What is numpy? (Numerical Python)

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

At the core of the NumPy package, is the ndarray object. This encapsulates n-dimensional arrays of homogeneous data types

Numpy Arrays Vs Python Sequences

- NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an ndarray will create a new array and delete the original.
- The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory.
- NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python's built-in sequences.
- A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays.
- initally data Science pe kaam MATLAB jaise tools ya R programming language me kya jata tha (approx 2008-09) uss time pe python ko prefer nahi kya jata tha data Science ke lye qki ye bahut slow language tha tabhi Numpy Library banaya (basically numpy likha gya hai c me) jisme ki array ka concept aaya jisse ki ye language bhi bahut fast ho gya

creating Numpy Array

```
# Importing the Numpy
import numpy as np  # np(alies name) ek short name dye hain bass taki aage baar baar Numpy nahi likhna pade np se kaam chal jaye

#1D array   -> simply karna kuch nahi hai bass 1D list provide kara dena hai
a=np.array([1,2,3])
print(a)
print(type(a))  # 1D also called as  vector
```

```
[1 2 3]
<class 'numpy.ndarray'>
```

```
a=np.array(['jay','ram'])
a
```

```
array(['jay', 'ramjsdb'], dtype='<U7')
```

```
a=np.array(['jay','ram',2])
a
```

```
array(['jay', 'ram', '2'], dtype='<U11')
```

```
#2D & 3D array   -> simply karna kuch nahi hai bass 2D & 3D list provide kara dena hai
b=np.array([[1,2,3],[3,4,5]])
print(b)
print(type(b))   # 2D also called as  matrix
```

```
[[1 2 3]
 [3 4 5]]
<class 'numpy.ndarray'>
```

```
# ndmin     =>aap apne according dimension define karte ho
arr=np.array([1,2,3,4],ndmin=10)
arr
```

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

```
# dtype => apne man se array ka type bana sakte hain by default int hota hai isse float , bool , complex etc bana sakte hain
np.array([1,2,3],dtype=float)
```

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

```
np.array([1,2,3,0],dtype=bool)
```

```
array([ True,  True,  True, False])
```

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

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

```
# np.arange => similar as range function
np.arange(1,11)
```

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

```
np.arange(1,11,2)
```

```
array([1, 3, 5, 7, 9])
```

```
# with reshape  => basically reshape karne me kaam aata hai
np.arange(1,11).reshape(2,5)  # ye sare element ko jo 1D me hona tha usko reshape karke 2D me with 2-rows and 5-columns me convert kar
```

```
         # dega ye tabhi hoga jabki possible hai ye nahi ki hum bale 5 x 5 kar do jabki itne element hi nahi hain humare pass
```

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

```
np.arange(1,11).reshape(5,2)
```

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

```
# np.ones and np.zeros => aap isko use karke ek numpy array bana sakte ho jiske sare ke sare element either zero ho ya one bass aapko tuples me 2D array
#ka size dena hai
np.ones((3,4))
```

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

```
np.zeros((3,4))
```

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

```
# yadi kisi random number se initialize karana ho to
#np.random
np.random.random((3,4))
```

```
array([[0.65367038, 0.61962895, 0.36920657, 0.82015751],
       [0.56679943, 0.17207025, 0.70460235, 0.80658213],
       [0.95347863, 0.40823868, 0.16345232, 0.82663729]])
```

```
#np.linspace =>linear space isme aap 3 chize provide karate ho (i) lower range ,ii) heigher range iii) the total number that you want
np.linspace(-10,10,10)  #-10 se 10 ke bich me 10 number equal distance pe de dega(initial and final both are included) matlab 2 ka
                        # difference humesa same hoga
```

```
array([-10.        ,  -7.77777778,  -5.55555556,  -3.33333333,
       -1.11111111,   1.11111111,   3.33333333,   5.55555556,
        7.77777778,  10.        ])
```

```
# by default sara  number float me hota hai but agar aapko int me chahaiye to bass {dtype=int} likh do
np.linspace(-10,10,10,dtype=int)
```

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

# np.identity => to print identity matrix
np.identity(3)  # bass size paass kar do

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

Array Attributes

- har numpy array numpy class ka object hai toh har numpy array jo aap bana rahe ho wo numpy class ki kuch arrtibutes ko access kar sakta hai

```
a1=np.arange(10,dtype=np.int32) # dtype compulsury nahi hai but agar aap 32 bit bale ke sath ya 64 bit ke sath kaam kaena chahate hain ypou choice
a2=np.arange(12,dtype=float).reshape(3,4)
a3=np.arange(8,dtype=np.int64).reshape(2,2,2)  # simply (2,2,2) batata hai ki 2x2 size ka andar me 2 (pahla number) 2D array hai

a1 #1D array  or vector

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

a2 #2D array

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

a3  #3D array or tensor

array([[[0, 1],
        [2, 3]],

       [[4, 5],
        [6, 7]]])

# ndim (Number of Dimension)  => simply ki aapka jo array hai wo kitne dimension ka hai
a1.ndim

1

a2.ndim

2

a3.ndim

3
```

```
# shape (ye batayega ki har dimension me kitne item hain)
a1.shape
```

```
(10,)
```

```
a2.shape
```

```
(3, 4)
```

```
a3.shape
```

```
(2, 2, 2)
```

```
# size => number of items kitne hain
a1.size
```

```
10
```

```
a2.size
```

```
12
```

```
a3.size
```

```
8
```

```
# itemsize => basically batata hai ki memory me har item kitna size occupy karta hai
a1.itemsize
```

```
4
```

```
a2.itemsize
```

```
8
```

```
a3.itemsize
```

```
8
```

```
# dtype =>ye batata hai ki aapke item ka datatype kya hai
print(a1.dtype)
print(a2.dtype)
print(a3.dtype)
```

```
int32
float64
int64
```

changing Datatype

```
# astype  => you can change your data type
a4=a3.astype(np.int32)
a4
```

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

       [[4, 5],
        [6, 7]]])
```

```
print(a4.dtype)
```

```
int32
```

```
print(a3.dtype)   # it shows ki changes original bale me nahi hota hai
```

```
int64
```

```
a5=a3.astype(float)
a5
```

```
array([[[0., 1.],
        [2., 3.]],

       [[4., 5.],
        [6., 7.]]])
```

```
print(a5.dtype)
```

```
float64
```

Array Operations

```
a1 = np.arange(12).reshape(3,4)
a2 = np.arange(12,24).reshape(3,4)
```

```
a2
```

```
array([[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

```
## scalar Operation
```

```
# arithmetic  => kisi particular number se arithmatic operation kara do
a1
```

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

```
a1*2
```

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

```
a1+2
```

```
array([[ 2,  3,  4,  5],
       [ 6,  7,  8,  9],
       [10, 11, 12, 13]])
```

```
a1-2
```

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

```
a1/2
```

```
array([[0. , 0.5, 1. , 1.5],
       [2. , 2.5, 3. , 3.5],
       [4. , 4.5, 5. , 5.5]])
```

```
a1//2
```

```
array([[0, 0, 1, 1],
       [2, 2, 3, 3],
       [4, 4, 5, 5]], dtype=int32)
```

```
a1%2
```

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

```
a1**2
```

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

```
# relational  => sabke sab relational operator apply kar sakte ho
a2
```

```
array([[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

```
a2>5  # each element se compare karega ki a2 ka jo each element hai wo 5 se grater hai ki nahi and ek boolean array return karega
```

```
array([[ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True]])
```

```
a2>15
```

```
array([[False, False, False, False],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True]])
```

```
a2==15
```

```
array([[False, False, False,  True],
       [False, False, False, False],
       [False, False, False, False]])
```

```
# vector operation  => shape same hona chahaiye


# arithmetic
```

```
a1
```

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

```
a2
```

```
array([[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

```
a1+a2
```

```
array([[12, 14, 16, 18],
       [20, 22, 24, 26],
       [28, 30, 32, 34]])
```

```
a1*a2  # corrosponding element product hoga
```

```
array([[  0,  13,  28,  45],
       [ 64,  85, 108, 133],
       [160, 189, 220, 253]])
```

```
a1**a2
```

```
array([[          0,           1,       16384,    14348907],
       [          0, -1564725563,  1159987200,   442181591],
       [          0,  1914644777, -1304428544,  -122979837]])

a1/a2

array([[0.        , 0.07692308, 0.14285714, 0.2       ],
       [0.25      , 0.29411765, 0.33333333, 0.36842105],
       [0.4       , 0.42857143, 0.45454545, 0.47826087]])
```

Array function

```
a1 = np.random.random((3,3))
a1 = np.round(a1*100)

# random number -> rand,randn,ranf,randint

a=np.random.rand(10)
a

array([0.9033302 , 0.85146063, 0.82667666, 0.40605734, 0.66259751,
       0.77309725, 0.09060155, 0.55174489, 0.02175932, 0.477554  ])

a=np.random.rand(2,4)
a

array([[0.40843536, 0.98472908, 0.15423192, 0.05024507],
       [0.84223149, 0.11024095, 0.62546642, 0.62493099]])

a=np.random.ranf(10)
a

array([0.50113025, 0.50028113, 0.92899553, 0.91632737, 0.77442113,
       0.94794648, 0.69412541, 0.39085533, 0.58093387, 0.08175353])

a=np.random.ranf(2,4)
a

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call
last)
Cell In[14], line 1
----> 1 a=np.random.ranf(2,4)
      2 a

File numpy\\random\\mtrand.pyx:4879, in numpy.random.mtrand.ranf()

File numpy\\random\\mtrand.pyx:385, in
numpy.random.mtrand.RandomState.random_sample()
```

```
TypeError: random_sample() takes at most 1 positional argument (2
given)
```

```
a=np.random.ranf((2,4))  # agar multidimension chahaiye to aap simply
ek tuple pass kar do
a
```

```
array([[0.98842586, 0.85284128, 0.65288041, 0.75446212],
       [0.35332838, 0.54494872, 0.02990169, 0.11542688]])
```

```
a=np.random.randn(4)  # random negative
a
```

```
array([-0.66633551,  0.39187368, -1.4242592 , -0.23321945])
```

```
a=np.random.randint(18,25,10)  # 18-25 ke bich me 10 integer value
```

```
a
```

```
array([21, 24, 18, 22, 23, 23, 23, 21, 20, 24])
```

```
a1
```

```
array([[68., 40., 82.],
       [11., 29., 36.],
       [84., 63., 66.]])
```

```
# max/min/sum/prod
np.max(a1)
```

```
84.0
```

```
np.min(a1)
```

```
11.0
```

```
np.sum(a1)  # jitna bhi element hai array me sabka sum find karke de
dega
```

```
479.0
```

```
np.prod(a1)   # jitna bhi element hai array me sabka product find
karke de dega
```

```
894622283089920.0
```

```
a1
```

```
array([[68., 40., 82.],
       [11., 29., 36.],
       [84., 63., 66.]])
```

```
# hum each row ke sath bhi kaam kar sakte hain matlab ki agar need hai ki each row ya each column ka max/min element find karo ya each row
# ya each column ka sum/product

# axis=0 -> col and 1-> row
np.max(a1,axis=0)  # columne wise maximum element fing karke de dega
```

```
array([84., 63., 82.])
```

```
np.min(a1,axis=0)
```

```
array([11., 29., 36.])
```

```
np.sum(a1,axis=0)
```

```
array([163., 132., 184.])
```

```
np.prod(a1,axis=0)
```

```
array([ 62832.,  73080., 194832.])
```

```
np.max(a1,axis=1)
```

```
array([82., 36., 84.])
```

```
np.min(a1,axis=1)
```

```
array([40., 11., 63.])
```

```
np.sum(a1,axis=1)
```

```
array([190.,  76., 213.])
```

```
np.prod(a1,axis=1)
```

```
array([223040.,  11484., 349272.])
```

```
# mean/median/std/var

np.mean(a1)  # sare element ka mean find karke de dega
```

```
53.22222222222222
```

```
# and ofcourse yaha bhi flexibility hai row wise and column wise bhi find kar sakte ho
np.mean(a1,axis=1)
```

```
array([63.33333333, 25.33333333, 71.        ])
```

```
np.median(a1)
```

```
63.0
```

```
np.median(a1,axis=1)
```

```
array([68., 29., 66.])
```

```
np.var(a1)  # variance
```

```
565.9506172839506
```

```
np.std(a1)  # standard deviation
```

```
23.789716628912387
```

```
# trigonometric function
np.sin(a1)
```

```
array([[-0.89792768,  0.74511316,  0.31322878],
       [-0.99999021, -0.66363388, -0.99177885],
       [ 0.73319032,  0.1673557 , -0.02655115]])
```

```
# dot product => matrix product
a2=np.arange(12).reshape(3,4)
a3=np.arange(12,24).reshape(4,3)
```

```
np.dot(a2,a3)
```

```
array([[114, 120, 126],
       [378, 400, 422],
       [642, 680, 718]])
```

```
# log and exponent
np.log(a1)  # har element ka log
```

```
array([[4.21950771, 3.68887945, 4.40671925],
       [2.39789527, 3.36729583, 3.58351894],
       [4.4308168 , 4.14313473, 4.18965474]])
```

```
np.exp(a1)  # har element ka exponent
```

```
array([[3.40427605e+29, 2.35385267e+17, 4.09399696e+35],
       [5.98741417e+04, 3.93133430e+12, 4.31123155e+15],
       [3.02507732e+36, 2.29378316e+27, 4.60718663e+28]])
```

```
# round /floop/ceil
a=np.random.random((2,3))*100
```

```
a
```

```
array([[94.92026349,  5.59729856, 66.3819704 ],
       [71.70719749, 44.80901264,  8.01456459]])
```

```
np.round(a)
```

```
array([[95.,  6., 66.],
       [72., 45.,  8.]])
```

```
np.floor(a)
```

```
array([[94.,  5., 66.],
       [71., 44.,  8.]])
```

```
np.ceil(a)
```

```
array([[95.,  6., 67.],
       [72., 45.,  9.]])
```

Indexing and Slicing

```
a1=np.arange(10)
a2=np.arange(12).reshape(3,4)
a3=np.arange(8).reshape(2,2,2)
```

```
a1
```

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

```
# normal jaise pahle indexing hota tha and numpy also support +ve as well -ve indexes
a1[-1]
```

```
9
```

```
a1[0]
```

```
0
```

```
# 2D
a2
```

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

```
#just we have to find 6 => sabse pahle row batao and comma ke baad column batao and row column ka indexing 0 se suru hota hai
a2[1,2]
```

```
6
```

```
# 4
a2[1,0]
```

```
4
```

```
# 3D
a3
```

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

       [[4, 5],
        [6, 7]]])
```

```
# iss 3D array me 2 - 2D array  hai => to pahle ye batao ki kon se 2D array se element find karna hai
# find 5
a3[1,0,1]  # 1st 2D array , 0th row and 1st column
```

```
5
```

```
# 0
a3[0,0,0]
```

```
0
```

```
#6
a3[1,1,0]
```

```
6
```

Slicing => ek sath multiple item ko bhi nikal sakte ho

```
a1
```

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

```
# 1D me sab kuch same hai python kee jaisa
# find 2,3,4
a1[2:5]
```

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

```
a1[2:5:2]
```

```
array([2, 4])
```

```
# 2D
a2
```

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

```
# find 1st row
a2[0,:]  # kon sa row chahaiye and kon sa column chahaiye to mujhe sare column chahaiye 1st row ka to sirf colon(seperate with column)
```

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

```
# find 2nd column
a2[:,2]
```

```
array([ 2,  6, 10])
```

```
# 5,6,
# 9,10
a2[1:,1:3] #=> 1st row se suru karo and 1st and 2nd column since last is exclusive so 3rd column print nahi hoga
```

```
array([[ 5,  6],
       [ 9, 10]])
```

```
 #0,3
 #8,11
a2[::2,::3]  # comma ke pahla bala batata hai ki kon sa row chahaiye to mujhe sare row chahaiye with 1 alternate and comma ke baad
              #bala batata hai ki kon kon sa column chahaiye to sare chahaiye but 2 column alternative
```

```
array([[ 0,  3],
       [ 8, 11]])
```

```
# 1,3
# 9,11
a2[::2,1::2]
```

```
array([[ 1,  3],
       [ 9, 11]])
```

```
# 4,7
a2[1,::3]
```

```
array([4, 7])
```

```
#1,2,3
#5,6,7
a2[0]
```

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

```
a2[0:2,1:]
```

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

```
#1,3
#5,7
a2[0:2,1::2]
```

```
array([[1, 3],
       [5, 7]])
```

```
#3D
a3=np.arange(27).reshape(3,3,3)
a3
```

```
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]]])
```

```
# bich bala array find karo
a3[1]
```

```
array([[ 9, 10, 11],
       [12, 13, 14],
       [15, 16, 17]])
```

```
# 1st and last bala chahaiye
a3[0::2] #or,a3[::2]
```

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

       [[18, 19, 20],
        [21, 22, 23],
        [24, 25, 26]]])
```

```
#1st 2D array ka 1st indexed row (3,4,5)
a3[0,1,:]
```

