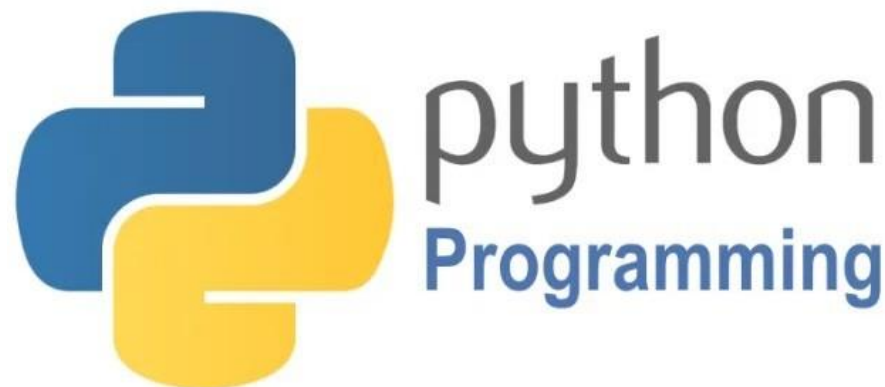


PYTHON PROGRAMMING LANGUAGE



What is python?

1. Python is powerful programming language.
2. Specially used in data science, machine learning to solve daily life real-time problems and providing the solutions.
3. Python is a high-level, interpreted, interactive and object oriented scripting language.

History of Python language:

1. Python was developed by "Guido Van Rossum" in late 1980's in National Research Institute for Mathematics and Computer Science in Netherlands.
2. Python was derived from many other languages including ABC, Modulo-3, C, C++, Algol-68, SmallTalk, Unix shell and other scripting languages.

Why python?

1. Easy to understand
2. Easy syntax
3. Beginner friendly

4.Supports multiple libraries

5.Supports OOP's

Applications of python

1.Web applications

2.Desktop applications

3.Database applications

4.Web scraping

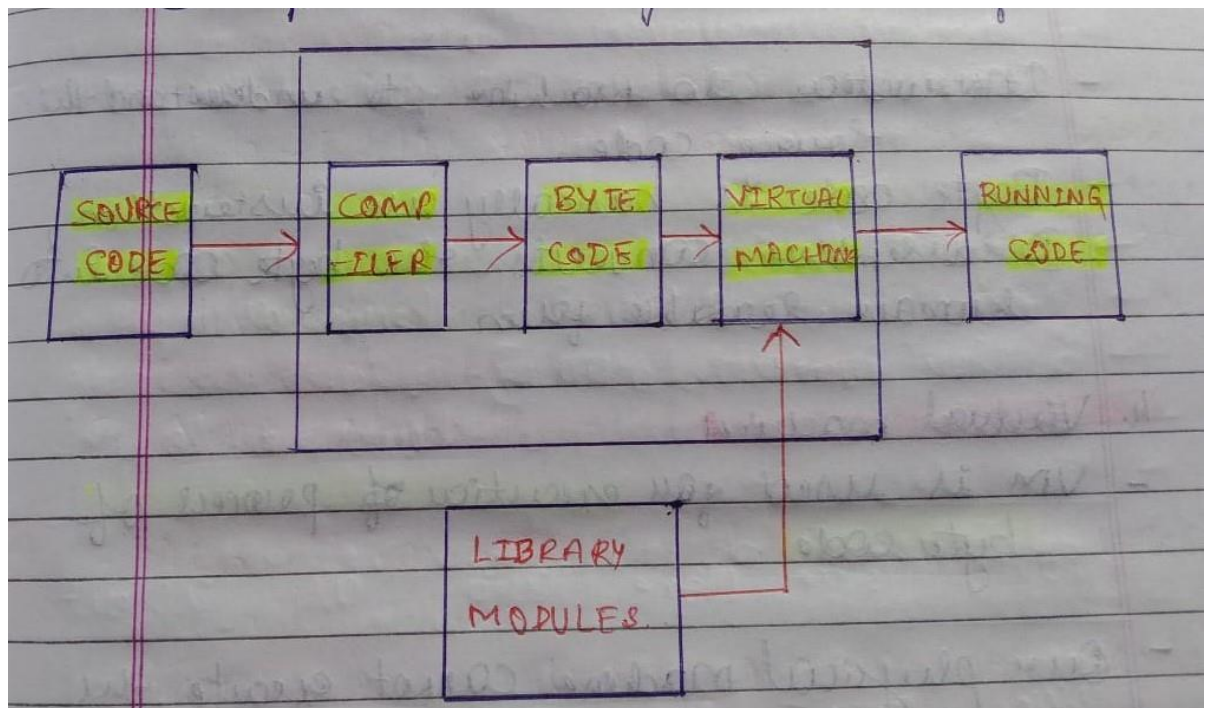
5.Wen mapping

6.Data analysis

7.Interactive web visualization

8.Computer vision and image processing etc.,

INTERNAL WORKING OF PYTHON



1.Python identifiers:

1.A Python identifier is a name used to identify variable,function,class ,module or other objects.

2.A identifier starts with a letter (a-z),(A-Z) or an underscore(_) followed by zero or more letters,and digits(0-9)

Naming conventions for identifiers:

- 1.class names start with an uppercase letter .All the other identifiers start with a lower case letter.
2. Starting an identifier with a single leading underscore indicates that the identifier is "private".
- 3.starting an identifier with two leading underscores indicates "strongly private".
- 4.If the identifier also ends with two trailing underscore ,the identifier is a language defined special name.

Python keywords:

- 1.Keywords are reserved words.
- 2.cannot use them as constant or variable.
- 3.all the keywords except "True","False" and "None" are in lowercase.
- 4.they are 35 keywords present in python

In [42]: *#code to print all the keywords present in python*

```
import keyword
print(keyword.kwlist)
a=len(keyword.kwlist)
print(a)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', '
class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for',
'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'o
r', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
35
```

Indentation in python:

- 1.In C,C++,the indentation is specified by using the flower brackets{}.
- 2.In python,indentation is done by 4 white spaces or tab.

Python comments:

In python we have 2 types of comments such as follow:

- 1.single line comment(specified using single or double quotes).

2. multi line comment(specified using doc string).

```
In [43]: #single line comment--->shows the single line comment.  
print('hello my dear friends')
```

hello my dear friends

```
In [44]: """  
multi line comment  
how are you all  
have a nice day guys  
"""  
print('hello CSE people')
```

hello CSE people

PYTHON VARIABLES:

1. Variables are nothing but reserved memory locations to store values.

2. When we create a variable ,automatically some space in memory is reserved.

3. Based on the data types the interpreter allocates the memory.

4. Variable can take any data type such as integer,float numbers,strings etc.

```
In [45]: #example 1:-  
#here 3 variables have been created and stored the values as shown below  
a=50  
b=50  
c=a+b  
print(c)  
  
#example 2 on strings  
  
x='hello'  
x
```

100

Out[45]: 'hello'

Python Data types:

1. Integer data type

2. Float point numbers

3. Strings data type

4.Boolean data type

Python Data Structures

1. In python we have four types of data structures

2. These data structures are very useful and important while working with advance topics such as machine learning ,data science.

1.List

2.Tuple

3.Dictionary

4.Sets

1.LISTS IN PYTHON:

*List is a python data structure

*it is a combination of any data type.

*enclosed within [] and separated by commas(,).

*list is sequence of values called items or elements.

*list is similar arrays.

*list mutable.

*list is indexible.

In [46]: *#Creating a list with constructor()*

```
a=[1,2,3,4,5]
b=[2.4,7.0,6.8]
c=['hello','students']
print(a)
print(b)
print(c)
```

```
[1, 2, 3, 4, 5]
[2.4, 7.0, 6.8]
['hello', 'students']
```

In [47]: `type(a)`

Out[47]: `list`

In [48]: *#creating list with heterogenous data:-*

```
a=(['hello',8,9,20.77])  
a
```

Out[48]: ['hello', 8, 9, 20.77]

In [49]: *#create list using range function:-*

```
L=list(range(0,6))  
print(L)
```

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

In [50]: *#creating a list without constructor:-*

```
days=['monday','tuesday','wednesday','thursday','friday','saturday','sunday']  
days
```

Out[50]: ['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunday']

In [51]: *#accessing list items using index values:-*

```
days[1]
```

Out[51]: 'tuesday'

In [52]: *#negative indexing:-*

#creating a list without constructor:-

```
days=['monday','tuesday','wednesday','thursday','friday','saturday','sunday']  
days[-1]
```

Out[52]: 'sunday'

In [53]: days[:]

Out[53]: ['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunday']

In [54]: *#list slicing:-*

```
days[1:4]
```

Out[54]: ['tuesday', 'wednesday', 'thursday']

In [55]: *#list slicing with step size:-*

```
days=['monday','tuesday','wednesday','thursday','friday','saturday','sunday']  
days[1:4:2]
```

Out[55]: ['tuesday', 'thursday']

List built-in function:

Function	Meaning
len()	Returns the number of elements in a list
max()	Returns element with greater value
min()	Returns the element with lowest value
sum()	Returns the sum of all elements in the list
random.shuffle()	Shuffle the elements in a list randomly -Here we make use of a random package .

```
In [56]: #example 1:-
days=['monday','tuesday','wednesday','thursday','friday','saturday','sunday']
len(days)
```

Out[56]: 7

```
In [57]: numbers=[1,2,3,4,5,6,7,8]
max(numbers)
```

Out[57]: 8

```
In [58]: numbers=[1,2,3,4,5,6,7,8]
min(numbers)
```

Out[58]: 1

```
In [59]: numbers=[1,2,3,4,5,6,7,8]
sum(numbers)
```

Out[59]: 36

```
In [60]: import random
numbers=[1,2,3,4,5,6,7,8]
random.shuffle(numbers)
print(numbers)
```

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

List Operators

Operator	Meaning
"+"	Concatenates two lists with each other.
"*"	Replicates the elements of lists
"in"	-Used to determine whether the element is present in the list or not . -Returns true if the element is present in the list -Returns false if the element is not present.
"is"	Used to check whether two variables refer to the same object or not .
"del"	-Used to remove elements from the list.

In [61]: *#examples on operator:-*

```
a=[1,2,3]
b=[4,5,6]
c=a+b
print(c)
```

[1, 2, 3, 4, 5, 6]

In [63]:

```
a=[10,20]
b=3*a
print(b)
```

[10, 20, 10, 20, 10, 20]

In [64]:

```
days=['monday','tuesday','wednesday','thursday','friday','saturday','sunday']
'wednesday' in days
```

Out[64]: True

In [65]:

```
days=['monday','tuesday','wednesday','thursday','friday','saturday','sunday']
'wednesday' not in days
```

Out[65]: False


```
In [66]: a='Arshiya'
         b='Arshiya'
         a is b
```

Out[66]: True

```
In [67]: a='Arshiya'
         b='Lubna'
         a is b
```

Out[67]: False

```
In [69]: days=['monday','tuesday','wednesday','thursday','friday','saturday','sunday']
         del days[1]
         days
```

Out[69]: ['monday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunday']

List Methods:

Methods	Meaning
append()	Adds element at the end of the list
clear()	Clear the contents of the list
count()	Gives count of repeated items
copy()	Copy list from one list to another list
extend ()	Joins the elements of 2 lists

```
In [70]: a=['a','b','c','d']
         print(a)
         a.append('e')
         print(a)
```

```
['a', 'b', 'c', 'd']
['a', 'b', 'c', 'd', 'e']
```

```
In [71]: list=['red','orange','pink','black']
print(list)
list.clear()
print(list)

['red', 'orange', 'pink', 'black']
[]
```

```
In [72]: nums=[1,2,3,4,1,4,1,7,8,1]
print(nums)
nums.count(1)

[1, 2, 3, 4, 1, 4, 1, 7, 8, 1]
```

Out[72]: 4

```
In [73]: list_1=[1,2,3,4]
print(list_1)
list_2=list_1.copy()
print(list_2)

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

```
In [74]: A=[1,3,2,4,5]
B=[5,6,7,3,6]
print(A)
print(B)
A.extend(B)
print(A)

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

NOTE:we have many other methods such as index(),insert(),pop(),remove(),reverse(),sort() and so on.

