Numpy array vs Python lists

- speed comparision

```
# list
a=[i for i in range(10000000)]
b=[i for i in range(10000000,20000000)]

c=[]
import time

start = time.time()

for i in range(len(a)):
    c.append(a[i] + b[i])

print(time.time() - start)
```

```
3.019468307495117
```

```
# list execute hone me approx 3 sec ka time le raha hai

# numpy array
import numpy as np
a=np.arange(10000000)
b=np.arange(10000000,20000000)
start = time.time()
c=a+b
print(time.time() - start)
```

```
0.02157139778137207
```

```
# but numpy array same kaam ko karne me bass 0.02 second le raha hai

3.01/0.02
```

```
150.5
```

```
# approx 150 times faster hai array iss scenario me
```

- memory comparision

```
# list
a=[i for i in range(10000000)]

import sys

sys.getsizeof(a) # ye memory me kitna size occupy kar raha hai wo
nikal kar de deta hai (bytes me)
```

```
89095160
```

```
# numpy array
a=np.arange(10000000)     # isme hum dtype ke help se need ke according size decrease bhi kar sakte hain
sys.getsizeof(a)
```

```
40000112
```

```
a=np.arange(10000000,dtype=np.int64)  # waise approx memory same hi use karta hai but ye aacha hai qki isme size change karne ka option hai
sys.getsizeof(a)
```

```
80000112
```

```
a=np.arange(10000000,dtype=np.int32)  # waise approx memory same hi use karta hai but ye aacha hai qki isme size change karne ka option hai
sys.getsizeof(a)
```

```
40000112
```

```
a=np.arange(10000000,dtype=np.int16)  # waise approx memory same hi use karta hai but ye aacha hai qki isme size change karne ka option hai
sys.getsizeof(a)
```

```
20000112
```

```
a=np.arange(10000000,dtype=np.int8)  # waise approx memory same hi use karta hai but ye aacha hai qki isme size change karne ka option hai
sys.getsizeof(a)
```

```
10000112
```

- convineance => syntax is easy in numpy array to use

inn summary

- numpy array is much faster then the python list
- numpy array takes less memory as compare to list bcos of flexibily to cahnge dtype in numpy array
- numpy array is convineance to use bcos of sytax is easy compare to list

Advance indexing and slicing

```
# normal indexing and slicing

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

```
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]])
```

```
a[1,2]
```

```
6
```

```
a[1:3,1:3]
```

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

fancy indexing

```
a
```

```
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]])
```

```
# 1st,3rd and 4th row nikal ke dikhao -> ye aap normal indexing se
nahi nikal sakte
```

```
a[[0,2,3]]  # fancy indexing me simply ek list pass kar dete hain jis
jis row ka bhi need hota hai
```

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

```
# 1st,3rd,4th,6th row
a[[0,2,3,5]]
```

```
array([[ 0,  1,  2,  3],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [20, 21, 22, 23]])
```

```
# 1st,3rd qnd 4th column
a[:,[0,2,3]]  # sare row chahiye so colon(:)
```

```
array([[ 0,  2,  3],
       [ 4,  6,  7],
       [ 8, 10, 11],
       [12, 14, 15],
       [16, 18, 19],
       [20, 22, 23]])
```

boolean indexing

```
# Boolean Indexing
a = np.random.randint(1,100,24).reshape(6,4)   # 1 se 100 ke bich me 24 random element
a
```

```
array([[32, 59, 96, 64],
       [96, 86, 52, 46],
       [27, 35, 89, 16],
       [80, 72, 49, 74],
       [93, 63, 39, 34],
       [13, 75, 17, 83]])
```

```
# find all numbers greater than 50
a > 50
```

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

```
a[a>50]   # jaha jaha pe true hai wo element rah jayega baki sab remove ho jayega
```

```
array([59, 96, 64, 96, 86, 52, 89, 80, 72, 74, 93, 63, 75, 83])
```

```
# find out even numbers
a[a%2==0]
```

```
array([32, 96, 64, 96, 86, 52, 46, 16, 80, 72, 74, 34])
```

```
  (a>50) & (a%2==0)    # yaha pe bitwise and islye qki a>50 se ek boolean array milega jisme ki true and false hai and  a%2==0 se se ek boolean array milega so boolean
                       #value ko compare karne ke lye hum bitwise and ka use karenge
```

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

```
# find all numbers greater than 50 and are even
a[(a>50) & (a%2==0)]
```

```
array([96, 64, 96, 86, 52, 80, 72, 74])
```

```
# find all numbers not divisible by 7
a[~(a%7==0)]
```

```
array([32, 59, 96, 64, 96, 86, 52, 46, 27, 89, 16, 80, 72, 74, 93, 39,
34,
       13, 75, 17, 83])
```

Broadcasting

The term broadcasting describes how NumPy treats arrays with different shapes during arithmetic operations.

The smaller array is “broadcast” across the larger array so that they have compatible shapes. (basically hota kya hai chote size ke array ko broadcast karke bade size ke array ke compatible banata hai)

```
# broadCasting batata hai ki agar do alag alag  shape ke array hai to
usse kaise treate karte hain arithmetic operation kee time

# same shape
import numpy as  np
a = np.arange(6).reshape(2,3)
b = np.arange(6,12).reshape(2,3)
print(a)
print(b)
print(a+b)
```

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

