Numpy :Numpy is a library or package Numpy contains the buit-in methods or function combined with other packages,pandas,scikit,metplotlib,Tenserflow etc. Numpy can perform statistical data analysis,matrix manipulations,linear algebara.etc. Numpy is used in the applications of data scientist.

Numpy provides multidimensional data in the form of arrays

Arrays are the collection of homogeneous data.

numpy array is faster as compare to python list

python list contains different type of data but numpy array contains similar type of data

installing numpy

In [1]:
```
pip install numpy
```
```
Requirement already satisfied: numpy in c:\users\sumit\anaconda3\lib\site-packages (1.16.2)
Note: you may need to restart the kernel to use updated packages.
```

using numpy

In [2]:
```
import numpy as np
```

In [3]:
```
a=np.array([1,2,3,4,5,6]) # passing list
```

In [5]:
```
print(a)
```
```
[1 2 3 4 5 6]
```

In [6]:
```
b=np.array((1,2,3,4,5))# passing tuple
```

In [7]:
```
print(b)
```
```
[1 2 3 4 5]
```

In [8]:
```
c=np.array({1,2,3,4,5})# passing set
```

In [9]:
```
print(c)
```
```
{1, 2, 3, 4, 5}
```

In [10]:
```
print(type(a)) # a belongs to ndarray class(N dimension array(max32))
```
```
<class 'numpy.ndarray'>
```

In [11]:
```
# multidemensional array
#1-D array---vectors
#2-D array----matrix or tabular form
#3-D or higher dimensions-----tensor
```

In [12]:
```
#2-D array is the collection of 1-D arrays
b=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(b)
```
```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [13]:
```
# 3-D array is the collection of 2-D arrays
c=np.array([
    [
        [1,2,3],[4,5,6]
    ],
    [
        [7,8,9],[10,11,12]
    ],
    [
        [11,12,13],[14,15,16]
    ]
])
```

In [14]:
```
print(c)
```
```
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]

 [[11 12 13]
  [14 15 16]]]
```

In [1]:
```
# using attributes in ndarray object
# by using attributes we can get some iformation about object
#1.ndim-provides dimensions of ndarray object
#2.size-provides no of element stored in ndarray object
#3.shape-provide order of ndarray object
#4.dtype- provides data type of elements
#5itemsize-provide size of each item in the array
#6.nbytes-provide totaol no of bytes occupied by ndarray object
```

In [2]:
```
import numpy as np
a=np.array([1,2,3,4,5,6])
b=np.array([[1,2,3],[4,5,6],[7,8,9]])
c=np.array([
    [
        [1,2,3],[4,5,6]
    ],
    [
        [7,8,9],[10,11,12]
    ],
    [
        [11,12,13],[14,15,16]
    ]
])
```

In [3]:
```
print(a)
print(b)
print(c)
```
```
[1 2 3 4 5 6]
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]

 [[11 12 13]
  [14 15 16]]]
```

In [6]:
```
print(a.ndim)
print(b.ndim)
print(c.ndim)
```
```
1
2
3
```

In [9]:
```
#using size
print(a.size)
print(b.size)
print(c.size)
```
```
6
9
18
```

In [12]:
```
#using shape
print(a.shape)
print(b.shape)#(output is (r,c) where r is no of rows and  c is no of cols)
print(c.shape)#(output is (r,c,t) where r is no 2d array,c is no 1d array and t is no element in 1-d ar
ray)
```
```
(6,)
(3, 3)
(3, 2, 3)
```

In [13]:
```
print(c)
```
```
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]

