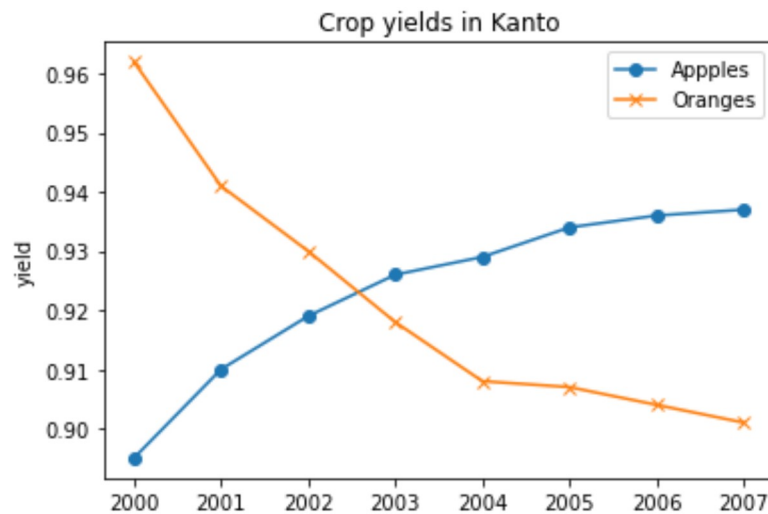


## Line Markers:

1. We can also show markers for the data points on each line using marker argument of plt.plot.
2. Matplotlib supports many different types of markers like circle, cross, square, diamond etc.

```
In [106]: plt.plot(year,apples,marker='o')
plt.plot(year,oranges,marker='x')

plt.ylabel('yield')
plt.title('Crop yields in Kanto')
plt.legend(['Apples','Oranges']);
```



## Styling lines and markers:

1. `plt.plot` function supports many arguments for styling lines and markers.

`color` or `c`: set the color of the line.

`linestyle` or `ls`: choose between a solid or dashed line.

`linewidth` or `lw`: set the width of the line

`markersize` or `ms`: set the size of markers

`markeredgecolor` or `mec`: set the edge color for markers

`markeredgewidth` or `mew`: set the edge width for markers

markerfacecolor or mfc: set the fill color for markers

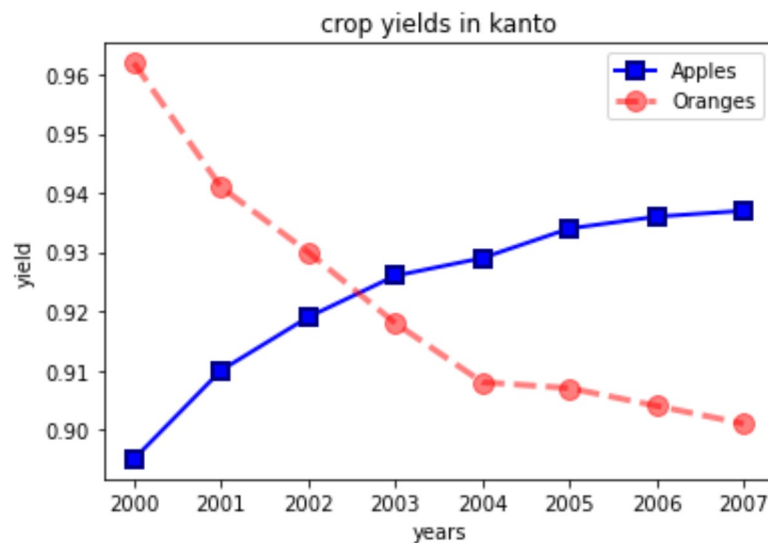
alpha :opacity of the plot

```
In [107]: plt.plot(year,apples,marker='s',c='b',ls='-',lw=2,ms=8,mew=2,mec='navy')
plt.plot(year,oranges,marker='o',c='r',ls='--',lw=3,ms=10,alpha=0.5)

plt.xlabel('years')
plt.ylabel('yield')

plt.title('crop yields in kanto')
plt.legend(['Apples','Oranges'])
```