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

```
#2nd bala array ka 2nd column(10,13,16)
a3[1,:,1]
```

```
array([10, 13, 16])
```

```
#22,23
#25,26
a3[2,1:3,1:3]
```

```
array([[22, 23],
       [25, 26]])
```

```
#0,2
#18,20
a3[::2,0,::2]
```

```
array([[ 0,  2],
       [18, 20]])
```

```
import numpy as np

a=np.arange(9).reshape(3,3)
a
```

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

```
np.transpose(a)
```

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

```
a=np.matrix([[1,2,3],[4,5,6],[7,8,9]])
a
```

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

Iterating

- simply hum Numpy array pe loop chala sakte hain

```
a1
```

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

```
for i in a1:
    print(i)
```

```
0
1
2
3
4
5
6
```

```
7
8
9
```

```
a2
```

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

```
for i in a2:
    print(i)  # ek iteration me ek complete row print hota hai so in this code total iteration is 3
```

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

```
for i in a2:
    print(i)
    print()
```

```
[0 1 2 3]

[4 5 6 7]

[ 8  9 10 11]
```

```
# yadi har element ko seperately print karana hai to
# nditer  => inbult function ka help lena hoga
for i in np.nditer(a2):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
```

```
a3
```

```
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]]])
```

```
for i in a3:
    print(i)  # har iteration me ek 2D array print hota hai
```

```
[[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]]
```

```
for i in a3:
    print(i)
    print()
```

```
[[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]]
```

```
# yadi har element ko seperately print karana hai to
# nditer  => inbult function ka help lene hoga
for i in np.nditer(a3):
    print(i)
```

```
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
```

reshaping

```
# reshape => Already done

# Transpose => simply jaise matrix me kaam karta hai
a2
```

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

```
np.transpose(a2)
```

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

```
# another syntax for transpose
a2.T
```

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

```
# ravel => kitne bhi dimension ke array ko 1D me convert kar deta hai
a2
```

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

```
a2.ravel()
```

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

```
a3
```

```
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]]])
```

```
a3.ravel()
```

```
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])
```

```
# ye saare temporary changes ho raha hai
```

Stacking

- simply multiple array ko ek dusre ke sath chipka sakte hain
- jab multiple jagah se data aata hai and usse analyse karna ho to Stacking ka use karte hain

```
a4=np.arange(12).reshape(3,4)
a5=np.arange(12,24).reshape(3,4)
a4
```

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

```
a5
```

```
array([[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])

# horizontal stacking  => shape same hona jaruri hai
np.hstack((a4,a5))  # andar me tuples ke form me array ko pass karte
hain

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

np.hstack((a4,a5,a4))

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

# vertical Stacking
np.vstack((a4,a5))

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]])
```

Splitting

- array ko kaat deta hai and agar | => kaat raha hai to horizontal spliting and if -- => kaat raha hai to vertical split confuse nahi hona hai
- jab kabhi ek data set se multiple data set create karna ho to iska use karte hain

```
# horizontal Spliting
a4

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

# a4 ko horizontaly 2- parts me split karna hai
np.hsplit(a4,2)  # parameter => 1st - ki kon se array ko split karna
hai &&&& 2nd - ki kitna part me katna hai

[array([[0, 1],
        [4, 5],
        [8, 9]]),
```

```
 array([[ 2,  3],
        [ 6,  7],
        [10, 11]])]

np.hsplit(a4,3) # error qki 3 equal part me split ho hi nahi sakta hai

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call
last)
Cell In[187], line 1
----> 1 np.hsplit(a4,3)

File ~\anaconda3\Lib\site-packages\numpy\lib\shape_base.py:938, in
hsplit(ary, indices_or_sections)
    936     raise ValueError('hsplit only works on arrays of 1 or more
dimensions')
    937 if ary.ndim > 1:
--> 938     return split(ary, indices_or_sections, 1)
    939 else:
    940     return split(ary, indices_or_sections, 0)

File ~\anaconda3\Lib\site-packages\numpy\lib\shape_base.py:864, in
split(ary, indices_or_sections, axis)
    862     N = ary.shape[axis]
    863     if N % sections:
--> 864         raise ValueError(
    865             'array split does not result in an equal
division') from None
    866 return array_split(ary, indices_or_sections, axis)

ValueError: array split does not result in an equal division

# vertical split
a5

array([[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])

np.vsplit(a5,3)

[array([[12, 13, 14, 15]]),
 array([[16, 17, 18, 19]]),
 array([[20, 21, 22, 23]])]
```