Learning Numpy : NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy stands for Numerical Python To work with numpy we have to import numpy package.

In [1]:
```
import numpy as np
```

In [2]:
```
np.version.version
```

Out[2]:
```
'1.20.3'
```

In [3]:
```
mylist=[1,2,3]
mylist
```

Out[3]:
```
[1, 2, 3]
```

array method :A Python Array is a collection of common type of data structures having elements with same data type.

In [4]:
```
#array

np.array(mylist)   # 1D one-dimensional array to perform mathematical operations.
```

Out[4]:
```
array([1, 2, 3])
```

In [5]:
```
# will 'matrices' i.e. two dimensional which was having colums and rows.
```

In [6]:
```
my_matrices=[[1,3],[4,6]]
my_matrices
```

Out[6]:
```
[[1, 3], [4, 6]]
```

In [7]:
```
np.array(my_matrices) # 2D array
```

Out[7]:
```
array([[1, 3],
       [4, 6]])
```

In [8]:
```
# methods of numpy
```

In [9]:
```
#arrange  ( start - stop - step )

np.arange (0,10)  # IT WILL GENERATE number from 0 to 9 excluding last no. i.e.10-1 =1 it's like range
```

Out[9]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [10]:
```
np.arange (20,25)   # started from 20 and stop at 24 there is no step
```

Out[10]:
```
array([20, 21, 22, 23, 24])
```

In [11]:
```
np.arange (0,12,2)          # here, star value is'0', stop value'12' and step size value'2' ,
                            #it will add +2 like 0+2=2,2+2=4,2+4=6 ...excluding stop value i.e.'12'
```

Out[11]:
```
array([ 0,  2,  4,  6,  8, 10])
```

In [12]:
```
np.arange(0,12,3)
```

Out[12]:
```
array([0, 3, 6, 9])
```

In [13]:
```
np.arange(7,12,3)
```

Out[13]:
```
array([ 7, 10])
```

In [14]:
```
np.arange(30,15,-3)   # start'30'-stop'15'-step'-3'  i.e. 30-3=27,27-3=24,24-3=21 ....
```

Out[14]:
```
array([30, 27, 24, 21, 18])
```

zeros and one method- The numpy.zeros() function returns a new array of given shape and type, with zeros.

the use of this method,when we do the vector operations.

In [15]:
```
np.zeros(4)
```

Out[15]:
```
array([0., 0., 0., 0.])
```

In [16]:
```
np.zeros((5,5))  # it will generates 5 nos of columns and 5 nos rows
```

Out[16]:
```
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., 0.]])
```

In [17]:
```
np.ones ((3,3))    # Unit vector initialize with '1'
```

Out[17]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

Linspace method: The linspace() function returns evenly spaced numbers over a specified interval [start, stop]. The endpoint of the interval can optionally be excluded.

In [18]:
```
np.linspace(0,10,3)  # it returns evenly spaced numbers,it  generate 3 interval values between 0 to 10 so that distance btween each is equal
```

Out[18]:
```
array([ 0.,  5., 10.])
```

In [19]:
```
np.linspace (0,12,30)  # it  generate 30 intervals values between 0 to 12 so that distance btween each is equal
```

Out[19]:
```
array([ 0.        ,  0.4137931 ,  0.82758621,  1.24137931,  1.65517241,
        2.06896552,  2.48275862,  2.89655172,  3.31034483,  3.72413793,
        4.13793103,  4.55172414,  4.96551724,  5.37931034,  5.79310345,
        6.20689655,  6.62068966,  7.03448276,  7.44827586,  7.86206897,
        8.27586207,  8.68965517,  9.10344828,  9.51724138,  9.93103448,
       10.34482759, 10.75862069, 11.17241379, 11.5862069 , 12.        ])
```

In [20]:
```
np.linspace (0,27,3)
```

Out[20]:
```
array([ 0. , 13.5, 27. ])
```

An identity matrix : is a square matrix in which the elements of the main diagonal are equal to one and the other elements equal to zero. Return a 2-D array with ones on the diagonal and zero elsewhere.