```
# diff shape
a = np.arange(6).reshape(2,3)
b = np.arange(3).reshape(1,3)

print(a)
print()
print(b)
print()
print(a+b) # do alag alag shape ka array hai waise to addition
possible nahi hota hai but numpy me ye feature hai
```

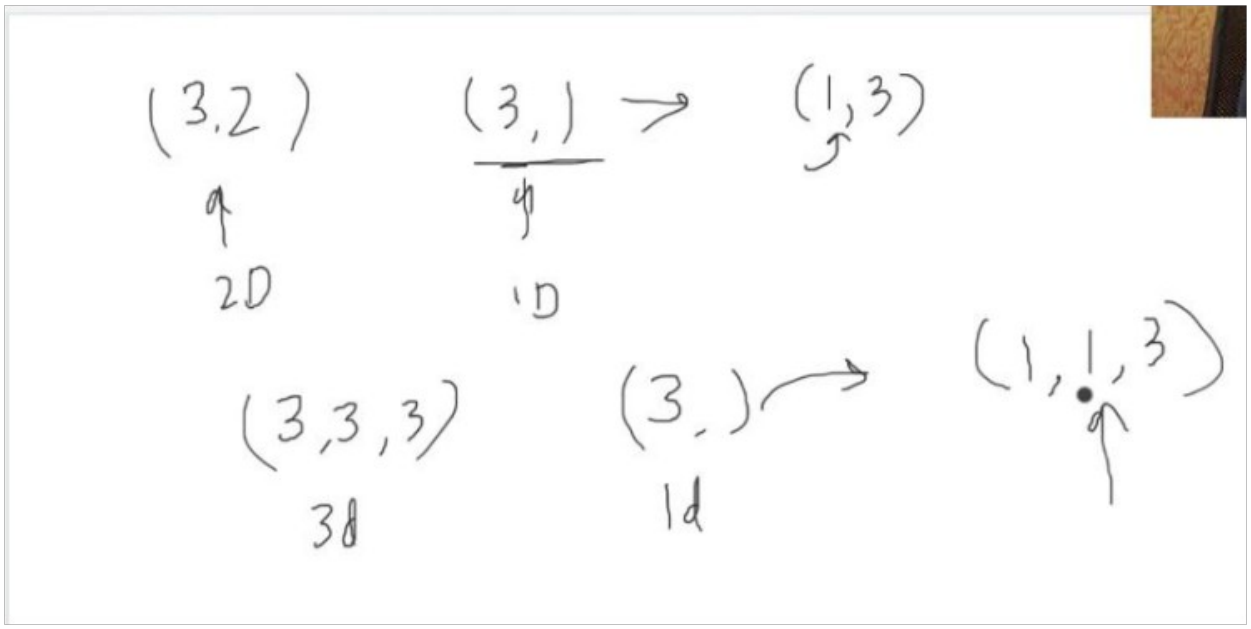
```
[[0 1 2]
 [3 4 5]]

[[0 1 2]]
```

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

Broadcasting Rules

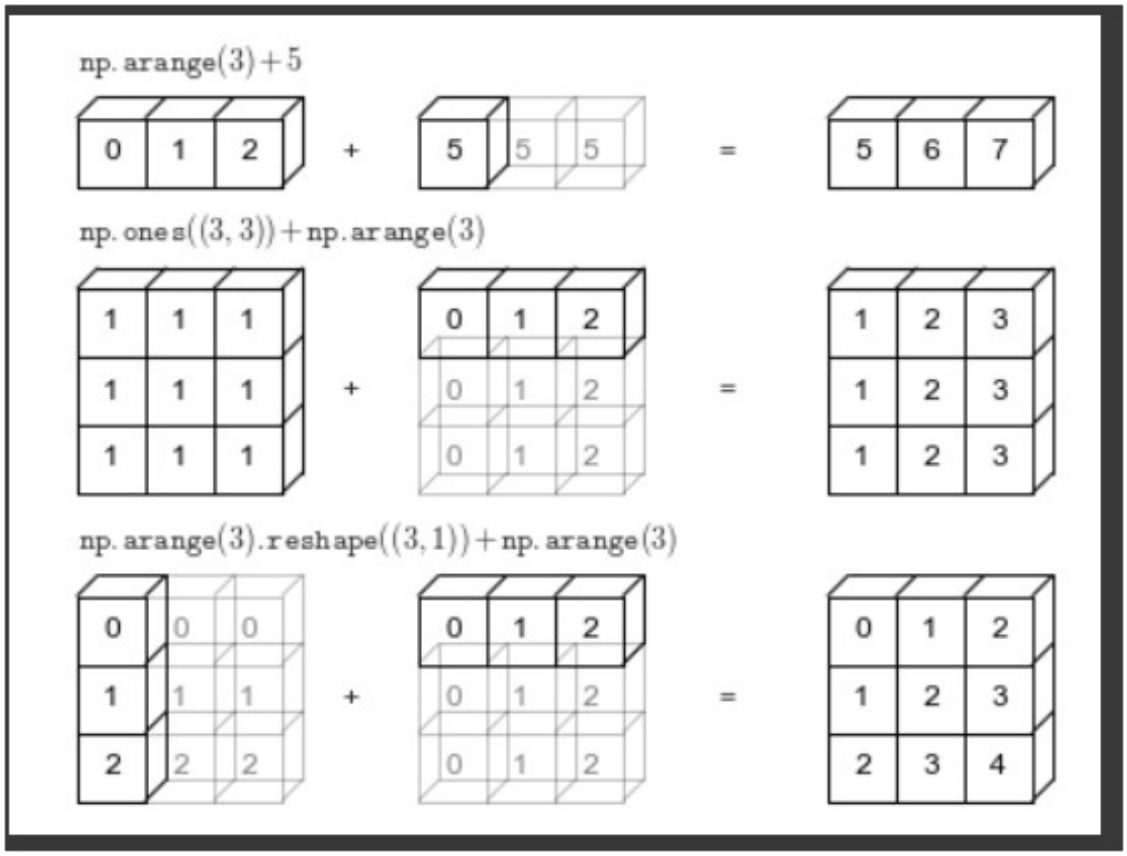
1. Make the two arrays have the same number of dimensions.

- If the numbers of dimensions of the two arrays are different, add new dimensions with size 1 to the head of the array with the smaller dimension.



2. Make each dimension of the two arrays the same size.

- If the sizes of each dimension of the two arrays do not match, dimensions with size 1 are stretched to the size of the other array.
- (upar ke example me jo 1 hai usko streached karte karte bade bale ke dimension ke equal karo)
- If there is a dimension whose size is not 1 in either of the two arrays, it cannot be broadcasted, and an error is raised.

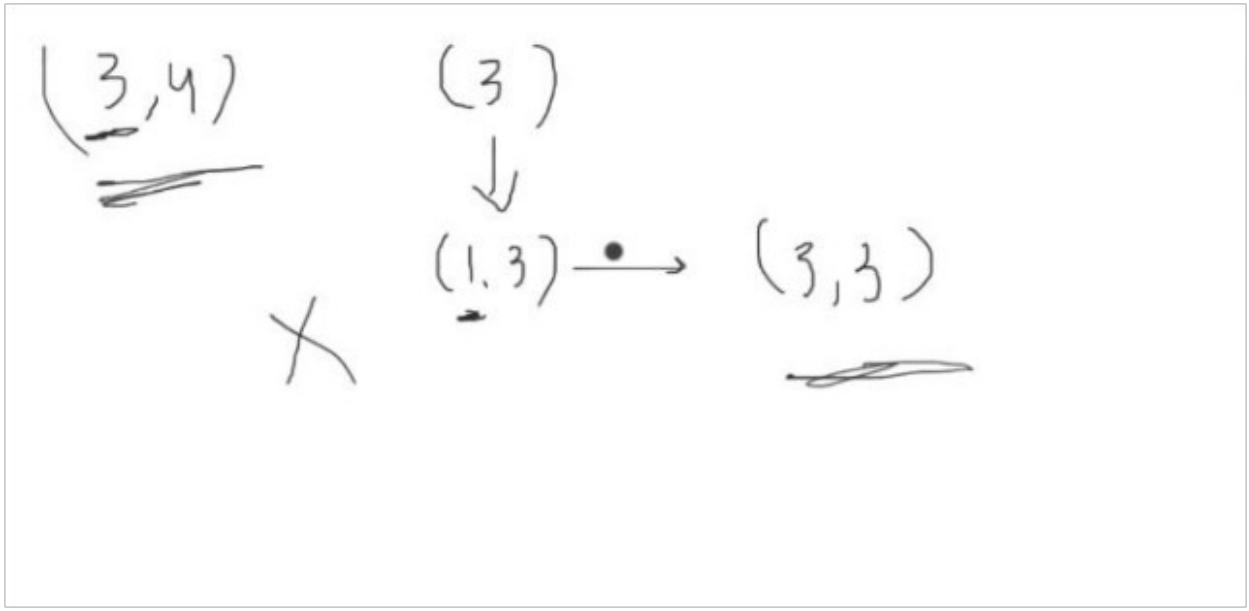



```
# More examples

a = np.arange(12).reshape(4,3)
b = np.arange(3)

print(a)
print()
print(b)
print()

print(a+b)
```

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

[0 1 2]

[[ 0  2  4]
 [ 3  5  7]
 [ 6  8 10]
 [ 9 11 13]]
```

```
# More examples

a = np.arange(12).reshape(3,4)
b = np.arange(3)

print(a)
print()
print(b)
print()

print(a+b)
```

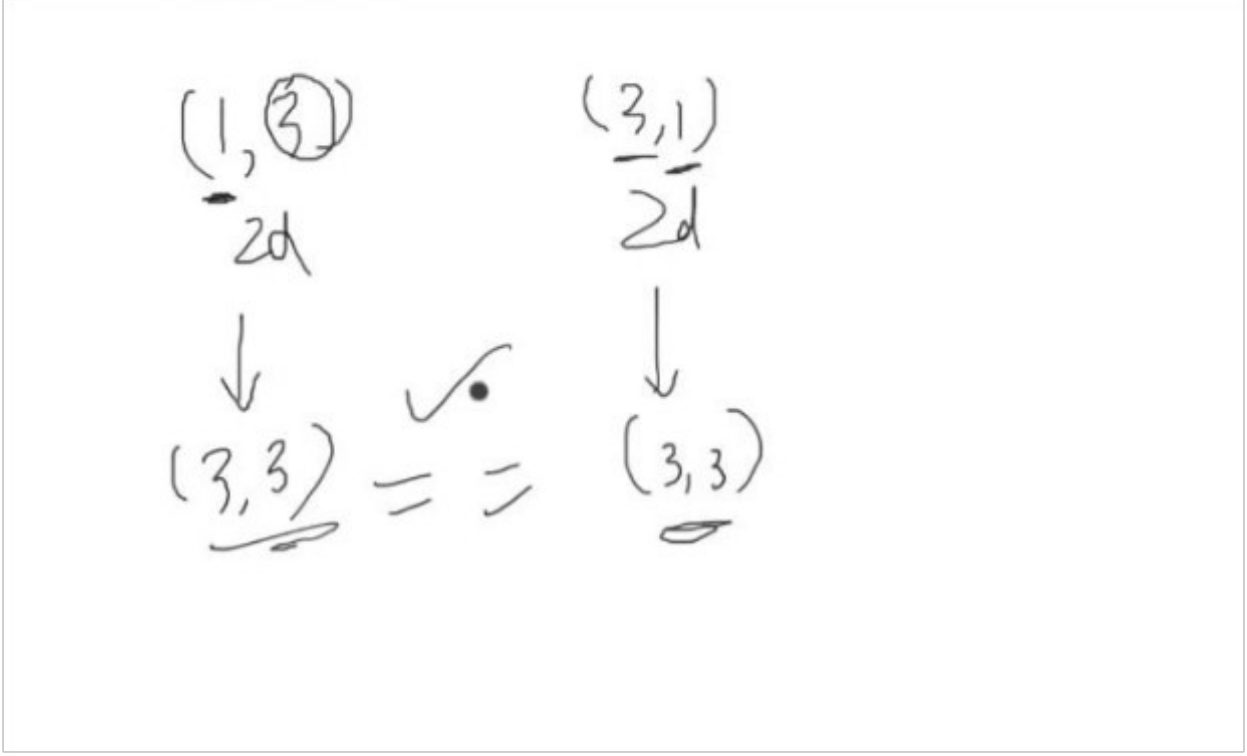
```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

[0 1 2]


---------------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
Cell In[37], line 11
      8 print(b)
      9 print()
---> 11 print(a+b)

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



2nd bale array me utna element hi nahi hai

```
a = np.arange(3).reshape(1,3)
b = np.arange(3).reshape(3,1)

print(a)
print()
print(b)
print()

print(a+b)
```

```
[[0 1 2]]

[[0]
 [1]
 [2]]

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




```
a = np.arange(3).reshape(1,3)
b = np.arange(4).reshape(4,1)

print(a)
print()
print(b)
```

```
print()

print(a + b)
```

```
[[0 1 2]]

[[0]
 [1]
 [2]
 [3]]

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

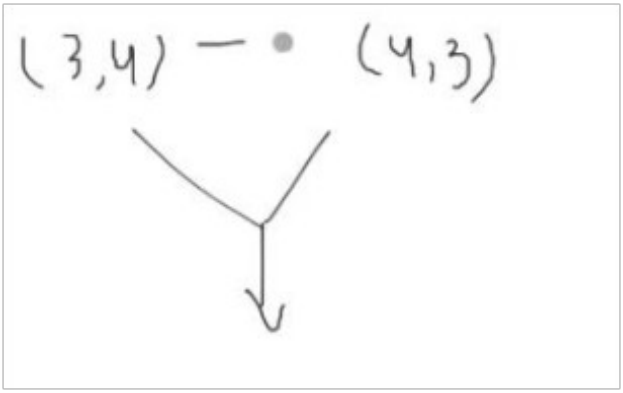
```
a = np.array([1])
# shape -> (1,1)
b = np.arange(4).reshape(2,2)
# shape -> (2,2)

print(a)
print()
print(b)
print()

print(a+b)
```

```
[1]

[[0 1]
 [2 3]]

[[1 2]
 [3 4]]
```

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

print(a)
print()
print(b)
print()

print(a+b)
```

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

[[ 0  1  2]
```

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


---------------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
Cell In[43], line 9
      6 print(b)
      7 print()
----> 9 print(a+b)

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



at least kisi ek me suru me either 1 row ya 1 column available hona chahaiye broadcast hone ke lye

```
a = np.arange(16).reshape(4,4)
b = np.arange(4).reshape(2,2)

print(a)
print()
print(b)
print()

print(a+b)  # stretch karne ke lye 1 hai hi nahi to error aayega
```

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

[[0 1]
 [2 3]]
```

```
---------------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
Cell In[46], line 9
      6 print(b)
      7 print()
----> 9 print(a+b)

ValueError: operands could not be broadcast together with shapes (4,4)
(2,2)
```

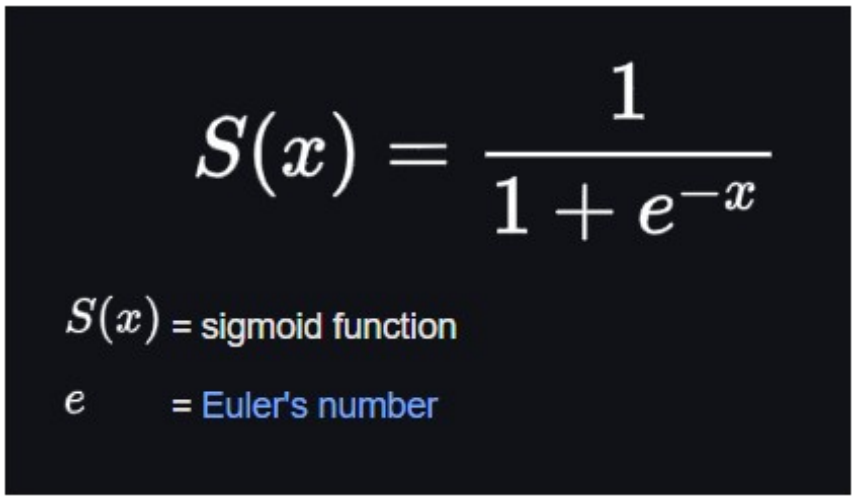
broadcasting ka main use hota hai vectorizing me

Working with mathematical formulas

```
a = np.arange(10)
np.sin(a)
```

```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -
0.7568025 ,
       -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,
0.41211849])
```

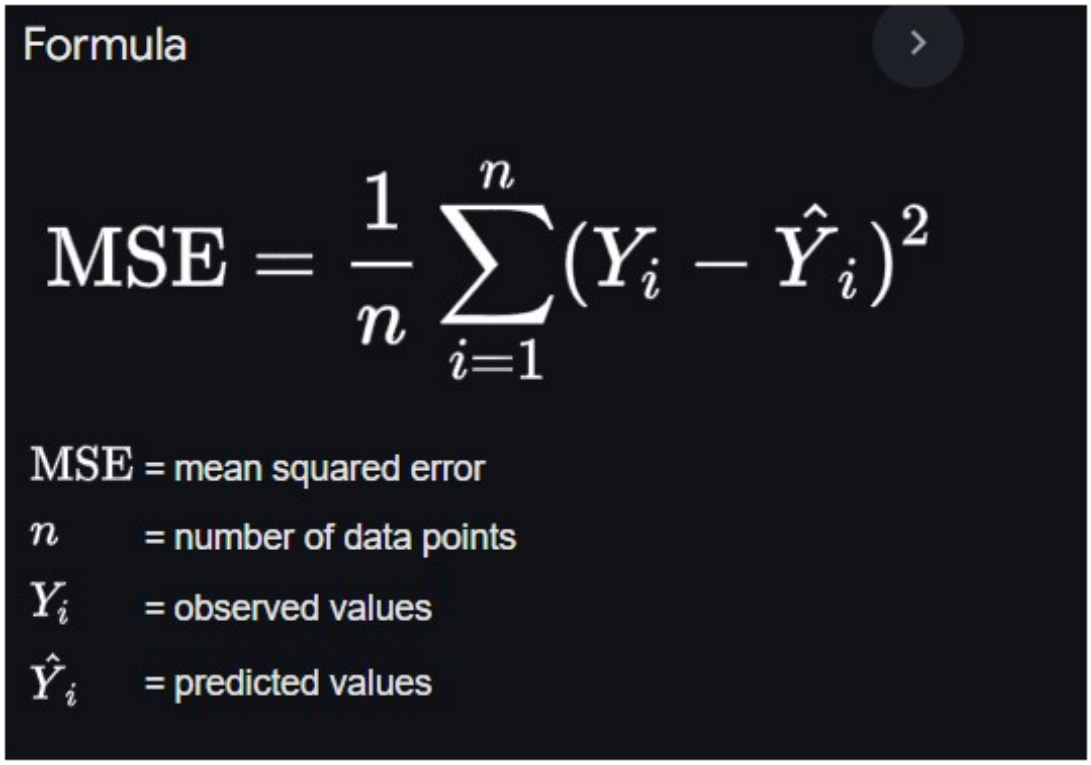
```
# sigmoid => how we can find sigmoid of all the elements of array
```




```
def sigmoid(array):
    return 1/(1 + (1/np.exp(array)))

a = np.arange(10)
sigmoid((a))
```

```
array([0.5       , 0.73105858, 0.88079708, 0.95257413, 0.98201379,
       0.99330715, 0.99752738, 0.99908895, 0.99966465, 0.99987661])
```

mean square error




```
actual = np.random.randint(1,50,25)
predicted = np.random.randint(1,50,25)

predicted.size

25

predicted

array([25,  3, 42,  6, 16, 24, 12, 32,  9, 35, 19, 45, 48,  8,  5, 41,
35,
       42, 19, 18,  4, 16, 12, 23, 16])

actual

array([41, 14, 41,  1, 11, 36,  2, 43,  5, 23, 16,  4, 27, 25, 23, 42,
47,
       30, 49, 22, 33, 39, 40, 19, 24])
```

```
def mse(actual,predicted):
    return np.mean((actual-predicted)**2)

mse(actual,predicted)
```

```
285.44
```

```
# HW categorical cross function
```

hw make a function for binary loss entropy

$$\text{Log loss} = \frac{1}{N} \sum_{i=1}^{N} -\left(y_i * \log(p_i) + (1-y_i) * \log(1-p_i) \right)$$

```
def ble(actual,pridected):
    return 1/(actual.size) *np.mean(-(actual*np.log(predicted)) + (1 -
actual) * np.log(1-predicted))

ble(actual,predicted)
```

```
C:\Users\jayra\AppData\Local\Temp\ipykernel_13720\3112359830.py:2:
RuntimeWarning: invalid value encountered in log
  return 1/(actual.size) *np.mean(-(actual*np.log(predicted)) + (1 -
actual) * np.log(1-predicted))

nan
```

Working with missing value

```
# Working with missing values -> np.nan   # nan basically missing
value banata hai  (dhyan rakhna hai ki nan aur none different hota hai
python me)
a = np.array([1,2,3,4,np.nan,6])
a
```

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

```
# nan ko remove kar dena hai
np.isnan(a)   # abb ye boolean array aa gya
```

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

```
a[np.isnan(a)]
```

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

```
a[~(np.isnan(a))]  # remove ho gya nan number
```

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

Plotting Graphs

```
# plotting a 2D plot

# x=y

x = np.linspace(-10,10,100)
y=x
```

```
x
```

```
array([-10.        ,  -9.7979798 ,  -9.5959596 ,  -9.39393939,
        -9.19191919,  -8.98989899,  -8.78787879,  -8.58585859,
        -8.38383838,  -8.18181818,  -7.97979798,  -7.77777778,
        -7.57575758,  -7.37373737,  -7.17171717,  -6.96969697,
        -6.76767677,  -6.56565657,  -6.36363636,  -6.16161616,
        -5.95959596,  -5.75757576,  -5.55555556,  -5.35353535,
        -5.15151515,  -4.94949495,  -4.74747475,  -4.54545455,
        -4.34343434,  -4.14141414,  -3.93939394,  -3.73737374,
        -3.53535354,  -3.33333333,  -3.13131313,  -2.92929293,
        -2.72727273,  -2.52525253,  -2.32323232,  -2.12121212,
        -1.91919192,  -1.71717172,  -1.51515152,  -1.31313131,
        -1.11111111,  -0.90909091,  -0.70707071,  -0.50505051,
        -0.3030303 ,  -0.1010101 ,   0.1010101 ,   0.3030303 ,
         0.50505051,   0.70707071,   0.90909091,   1.11111111,
         1.31313131,   1.51515152,   1.71717172,   1.91919192,
         2.12121212,   2.32323232,   2.52525253,   2.72727273,
         2.92929293,   3.13131313,   3.33333333,   3.53535354,
         3.73737374,   3.93939394,   4.14141414,   4.34343434,
         4.54545455,   4.74747475,   4.94949495,   5.15151515,
         5.35353535,   5.55555556,   5.75757576,   5.95959596,
         6.16161616,   6.36363636,   6.56565657,   6.76767677,
         6.96969697,   7.17171717,   7.37373737,   7.57575758,
         7.77777778,   7.97979798,   8.18181818,   8.38383838,
         8.58585859,   8.78787879,   8.98989899,   9.19191919,
         9.39393939,   9.5959596 ,   9.7979798 ,  10.        ])
```

```
y
```

```
array([-10.        ,  -9.7979798 ,  -9.5959596 ,  -9.39393939,
        -9.19191919,  -8.98989899,  -8.78787879,  -8.58585859,
        -8.38383838,  -8.18181818,  -7.97979798,  -7.77777778,
        -7.57575758,  -7.37373737,  -7.17171717,  -6.96969697,
```

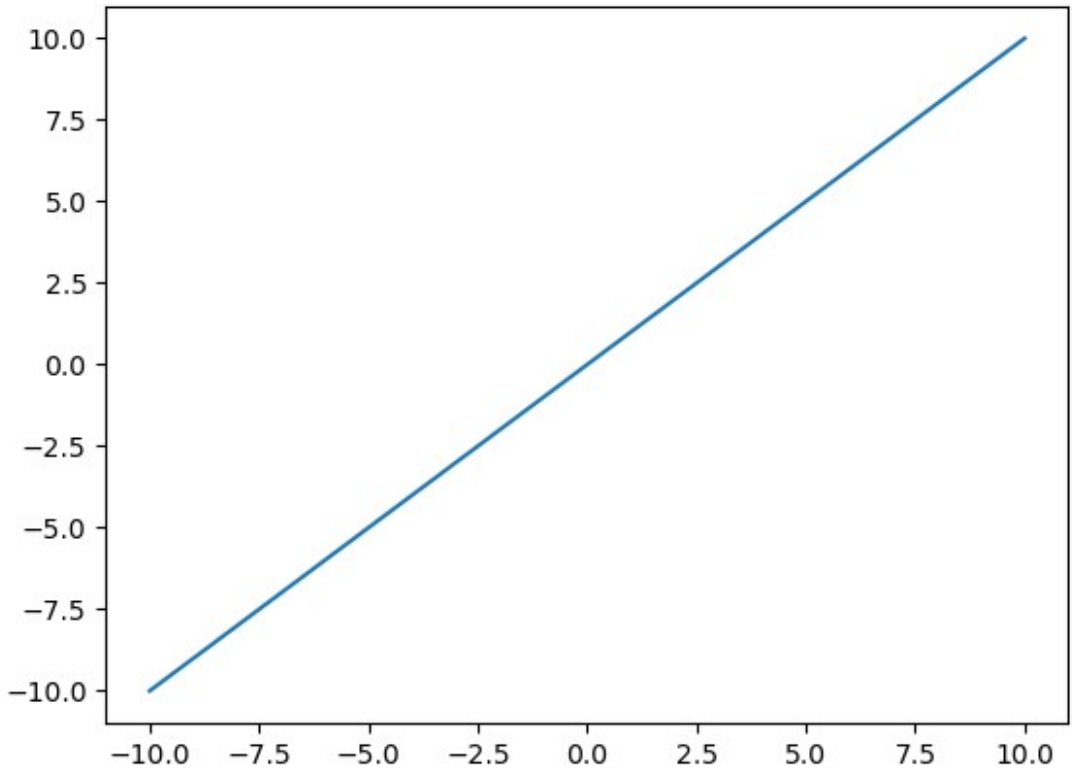
```
       -6.76767677,  -6.56565657,  -6.36363636,  -6.16161616,
       -5.95959596,  -5.75757576,  -5.55555556,  -5.35353535,
       -5.15151515,  -4.94949495,  -4.74747475,  -4.54545455,
       -4.34343434,  -4.14141414,  -3.93939394,  -3.73737374,
       -3.53535354,  -3.33333333,  -3.13131313,  -2.92929293,
       -2.72727273,  -2.52525253,  -2.32323232,  -2.12121212,
       -1.91919192,  -1.71717172,  -1.51515152,  -1.31313131,
       -1.11111111,  -0.90909091,  -0.70707071,  -0.50505051,
       -0.3030303 ,  -0.1010101 ,   0.1010101 ,   0.3030303 ,
        0.50505051,   0.70707071,   0.90909091,   1.11111111,
        1.31313131,   1.51515152,   1.71717172,   1.91919192,
        2.12121212,   2.32323232,   2.52525253,   2.72727273,
        2.92929293,   3.13131313,   3.33333333,   3.53535354,
        3.73737374,   3.93939394,   4.14141414,   4.34343434,
        4.54545455,   4.74747475,   4.94949495,   5.15151515,
        5.35353535,   5.55555556,   5.75757576,   5.95959596,
        6.16161616,   6.36363636,   6.56565657,   6.76767677,
        6.96969697,   7.17171717,   7.37373737,   7.57575758,
        7.77777778,   7.97979798,   8.18181818,   8.38383838,
        8.58585859,   8.78787879,   8.98989899,   9.19191919,
        9.39393939,   9.5959596 ,   9.7979798 ,  10.        ])
```

```
# x and y ka multiple value mil gya abb isse mila ke graph bana denge
```

```
import matplotlib.pyplot as plt
```