2.PYTHON TUPLES

1.Tuple is another important data structures present in python.

2.Allow multiple data types.

3.Enclosed () and separated by commas(,).

4.Tuples are not mutable

5.Tuples are indexible

6.Tuples are ordered

In [79]: *#creating tuple with single element:-*

```
a=(5,)
print(a)
print(type(a))
```

```
(5,)
<class 'tuple'>
```

In [80]: *#creating tuple with integers*

```
a=(1,2,3,4,5)
print(a)
```

```
(1, 2, 3, 4, 5)
```

In [81]: *#creating tuple with float numbers*

```
b=(6.5,5.9,2.0)
print(b)
```

```
(6.5, 5.9, 2.0)
```

In [82]: *#creating tuple with strings*

```
c=('one','two','three')
c
```

Out[82]: ('one', 'two', 'three')

In [84]: *#creating tuple multiple data types*

```
mix=('hello',708,66.8,1,2)
mix
```

Out[84]: ('hello', 708, 66.8, 1, 2)

In [86]: *#accessing items of tuple*

```
a=(1,2,3,4,'hello')
print(a[4])
```

```
hello
```

Tuple Methods

Method	Meaning
count()	Returns the number of times a specified value occurs in a tuple.
index()	Searches the tuple for a specific value and

```
index()
```

Searches the tuple for a specific value and returns the position of where it was found

3. Python Dictionary

1. Dictionaries are used to store the values in key-value pair.
2. It's a combination of keys and values
3. Elements are accessed based on the keys
4. It is ordered, mutable, and doesn't allow duplicate elements.
5. Enclosed within {} with key value pair

```
In [87]: #create a dictionary
a={
    'name': 'joy',
    'age': 23,
    'education': 'Engineer'
}
print(a)

{'name': 'joy', 'age': 23, 'education': 'Engineer'}
```

```
In [88]: len(a)
```

```
Out[88]: 3
```

```
In [89]: type(a)
```

```
Out[89]: dict
```

```
In [91]: x=a.keys()
x
```

```
Out[91]: dict_keys(['name', 'age', 'education'])
```

```
In [92]: y=a.values()
y
```

```
Out[92]: dict_values(['joy', 23, 'Engineer'])
```

```
In [93]: z=a.items()
z
```

```
Out[93]: dict_items([('name', 'joy'), ('age', 23), ('education', 'Engineer')])
```

Dictionary methods:

Methods	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of dictionary
get()	Returns the value of specified key
items()	Returns the list containing tuple pair of keys and values
keys(),values()	key()-Returns the keys of the dictionary values()-Returns the values of the dictionary

4.Python Sets:

- 1.Sets are used to store multiple items in a single variable.
- 2.Sets are enclosed withing {}.
3. Sets are unordered,not mutable and unidexible.
4. Sets items are unchangable but we can remove and add new items.

```
In [94]: #Examp/e:-
set1={1,2,3,4}
print(set1)

{1, 2, 3, 4}
```

```
In [95]: 3duplicates not allowed
set={1,2,3,4,5,6,7,1,1}
set
```

```
Out[95]: {1, 2, 3, 4, 5, 6, 7}
```

```
In [96]: len(set)
```

```
Out[96]: 7
```

```
In [97]: type(set)
```

```
Out[97]: set
```

Set Methods:

Methods	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
Difference()	Returns a set containing difference between 2 or more sets
Discard()	Removes the specific item
Copy()	Returns a copy of the set

Note: We have still more methods in sets in the above table few are discussed. Please for more info refer this link(https://www.w3schools.com/python/python_sets.asp)
(https://www.w3schools.com/python/python_sets.asp)

Python Operators

```
In [ ]: We have various python operators which are useful in performing various operations
```

Operators:

1. Arithmetic operators
2. Comparison operators
3. Logical operators
4. Assignment operators
5. Special operators
 - a. Identity operators
 - b. Membership operators

1.Arthematic operators

Arthematic Operators	Description
+	Used for addition
-	Used for subtraction
*	Used for multiplication
/	Used for division
//	Used for floor division
%	Used for modules
**	Used for exponent or power

2.Comparison operators

a -first operand

b-second operand

Comparison operator	Description
>	Checks whether a is greater than b or not
<	Checks whether a is less b or not
>=	Check whether the operators are greater than equal to each other
<=	Check whether operands are less than equals to each others
==	Checks whether 2 operands are equal
!=	Check whether 2 operands are not equal

3.Logical operators

Logical operators	Description
and	True if both the operands are true
or	True if either of the operands is true

not	Returns the compliment of the operand
-----	---------------------------------------

4. Assignment operators

Assignment Operators	Description
=	Assign value of right side of expression to left side operand.
+=	Add and assign: add right side operand with left side operand then assign to left operand
-=	Subtract and assign: subtract right operand with left operand and then assign to left operand
&	Perform AND operation on operands and assign value to left.
**	Perform exponent operation
%=	Perform modulus AND operation
/=	Perform divide AND operation

5. SPECIAL OPERATORS

1. Identity operators:

*Used to check **if** the two values are located on the same part of the memory.

*Returns **true** or **false** depending on conditions.

*two identity operators are '**is**' and '**is not**'.

2. Membership operators:

*used to check whether a value is found in a sequence.

*two membership operators are '**in**' and '**not in**'.

CONTROL FLOW IN PYTHON:

1. if statement

2. if-else statement

3.nested if statement

LOOPS:

1.for loop

2.while loop

Python functions

1.Python functions is a block of statements that performs a specific task.

2.Get executed when the function is called.

3.goal is to put some repeatedly done task together and make a function, so instead of writing the same code again and again for different inputs, we can perform the function calls to reuse the code contained in the function.

4.functions are defined by using "def" keyword.

5.We have 2 types of functions:

a.User-defined function

b.Built-in function

Syntax:

```
def function_name(parameters):  
    #####statements  
function_name(arguments)
```

```
In [ ]: def add():  
        a=2  
        b=3  
        sum=a+b  
        return sum  
add()  
print(add())
```

1.Built-in function:

*python predefined functions that are readily available.

*such as min(),max(),print(),sum() etc

2.User-defined functions:

3. Anonymous function:

*function that we define without a name.

*anonymous functions are also called as lambda function.

*they are not declared without def keyword.

```
In [ ]: function modularity:

def collectdata():
    'function to collect data'
    statement(s)

def cleandata():
    'function to clean data'
    statement(s)

def processdata():
    'function to pre-process the data'
    statement(s)

def exploredate():
    'function to explore data'
    statement(s)

def visualization():
    'function to visualize data'
    statement(s)

#main program:-
collectdata()
cleandata()
processdata()
exploredata()
visualization()
```

NOTE:

1. I request all the students to go through the basics of python.

2. This notebook just covers basics for refreshing the topics what you have studied in previous semester.

3. Requesting to be very thorough with python concepts

4. Python is essential and pre-requisite for machine learning.

Thank You!

In []:

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.

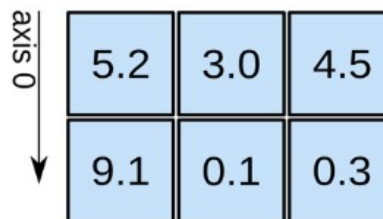
Numpy array can accept only one type of elements. We cannot store different data types into same arrays.

ARRAY STRUCTURE IN NUMPY

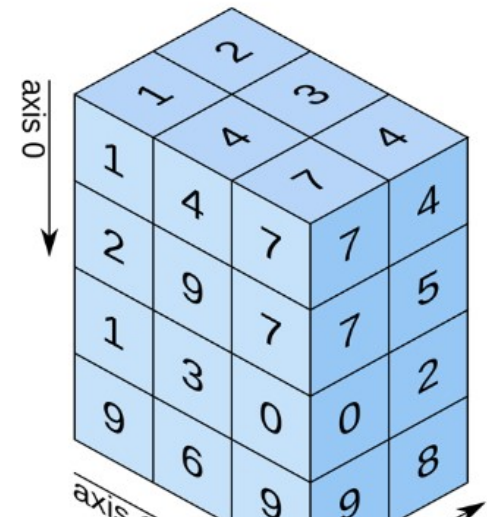
1D array



2D array



3D array



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

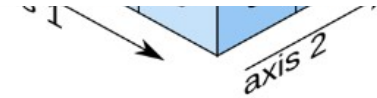
axis 0

shape: (4,)

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

axis 1

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 cube is shown in the first octant, with its edges aligned with the axes.

axis 1
axis 2
axis 0

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 used 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.

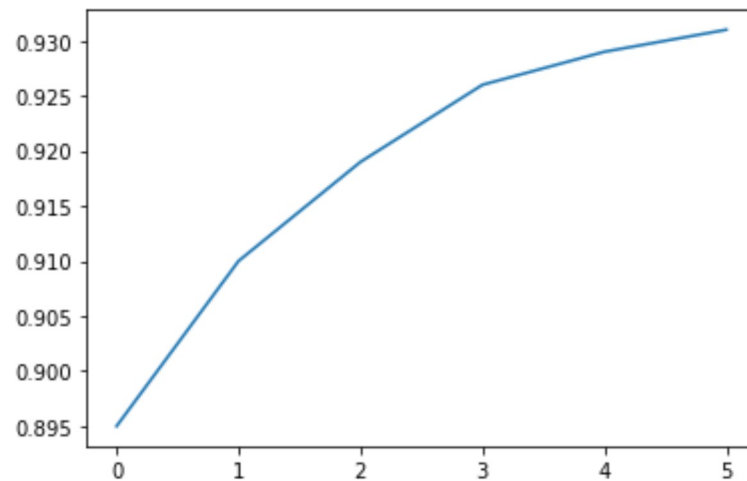
*We can customize the shape,size,color and other aesthetic elements of the markers and lines for better visualization.

```
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>]
```



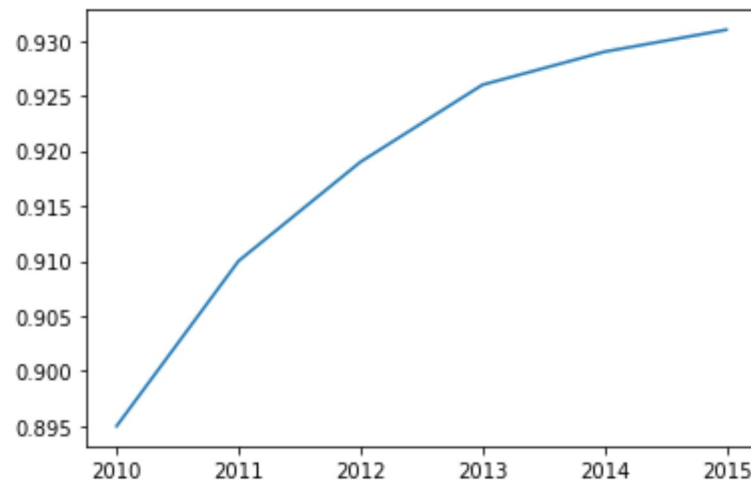
Customizing x-axis:

