Numpy

In [1]:

```
import numpy as np
a= np.array([1,2,3,4,5,6,7,8,9,10])
a*5
```

Out[1]:

```
array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50])
```

In [4]:

```
b = np.array([1.5,2.5,3])
b*2
```

Out[4]:

```
array([3., 5., 6.])
```

In [5]:

```
type(a)
```

Out[5]:

```
numpy.ndarray
```

In [6]:

```
type(b)
```

Out[6]:

```
numpy.ndarray
```

In [7]:

```
c = np.array([1,2], dtype = "float32")
c
```

Out[7]:

```
array([1., 2.], dtype=float32)
```

In [13]:

```
d = np.array([1.1,2.5,6.9], dtype = "int8")  #not flooring not ceiling
d
```

Out[13]:

```
array([1, 2, 6], dtype=int8)
```

In [14]:

```
print (type(d))
```

```
<class 'numpy.ndarray'>
```

In [15]:

```
type(d)  #OR look down for mentioning dtype in quote n without quotes
```

Out[15]:

```
numpy.ndarray
```

In [16]:

```
e = np.array([1.1,2.5,6.9], dtype = np.int8)  #not flooring not ceiling
e
```

Out[16]:

```
array([1, 2, 6], dtype=int8)
```

In [17]:

```
type(e)
```

Out[17]:

```
numpy.ndarray
```

In [27]:

```
# to create random numbers
randomarray1 = np.random.rand(3)
randomarray1
```

Out[27]:

```
array([0.97332176, 0.0600249 , 0.29276713])
```

In [36]:

```
randomarray1 = np.random.rand(6)
print(randomarray1)
print(randomarray1 *10)
print(randomarray1 +10)
print(randomarray1 - 10)
print(randomarray1 /2)
print(randomarray1 **2)
print(randomarray1 // 2)
```

```
[0.86269731 0.58893429 0.6660942  0.24454413 0.10712988 0.22099194]
[8.62697311 5.88934286 6.66094201 2.44544131 1.07129879 2.20991936]
[10.86269731 10.58893429 10.6660942  10.24454413 10.10712988 10.22099194]
[-9.13730269 -9.41106571 -9.3339058  -9.75545587 -9.89287012 -9.77900806]
[0.43134866 0.29446714 0.3330471  0.12227207 0.05356494 0.11049597]
[0.74424665 0.34684359 0.44368148 0.05980183 0.01147681 0.04883744]
[0. 0. 0. 0. 0. 0.]
```

In [41]:

```
randomarray2 = np.random.rand(6).reshape(3,2)
print(randomarray2)
print(randomarray2 * 2)
print(type(randomarray2))
```

```
[[0.72522076 0.81790041]
 [0.59787396 0.05756699]
 [0.14961437 0.93401208]]
[[1.45044152 1.63580082]
 [1.19574791 0.11513397]
 [0.29922873 1.86802417]]
<class 'numpy.ndarray'>
```

In [ ]:

```
# reshape create 2 dimentional array as double squares
```

In [ ]:

```
# arange creates same as range in list
```

In [ ]:

```
# reshape can apply on arange(4).rehape(2,2) and random.rand(4).reshape(2,2)
```

In [42]:

```
# to generate range numbers
randomarray3 = np.arange(3)
print(randomarray3)
print(randomarray3 *2)
print(type(randomarray3))
```

```
[0 1 2]
[0 2 4]
<class 'numpy.ndarray'>
```

In [21]:

```
randomarray = np.arange(6).reshape(3,2)
randomarray
```

Out[21]:

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

In [23]:

```
newrandomarray = np.array([2,3.4,5.6,66])
print(newrandomarray)
print(newrandomarray.reshape(2,2)) # reshape jiska lena ho apply array pe hoga not on
```

```
[ 2.   3.4  5.6 66. ]
[[ 2.   3.4]
 [ 5.6 66. ]]
```

In [24]:

```
randomarray = np.arange(12).reshape(4,3) # 4 rows and 3 columns
randomarray
```

Out[24]:

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

In [44]:

```
# to check dimention of array
newrandomarray.ndim
```

Out[44]:

```
1
```

In [45]:

```
randomarray.ndim  #answer 2 = double square brackets
```

Out[45]:

```
2
```

2 Dimentional Array

In [46]:

```
#array>> vector  (1D)
#matrix>> tensor(Multi dimentional)
```

In [65]:

```
# making tensor 2D = x and y axis
twoD = np.array([[1,2,3],[4,5,6],[7,8,9],[2,3,4]])
print(twoD)
print(twoD.ndim)
print(twoD.shape)
print(type(twoD))
print(twoD.reshape(3,4))
print(np.array([[1,2],[2,4]]))
print(np.array([[1,2],[2,4]],dtype="float32"))
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]
 [2 3 4]]
2
(4, 3)
<class 'numpy.ndarray'>
[[1 2 3 4]
 [5 6 7 8]
 [9 2 3 4]]
[[1 2]
 [2 4]]
[[1. 2.]
 [2. 4.]]
```

In [49]:

```
twoD2 = np.array([[4,5,6],[7,8,9],[2,3,4]])
twoD2
```

Out[49]:

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

In [57]:

```
twoD2.reshape(3,3)
```

Out[57]:

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

In [59]:

```
twoD.reshape(3,4) # bcz it has 12 numbers make pair
```

Out[59]:

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

In [60]:

```
1 twoD.reshape(6,2) # 12 values
```

Out[60]:

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

Below showing scalar,vector(1D),2D arrays etc

In [75]:

```
1 zeroD = np.array(100)
2 print (zeroD.ndim)    #dimension is 0
3 zeroD.shape #() ki koi shape nhi, scalar ki koi shape nhi
```

```
0
```

Out[75]:

```
()
```

In [ ]:

```
1
```

In [68]:

```
1 miscD = np.array([99]) # 1 dimentional can b x or y;element is single
2 print(miscD)
3 print(miscD.ndim)
4 print(miscD.shape) #comma means vector
```

```
[99]
1
(1,)
```

In [69]:

```
1 miscD2 = np.array([[99]]) # now its 2dimentional x and y axid means matrix; element is
2 print(miscD2)
3 print(miscD2.ndim)
4 print(miscD2.shape)
```

```
[[99]]
2
(1, 1)
```

In [70]:

```
miscD3 = np.array([[[99]]]) # now its 3dimentional x and y and z axis means matrix; ele
print(miscD3)
print(miscD3.ndim)
print(miscD3.shape)
```

```
[[[99]]]
3
(1, 1, 1)
```

In [73]:

```
miscD5 = np.array([[[[[99]]]]]) # now its 5dimentional x and y and z and 1 more axis me
print(miscD5)
print(miscD5.ndim)
print(miscD5.shape)
```

```
[[[[[99]]]]]
5
(1, 1, 1, 1, 1)
```

arange then ravel

In [7]:

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

Out[7]:

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

In [9]:

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

Out[9]:

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

In [16]:

```
# now ravel means sedha krna
arrayravel = np.ravel(arrayarange)
print(arrayravel)
print(arrayravel.ndim)
print(arrayravel.shape)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
1
(12,)
```

In [15]:

```
newreshape = arrayravel.reshape(4,3)
print(newreshape)
print(newreshape.ndim)
print(newreshape.shape)
```

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

In [13]:

```
newreshape = arrayravel.reshape(6,2)
newreshape
```

Out[13]:

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

In [ ]:

```

```

In [25]:

```
twoD1 = np.arange(12).reshape(4,3)
print(twoD1)
print(twoD1.ndim)
print(twoD1.shape)
print(np.ravel(twoD1))
threeD1 = twoD1.reshape(2,2,3)
print(threeD1) # 3D array
print(threeD1.ndim)
print(threeD1.shape)
threeDravel = np.ravel(threeD1)
print(threeDravel)
print(threeDravel.ndim)
print(threeDravel.shape)
new2D = threeDravel.reshape(4,3)
print(new2D)
print(new2D.ndim)
print(new2D.shape)
print(new2D.reshape(6,2))
```

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

 [[ 6  7  8]
  [ 9 10 11]]]
3
(2, 2, 3)
[ 0  1  2  3  4  5  6  7  8  9 10 11]
1
(12,)
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
2
(4, 3)
[[ 0  1]
 [ 2  3]
 [ 4  5]
 [ 6  7]
 [ 8  9]
 [10 11]]
```

Creating 3D array

3D means 3 square brackets, 2D means 2 brackets

In [35]:

```
threeD2 = np.array([[[1,2,3],
                     [4,5,6],
                     [7,8,9]],
                    [[3,56,8],
                     [8,7,6],
                     [9,8,7]]])
print(threeD2)#3 d k andar dou 2ds hen n wo 2ds k andar 1  dimensionsl thy
print(threeD2.ndim)
print(threeD2.shape) # means 18
```

```
[[[ 1  2  3]
  [ 4  5  6]
  [ 7  8  9]]

 [[ 3 56  8]
  [ 8  7  6]
  [ 9  8  7]]]
3
(2, 3, 3)
```

In [38]:

```
twoDback = threeD2.reshape(6,3)
print(twoDback)
print(np.ravel(twoDback))
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [ 3 56  8]
 [ 8  7  6]
 [ 9  8  7]]
[ 1  2  3  4  5  6  7  8  9  3 56  8  8  7  6  9  8  7]
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [26]:

```
1 twoD1 + twoD1
```

Out[26]:

```
array([[ 0,  2,  4],
       [ 6,  8, 10],
       [12, 14, 16],
       [18, 20, 22]])
```

In [27]:

```
1 twoD1 - twoD1
```

Out[27]:

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

In [28]:

```
1 twoD1 / twoD1
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: RuntimeW
arning: invalid value encountered in true_divide
  """Entry point for launching an IPython kernel.
```

Out[28]:

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

In [29]:

```
1 twoD1 // twoD1
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: RuntimeW
arning: divide by zero encountered in floor_divide
  """Entry point for launching an IPython kernel.
```

Out[29]:

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

In [31]:

```
1 twoD1 * twoD1
```

Out[31]:

```
array([[  0,   1,   4],
       [  9,  16,  25],
       [ 36,  49,  64],
       [ 81, 100, 121]])
```

In [32]:

```
1 twoD1 ** twoD1
```

Out[32]:

```
array([[         1,          1,          4],
       [        27,        256,       3125],
       [     46656,     823543,   16777216],
       [ 387420489, 1410065408, 1843829075]], dtype=int32)
```

In [52]:

```
1 m1 = np.arange(12).reshape(4,3)
2 m2 = np.random.randn(12).reshape(4,3)
3 m1 + m2
```

Out[52]:

```
array([[-1.45443735,  0.65484139,  4.06793497],
       [ 3.56680494,  5.09085585,  3.96920617],
       [ 6.12238992,  7.25578511,  8.65774725],
       [ 8.44296138, 10.14523869, 11.91984337]])
```

In [53]:

```
1 m4 = np.arange(12).reshape(4,3)
2 m5 = np.random.randn(16).reshape(4,4)
3 m4 + m5 # error bcz addition etc k liye same shape is necessary
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-53-70af96d1ea4f> in <module>
      1 m4 = np.arange(12).reshape(4,3)
      2 m5 = np.random.randn(16).reshape(4,4)
----> 3 m4 + m5 # error bcz addition etc k liye same shape is necessary

ValueError: operands could not be broadcast together with shapes (4,3) (4,4)
```

In [41]:

```
1 #in multiplaction of matrix first row elements  equal to first coloumn elements 2x3 2x
2 # order is shape
```

In [54]:

```
m3 = np.arange(18).reshape(3,6)
print(m3)
print(m3.shape)
print(m1.shape) # rows of first  matrix= colums of other
```

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

In [44]:

```
print(m1@m3) # for mirror multiplication need rows = col of other
```

```
[[ 30  33  36  39  42  45]
 [ 84  96 108 120 132 144]
 [138 159 180 201 222 243]
 [192 222 252 282 312 342]]
```

In [45]:

```
print(m1*m3) # for this matrix equal size
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-45-8f9b6b6c7384> in <module>
----> 1 print(m1*m3)

ValueError: operands could not be broadcast together with shapes (4,3) (3,6)
```

