+ Code + Text

```
#creating numpy/n-d arrays
import numpy as np

arr1 = np.array([1,2,3,4,5,6])
arr1
```

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

```
type(arr1)
```

```
numpy.ndarray
```

```
#this is a 2d array
arr2 = np.array([[1,2,3],[4,5,6]])
arr2
```

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

```
arr3 = np.zeros((2,3))
arr3
```

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

```
arr4 = np.ones((3,3))
arr4
```

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

```
arr5 = np.identity(5)
arr5
```

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

```
arr6 = np.arange(10)
arr6
```

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

```
arr7 = np.arange(5,16,2)
arr7
```

```
array([ 5,  7,  9, 11, 13, 15])
```

```
arr8 = np.linspace(10,20,10)
arr8
```

```
array([10.        , 11.11111111, 12.22222222, 13.33333333, 14.44444444,
       15.55555556, 16.66666667, 17.77777778, 18.88888889, 20.        ])
```

```
arr9 = arr8.copy()
arr9
```

```
array([10.        , 11.11111111, 12.22222222, 13.33333333, 14.44444444,
       15.55555556, 16.66666667, 17.77777778, 18.88888889, 20.        ])
```

```
arr10 = np.array([[[1,2],[3,4]],[[1,2],[3,4]]])
arr10
```

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

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

```
arr10.shape
```

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

```
arr10.ndim
```

```
3
```

```
arr10.size
```

```
8
```

```
arr10.itemsize
```

```
8
```

```
arr10.dtype
```

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

```
arr10.astype
```

```
<function ndarray.astype>
```

```
#convert int to float
arr10.astype('float')
```

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

       [[1., 2.],
        [3., 4.]]])
```

```
#python list vs numpy array
#np arrays are faster, convinient and less memory consumption
lista = range(100)
array_a = np.arange(100)

import sys

print(sys.getsizeof(87)*len(lista))
print("array size",array_a.itemsize*array_a.size)
```

```
2800
array size 800
```

```
#for time diference...dekho ab
import time
x = range(10000000)
```

```
y = range(10000000,20000000)
start = time.time()
c= [(x+y) for x,y in zip(x,y)]
print(time.time()-start)
```

```
1.4595870971679688
```

```
x = np.arange(10000000)
y = np.arange(10000000,20000000)
start = time.time()
c= x+y
print(time.time()-start)
```

```
0.10508298873901367
```

```
arr11 = np.arange(24).reshape(6,4)
arr11
```

```
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]])
```

```
#row printing
arr11[:2]
```

```
#column printing
arr11[:,2:3]
```

```
#middle name
arr11[2:4,1:3]
```

```
array([[ 9, 10],
       [13, 14]])
```

```
for i in arr11:
  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]
```

```
#for printing proper inside numbers we have builtin nditer function
for i in np.nditer(arr11):
  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
```

```
#Numpy Operations
import numpy as np
arr1 = np.array([1,2,3,4,5,6])
arr2 = np.array([4,5,6,7,8,9])

#subtraction
arr1-arr2
#multiplication
arr1*arr2
#greater than
arr1>3  #output will be false
```

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

```
#now arange function acts like for eg np.arange(6,12) will give 6-12
arr2 = np.arange(6).reshape(2,3)
arr3 = np.arange(6,12).reshape(3,2)
#arr3
arr2.dot(arr3)
```

```
array([[ 28,  31],
       [100, 112]])
```

```
# to find largest number from an array
#arr2.max()

arr2.min()
```

```
0
```

```
#x axis is zero column wise
arr2.min(axis=0)
```

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

```
#x axis is one row wise
arr2.min(axis=1)
```

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

```
arr2.sum()
```

```
15
```

```
arr2.sum(axis=0)
```

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

```
arr2.mean()
```

```
2.5
```

```
arr2.std()
```

```
1.707825127659933
```

```
np.median(arr2)
```

```
np.sin(arr2)
```

```
array([[ 0.        ,  0.84147098,  0.90929743],
       [ 0.14112001, -0.7568025 , -0.95892427]])
```

```
np.exp(arr2)
```

```
array([[  1.        ,   2.71828183,   7.3890561 ],
       [ 20.08553692,  54.59815003, 148.4131591 ]])
```