- 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)
```

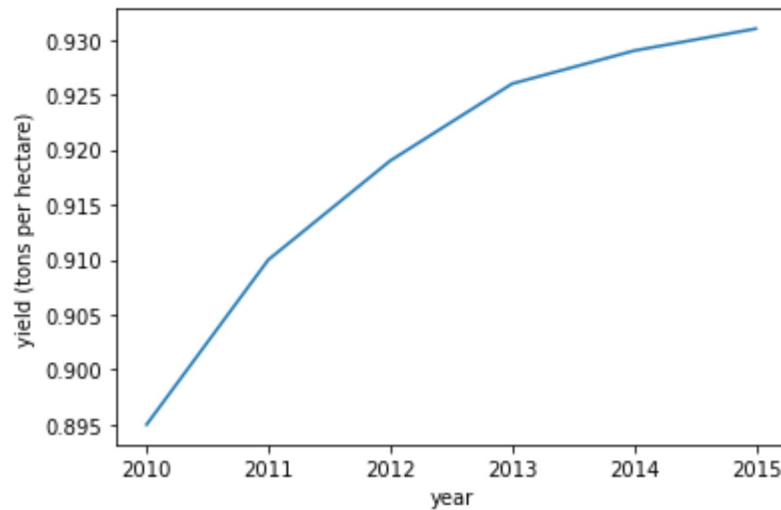
```
Out[101]: [<matplotlib.lines.Line2D at 0x2ab7d92f190>]
```



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')
```

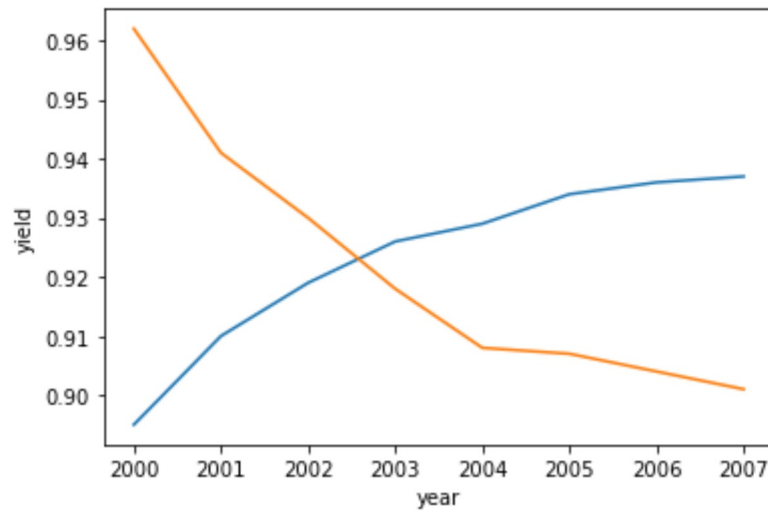


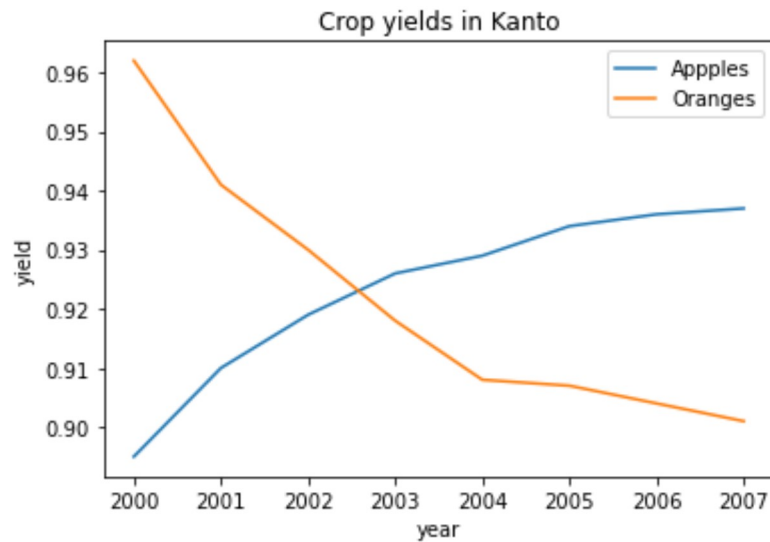
Chart title and legend:

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']);
```

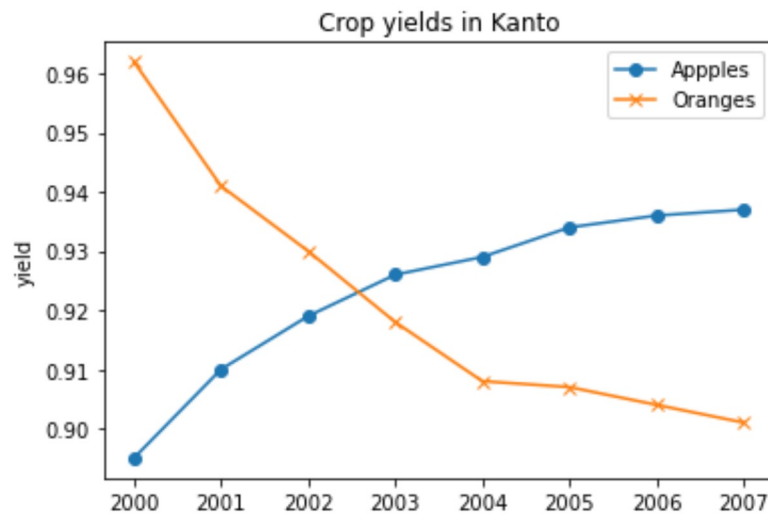


Line Markers:

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

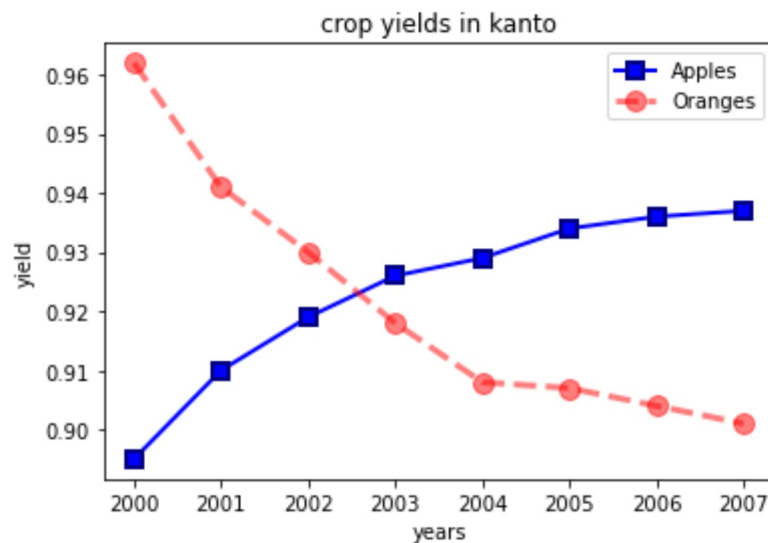
alpha :opacity of the plot

```
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

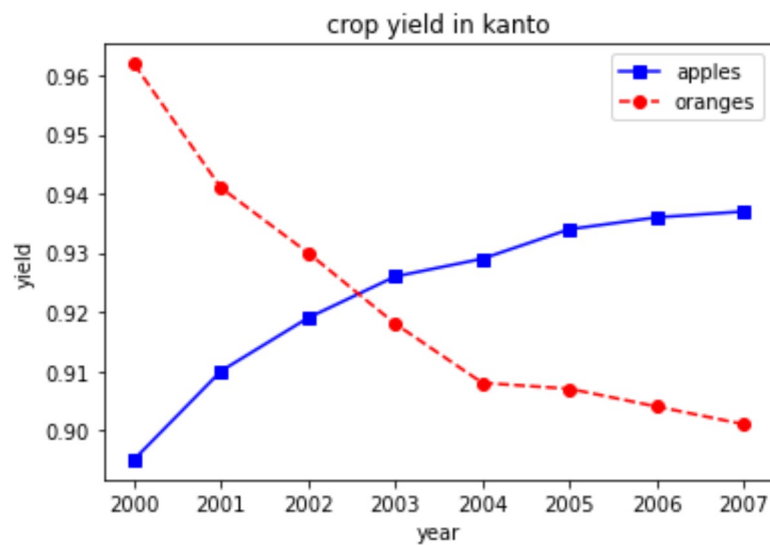
fmt='[marker][line][color]'

```
In [108]: plt.plot(year,apples,'s-b')
plt.plot(year,oranges,'o--r')

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

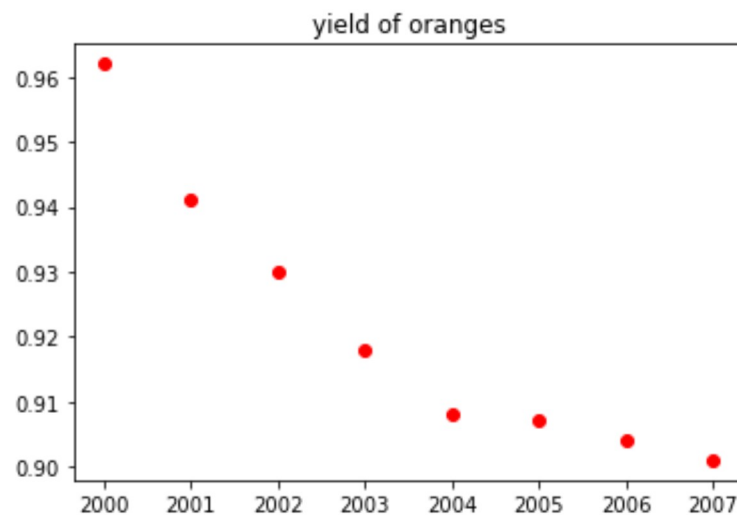
plt.title('crop yield in kanto')
plt.legend(['apples','oranges'])
```

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')
```

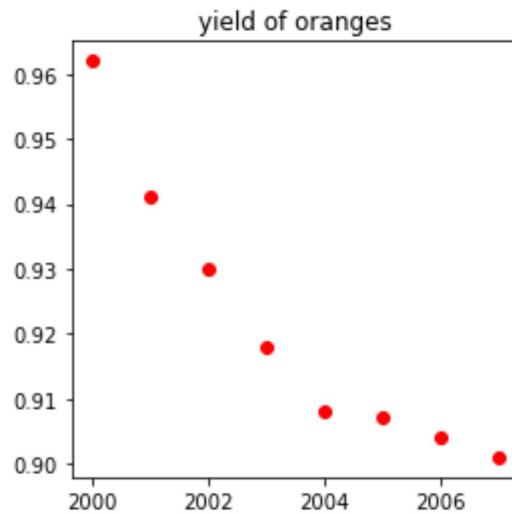


Changing the figure size:

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:

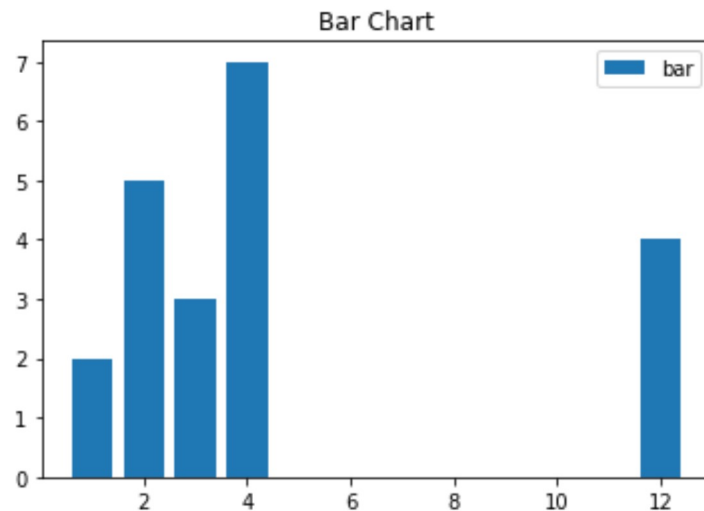
```
plt.bar(x, height, width, bottom, align)
```

```
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:

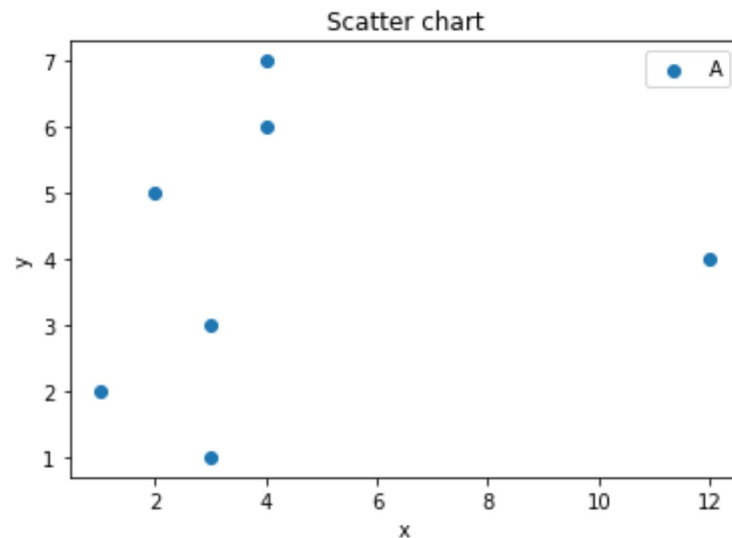
- 1.Scatter plots are used to observe the relationship between variables and use dots to represent the relationship between them.
- 2.The scatter() method in the matplotlib library is used to draw a scatter plot.

```
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 []:

1. Create a Pandas data frame for empdata.csv

```
import pandas as pd
df = pd.read_csv("/content/empdata.csv")
```

```
df.head()
```

	Empid	Ename	Salary	DOJ
0	1001	Ganesh	1000.00	10-10-2000
1	1002	Anil	23000.50	3/20/2002
2	1003	Gaurav	NaN	03-03-2002
3	1004	Hema Chandra	16500.50	09-10-2000
4	1005	Laxmi Prasanna	12000.75	10-08-2000

```
df.tail(2)
```

	Empid	Ename	Salary	DOJ
4	1005	Laxmi Prasanna	12000.75	10-08-2000
5	1006	Anant	9999.99	09-09-1999

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 4 columns):
#   Column  Non-Null Count  Dtype
---  -
0   Empid   6 non-null        int64
1   Ename   6 non-null        object
2   Salary  5 non-null        float64
3   DOJ     6 non-null        object
dtypes: float64(1), int64(1), object(2)
memory usage: 320.0+ bytes
```

```
df.describe()
```

	Empid	Salary
count	6.000000	5.000000
mean	1003.500000	12500.348000
std	1.870829	8139.622234

```
df.shape
```

```
(6, 4)
```

```
type(df)
```

```
pandas.core.frame.DataFrame
```

2. To retrieve column names

```
df.columns
```

```
Index(['Empid', 'Ename', 'Salary', 'DOJ'], dtype='object')
```

4. To retrieve column data

```
df.Empid
```

```
0    1001
1    1002
2    1003
3    1004
4    1005
5    1006
Name: Empid, dtype: int64
```

5. To retrieve a set of columns

```
df[['Empid', 'Ename']]
```

Empid **Ename**

Check for duplicates and remove them

```

4      1000      Aali
df1 = df.append(df)
print('Dimensions of the original frame', df.shape)
print('Dimensions of the frame with duplicates', df1.shape)
#remove the duplicates
df1 = df1.drop_duplicates() #or use this statement df1.drop_duplicates(inplace = True)
print('Dimensions of the frame after removing duplicates', df1.shape)

```

```

Dimensions of the original frame (6, 4)
Dimensions of the frame with duplicates (12, 4)
Dimensions of the frame after removing duplicates (6, 4)
<ipython-input-22-b62acb36e572>:1: FutureWarning: The frame.append method is deprecated
df1 = df.append(df)

```

```

#change all column names to Upper case
df.columns = [i.upper() for i in df]
print(df.columns)

```

```

Index(['EMPID', 'ENAME', 'SALARY', 'DOJ'], dtype='object')

```

Handling missing values

```

df.isna().sum()
#df.isnull().sum()

#print(df.isnull())
print('The no. of nulls in each column is \n',df.isnull().sum())
df.dropna(axis = 1, inplace = False)

```

The no. of nulls in each column is

```
EMPID      0
```

```
df.isna().sum()
```

```
EMPID      0
ENAME      0
SALARY      1
DOJ         0
dtype: int64
```

```
1    1002      Anil    3/20/2002
```

```
df
```

	EMPID	ENAME	SALARY	DOJ
0	1001	Ganesh	1000.00	10-10-2000
1	1002	Anil	23000.50	3/20/2002
2	1003	Gaurav	NaN	03-03-2002
3	1004	Hema Chandra	16500.50	09-10-2000
4	1005	Laxmi Prasanna	12000.75	10-08-2000
5	1006	Anant	9999.99	09-09-1999