In [55]:

```
print(m1 * m2)
# shape of both is 4 x 3 same shape for multiplication index to index
# mirror is 1st row 1st col like this
# same shape pe @ krenge tu answer diff aega

```

Out[55]:

```
array([[-0.        , -0.34515861,  4.13586993],
       [ 1.70041483,  4.36342341, -5.15396916],
       [ 0.73433949,  1.79049575,  5.26197801],
       [-5.01334757,  1.45238687, 10.11827703]])
```

In [60]:

```
n1 = np.arange(6).reshape(3,2)
n2 = np.arange(6).reshape(3,2)
print(n1)
print(n2)
print(n1*n2)
n3= n2.reshape(2,3)
print(n1 @ n3)
```

```
[[0 1]
 [2 3]
 [4 5]]
[[0 1]
 [2 3]
 [4 5]]
[[ 0  1]
 [ 4  9]
 [16 25]]
[[ 3  4  5]
 [ 9 14 19]
 [15 24 33]]
```

In [62]:

```
print(np.dot(n1,n3)) # other way of mirror multiplication
print(n1.dot(n3))
```

```
[[ 3  4  5]
 [ 9 14 19]
 [15 24 33]]
[[ 3  4  5]
 [ 9 14 19]
 [15 24 33]]
```

In [ ]:

```

```

In [65]:

```
#builtin matrix zero  1D
zeroos = np.zeros(4)
print(zeroos)
print(np.zeros(4,dtype = "int8"))
print(zeroos.ndim)
print(zeroos.shape)
```

```
[0. 0. 0. 0.]
[0 0 0 0]
1
(4,)
```

In [71]:

```
# 2D zero matrix
twoDzero = np.zeros((4,3),dtype="int8") # must give dtype in 2D otherwise not run
print(twoDzero)
print(twoDzero.ndim)
print(twoDzero.shape)
print(twoDzero.reshape(6,2))
```

```
[[0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]]
2
(4, 3)
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]]
```

In [77]:

```
#creating 1 D 2 D 3 D matriz of ones
oneDone = np.ones(4)
print(oneDone)
twoDones = np.ones((4,5),dtype ="int8") # tuple he 4 5
print(twoDones)
print(oneDone.ndim)
print(twoDones.ndim)
threeDones = np.ones((3,3,4),dtype = "float32")
print(threeDones)
print(threeDones.ndim)
print(threeDones *2)
```

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

 [[2. 2. 2. 2.]
  [2. 2. 2. 2.]
  [2. 2. 2. 2.]]

 [[2. 2. 2. 2.]
  [2. 2. 2. 2.]
  [2. 2. 2. 2.]]]
```

In [82]:

```
# creating full of the given number in 1 D 2D and 3D array
oneDfull = np.full(4,50)  #other way kis se full kra he
print(oneDfull)
twoDfull = np.full((3,3),50)
print(twoDfull)
threeDfull = np.full((3,3,3),50)
print(threeDfull)
print(twoDfull*2)
```

```
[50 50 50 50]
[[50 50 50]
 [50 50 50]
 [50 50 50]]
[[[50 50 50]
  [50 50 50]
  [50 50 50]]

 [[50 50 50]
  [50 50 50]
  [50 50 50]]

 [[50 50 50]
  [50 50 50]
  [50 50 50]]]
[[100 100 100]
 [100 100 100]
 [100 100 100]]
```

In [87]:

```
# empty create zeros but bcz of garbage value(no refreence )showing this data
oneDempty = np.empty(2)
print(oneDempty)
twoDempty = np.empty((2,2),dtype="int8") # in 2D must define data type
print(twoDempty)
threeDempty = np.empty((2,2,2))
print(threeDempty) # empty create zero but bcz of garbage value(no reference)show those
fourDempty = np.empty((2,2,2,2))
print(fourDempty)
```

```
[1.0609979e-312 1.0609979e-312]
[[0 0]
 [0 0]]
[[[0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]]]
[[[[-0.4637675  -0.3077015 ]
   [-0.37730758 -0.92496697]]