In [21]:
```
np.eye(3)
```

```
array([[1., 0., 0.],
```

Out[21]:

```
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [22]:

```
np.eye(9)
```

Out[22]:

```
array([[1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1.]])
```

Random = The random() method returns a random floating number between 0 and 1. The numpy.random.rand() function creates an array of specified shape and fills it with random values

In [23]:

```
np.random.rand(4)        # from uniform distribution , values always between 0 to 1
```

Out[23]:

```
array([0.81455823, 0.40096973, 0.01232438, 0.00709984])
```

In [24]:

```
np.random.rand(4,4)
```

Out[24]:

```
array([[0.62083955, 0.37746846, 0.70855979, 0.05354033],
       [0.22244406, 0.21905723, 0.62513075, 0.96985507],
       [0.06305229, 0.66285974, 0.7874476 , 0.11088002],
       [0.15745661, 0.03893476, 0.59755597, 0.55882512]])
```

The numpy. random. randn() function= it creates an array of specified shape and fills it with random values as per standard normal distribution

In [25]:

```
np.random.randn(4)                    # from std.normal distribution
```

Out[25]:

```
array([ 1.89138406, -0.56599777,  0.38353166, -1.74501793])
```

In [26]:

```
np.random.randn(3,2)
```

Out[26]:

```
array([[ 0.66112592,  1.70638371],
       [-0.72215241, -0.01777903],
       [ 0.23983795,  0.09049794]])
```

numpy.random.randint()= is one of the function for doing random sampling in numpy. It returns an array of specified shape and fills it with random integers from low (inclusive) to high (exclusive), i.e. in the interval [low, high).

In [27]:

```
np.random.randint(1,50,3)   # it will generate value in integer without decimel
                            # everytime will get different numbers
```

Out[27]:

```
array([46, 36, 47])
```

Reshaping= What is reshaping in Python? Reshaping means changing the shape of an array. The shape of an array is the number of elements in each dimension. By reshaping we can add or remove dimensions or change number of elements in each dimension.

In [28]:

```
np.arange(25)
```

Out[28]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24])
```

In [29]:

```
# will assign this array with 'arr'
```

In [30]:

```
arr=np.arange(25)  # we have created 1-D array now will convert to 2-D with 'reshap' function
arr
```

Out[30]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24])
```

In [31]:

```
a=arr.reshape (5,5)
a                  # converted from 1D to 2D
```

Out[31]:

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

In [32]:

```
arr1=np.arange(12)
arr1
```

Out[32]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

In [33]:

```
arr1.reshape (4,3)
```

Out[33]:

```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

In [34]:

```
a.shape   # will check the shape
```

Out[34]:

```
(5, 5)
```

min function=The min() function returns the item with the lowest value, or the item with the lowest value in an iterable.

In [35]:

```
mimxarr=np.random.randint(0,50,20)                # will find out minimum no. out of this array
mimxarr
```

Out[35]:

```
array([ 8, 15, 39, 20, 23,  5, 40,  0, 33,  4, 33,  3, 13, 24, 31,  5, 27,
       35, 15, 20])
```

In [36]:

```
mimxarr.min()          # got minimum no. in array
```

Out[36]:

```
0
```

In [37]:

```
mimxarr.max()          # got maximum no. in array
```

Out[37]:

```
40
```

Numpy argmin & arhmax = is a function in python which returns the index of the minimum/maximum element from a given array along the given axis.

In [38]:

```
mimxarr.argmin()  # index no. 12 having minimum value in mimxarr i.e.3 ( in numpy index start from '0')
```

Out[38]:

```
7
```

In [39]:

```
mimxarr.argmax()  # index no. 2 having minimum value in mimxarr i.e.47 ( in numpy index start from '0')
```

Out[39]:

```
6
```

In [40]:

```
mimxarr[7]     # we can get specific no value in array by enterring index no.like here we got value 4 on index no.7
```

Out[40]:

```
0
```

In [41]:
```
mimxarr[1:7]       # here slicing also works, here value start from 1 and end to 6th excluding 7th value
```
Out[41]:
```
array([15, 39, 20, 23,  5, 40])
```
In [42]:
```
mimxarr[:]     # will get all the values
```
Out[42]:
```
array([ 8, 15, 39, 20, 23,  5, 40,  0, 33,  4, 33,  3, 13, 24, 31,  5, 27,
       35, 15, 20])
```
In [43]:
```
# will make sub-set from mixarr array
```
In [44]:
```
new_mimx= mimxarr[1:7]
new_mimx
```
Out[44]:
```
array([15, 39, 20, 23,  5, 40])
```
In [45]:
```
new_mimx[2:5]=15        # will update values in 'mimxarr'
new_mimx
```
Out[45]:
```
array([15, 39, 15, 15, 15, 40])
```
In [46]:
```
new_mimx
```
Out[46]:
```
array([15, 39, 15, 15, 15, 40])
```
In [47]:
```
b=mimxarr.copy()     # we have copied same array with new name to protect original array ,
                     #so that update won't affect the original array
b
```
Out[47]:
```
array([ 8, 15, 39, 15, 15, 15, 40,  0, 33,  4, 33,  3, 13, 24, 31,  5, 27,
       35, 15, 20])
```
two dimensional metrics 2D

In [48]:
```
array_2d=np.array([[1,2,3],[6,7,8],[9,10,11]])
```
In [49]:
```
array_2d
```
Out[49]:
```
array([[ 1,  2,  3],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```
In [50]:
```
array_2d[1]    # we have indexing row no.1 ,index start from '0' zero
```
Out[50]:
```
array([6, 7, 8])
```
In [51]:
```
array_2d [0,2]   # single bracket, will get data from 1st row  (format=row,column)
```
Out[51]:
```
3
```
In [52]:
```
array_2d [[0,2]]   # double bracket, will get data from  row wise  ( row index,row column)
```
Out[52]:
```
array([[ 1,  2,  3],
       [ 9, 10, 11]])
```
In [53]:
```
array_2d [:2,1:]    # pick row (0,1),pick column from 1 till end
```
Out[53]:
```
array([[2, 3],
       [7, 8]])
```
In [54]:
```
array_2d [2:]    # 2nd row ,all columns
```
Out[54]:
```
array([[ 9, 10, 11]])
```
What is a boolean in Python? The python data type bool is used to store two values i.e True and False . Bool is used to test whether the result of an expression is true or false.

In [55]:
```
array_2d
```
Out[55]:
```
array([[ 1,  2,  3],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```
In [56]:
```
array_2d >4
```
Out[56]:
```
array([[False, False, False],
       [ True,  True,  True],
       [ True,  True,  True]])
```
In [57]:
```
array_2d[array_2d >4 ]      # by this condition will get all true value gets highlight
```
Out[57]:
```
array([ 6,  7,  8,  9, 10, 11])
```
logical operator like <,>,+,-,/,*

In [58]:
```
array_2d + array_2d          # addition taken place
```
Out[58]:
```
array([[ 2,  4,  6],
       [12, 14, 16],
       [18, 20, 22]])
```
In [59]:
```
array_2d * array_2d        # multiplication
```
Out[59]:
```
array([[  1,   4,   9],
       [ 36,  49,  64],
       [ 81, 100, 121]])
```
matmul function

In [60]:
```
a=np.array([[1,2],[3,4]])
a
```
Out[60]:
```
array([[1, 2],
       [3, 4]])
```
In [61]:
```
b=np.array([5,6])
b
```
Out[61]:
```
array([5, 6])
```
In [62]:
```
np.matmul(b,a)
```
Out[62]:
```
array([23, 34])
```
In [63]:
```
list(b)          # conversion to list
```

Out[63]:
```
[5, 6]
```

In [65]:
```
b.tolist()         # another method to convert to list
```

Out[65]:
```
[5, 6]
```

In [64]:
```
tuple(b)           # conversion to tuple
```

Out[64]:
```
(5, 6)
```

PANDAS LEARNING -it's apython library -it use to analyse data

In [66]:
```
import pandas as pd
```

In [69]:
```
pd.__version__
```

Out[69]:
```
'1.3.4'
```

-Pandas Series = is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.).

In [71]:
```
arr=np.array([38,12,90])
arr
```

Out[71]:
```
array([38, 12, 90])
```

In [72]:
```
pd.Series(arr)
```

Out[72]:
```
0    38
1    12
2    90
dtype: int32
```

What is a pandas label? Image result for label in pandas series Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.).