6. Find the highest and lowest salary

```
print('Highest Salary is',df['SALARY'].max())
print('Lowest Salary is', df['SALARY'].min())
```

```
Highest Salary is 23000.5
Lowest Salary is 1000.0
```

```
# This is formatted as code
```

Display the details of employees whose salary is above 20000

```
df[df.SALARY > 20000]
```

	EMPID	ENAME	SALARY	DOJ
1	1002	Anil	23000.5	3/20/2002

Display only the id and names of employees whose salary is greater than 20000

```
df[['EMPID', 'ENAME']] [df.SALARY > 20000]
```

	EMPID	ENAME
1	1002	Anil

Display the Eid and name of the highest paid employee

```
df[['EMPID','ENAME']] [df.SALARY == df.SALARY.max()]
```

	EMPID	ENAME
1	1002	Anil

Display the enames whose salary is above the average salary

```
print('Average Salary is', df.SALARY.mean())
df['ENAME'][df.SALARY > df.SALARY.mean()]
```

```
Average Salary is 12500.348
1      Anil
3  Hema Chandra
Name: ENAME, dtype: object
```

Sort in ascending order of DOJ and store the result in another frame

```
df1['DOJ'] = pd.to_datetime(df1['DOJ'])    #convert DOJ to date type
df1.info()
print('Frame before sorting\n', df1)
df1.sort_values("DOJ", inplace = True)
print('Frame after sorting\n', df1)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6 entries, 0 to 5
Data columns (total 4 columns):
#   Column  Non-Null Count  Dtype
---  -
0   Empid   6 non-null         int64
1   Ename    6 non-null         object
2   Salary   5 non-null         float64
3   DOJ      6 non-null         datetime64[ns]
dtypes: datetime64[ns](1), float64(1), int64(1), object(1)
memory usage: 240.0+ bytes
Frame before sorting
   Empid      Ename      Salary      DOJ
0   1001      Ganesh    1000.00  2000-10-10
1   1002        Anil   23000.50  2002-03-20
2   1003      Gaurav      NaN  2002-03-03
3   1004  Hema Chandra  16500.50  2000-09-10
4   1005  Laxmi Prasanna  12000.75  2000-10-08
5   1006        Anant    9999.99  1999-09-09
Frame after sorting
   Empid      Ename      Salary      DOJ
```

```

5  1006      Anant      9999.99 1999-09-09
3  1004  Hema Chandra 16500.50 2000-09-10
4  1005  Laxmi Prasanna 12000.75 2000-10-08
0  1001      Ganesh      1000.00 2000-10-10
2  1003      Gaurav      NaN 2002-03-03
1  1002      Anil      23000.50 2002-03-20

```

Sort in descending order of dates

```
df1.sort_values("DOJ", ascending = False, inplace = True)
df1
```

	Empid	Ename	Salary	DOJ
1	1002	Anil	23000.50	2002-03-20
2	1003	Gaurav	NaN	2002-03-03
0	1001	Ganesh	1000.00	2000-10-10
4	1005	Laxmi Prasanna	12000.75	2000-10-08
3	1004	Hema Chandra	16500.50	2000-09-10
5	1006	Anant	9999.99	1999-09-09

Sort DOJ in descending and salary in ascending order

```
df1.sort_values(by = ['DOJ', 'Salary'], ascending = [False, True], inplace = True)
df1
```

	Empid	Ename	Salary	DOJ
1	1002	Anil	23000.50	2002-03-20
2	1003	Gaurav	NaN	2002-03-03
0	1001	Ganesh	1000.00	2000-10-10
4	1005	Laxmi Prasanna	12000.75	2000-10-08
3	1004	Hema Chandra	16500.50	2000-09-10
5	1006	Anant	9999.99	1999-09-09



LABSHEET-4

Import necessary libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing the dataset

```
df = pd.read_csv('/content/advertising.csv')
df.head(10)
df.info()
print(df.shape)
```

View descriptive statistics

```
print(df.describe())
```

Outlier Analysis

```
fig, axs = plt.subplots(3, figsize = (5,5))
plt1 = sns.boxplot(df['TV'], ax = axs[0])
plt2 = sns.boxplot(df['Newspaper'], ax = axs[1])
plt3 = sns.boxplot(df['Radio'], ax = axs[2])
plt.tight_layout()
```

Declare feature variable and target variable

```
X = df['TV']
y = df['Sales']
```

Plot scatter plot between X and y

```
plt.scatter(X, y, color = 'blue', label='Scatter Plot')
plt.title('Relationship between TV and Sales')
```

```
plt.xlabel('TV')
```

```
plt.ylabel('Sales')
```

```
plt.legend()
```

```
plt.show()
```

Print the dimensions of X and y

```
print(X.shape)
print(y.shape)
```

```
X=np.array(X)
y=np.array(y)
```

Reshape X and y

```
X = X.reshape(-1,1)
```

```

y = y.reshape(-1,1)

# Print the dimensions of X and y after reshaping

print(X.shape)
print(y.shape)

# Split X and y into training and test data sets
#random_state--the set of data does not change

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size=0.30,
random_state=42)

# Print the dimensions of X_train,X_test,y_train,y_test
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)

# Fit the linear model

# Instantiate the linear regression object lm
from sklearn.linear_model import LinearRegression
lm = LinearRegression()

# Train the model using training data sets
lm.fit(X_train,y_train)

# Predict on the test data
y_pred=lm.predict(X_test)

# Visualising the Training set results
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, lm.predict(X_train), color = 'blue')

# Visualising the Test set results
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_test, lm.predict(X_test), color = 'blue')
plt.title('Test set results')
plt.xlabel('TV')
plt.ylabel('Sales')
plt.show()

# Compute model slope and intercept

slope = lm.coef_
intercept = lm.intercept_,

```

```
print("Estimated model slope:" , slope)
print("Estimated model intercept:" , intercept)

X_new = [[200]]

lm.predict(X_new)
```