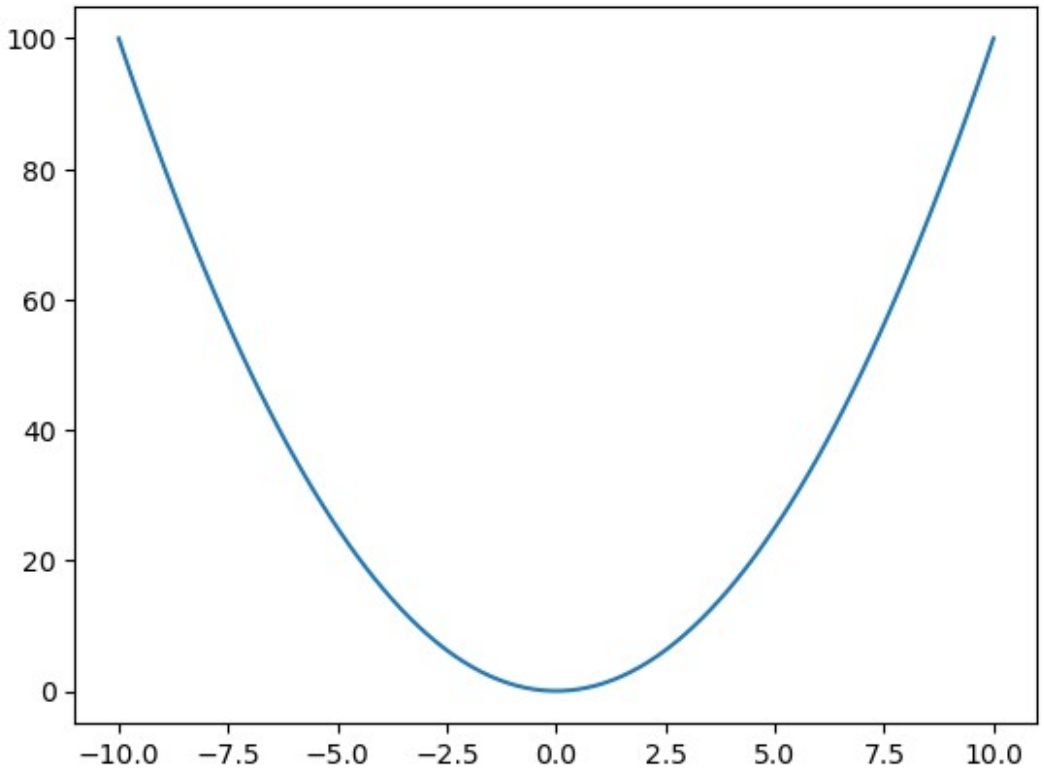
```
plt.plot(x,y)    # ofcoure dono array ka size same hona chahaiye
```

```
[<matplotlib.lines.Line2D at 0x227f074d070>]
```