  [[-0.01230615  0.7877795 ]
   [ 1.5481059  -0.13274204]]]


 [[[ 1.36829285 -1.59731338]
   [ 0.07365215 -1.50495364]]

  [[ 0.64683316 -0.53858111]
   [ 0.30888765 -0.61745263]]]]
```

In [94]:

```
#zeros like measn creates zeros but shape and data type uthatega m3 ki
print(m3)
likezero = np.zeros_like(m3)
print(likezero)
likeones = np.ones_like(m3)
print(likeones)

```

```
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]]
[[0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]]
[[1 1 1 1 1 1]
 [1 1 1 1 1 1]
 [1 1 1 1 1 1]]
```

In [106]:

```
#identity works on 2D
i1 = np.eye(4) #idenetity diagnol 1 n determinant ans is 1 # 4 means 4 by 4 matriz
i2 = np.eye(4,2)
i3 = np.eye(3,3,3) # 3 by 3 matriz but showing zeros not 1 in diagnol.note
print(i1)
print(i2)
print(i3)

```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
[[1. 0.]
 [0. 1.]
 [0. 0.]
 [0. 0.]]
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

In [112]:

```
# change data type of matriz
changedatatype = i2.astype("int8")
print(changedatatype)
```

```
[[1 0]
 [0 1]
 [0 0]
 [0 0]]
```

In [115]:

```
print(changedatatype>i2)  # should be same shape
```

```
[[False False]
 [False False]
 [False False]
 [False False]]
```

Basic Indexing and Slicing

In [117]:

```
alist = [99,77,55,33,11]
print(alist[0])
print(type(alist))
print(type(alist[0]))
print(type(alist[0:3]))

```

```
99
<class 'list'>
<class 'int'>
<class 'list'>
```

In [119]:

```
#indexing se value ati he jo data type apne di hogi jo k int he
#in slicing return jid data type ka slice woi hoga mtlb list se list niklega
#pizza ka slice pizza he but pizza mei mushroom pizza nhi he
```

In [138]:

```
tdarray = np.arange(36).reshape(6,6)
print(tdarray)

# now get 20 number which is 4th row and 3rd col but indexing starts with 0 always reme
print(tdarray[3,2])

#slicing 13 14 15 19 20 21 26 27 28 (3x3 matrix)
#tdarray[row,colomn]
#tdarray[start:end, start:end]
print(tdarray[2:5, 1:4]) # end at 5 means included 4. 5 not included

#33 find using indexing - show u 1D
print(tdarray[5,3])

# 33 find using slicing - show u 2D
print(tdarray[5:,3:4])

#
print(tdarray[1:2,2:5])

print(tdarray[2:4,1:3])

print(tdarray[1:6:2,1:6:2])
print(tdarray[::2,::2])
print(tdarray[::-2,::-2])
print(tdarray[::2,::-2])
print(tdarray[0:6,0])  #showing 1D
print(tdarray[0:6,0:1]) # showing 2D
```

```
[[ 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 25 26 27 28 29]
 [30 31 32 33 34 35]]
20
[[13 14 15]
 [19 20 21]
 [25 26 27]]
33
[[33]]
[[ 8  9 10]]
[[13 14]
 [19 20]]
[[ 7  9 11]
 [19 21 23]
 [31 33 35]]
[[ 0  2  4]
 [12 14 16]
 [24 26 28]]
[[35 33 31]
 [23 21 19]
 [11  9  7]]
[[ 5  3  1]
 [17 15 13]
 [29 27 25]]
[ 0  6 12 18 24 30]
[[ 0]
 [ 6]
```

```
[12]
[18]
[24]
[30]]
```

In [4]:

```
arrw = np.arange(12).reshape(2,3,2)
arrw

```

Out[4]:

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

       [[ 6,  7],
        [ 8,  9],
        [10, 11]]])
```

In [5]:

```
flat=arrw.flatten()
flat
```

Out[5]:

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

In [6]:

```
arrw
```

Out[6]:

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

       [[ 6,  7],
        [ 8,  9],
        [10, 11]]])
```

In [7]:

```
ravel = arrw.ravel()
ravel
```

Out[7]:

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

In [8]:

```
ravel[3:6]=1000  # 3 till 5 value will be 1000
```

In [9]:

```
1 ravel
```

Out[9]:

```
array([   0,    1,    2, 1000, 1000, 1000,    6,    7,    8,    9,   10,
        11])
```

In [8]:

```
1 flat[3:5]=444
2 flat
```

Out[8]:

```
array([  0,   1,   2, 444, 444,   5,   6,   7,   8,   9,  10,  11])
```

In [9]:

```
1 arrw
```

Out[9]:

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

       [[ 6,  7],
        [ 8,  9],
        [10, 11]]])
```

In [10]:

```
1 arr1w = np.array([[1, 2, 3], [4, 5, 6]])
2 arr1w
```

Out[10]:

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

In [11]:

```
1 arr2w = np.array([[7, 8, 9], [10, 11, 12]])
2 arr2w
```

Out[11]:

```
array([[ 7,  8,  9],
       [10, 11, 12]])
```

In [12]:

```
1 np.concatenate([arr1w,arr2w])
```

Out[12]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [14]:

```
1 np.concatenate([arr1w,arr2w],axis=0)
```

Out[14]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [15]:

```
1 np.concatenate([arr1w,arr2w],axis=1)
```

Out[15]:

```
array([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 11, 12]])
```

In [16]:

```
1 np.vstack([arr1w,arr2w])  #vstack or axis 0 or concatenate
```

Out[16]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [18]:

```
1 np.hstack([arr1w,arr2w])
```

Out[18]:

```
array([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 11, 12]])
```

In [5]:

```
sp = np.arange(36).reshape(6,6)
sp
```

Out[5]:

```
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, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35]])
```

In [20]:

```
np.vsplit(sp,3)
```

Out[20]:

```
[array([[ 0,  1,  2,  3,  4,  5],
        [ 6,  7,  8,  9, 10, 11]]), array([[12, 13, 14, 15, 16, 17],
        [18, 19, 20, 21, 22, 23]]), array([[24, 25, 26, 27, 28, 29],
        [30, 31, 32, 33, 34, 35]])]
```

In [22]:

```
np.vsplit(sp,1)
```

Out[22]:

```
[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, 25, 26, 27, 28, 29],
        [30, 31, 32, 33, 34, 35]])]
```

In [24]:

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

Out[24]:

```
[array([[ 0,  1,  2,  3,  4,  5],
        [ 6,  7,  8,  9, 10, 11],
        [12, 13, 14, 15, 16, 17]]), array([[18, 19, 20, 21, 22, 23],
        [24, 25, 26, 27, 28, 29],
        [30, 31, 32, 33, 34, 35]])]
```

In [27]:

```
1 np.hsplit(sp,3)
```

Out[27]:

```
[array([[ 0,  1],
        [ 6,  7],
        [12, 13],
        [18, 19],
        [24, 25],
        [30, 31]]), array([[ 2,  3],
        [ 8,  9],
        [14, 15],
        [20, 21],
        [26, 27],
        [32, 33]]), array([[ 4,  5],
        [10, 11],
        [16, 17],
        [22, 23],
        [28, 29],
        [34, 35]])]
```

In [28]:

```
1 np.hsplit(sp,2) #Split an array into multiple sub-arrays horizontally (column-wise).
2 #Please refer to the split documentation. hsplit is equivalent to split with axis=1,
3 #the array is always split along the second axis regardless of the array dimension.
4
```

Out[28]:

```
[array([[ 0,  1,  2],
        [ 6,  7,  8],
        [12, 13, 14],
        [18, 19, 20],
        [24, 25, 26],
        [30, 31, 32]]), array([[ 3,  4,  5],
        [ 9, 10, 11],
        [15, 16, 17],
        [21, 22, 23],
        [27, 28, 29],
        [33, 34, 35]])]
```

In [29]:

```
1 np.hsplit(sp,1)
```

Out[29]:

```
[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, 25, 26, 27, 28, 29],
        [30, 31, 32, 33, 34, 35]])]
```

