```
import numpy as np

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

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

for arrays in output we see arrays written and then square brackets with values but for list we just see the numbers in square brackets

```
arr=np.arange(1,11)
arr

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

sort of array creation defining range

```
arr=np.zeros((6,6))
arr

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

just like this zeros there is another are others like ones and all with which we can generate arrays of those numbers

```
arr=np.linspace(0,1,100)
arr

array([0.        , 0.01010101, 0.02020202, 0.03030303, 0.04040404,
       0.05050505, 0.06060606, 0.07070707, 0.08080808, 0.09090909,
       0.1010101 , 0.11111111, 0.12121212, 0.13131313, 0.14141414,
       0.15151515, 0.16161616, 0.17171717, 0.18181818, 0.19191919,
       0.2020202 , 0.21212121, 0.22222222, 0.23232323, 0.24242424,
       0.25252525, 0.26262626, 0.27272727, 0.28282828, 0.29292929,
       0.3030303 , 0.31313131, 0.32323232, 0.33333333, 0.34343434,
       0.35353535, 0.36363636, 0.37373737, 0.38383838, 0.39393939,
       0.4040404 , 0.41414141, 0.42424242, 0.43434343, 0.44444444,
       0.45454545, 0.46464646, 0.47474747, 0.48484848, 0.49494949,
       0.50505051, 0.51515152, 0.52525253, 0.53535354, 0.54545455,
       0.55555556, 0.56565657, 0.57575758, 0.58585859, 0.5959596 ,
       0.60606061, 0.61616162, 0.62626263, 0.63636364, 0.64646465,
       0.65656566, 0.66666667, 0.67676768, 0.68686869, 0.6969697 ,
       0.70707071, 0.71717172, 0.72727273, 0.73737374, 0.74747475,
       0.75757576, 0.76767677, 0.77777778, 0.78787879, 0.7979798 ,
       0.80808081, 0.81818182, 0.82828283, 0.83838384, 0.84848485,
       0.85858586, 0.86868687, 0.87878788, 0.88888889, 0.8989899 ,
       0.90909091, 0.91919192, 0.92929293, 0.93939394, 0.94949495,
       0.95959596, 0.96969697, 0.97979798, 0.98989899, 1.        ])
```

lin space is different from range as the first 2 elements are the range between which we want values and the last third tells us how many numbers we want between those numbers and all number are linearly spaced means the diff between each element is same

```
np.random.rand(5)

array([0.06514258, 0.3135005 , 0.24933622, 0.02251835, 0.42676   ])
```

rand creates 5 random numers ,its linear a vector nota a matrix as matrix usually ends with two brackets . this is a case of normalization as the values are between 0 and 1 as for the case of standardization the values lie between 3 to -3

```
np.random.randn(10)

array([-0.5067467 , -1.27746252,  0.33335425, -1.01639158,
1.25048165,
       -0.34883539, -0.36164974,  1.1418113 , -0.01675214, -
0.09445982])
```

this is the case to have standardization we use randn instead of just rand

```
np.random.randint(6)

3
```

this number 6 tells that i want a random number between 0 and that number

```
np.random.randint(10,20)

16
```

gives a random number between 10 and 20

```
np.random.randint(10,20,10)

array([16, 19, 14, 19, 17, 19, 10, 18, 17, 18], dtype=int32)
```

10 and 20 are the range between which i want numbewrs and the last number 10 tells us how ,

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

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

```
arr.shape
```

```
(3, 3)
```

means it gives 3 rows and 3 columns , these are attributes like shape size,etc they are written without brackets

```
arr.size
```

```
9
```

tells how many elements are there in array

```
arr.dtype
```

```
dtype('int64')
```

tells all are integer type

```
arr
```

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

```
arr.min()
```

```
np.int64(1)
```

```
arr.max()
```

```
np.int64(9)
```

```
arr.sum()
```

```
np.int64(45)
```

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

```
array([12, 15, 18])
```

axis =0 gives column wise sum result and axis=1 will give row wise sum result

```
arr.mean()
```

```
np.float64(5.0)
```

```
arr.std()
```

```
np.float64(2.581988897471611)
```

```
arr.argmax()
```

```
np.int64(8)
```

gives index of the maximum element

```
arr.argmin()
```

```
np.int64(0)
```

index of the minimum number

```
arr=np.arange(1,31)
arr
```

```
array([ 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])
```

```
arr.reshape(6,5)
```

```
array([[ 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]])
```

while reshaping we have to keep in mind the numbers we put should include all the elements

```
arr=np.arange(11,21)
arr
```

```
array([11, 12, 13, 14, 15, 16, 17, 18, 19, 20])
```

```
arr[5]
```

```
np.int64(16)
```

```
arr[1:5]
```

```
array([12, 13, 14, 15])
```

```
arr_slice=arr[3::2]
arr_slice
```

```
array([14, 16, 18, 20])
```

```
arr=np.arange(1,31).reshape(6,5)
arr
```

```
array([[ 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]])
```

```
arr[5]
```

```
array([26, 27, 28, 29, 30])
```

on giving a single digit in case of matrix it will target rows

```
arr[0,0]
```

```
np.int64(1)
```

gives the element at 0 row and 0 column

```
slice=arr[0:2,1:3]
slice
```

```
array([[2, 3],
       [7, 8]])
```

rows and columns are separately sliced as we want the element from 0 to 1 row and column from 1 to 2 column

```
slice2=arr[3:,3: ]
slice2
```

```
array([[19, 20],
       [24, 25],
       [29, 30]])
```

```
arr[:,2]
```

```
array([ 3,  8, 13, 18, 23, 28])
```

this way we have elements of a column rows we ant to select all so colon is used

```
arr=np.arange(11,21)
arr
```

```
array([11, 12, 13, 14, 15, 16, 17, 18, 19, 20])
```

```
bool_index=arr%2==0
bool_index
```

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

```
arr=arr[bool_index]
arr
```

```
array([12, 14, 16, 18, 20])
```

returns the number that return true

```
a1=np.array([1,2,3,4])
a2=np.array([5,6,7,8])
```

```
a1+a2
```

```
array([ 6,  8, 10, 12])
```

if u want to perform array operations size of both arrays must be same

```
l=[10,20,30,40]
arr=np.array(l)
```

```
arr+10
```

```
array([20, 30, 40, 50])
```

in case of list we need to use a loop and then add ten to each element but such is not the case of array we can directly do that using broadcasting

```
arr2=np.arange(1,26).reshape(5,5)
arr2
```

```
array([[ 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]])
```

```
arr2+10
```

```
array([[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]])
```

```
a=np.arange(1,21)
a
```

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

```
slice=a[:5]
slice=slice*10
slice
```

```
array([10, 20, 30, 40, 50])
```

```
a
```

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

no changes in a but if we do the same thing in list it will show , this is the concept of shallow copy

```
a
b=a
```

```
b[0]=99
```

```
b
```

```
array([99,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
17,
       18, 19, 20])
```

```
a
```

```
array([99,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
17,
       18, 19, 20])
```

concept of deep copy

```
a
b=a.copy()
```

```
b[0]=89
```

```
b
```

```
array([89,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
17,
       18, 19, 20])
```

```
a
```

```
array([99,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
17,
       18, 19, 20])
```

this is the concept of shallow copy

```
A=np.array([[1,2],[3,4]])
B=np.array([[5,6],[7,8]])

A

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

B

array([[5, 6],
       [7, 8]])

A@B

array([[19, 22],
       [43, 50]])
```

this gives the multiplication of 2 matrix

```
np.dot(A,B)

array([[19, 22],
       [43, 50]])
```

its a dot product this also gives the same answer

```
A.T

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

this gives the transpose of a amtrix meaning elemets get rearranged rows become columns

```
A

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

B

array([[5, 6],
       [7, 8]])
```

you cannot stack 2 matrix

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

np.vstack((a,b))

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

we need to pass it in tuple form as it can take only one argument and vstyack means verticle stack means row wise stack

```
np.hstack((a,b))

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

np.column_stack((a,b))

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

to keep the matrix in 2d format

```
c=np.arange(0,16).reshape(4,4)
c

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

np.hsplit(c,2)

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

hsplit splits horizontally c is the array and 2 tells how many parts we want to split it in

```
np.vsplit(c,2)

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

v split is vertical split and the number you write to tell how many divisions you want should be such that it results in equal splits else it would give error like if we write 3 it will give error as it does not split it in equal parts

VALID SUDOKU

```
s=np.array([
           [5,3,4,6,7,8,9,1,2],
            [6,7,2,1,9,5,3,4,8],
            [1,9,8,3,4,2,5,6,7],

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

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

b=np.sum(s,axis=1)
for i in b:
    if i!=45:
        print("sudoku is not valid")
        break
else:
    print("sudoku is currently valid for rows")
```

```
sudoku is currently valid for rows
```

```
c=np.sum(s,axis=0)
for i in b:
    if i!=45:
        print("sudoku is not valid")
        break
else:
    print("sudoku is currently valid for columns")
```

```
sudoku is currently valid for columns
```

```
for i in range(0,9,3):
    for j in range(0,9,3):
        n=s[i:i+3,j:j+3]
        print(n.sum())
```

```
45
45
45
45
45
45
45
45
45
```

this is done to check if all 3 by 3 columns have the same sum

GENERAL QUESTION

columns show [Age,Math marks,Science Marks]

```
data=np.array([
    [18,85,78],
    [19,92,88],
    [17,76,95],
    [18,65,70],
    [20,90,85]
])

#get shape of matrix
data.shape
```

```
(5, 3)
```

```
#get average age of students
np.mean(data[:,0])
```

```
np.float64(18.4)
```

```
#extract math marks of all students
data[:,1]
```

```
array([85, 92, 76, 65, 90])
```

```
#find the highest science marks
np.max(data[:,2])
```

```
np.int64(95)
```

```
#details of the students who scored more than 90 in maths
data[data[:,1]>90]
```

```
array([[19, 92, 88]])
```

```
#increase math marks of all students by 5
data[:,1]+=5
data
```

```
array([[18, 90, 78],
       [19, 97, 88],
       [17, 81, 95],
       [18, 70, 70],
       [20, 95, 85]])
```

```
#find how many students are younger than 19
len(data[data[:,0]<19])
```

```
3
```

```
#calculate the average marks in each subject (column wise mean)
np.mean(data[:,1:],axis=0)
```

```
array([86.6, 83.2])
```

```
#get data of students who scored atleast 80 in both subject
data[(data[:,1]>=80)&(data[:,2]>=80)]
```

```
array([[19, 97, 88],
       [17, 81, 95],
       [20, 95, 85]])
```

```
#replace all science marks<75 with 0
data[:,2][data[:,2]<75]=0
data
```

```
array([[18, 90, 78],
       [19, 97, 88],
       [17, 81, 95],
       [18, 70,  0],
       [20, 95, 85]])
```