Out[107]: <matplotlib.legend.Legend at 0x2ab7d680850>



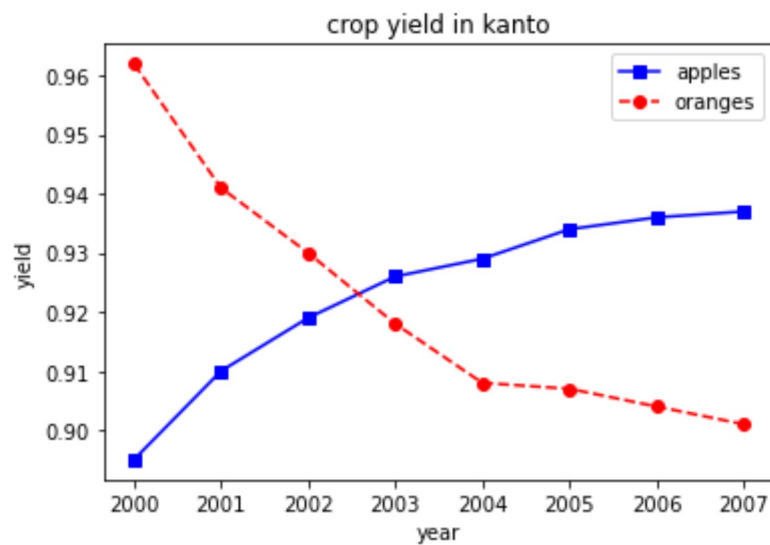
The fmt argument a shorthand for specifying the line style,marker and line color.

it can be provided as the third argument to plt.plot

fmt='[marker][line][color]'

```
In [108]: plt.plot(year,apples,'s-b')  
plt.plot(year,oranges,'o--r')  
  
plt.xlabel('year')  
plt.ylabel('yield')  
  
plt.title('crop yield in kanto')  
plt.legend(['apples','oranges'])
```

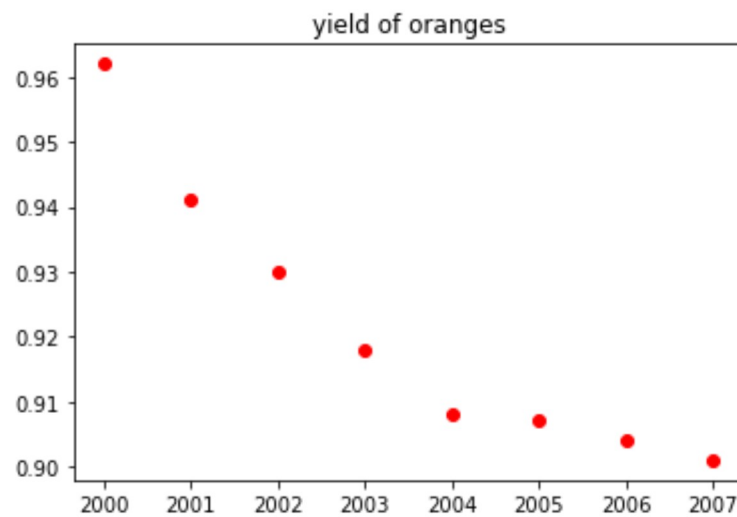
Out[108]: <matplotlib.legend.Legend at 0x2ab7d58f610>





```
In [109]: #if no line is specified in fmt, only markers are drawn.  
plt.plot(year,oranges,'or')  
plt.title('yield of oranges')
```

```
Out[109]: Text(0.5, 1.0, 'yield of oranges')
```

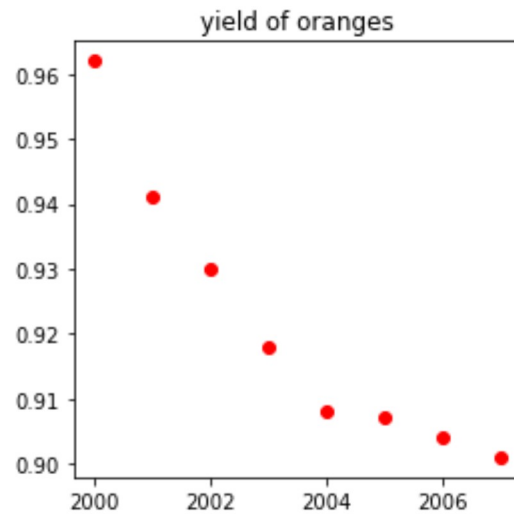


## Changing the figure size:

you can use the plt.figure function to change the size of the figure

```
In [110]: plt.figure(figsize=(4,4))  
  
plt.plot(year,oranges,'or')  
plt.title('yield of oranges')
```

```
Out[110]: Text(0.5, 1.0, 'yield of oranges')
```



## BAR PLOT:

- 1.A bar plot or bar chart is a graph that represents the category of data with rectangular bars with lengths and heights that is proportional to the values which they represent.
- 2.The bar plots can be plotted horizontally or vertically.
- 3.A bar chart describes the comparisons between the discrete categories. It can be created using the bar() method.