In [7]:

```
1 np.split(sp,2,axis=1) #hsplit
```

Out[7]:

```
[array([[ 0,  1,  2],
        [ 6,  7,  8],
        [12, 13, 14],
        [18, 19, 20],
        [24, 25, 26],
        [30, 31, 32]]), array([[ 3,  4,  5],
        [ 9, 10, 11],
        [15, 16, 17],
        [21, 22, 23],
        [27, 28, 29],
        [33, 34, 35]])]
```

In [8]:

```
1 np.split(sp,2)
```

Out[8]:

```
[array([[ 0,  1,  2,  3,  4,  5],
        [ 6,  7,  8,  9, 10, 11],
        [12, 13, 14, 15, 16, 17]]), array([[18, 19, 20, 21, 22, 23],
        [24, 25, 26, 27, 28, 29],
        [30, 31, 32, 33, 34, 35]])]
```

In [9]:

```
1 np.hsplit(sp,2)
```

Out[9]:

```
[array([[ 0,  1,  2],
        [ 6,  7,  8],
        [12, 13, 14],
        [18, 19, 20],
        [24, 25, 26],
        [30, 31, 32]]), array([[ 3,  4,  5],
        [ 9, 10, 11],
        [15, 16, 17],
        [21, 22, 23],
        [27, 28, 29],
        [33, 34, 35]])]
```

In [10]:

```
1 sp
```

Out[10]:

```
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, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35]])
```

In [13]:

```
w=np.hsplit(sp,3)
w
```

Out[13]:

```
[array([[ 0,  1],
        [ 6,  7],
        [12, 13],
        [18, 19],
        [24, 25],
        [30, 31]]), array([[ 2,  3],
        [ 8,  9],
        [14, 15],
        [20, 21],
        [26, 27],
        [32, 33]]), array([[ 4,  5],
        [10, 11],
        [16, 17],
        [22, 23],
        [28, 29],
        [34, 35]])]
```

In [16]:

```
np.shape(w)
```

Out[16]:

```
(3, 6, 2)
```

In [17]:

```
np.shape(sp)
```

Out[17]:

```
(6, 6)
```

In [21]:

```
y=np.hsplit(sp,1)
print(y)
print(np.shape(y))
```

```
[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, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35]])]
(1, 6, 6)
```

In [22]:

```
q=y=np.hsplit(sp,2)
print(q)
print(np.shape(q))
```

```
[array([[ 0,  1,  2],
       [ 6,  7,  8],
       [12, 13, 14],
       [18, 19, 20],
       [24, 25, 26],
       [30, 31, 32]]), array([[ 3,  4,  5],
       [ 9, 10, 11],
       [15, 16, 17],
       [21, 22, 23],
       [27, 28, 29],
       [33, 34, 35]])]
(2, 6, 3)
```

In [23]:

```
s=y=np.vsplit(sp,1)
print(s)
print(np.shape(s))
```

```
[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, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35]])]
(1, 6, 6)
```

In [24]:

```
t= np.vsplit(sp,2)
print(t)
print(np.shape(t))
```

```
[array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17]]), array([[18, 19, 20, 21, 22, 23],
       [24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35]])]
(2, 3, 6)
```

In [25]:

```
u= np.vsplit(sp,3)
print(u)
print(np.shape(u))
```

```
[array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11]]), array([[12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23]]), array([[24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35]])]
(3, 2, 6)
```

In [28]:

```
a = np.arange(6.0)
np.split(a, [4, 6, 6, 7])
```

Out[28]:

```
[array([0., 1., 2., 3.]),
 array([4., 5.]),
 array([], dtype=float64),
 array([], dtype=float64),
 array([], dtype=float64)]
```

In [27]:

```
a
```

Out[27]:

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

In [31]:

```
a = np.arange(7.0)  #till 6 tk values lega
np.split(a, [4, 5, 6, 7])
```

Out[31]:

```
[array([0., 1., 2., 3.]),
 array([4.]),
 array([5.]),
 array([6.]),
 array([], dtype=float64)]
```

In [33]:

```
np.arange(12)
```

Out[33]:

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

In [ ]:

```

```