 [[11 12 13]
  [14 15 16]]]
```

In [16]:
```
print(a.dtype)# int32 means memory occupied is 32 bits
print(b.dtype)# int32
print(c.dtype)
```
```
int32
int32
int32
```

In [17]:
```
print(a.itemsize) # returns memeory occupied by item in bytes
```
```
4
```

In [19]:
```
print(b.itemsize)
print(c.itemsize)
```
```
4
4
```

In [21]:
```
print(a.size*a.itemsize) # total no of bytes occupied by ndarrya object
print(b.size*b.itemsize)
print(c.size*c.itemsize)
```
```
24
36
72
```

In [24]:
```
#nbytes
print(a.nbytes)
print(b.nbytes)
print(c.nbytes)
```
```
24
36
72
```

In [25]:
```
# dtype:data type
a=np.array([1,2,3,4,5,6],dtype='int64')
```

In [26]:
```
print(a)
```
```
[1 2 3 4 5 6]
```

In [27]:
```
print(a.dtype)
```
```
int64
```

In [28]:
```
print(a.itemsize)
```
```
8
```

In [29]:
```
a=np.array([1,2,3,4,5.0,6.3])# will ot give error but data wil be homogenous
```

In [30]:
```
print(a.dtype)
```
```
float64
```

In [31]:
```
print(a)
```
```
[1.  2.  3.  4.  5.  6.3]
```

In [32]:
```
b=a.astype(int) # return new array with the given data type
```

In [33]:
```
print(b)
```
```
[1 2 3 4 5 6]
```

In [34]:
```
print(b.dtype)
```
```
int32
```

In [35]:
```
print(a.dtype)
```
```
float64
```

access/change the elements in the array

In [39]:
```
print(b[1])
print(b[3])
```
```
2
4
```

In [40]:
```
print(b[-2])
```
```
5
```

In [41]:
```
b[1]=23
print(b)
```
```
[ 1 23  3  4  5  6]
```

In [50]:
```
b[[1,5]] # mutiple indexing in 1-d array
```
Out[50]:
```
array([23,  6])
```

In [43]:
```
x=np.array([[1,2,3,4],[5,6,7,8]])
print(x)
```
```
[[1 2 3 4]
 [5 6 7 8]]
```

In [44]:
```
x[0][1]
```
Out[44]:
```
2
```

In [47]:
```
print(x[0,1])
print(x[-1,-3])
```
```
2
6
```

In [51]:
```
x[(0,1),(1,2)] # multiple indexing in 2-d array
```
Out[51]:
```
array([2, 7])
```

In [52]:
```
x=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(x)
```
```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [53]:
```
x[(0,1,2),(0,1,2)] # diagonal elemnet
```
Out[53]:
```
array([1, 5, 9])
```

slicing in the array

In [1]:
```
import numpy as np
a=np.array([1,2,3,4,5,6])
b=np.array([[1,2,3],[4,5,6],[7,8,9]])
c=np.array([
    [
        [1,2,3],[4,5,6]
    ],
    [
        [7,8,9],[10,11,12]
    ],
    [
        [11,12,13],[14,15,16]
    ]
])
```

In [2]:
```
print(a)
```
```
[1 2 3 4 5 6]
```

In [3]:
```
print(a[1:4])
```
```
[2 3 4]
```

In [4]:
```
print(a[::-1])# a[start:stop:step/stride]
```
```
[6 5 4 3 2 1]
```

In [5]:
```
#2d array slicing
```

In [6]:
```
print(b)
```
```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [7]:
```
print(b[:2,:]) # b[row slicing,col slicing]
```
```
[[1 2 3]
 [4 5 6]]
```

In [8]:
```
print(b[1:,1:])
```
```
[[5 6]
 [8 9]]
```

In [9]:
```
# to find corner elements
print(b[::2,::2])
```
```
[[1 3]
 [7 9]]
```

In [10]:
```
print(b[::-1,::-1])
```
```
[[9 8 7]
 [6 5 4]
 [3 2 1]]
```

In [11]:
```
# 3d array
```

In [16]:
```
print(c[])# c[2d array slicing,1-d array slicing,elements in 1-d array slicing]
```
```
[[[16 15 14]
  [13 12 11]]

 [[12 11 10]
  [ 9  8  7]]

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

In [13]:
```
print(c)
```
```
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]

 [[11 12 13]
  [14 15 16]]]
```

In [21]:
```
c[1:,:,1:]
```
Out[21]:
```
array([[[ 8,  9],
        [11, 12]],

       [[12, 13],
        [15, 16]]])
```

In [22]:
```
# reshaping and resizing
```

In [24]:
```
#arange(): is equivalent to range function:syntax arange(start,stop,step) works on 1D only
a=np.arange(15)
print(a)
b=np.arange(1,50,2)
print(b)
```
```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
[ 1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47
 49]
```

In [25]:
```
#reshaping
print(b)
```
```
[ 1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47
 49]
```

In [26]:
```
b.size
```
Out[26]:
```
25
```

In [31]:
```
c=b.reshape((5,5))# returns new array with new shape having same size of given array
```

In [32]:
```
print(b)
```
```
[ 1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47
 49]
```

In [33]:
```
print(c)
```
```
[[ 1  3  5  7  9]
 [11 13 15 17 19]
 [21 23 25 27 29]
 [31 33 35 37 39]
 [41 43 45 47 49]]
```

In [35]:
```
b.reshape((1,5,5))
```
Out[35]:
```
array([[[ 1,  3,  5,  7,  9],
        [11, 13, 15, 17, 19],
        [21, 23, 25, 27, 29],
        [31, 33, 35, 37, 39],
        [41, 43, 45, 47, 49]]])
```

In [36]:
```
x=np.arange(20).reshape(4,5)
print(x)
```
```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

In [37]:
```
#resize:
```

In [38]:
```
print(b)
```
```
[ 1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47
 49]
```

In [42]:
```
np.resize(b,(5,7))# syntax is np.resize(array,newshape)# similer to reshape but it add/fill the remaini
ng elements by repeating the elements of existing array
```
Out[42]:
```
array([[ 1,  3,  5,  7,  9, 11, 13],
       [15, 17, 19, 21, 23, 25, 27],
       [29, 31, 33, 35, 37, 39, 41],
       [43, 45, 47, 49,  1,  3,  5],
       [ 7,  9, 11, 13, 15, 17, 19]])
```

In [41]:
```
print(b)
```
```
[ 1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47
 49]
```

In [43]:
```
#linspace(start,end,intervalpoints including start and end point)
```

In [45]:
```
a=np.linspace(1,6,4)
print(a)
```
```
[1.         2.66666667 4.33333333 6.        ]
```

In [46]:
```
a=np.linspace(1,6) # 5o points by default
print(a)
```
```
[1.         1.10204082 1.20408163 1.30612245 1.40816327 1.51020408
 1.6122449  1.71428571 1.81632653 1.91836735 2.02040816 2.12244898
 2.2244898  2.32653061 2.42857143 2.53061224 2.63265306 2.73469388
 2.83673469 2.93877551 3.04081633 3.14285714 3.24489796 3.34693878
 3.44897959 3.55102041 3.65306122 3.75510204 3.85714286 3.95918367
 4.06122449 4.16326531 4.26530612 4.36734694 4.46938776 4.57142857
 4.67346939 4.7755102  4.87755102 4.97959184 5.08163265 5.18367347
 5.28571429 5.3877551  5.48979592 5.59183673 5.69387755 5.79591837
 5.89795918 6.        ]
```

Initialize different type of data structure using numpy object(special functions)

In [1]:
```
#zeros function
import numpy as np
```

In [4]:
```
a=np.zeros((3,4),dtype=int) # fill allentris by zero and by default its datatype is float
print(a)
```
```
[[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
```

In [6]:
```
#ones function
b=np.ones((3,4),dtype=int) # fill all entries of shape by ones
print(b)
```
```
[[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
```

In [9]:
```
#full function
c=np.full((2,4,2),100)# fill all entries by a specific value
print(c)
```
```
[[[100 100]
  [100 100]
  [100 100]
  [100 100]]

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

In [10]:
```
c=np.full(b.shape,20)
print(c)
```
```
[[20 20 20 20]
 [20 20 20 20]
 [20 20 20 20]]
```

In [11]:
```
# full_like
d=np.full_like(b,30)# similer to full function but passs the argument array and specific number
print(d)
```
```
[[30 30 30 30]
 [30 30 30 30]
 [30 30 30 30]]
```

In [12]:
```
op=np.ones((5,5),dtype=int)
print(op)
```
```
[[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]]
```

In [16]:
```
ze=np.zeros((3,3),dtype=int)
print(ze)
ze[1,1]=15
print(ze)
```
```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
[[ 0  0  0]
 [ 0 15  0]
 [ 0  0  0]]
```

In [15]:
```
op[1:4,1:4]=ze
print(op)
```
```
[[ 1  1  1  1  1]
 [ 1  0  0  0  1]
 [ 1  0 15  0  1]
 [ 1  0  0  0  1]
 [ 1  1  1  1  1]]
```

In [18]:
```
# random number generated array
ra=np.random.rand(4,3) # to fill entries by random number between 0 to 1
print(ra)
```
```
[[0.43067586 0.95543267 0.48303641]
 [0.04997411 0.21314996 0.03072634]
 [0.75125641 0.47026551 0.74935257]
 [0.9771865  0.54415839 0.4649171 ]]
```

In [25]:
```
# integer values
ri=np.random.randint(-2,7,size=(3,3))
print(ri)
```
```
[[ 5  4  2]
 [-2  1  2]
 [ 6  2 -1]]
```

In [27]:
```
# identity matrix
a=np.identity(5) # creates identity matrix
print(a)
```
```
[[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 [29]:
```
b=np.diag([11,12,13]) # fill all diagonal elemenst with some specific values
```

In [30]:
```
print(b)
```
```
[[11  0  0]
 [ 0 12  0]
 [ 0  0 13]]
```

In [31]:
```
# repeat function
```

In [37]:
```
x=np.array([[1,2,3,4]])
r=np.repeat(x,3,axis=0)# repeat along with rows
print(r)
```
```
[[1 2 3 4]
 [1 2 3 4]
 [1 2 3 4]]
```

In [39]:
```
r=np.repeat(x,3,axis=1)#repeat along with cols
print(r)
```
```
[[1 1 1 2 2 2 3 3 3 4 4 4]]
```

copy and view methods

In [41]:
```
# view():used to create a view of riginal array that means any change in view willl be reflected in ori
gnal array
a=np.array([1,2,3,4])
b=a.view()
b[1]=100
print(b)
```
```
[  1 100   3   4]
```

In [42]:
```
print(a)
```
```
[  1 100   3   4]
```

In [46]:
```
# copy():used to create a copy of original array that means any change in copy will not  be reflected i
n orignal array
b=np.array([11,12,13,14])
c=b.copy()
print(c)
```
```
[11 12 13 14]
```

In [47]:
```
c[1]=200
print(c)
```
```
[ 11 200  13  14]
```

In [48]:
```
print(b)
```
```
[11 12 13 14]
```

In [49]:
```
#ravel and flatten
```

In [50]:
```
a=np.arange(1,13).reshape(4,3)
print(a)
```
```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

In [52]:
```
b=a.ravel() # use to flat multidimension array into 1-d and it creates view
```

In [53]:
```
print(b)
```
```
[ 1  2  3  4  5  6  7  8  9 10 11 12]
```

In [54]:
```
b[3]=300
print(b)
```
```
[  1   2   3 300   5   6   7   8   9  10  11  12]
```

In [55]:
```
print(a)
```
```
[[  1   2   3]
 [300   5   6]
 [  7   8   9]
 [ 10  11  12]]
```

In [56]:
```
f=a.flatten()  #use to flat multidimension array into 1-d and it creates copy of original
print(f)
```
```
[  1   2   3 300   5   6   7   8   9  10  11  12]
```

In [57]:
```
f[1]=200
print(f)
```
```
[  1 200   3 300   5   6   7   8   9  10  11  12]
```

In [58]:
```
print(a)
```
```
[[  1   2   3]
 [300   5   6]
 [  7   8   9]
 [ 10  11  12]]
```

Mathematics

In [1]:
```
import numpy as np
a=np.array([1,2,3,4,5,6])
print(a)
```
```
[1 2 3 4 5 6]
```

In [7]:
```
#arithmetic operation but does not chnage in original
print(a+2)
print(a-2)
print(a*2)
print(a/2)
print(a%2)
print(a**2)
```
```
[3 4 5 6 7 8]
[-1  0  1  2  3  4]
[ 2  4  6  8 10 12]
[0.5 1.  1.5 2.  2.5 3. ]
[1 0 1 0 1 0]
[ 1  4  9 16 25 36]
```

In [8]:
```
# save changes in the original
a+=2
```

In [9]:
```
print(a)
```
```
[3 4 5 6 7 8]
```

In [10]:
```
a-=2
print(a)
```
```
[1 2 3 4 5 6]
```

In [11]:
```
a**=2
print(a)
```
```
[ 1  4  9 16 25 36]
```

In [12]:
```
#2 D array
x=np.arange(10).reshape(2,5)
print(x)
```
```
[[0 1 2 3 4]
 [5 6 7 8 9]]
```

In [13]:
```
x+=2
print(x)
```
```
[[ 2  3  4  5  6]
 [ 7  8  9 10 11]]
```

In [14]:
```
x*=3
print(x)
```
```
[[ 6  9 12 15 18]
 [21 24 27 30 33]]
```

In [15]:
```
#trigonometry opertaion
np.sin(x)
```
Out[15]:
```
array([[-0.2794155 ,  0.41211849, -0.53657292,  0.65028784, -0.75098725],
       [ 0.83665564, -0.90557836,  0.95637593, -0.98803162,  0.99991186]])
```

In [16]:
```
np.cos(x)
```
Out[16]:
```
array([[ 0.96017029, -0.91113026,  0.84385396, -0.75968791,  0.66031671],
       [-0.54772926,  0.42417901, -0.29213881,  0.15425145, -0.01327675]])
```

In [17]:
```
np.tan(x)
```
Out[17]:
```
array([[ -0.29100619,  -0.45231566,  -0.63585993,  -0.8559934 ,
         -1.13731371],
       [ -1.52749853,  -2.1348967 ,  -3.2737038 ,  -6.4053312 ,
        -75.3130148 ]])
```

In [18]:
```
# docs.scipy.org
```

In [19]:
```
#Linear algebra
```

In [20]:
```
A=np.array([[2,0,1],[4,3,8],[9,8,7]])
B=np.array([[10,20,30],[40,50,60],[70,80,90]])
print(A)
print(B)
```
```
[[2 0 1]
 [4 3 8]
 [9 8 7]]
[[10 20 30]
 [40 50 60]
 [70 80 90]]
```

In [21]:
```
#Transpose of matrix
A.transpose()
```
Out[21]:
```
array([[2, 4, 9],
       [0, 3, 8],
       [1, 8, 7]])
```

In [22]:
```
# alternative
```

In [24]:
```
B.T
```
Out[24]:
```
array([[10, 40, 70],
       [20, 50, 80],
       [30, 60, 90]])
```

In [25]:
```
# Addition of 2 matrices
```

In [26]:
```
print(A+B)
```
```
[[12 20 31]
 [44 53 68]
 [79 88 97]]
```

In [27]:
```
# multiplication of two matrics
```

In [30]:
```
x=np.arange(6).reshape(2,3)
y=np.arange(12).reshape(3,4)
print(x)
print(y)
```
```
[[0 1 2]
 [3 4 5]]
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

In [34]:
```
# matrxi multiplication
print(x@y)
print(np.matmul(x,y))
print(y@x)# will raise error
```
```
[[20 23 26 29]
 [56 68 80 92]]
[[20 23 26 29]
 [56 68 80 92]]
```

In [37]:
```
#determinant of a matrix
np.linalg.det(A)
```
Out[37]:
```
-81.00000000000003
```

In [38]:
```
print(B)
```
```
[[10 20 30]
 [40 50 60]
 [70 80 90]]
```

In [39]:
```
# trace# summation of diagonal element
```

In [40]:
```
np.trace(B)
```
Out[40]:
```
150
```

In [41]:
```
# inverse of matrix
```

In [42]:
```
np.linalg.inv(A)
```
Out[42]:
```
array([[ 0.5308642 , -0.09876543,  0.03703704],
       [-0.54320988, -0.0617284 ,  0.14814815],
       [-0.0617284 ,  0.19753086, -0.07407407]])
```

In [43]:
```
np.linalg.inv(B)
```
Out[43]:
```
array([[ 7.03687442e+13, -1.40737488e+14,  7.03687442e+13],
       [-1.40737488e+14,  2.81474977e+14, -1.40737488e+14],
       [ 7.03687442e+13, -1.40737488e+14,  7.03687442e+13]])
```

In [44]:
```
# statistics
```

In [45]:
```
a=np.arange(12)
print(a)
```
```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
```

In [46]:
```
np.sum(a)
```
Out[46]:
```
66
```

In [47]:
```
np.mean(a)
```
Out[47]:
```
5.5
```

In [48]:
```
np.min(a)
```
Out[48]:
```
0
```

In [49]:
```
np.max(a)
```
Out[49]:
```
11
```

In [50]:
```
b=np.arange(1,13).reshape(3,4)
print(b)
```
```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

In [56]:
```
print(np.sum(b))
print(np.sum(b,axis=0))
print(np.sum(b,axis=1))
```
```
78
[15 18 21 24]
[10 26 42]
```

In [59]:
```
print(np.mean(b))
print(np.mean(b,axis=0))
print(np.mean(b,axis=1))
```
```
6.5
[5. 6. 7. 8.]
[ 2.5  6.5 10.5]
```

In [63]:
```
print(np.std(b))
print(np.std(b,axis=0))
print(np.std(b,axis=1))
```
```
3.452052529534663
[3.26598632 3.26598632 3.26598632 3.26598632]
[1.11803399 1.11803399 1.11803399]
```

In [64]:
```
print(np.max(b,axis=0))
```
```
[ 9 10 11 12]
```

In [65]:
```
print(np.min(b,axis=1))
```
```
[1 5 9]
```

reorganizing the arrays

In [67]:
```
# reshape method used to reorgnize the array
a=np.arange(10)
print(a)
x=a.reshape(2,5)
print(x)
```
```
[0 1 2 3 4 5 6 7 8 9]
[[0 1 2 3 4]
 [5 6 7 8 9]]
```

In [68]:
```
# vstack: used to arange the arrays in vertical form by concatenate the next array in the given order as argument in vstack method
v1=np.array([1,2,3,4])
v2=np.array([5,6,7,8])
v3=np.array([7,8,9,10])
v4=np.array([11,12,13,14])# but dimensions(cols) shud be matched
np.vstack([v1,v2,v3,v4])
```
Out[68]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 7,  8,  9, 10],
       [11, 12, 13, 14]])
```

In [69]:
```
np.vstack([v2,v1,v3,v4])
```
Out[69]:
```
array([[ 5,  6,  7,  8],
       [ 1,  2,  3,  4],
       [ 7,  8,  9, 10],
       [11, 12, 13, 14]])
```

In [71]:
```
v1=np.array([1,2,3,4])
v2=np.array([5,6,7,8])
v3=np.array([7,8,9,10])
v4=np.array([11,12,13,14])
np.vstack([v1,v2,v3,v4])
```
Out[71]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 7,  8,  9, 10],
       [11, 12, 13, 14]])
```

In [76]:
```
#hstack method
h1=np.ones((2,4))
h2=np.zeros((2,2))# no of rows shud be matched
print(h1)
print(h2)
```
```
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]]
[[0. 0.]
 [0. 0.]]
```

In [77]:
```
np.hstack([h1,h2])
```
Out[77]:
```
array([[1., 1., 1., 1., 0., 0.],
       [1., 1., 1., 1., 0., 0.]])
```

In [ ]:
```
# split methods
```