## Syntax:

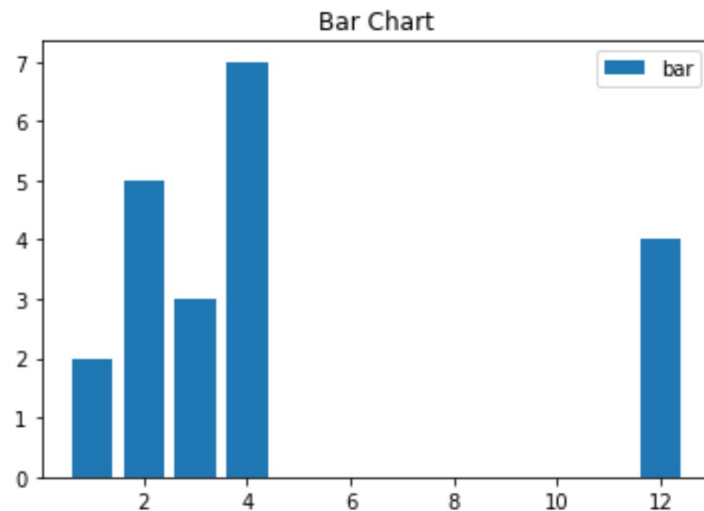
```
plt.bar(x, height, width, bottom, align)
```

```
In [111]: import matplotlib.pyplot as plt
# data to display on plots
x = [3, 1, 3, 12, 2, 4, 4]
y = [3, 2, 1, 4, 5, 6, 7]

# This will plot a simple bar chart
plt.bar(x, y)

# Title to the plot
plt.title("Bar Chart")

# Adding the Legends
plt.legend(["bar"])
plt.show()
```



## SCATTER PLOT:

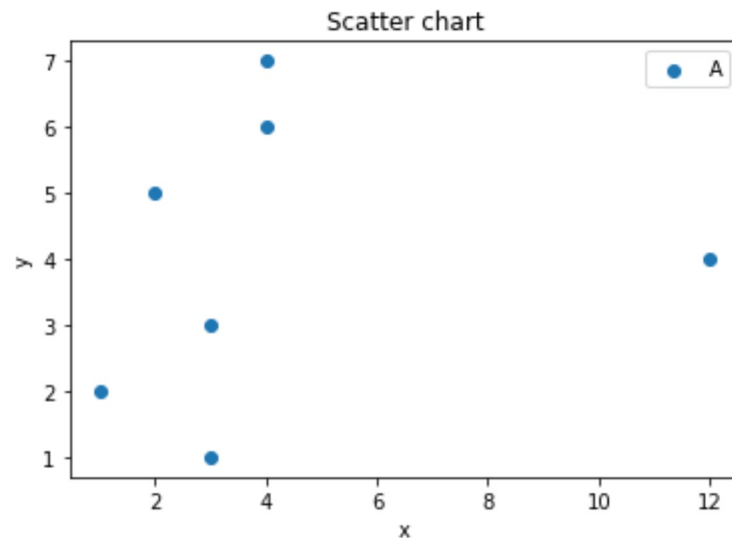
- 1.Scatter plots are used to observe the relationship between variables and use dots to represent the relationship between them.
- 2.The scatter() method in the matplotlib library is used to draw a scatter plot.

```
In [112]: import matplotlib.pyplot as plt
# data to display on plots
x = [3, 1, 3, 12, 2, 4, 4]
y = [3, 2, 1, 4, 5, 6, 7]

# This will plot a simple scatter chart
plt.scatter(x, y)

# Adding Legend to the plot
plt.legend("A")
plt.xlabel('x')
plt.ylabel('y')

# Title to the plot
plt.title("Scatter chart")
plt.show()
```



## PIE CHART:

- 1.A Pie Chart is a circular statistical plot that can display only one series of data.
- 2.The area of the chart is the total percentage of the given data.

3.The area of slices of the pie represents the percentage of the parts of the data.

4.The slices of pie are called wedges.

5.The area of the wedge is determined by the length of the arc of the wedge. It can be created using the pie() method.

```
In [113]: import matplotlib.pyplot as plt
# data to display on plots
x = [1, 2, 3, 4]

# this will explode the 1st wedge
# i.e. will separate the 1st wedge
# from the chart
e =(0.1, 0, 0, 0)

# This will plot a simple pie chart
plt.pie(x, explode = e)

# Title to the plot
plt.title("Pie chart")
plt.show()
```

Pie chart



References:

1. <https://matplotlib.org/stable/index.html> (<https://matplotlib.org/stable/index.html>)

2. <https://numpy.org/doc/> (<https://numpy.org/doc/>)

In [ ]: