In [ ]:

```
#agenda of today:
                       1. Data science Packages/libraries
                            (A) Numpy   - for Scientific Computing
                            (B)  Pandas - Data Analysis, Data Cleaning
                            (C) Matplotlib - Data Representation in the form of 2D Graphics.
```

In [ ]:

```
#Numpy-      Num+py = Numrical Python

Numpy is an open source library available and predefined library package used for scientific computing.
    Mainly Deals
               (i)   mathematical
               (ii)  Scientific
               (iii) Engineering
                (iv) Data Science programming.
                (v)  statical operations
                (vi) mutli-dimesinal arrays and matrices multiplication.
        #Author : Travis Oliphant
        #First Release : 1995
        #writeen in - Python Programming,C

#NOTE:
   Its Contains a powerful n-dimensional array object.
   Its also used in
                           1. linear algebra
                           2. random number capability

#what is numpy arrays?
     - Numpy array is powerful N-dimensional array object which is in the form of rows and columns.
     - Which can be initialize Numpy arrays from nested python lists
```

In [ ]:

```
#How to install numpy packages?
pip install numpy
```

In [6]:

```
#How to use numpy : (single dimensional Array)
import numpy as np
a=np.array([1,2,3])
a.shape
```

Out[6]:

```
(3,)
```

In [12]:

```
#Multi Dimensional Array:
import numpy as np
b=np.array([(5,6,7),(7,8,5)])
b.shape
```

Out[12]:

```
(2, 3)
```

In [16]:

```
#Numpy array attributes:
a1 = np.array([1,2,4,5])
print(a1)
print(a1.dtype)
a2 = np.array([1.6,7.8,8.9,3.4])
print(a2)
print(a2.dtype)
```

```
[1 2 4 5]
int32
[1.6 7.8 8.9 3.4]
float64
```

In [23]:

```
#shape of array:
import numpy as np
ar1 = np.array([1,2,3,4,5,6])              #1-d array
ar2 = np.array([[1,2,3],[4,5,6]])                       #2-d array
ar3 = np.array([[[1,3,5],[5,6,7]],[[5,6,7],[6,7,8]]])    #3-d array
print(ar1)
print(ar1.shape)
print(ar2)
print(ar2.shape)
print(ar3)
print(ar3.shape)
```

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

 [[5 6 7]
  [6 7 8]]]
(2, 2, 3)
```

In [28]:

```
#dimesion of an array:
#ndim():
import numpy as np
a1 = np.array([1,2,3,4,5])
print(a1.ndim)
a2 = np.array([[1,3,4],[6,7,9]])
print(a2.ndim)
a3 = np.array([[[1,3,5],[5,6,7]],[[5,6,7],[6,7,8]]])
print(a3.ndim)
```

```
1
2
3
```

In [44]:

```
#reshape of array:
import numpy as np
n1 = np.array([1,2,3,4,5,6,67,77])
print(n1.ndim)
print(n1.shape)
print("before reshape =",n1)
n2=n1.reshape(2,4)
print("after reshape =",n2)
print(n2.shape)
print(n2.ndim)
```

```
1
(8,)
before reshape = [ 1  2  3  4  5  6 67 77]
after reshape = [[ 1  2  3  4]
 [ 5  6 67 77]]
(2, 4)
2
```

In [52]:

```
#resize an array:
#resize() method modifies exiting shape and array itself.
import numpy as np
n1 = np.array([3,55,5,23,4,5,66,6])
print(n1)
n1.resize(3)
print(n1)
n1.resize(2,4)
print(n1)
n1.resize(3,3)
print(n1)
```

```
[ 3 55  5 23  4  5 66  6]
[ 3 55  5]
[[ 3 55  5  0]
 [ 0  0  0  0]]
[[ 3 55  5]
 [ 0  0  0]
 [ 0  0  0]]
```

In [64]:

```
#special functions for Numpy arrays generation:
#arange(): creates an array with specified spaced values b/w the start,end,internal values
#syntax:   arange(start,stop,intenal)
import numpy as np
a1 = np.arange(10)
print(a1)
a2 = np.arange(0,30,3)
print(a2)
print(a2.shape)
a3 = np.arange(9).reshape(3,3)
print(a3)
```

```
[0 1 2 3 4 5 6 7 8 9]
[ 0  3  6  9 12 15 18 21 24 27]
(10,)
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

In [68]:

```
#arrays with linspace():  functions generates an array with evenly spaced values b/w start,stop,internal values
import numpy as np
n1 = np.linspace(1,12,2)
print(n1)
n2 = np.linspace(1,12,4)
print(n2)
n3 = np.linspace(1,12,12).reshape(4,3)
print(n3)
```

```
[ 1. 12.]
[ 1.          4.66666667  8.33333333 12.        ]
[[ 1.  2.  3.]
 [ 4.  5.  6.]
 [ 7.  8.  9.]
 [10. 11. 12.]]
```

In [82]:

```
#Zero array:
import numpy as np
a1 = np.zeros(3)
a2 = np.zeros((2,4),dtype="int64")
a3 = np.zeros((3,6),dtype="int32")
print(a1)
print(a1.dtype)
print(a2)
print(a2.dtype)
print(a3.dtype)
```

```
[0. 0. 0.]
float64
[[0 0 0 0]
 [0 0 0 0]]
int64
int32
```

In [86]:

```
#one array: ones()
import numpy as np
np.ones(3,dtype="int32")
np.ones((6,5),dtype="int64")
```

Out[86]:

```
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, 1],
       [1, 1, 1, 1, 1]], dtype=int64)
```

In [92]:

```
#full array:  full(dimension,speciled number)
import numpy as np
a1 = np.full((3),100)
a2=np.full((2,5),99999)
print(a1)
print(a2)
```

```
[100 100 100]
[[99999 99999 99999 99999 99999]
 [99999 99999 99999 99999 99999]]
```

In [106]:

```
#eye array:  RETURNS an array with where all elements are equal to zero,expect
#           for the kth diagonal whose values are equal to one.
import numpy as np
a1 = np.eye(3,dtype="int32")
a2 = np.eye(5,k=0)
a2
```

Out[106]:

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

In [121]:

```
#randow number array:
#np.random.rand  (its generates uniformly distributed b/w 0 and 1)
#np.random.randn ( its generates normally distributed blw 0 an 1)
#np.random.randint (ist geneates uniformly distributed b/w 0 and given number)

import numpy as np
print(np.random.rand(3,2))
print(np.random.randn(3,2))
print(np.random.randint(15, size = (2,4)))
```

```
[[0.78518905 0.63476529]
 [0.62808953 0.24533248]
 [0.61474615 0.65761329]]
[[-1.12262877  0.37827565]
 [-0.13125665 -0.2953823 ]
 [-0.31092068 -0.05679359]]
[[14  1  3  6]
 [ 2  3  2  9]]
```

In [141]:

```
#Operations on numpy arrays:
#indexing:
import numpy as np
a1 = np.array([1,2,34,5,6,90])
a1[0]
a1[-1]
a1[3]
a2= np.array([[50,60,"python"],[80.9,90.5,"apssdc"],[45.6,"Abc",500]])
print(a2[1,2])
print(a2[0:,1])
print(a2[0:3,0])
print(a2[1:3,2])
```

```
apssdc
['60' '90.5' 'Abc']
['50' '80.9' '45.6']
['apssdc' '500']
```

In [146]:

```
#Joining and Stacking:
import numpy as np
a1 = np.array([[1,2,3],[6,7,9]])
a2 = np.array([[6,2,4],[8,9,3]])
a3 = np.hstack((a1,a2))   #Horizontal stacking
print(a3)
a4 = np.vstack((a1,a2))  #Vertical Stacking
print(a4)
a5= np.append(a1,a2,axis=0)
print(a5)
a6 = np.append(a1,a2,axis=1)
print(a6)
```

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

In [155]:

```
#Arithimatic operations:
import numpy as np
a1 = np.array([[4,5,6],[90,4,5]])
a2 = np.array([[56,7,7],[34,5,6]])
print(a1+a2)
print(a1-a2)
print(a1*a2)
print(a1/a2)
print(a1 ** a2)
#scalar arthimatic operations
print(a1+5)
print(a2-3)
print(a1*50)
```

```
[[ 60  12  13]
 [124   9  11]]
[[-52  -2  -1]
 [ 56  -1  -1]]
[[ 224   35   42]
 [3060   20   30]]
[[0.07142857 0.71428571 0.85714286]
 [2.64705882 0.8        0.83333333]]
[[     0  78125 279936]
 [     0   1024  15625]]
[[ 9 10 11]
 [95  9 10]]
[[53  4  4]
 [31  2  3]]
[[ 200  250  300]
 [4500  200  250]]
```

In [158]:

```
#Mathematical functions:
import numpy as np
a1 = np.array([[10,20,30],[60,40,90]])
print(np.sin(a1))
print(np.cos(a1))
print(np.tan(a1))
print(np.sqrt(a1))
print(np.exp(a1))
print(np.log10(a1))
```

```
[[-0.54402111  0.91294525 -0.98803162]
 [-0.30481062  0.74511316  0.89399666]]
[[-0.83907153  0.40808206  0.15425145]
 [-0.95241298 -0.66693806 -0.44807362]]
[[ 0.64836083  2.23716094 -6.4053312 ]
 [ 0.32004039 -1.11721493 -1.99520041]]
[[3.16227766 4.47213595 5.47722558]
 [7.74596669 6.32455532 9.48683298]]
[[2.20264658e+04 4.85165195e+08 1.06864746e+13]
 [1.14200739e+26 2.35385267e+17 1.22040329e+39]]
[[1.         1.30103    1.47712125]
 [1.77815125 1.60205999 1.95424251]]
```

In [162]:

```
#Matrix Transpose using:
import numpy as np
a1 = np.array([[1,2,3],[4,5,6]])
print("original array :",a1)
aT = a1.transpose()
print("Transpose array: ",aT)
```

```
original array : [[1 2 3]
 [4 5 6]]
Transpose array:  [[1 4]
 [2 5]
 [3 6]]
```

In [ ]: