In [ ]:

```
import numpy as np
```

In [7]:

```
a = np.arange(15).reshape(3,5)
a
type(a)  # type of array
a.ndim   # Dimensioan of arry
a.size   # size of an array
```

Out[7]:

```
15
```

In [9]:

```
a = np.array([2,3,4])
a
type(a)
```

Out[9]:

```
numpy.ndarray
```

In [17]:

```
b = np.array([[2,3,4], [5,6,7]])
b
```

Out[17]:

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

In [18]:

```
b = np.array([[1,2,3],[4,5,6]], dtype=complex)
b
```

Out[18]:

```
array([[1.+0.j, 2.+0.j, 3.+0.j],
       [4.+0.j, 5.+0.j, 6.+0.j]])
```

In [19]:

```
np.zeros
```

Out[19]:

```
<function numpy.zeros>
```

In [20]:

```
np.zeros([3,4])   # zero matrix with 3rows and 4 columns
```

Out[20]:

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

In [22]:

```
np.ones([2,3,4]) # 2 denotes 2 set of 3x4 ones matrix
```

Out[22]:

```
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 [27]:

```
c = np.arange(10,50,5) # range b/n 10 - 50 including 10 and excluding 50 with a interval of 5
c
```

Out[27]:

```
array([10, 15, 20, 25, 30, 35, 40, 45])
```

In [29]:

```
d = np.arange(10,15,0.25)  # interval range is defined so values wil be printed till the last
range of value defined
d
```

Out[29]:

```
array([10.  , 10.25, 10.5 , 10.75, 11.  , 11.25, 11.5 , 11.75, 12.  ,
       12.25, 12.5 , 12.75, 13.  , 13.25, 13.5 , 13.75, 14.  , 14.25,
       14.5 , 14.75])
```

In [37]:

```
f = np.linspace(10,15,10) # length of element = 10 mean 10 numbers should be printed b/n 10-15
irrespective of interval
f                         # with equal length and space
```

Out[37]:

```
array([10.        , 10.55555556, 11.11111111, 11.66666667, 12.22222222,
       12.77777778, 13.33333333, 13.88888889, 14.44444444, 15.        ])
```

In [50]:

```
a = np.array([[10,20,30,40], [5,6,7,8]])
b = np.array([[50,60,70,80], [1,2,3,4]])
print(a)
b
```

```
[[10 20 30 40]
 [ 5  6  7  8]]
```

Out[50]:

```
array([[50, 60, 70, 80],
       [ 1,  2,  3,  4]])
```

In [51]:

```
c = a+b
c
```

Out[51]:

```
array([[ 60,  80, 100, 120],
       [  6,   8,  10,  12]])
```

In [52]:

```
c = a*b  # Element wise product
c
```

Out[52]:

```
array([[ 500, 1200, 2100, 3200],
       [   5,   12,   21,   32]])
```

In [62]:

```
a = np.array ([[1,1],[0,1]])
b = np.array ([[2,0], [3,4]])
a
b
a@b
```

Out[62]:

```
array([[5, 4],
       [3, 4]])
```

In [6]:

```
a= np.random.random((2,3))
a
```

Out[6]:

```
array([[0.55504689, 0.53784705, 0.97108615],
       [0.94430267, 0.09198609, 0.67167239]])
```

In [11]:

```
b = a.sum() # sum of all elements as a whole
b
```

Out[11]:

```
3.7719412271345334
```

In [13]:

```
b = a.max()  # max value among the matrix
print(b)
c = a.min()
c
```

```
0.9710861466516325
```

Out[13]:

```
0.09198608896075455
```

In [14]:

```
d = np.arange(12).reshape(4,3)
d
```

Out[14]:

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

In [18]:

```
f = d.sum(axis=0) # axis= 0 --> column sum  axis = 1 --> Row sum
print(f)

f = d.sum(axis=1)
print(f)
```

```
[18 22 26]
[ 3 12 21 30]
```

In [20]:

```
f = d.cumsum(axis=0) # cumsum = cumulative sum -- Columns 0+3, 0+3, 0+3+9, 0+3+9+18
f
```

Out[20]:

```
array([[ 0,  1,  2],
       [ 3,  5,  7],
       [ 9, 12, 15],
       [18, 22, 26]], dtype=int32)
```

In [21]:

```
f = d.cumsum(axis=1) # Cumsum - cumuative sum - Rows
f
```

Out[21]:

```
array([[ 0,  1,  3],
       [ 3,  7, 12],
       [ 6, 13, 21],
       [ 9, 19, 30]], dtype=int32)
```

Indexing, Slicing, Iterating

In [28]:

```
I = np.arange(10)**3   # cube of 0-9 numbers
I
```

Out[28]:

```
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729], dtype=int32)
```

In [29]:

```
I[2] # value of I in index 2
```

Out[29]:

```
8
```

In [30]:

```
I[0:4] # includes 0 and exclude 4
```

Out[30]:

```
array([ 0,  1,  8, 27], dtype=int32)
```

In [31]:

```
I[2:6:2] # Start at 2 end at 5 at an interval of 2
```

Out[31]:

```
array([ 8, 64], dtype=int32)
```

In [32]:

```
I[::-1] # printing in reverse direction
```

Out[32]:

```
array([729, 512, 343, 216, 125,  64,  27,   8,   1,   0], dtype=int32)
```

Stacking

Stacking mean grouping together

In [6]:

```
import numpy as np
a = np.arange(6).reshape(2,3)
print(a)
```

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

In [5]:

```
b = np.arange(8).reshape(2,4)
print(b)
```

```
[[0 1 2 3]
 [4 5 6 7]]
```

In [7]:

```
# To perform horizontalstack - rows should be equal
np.hstack((a,b))
```

Out[7]:

```
array([[0, 1, 2, 0, 1, 2, 3],
       [3, 4, 5, 4, 5, 6, 7]])
```

In [8]:

```
c = np.arange(12).reshape(3,4)
c
```

Out[8]:

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

In [9]:

```
d = np.arange(20).reshape(5,4)
d
```

Out[9]:

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

In [11]:

```
# to perform verticlstack comlunms should be equal

np.vstack((c,d))
```

Out[11]:

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

Splitting of one array into smaller ones

In [17]:

```
a = np.floor(10*np.random.random((2,12)))
a
```

Out[17]:

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

In [18]:

```
np.hsplit(a,3) # Split the entire matrix a into 3 arrays # horizontal split hstack
```

Out[18]:

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

In [22]:

```
np.vsplit(a,2)
```

Out[22]:

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

In [23]:

```
b = np.floor(10*np.random.random((6,2)))
b
```

Out[23]:

```
array([[1., 0.],
       [1., 8.],
       [5., 1.],
       [4., 6.],
       [3., 6.],
       [0., 1.]])
```

In [24]:

```
np.vsplit(b,6)
```

Out[24]:

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

```
 array([[1., 8.]]),
 array([[5., 1.]]),
 array([[4., 6.]]),
 array([[3., 6.]]),
 array([[0., 1.]])]
```

In [28]:

```
z = np.arange(8)
z
```

Out[28]:

```
array([0, 1, 2, 3, 4, 5, 6, 7])
```

In [29]:

```
np.array_split(z,3)  # Eaually divide the matrix into given arrays and rest wil be printed in last array
```

Out[29]:

```
[array([0, 1, 2]), array([3, 4, 5]), array([6, 7])]
```

In [31]:

```
# Copying a array

x = np.array([1,2,3])
y=x
z = np.copy(x)

print(x)
print(y)
print(z)
```

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

In [33]:

```
x[0]= 2 ## Any changes in x will reflect in Y but not in Z since we have copied Z from x

print(x)
print(y)
print(z)
```

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

In [35]:

```
np.empty((2,2), dtype=int)  # retudn empty array
```

Out[35]:

```
array([[         0,          0],
       [         0, 1072693248]])
```

In [36]:

```
np.eye(3,4) # array with diagonal 1
```

Out[36]:

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

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

In [37]:

```
np.eye(3,4, k=1)  # k value positive mean upper diagaonal
```

Out[37]:

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

In [38]:

```
np.eye(3,4, k=-1) # k value negative mean lower diagonal
```

Out[38]:

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

In [41]:

```
np.identity((4))  # square array with one on the diagonal
```

Out[41]:

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

Converting array into float and integer

In [43]:

```
x = np.array([1,2,3])
x
```

Out[43]:

```
array([1, 2, 3])
```

In [44]:

```
x.astype(float)  # output will be in float
```

Out[44]:

```
array([1., 2., 3.])
```

In [45]:

```
y = np.array([1.2,3,4])
y
```

Out[45]:

```
array([1.2, 3. , 4. ])
```

In [46]:

```
y.astype(int) # output will be all values in integer
```

Out[46]:

```
array([1, 3, 4])
```

Cross Product of a matrix - Vector Cross Product

In [47]:

```
x = [1,2,3]
y = [4,5,6]

print(x)
print(y)
```

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

In [48]:

```
np.cross(x,y)
```

Out[48]:

```
array([-3,  6, -3])
```

In [56]:

```
m = [1,2,3],[4,5,6]
n = [4,5,6],[1,2,3]
print(m)
print(n)
```

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

In [57]:

```
np.cross(m,n)
```

Out[57]:

```
array([[-3,  6, -3],
       [ 3, -6,  3]])
```

In [ ]:

```

```