In [73]:
```
labels=['a','b','c']   # instead of 0,1,2 will change as a,b,c
```

In [74]:
```
pd.Series(arr,index=labels)
```

Out[74]:
```
a    38
b    12
c    90
dtype: int32
```

converting from dictionery to series

In [76]:
```
dict_one={'mom':76590,'dad':125078,'ram':9833455}
```

In [77]:
```
pd.Series(dict_one)
```

Out[77]:
```
mom      76590
dad     125078
ram    9833455
dtype: int64
```

will make our own sereis

In [80]:
```
series_1=pd.Series([5,99,577,900],index=['a',"b","c","d"])
series_1
```

Out[80]:
```
a      5
b     99
c    577
d    900
dtype: int64
```

In [81]:
```
series_1['c']    #will access data from
```

Out[81]:
```
577
```

In [82]:
```
series_1[3]  # with index no
```

Out[82]:
```
900
```

A Pandas DataFrame : is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

In [83]:
```
from numpy.random import randn
```

In [84]:
```
pd.DataFrame(randn (6,5),index=['one','two',"three",'four','five','six'])
```

Out[84]:

	0	1	2	3	4
one	0.353448	1.864996	-0.109023	0.973313	-0.452548
two	0.436256	-0.007463	0.467563	0.778698	0.182455
three	-0.245694	-0.182040	-0.161970	-1.248833	-0.033882
four	0.586884	-0.052189	1.108726	-0.826369	-0.584759
five	-0.357588	-1.190836	1.088814	0.433938	0.641741
six	-0.871334	1.855543	-1.018522	-1.332651	0.714253

In [110…
```
df=pd.DataFrame(randn (6,5),index=['one','two',"three",'four','five','six'],columns=['col1','col2',"col3",'col4','col5'])   # given other name to column instead of 1,2,3...
```

In [111…
```
pd.DataFrame(randn (6,5),index=['one','two',"three",'four','five','six'],columns=['col1','col2',"col3",'col4','col5'])   # given other name to column instead of 1,2,3...   # we have define 'df' name to tabl
df
```

Out[111…

	col1	col2	col3	col4	col5
one	1.767400	1.510039	-0.338397	1.628982	-0.484014
two	-1.642004	1.384606	0.186745	-0.906160	1.467255
three	-1.603896	-0.954926	-0.644413	-1.458476	-0.802828
four	1.663760	0.330509	-0.812248	0.825199	0.657622
five	0.229100	-0.952496	-0.574527	0.605673	-1.566942
six	0.565124	1.817735	-0.141439	-0.375414	0.019721

In [ ]:
```
# will access multiple columns
```

In [112…
```
df[['col1','col5']]
```

Out[112…

	col1	col5
one	1.767400	-0.484014
two	-1.642004	1.467255
three	-1.603896	-0.802828
four	1.663760	0.657622
five	0.229100	-1.566942
six	0.565124	0.019721

In [113…
```
df[['col2','col4']]
```

Out[113…

	col2	col4
one	1.510039	1.628982
two	1.384606	-0.906160
three	-0.954926	-1.458476
four	0.330509	0.825199
five	-0.952496	0.605673
six	1.817735	-0.375414

In [ ]:
```
# will check data type
```

In [114…
```
type(df.col1)
```

Out[114…
```
pandas.core.series.Series
```

In [ ]:
```
# will check type of data for multi columns
```

In [115…
```
type(df[['col1','col5']])
```

Out[115…
```
pandas.core.frame.DataFrame
```

In [ ]:
```
# add new column to 'df'
```

In [116…
```
df['col6']=df['col2']+ df['col4']
```

In [117…
```
df
```

Out[117…

	col1	col2	col3	col4	col5	col6
one	1.767400	1.510039	-0.338397	1.628982	-0.484014	3.139021
two	-1.642004	1.384606	0.186745	-0.906160	1.467255	0.478446
three	-1.603896	-0.954926	-0.644413	-1.458476	-0.802828	-2.413403
four	1.663760	0.330509	-0.812248	0.825199	0.657622	1.155708
five	0.229100	-0.952496	-0.574527	0.605673	-1.566942	-0.346824
six	0.565124	1.817735	-0.141439	-0.375414	0.019721	1.442322

In [ ]:
```
# how to delete specific columns
```

In [118…
```
df.drop('col6',axis=1)  # it will not delete permanently
```

Out[118…

	col1	col2	col3	col4	col5
one	1.767400	1.510039	-0.338397	1.628982	-0.484014
two	-1.642004	1.384606	0.186745	-0.906160	1.467255
three	-1.603896	-0.954926	-0.644413	-1.458476	-0.802828
four	1.663760	0.330509	-0.812248	0.825199	0.657622
five	0.229100	-0.952496	-0.574527	0.605673	-1.566942
six	0.565124	1.817735	-0.141439	-0.375414	0.019721

In [119…
```
df
```

Out[119…

	col1	col2	col3	col4	col5	col6
one	1.767400	1.510039	-0.338397	1.628982	-0.484014	3.139021
two	-1.642004	1.384606	0.186745	-0.906160	1.467255	0.478446
three	-1.603896	-0.954926	-0.644413	-1.458476	-0.802828	-2.413403
four	1.663760	0.330509	-0.812248	0.825199	0.657622	1.155708
five	0.229100	-0.952496	-0.574527	0.605673	-1.566942	-0.346824
six	0.565124	1.817735	-0.141439	-0.375414	0.019721	1.442322

In [ ]:
```
# to delete any column permanently will use'inplace=true'   NOTE: axis'0'= row  and  axis'1'=columns
```

In [120…
```
df.drop('col6',axis=1,inplace=True)
```

In [121…
```
df
```

Out[121…

	col1	col2	col3	col4	col5
one	1.767400	1.510039	-0.338397	1.628982	-0.484014
two	-1.642004	1.384606	0.186745	-0.906160	1.467255
three	-1.603896	-0.954926	-0.644413	-1.458476	-0.802828
four	1.663760	0.330509	-0.812248	0.825199	0.657622
five	0.229100	-0.952496	-0.574527	0.605673	-1.566942
six	0.565124	1.817735	-0.141439	-0.375414	0.019721

In [ ]:
```
# delete rows
```

In [123…
```
df.drop (['two','six'],axis=0)
```

Out[123…

	col1	col2	col3	col4	col5
one	1.767400	1.510039	-0.338397	1.628982	-0.484014
three	-1.603896	-0.954926	-0.644413	-1.458476	-0.802828
four	1.663760	0.330509	-0.812248	0.825199	0.657622
five	0.229100	-0.952496	-0.574527	0.605673	-1.566942

In [124…

```
df    # row no. 'two and six' gto deleted       # not permanently
```

Out[124…

	col1	col2	col3	col4	col5
one	1.767400	1.510039	-0.338397	1.628982	-0.484014
two	-1.642004	1.384606	0.186745	-0.906160	1.467255
three	-1.603896	-0.954926	-0.644413	-1.458476	-0.802828
four	1.663760	0.330509	-0.812248	0.825199	0.657622
five	0.229100	-0.952496	-0.574527	0.605673	-1.566942
six	0.565124	1.817735	-0.141439	-0.375414	0.019721

In [125…

```
df.drop (['two','six'],axis=0,inplace=True)  # row got permanently deleted
```

In [126…

```
df
```

Out[126…

	col1	col2	col3	col4	col5
one	1.767400	1.510039	-0.338397	1.628982	-0.484014
three	-1.603896	-0.954926	-0.644413	-1.458476	-0.802828
four	1.663760	0.330509	-0.812248	0.825199	0.657622
five	0.229100	-0.952496	-0.574527	0.605673	-1.566942

In [ ]:

```
# access rows by loc and iloc function
```

In [128…

```
df.loc['one']    # directly by repsetive column name
```

Out[128…

```
col1    1.767400
col2    1.510039
col3   -0.338397
col4    1.628982
col5   -0.484014
Name: one, dtype: float64
```

In [131…

```
df.iloc[1]    # with repsetive index no
```

Out[131…

```
col1   -1.603896
col2   -0.954926
col3   -0.644413
col4   -1.458476
col5   -0.802828
Name: three, dtype: float64
```

In [132…

```
df.iloc[2]
```

Out[132…

```
col1    1.663760
col2    0.330509
col3   -0.812248
col4    0.825199
col5    0.657622
Name: four, dtype: float64
```

In [134…

```
df.loc['one','col2']
```

Out[134…

```
1.5100393910717658
```

In [135…

```
df.loc['one','col5']
```

Out[135…

```
-0.48401411617236584
```

In [137…

```
df.loc[['one','four'],['col2','col4']]
```

Out[137…

	col2	col4
one	1.510039	1.628982
four	0.330509	0.825199