```
#ravel function is used to convert ndimensional array to 1d.
arr2.ravel()
```

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

```
arr2.transpose()
```

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

```
#stacking is appending the array on horizontal or vertical with the
import numpy as np
arr6= np.arange(12,18).reshape(2,3)
arr7= np.arange(20,26).reshape(2,3)

#np.hstack((arr6,arr7))
#array([[12, 13, 14, 20, 21, 22],
      #  [15, 16, 17, 23, 24, 25]])
```

```
np.vstack((arr6,arr7))
```

```
array([[12, 13, 14],
       [15, 16, 17],
       [20, 21, 22],
       [23, 24, 25]])
```

```
#splitting splits the array into equal parts
import numpy as np
np.hsplit(arr6,3)
np.vsplit(arr6,2)
```

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

```
#fancy indexing in numpy
arr6= np.arange(24).reshape(6,4)
arr6[[0,2,4]]
```

```
array([[ 0,  1,  2,  3],
       [ 8,  9, 10, 11],
       [16, 17, 18, 19]])
```

```
#indexing with boolean array
arr = np.random.randint(low =1 , high=100, size = 20).reshape(4,5)
#index overlapping as in print only those numbers whose value is grea
arr[arr>30]
```

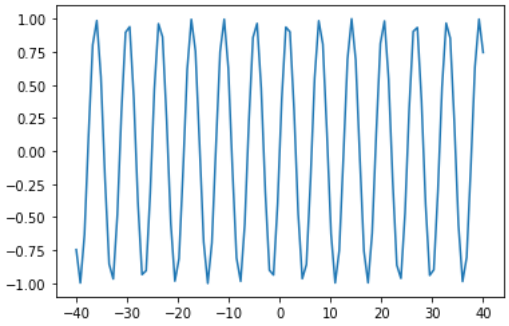
```
arr[(arr>30) & (arr%2!=0)] = 0
arr
```

```
array([[30, 68, 18, 64, 70],
       [80, 50,  0, 15, 66],
       [52, 18,  0,  0,  0],
       [82, 52, 62,  6,  0]])
```

```
#plotting graphs using numpy
x = np.linspace(-40,40,100)
y = np.sin(x)
```

```
import matplotlib.pyplot as plt
#this below line is used to show the graph. For the graph to be on s
%matplotlib inline
plt.plot(x,y)
```

```
[<matplotlib.lines.Line2D at 0x7f2183960cd0>]
```




```
y = x * x+2 * x+6
plt.plot(x,y)
```

```
#Broadcasting concept in numpy. Arithmetic operations can take place
#It adjust the size and then carries the operations
#note that one axis i.e y should be same
arr1  = np.arange(6).reshape(2,3)
arr2 = np.arange(10,13).reshape(1,3)
arr1+arr2
```

```
#random
import numpy as np
np.random.random()
```

```
#if you want a constant random value then seed function is used
np.random.seed(6)
```

```
np.random.random()
```

```
0.8928601514360016
```

```
#if you wnat any random function within given range then uniform fun
np.random.uniform(3,10)
```

```
3.3813215547564823
```

```
#if you wnat any array of random function within given range then un
np.random.uniform(3,10,6).reshape(2,3)
```

```
array([[8.03046066, 8.61519394, 8.15484651],
       [7.96392226, 6.78655798, 3.87376922]])
```

```
#if you wnat any array of integer random function within given range
np.random.randint(3,10,6).reshape(2,3)
```

```
array([[8, 5, 5],
       [9, 3, 8]])
```

```
#basic functions on 1d randint array
a = np.random.randint(3,10,6)

np.max(a)
```

```
6
```

```
np.min(a)
```

```
3
```

```
#index of maximun value
np.argmax(a)
```

```
0
```

```
#np.where is like conditional statement if condition is true then wha
np.where(a%2==1,-1,a)

#sort funtion does the sorting
np.sort(a)

#percentile is the function which will gives us that number below is
np.percentile(a,50)
#i.e. 50% numbers in this array is less than 4.5 and rest 50% is grea
```

```
4.5
```