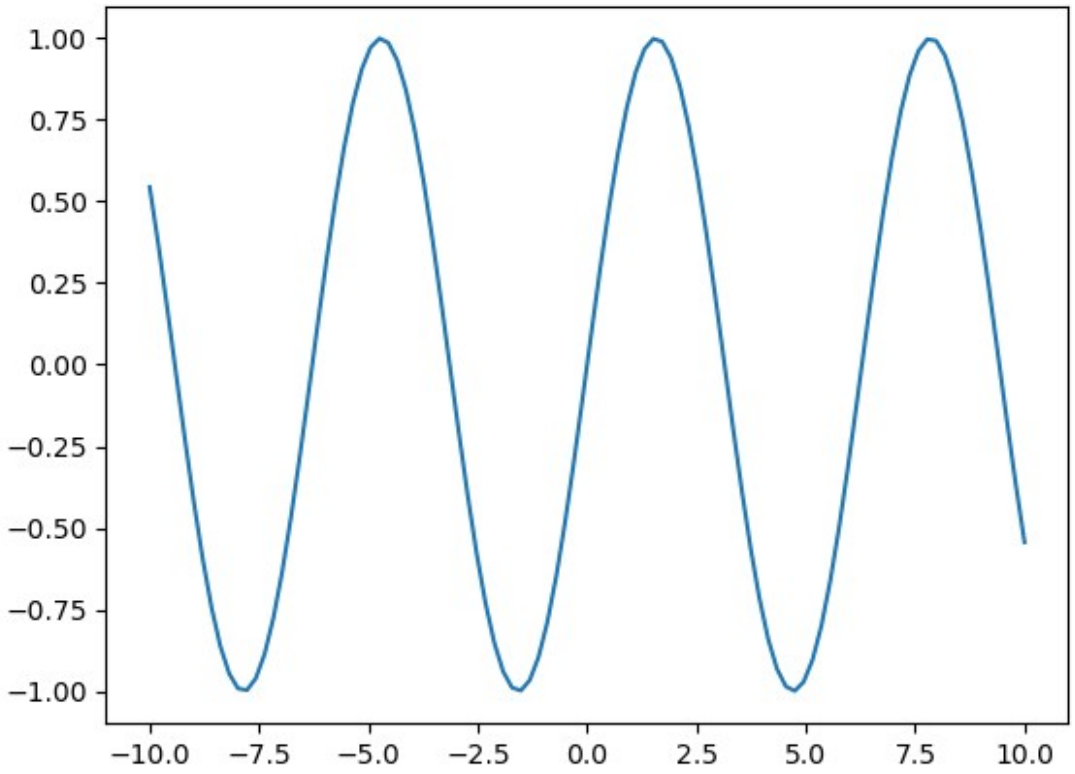

```
# y = x^2
x = np.linspace(-10,10,100)
y = x**2

plt.plot(x,y)
```

```
[<matplotlib.lines.Line2D at 0x227f0a525a0>]
```




```
# sin(x)
x = np.linspace(-10,10,100)
y = np.sin(x)

plt.plot(x,y)
```

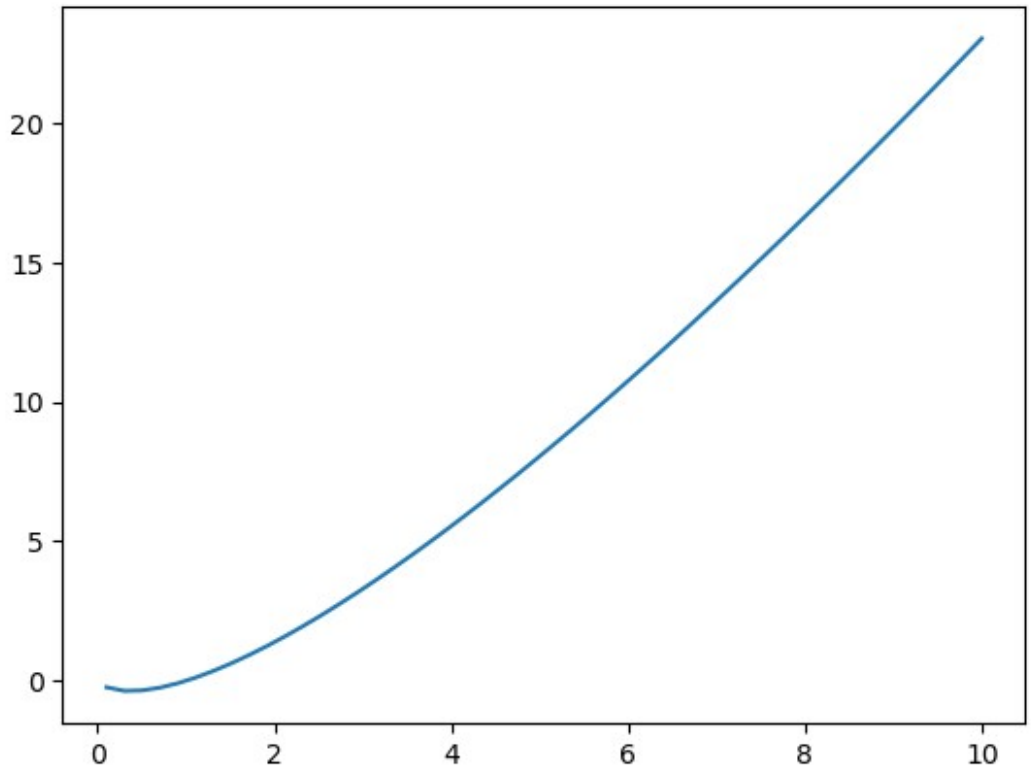
```
[<matplotlib.lines.Line2D at 0x227f23640b0>]
```




```
# y = xlog(x)
x = np.linspace(-10,10,100)
y = x*np.log(x)

plt.plot(x,y)
```

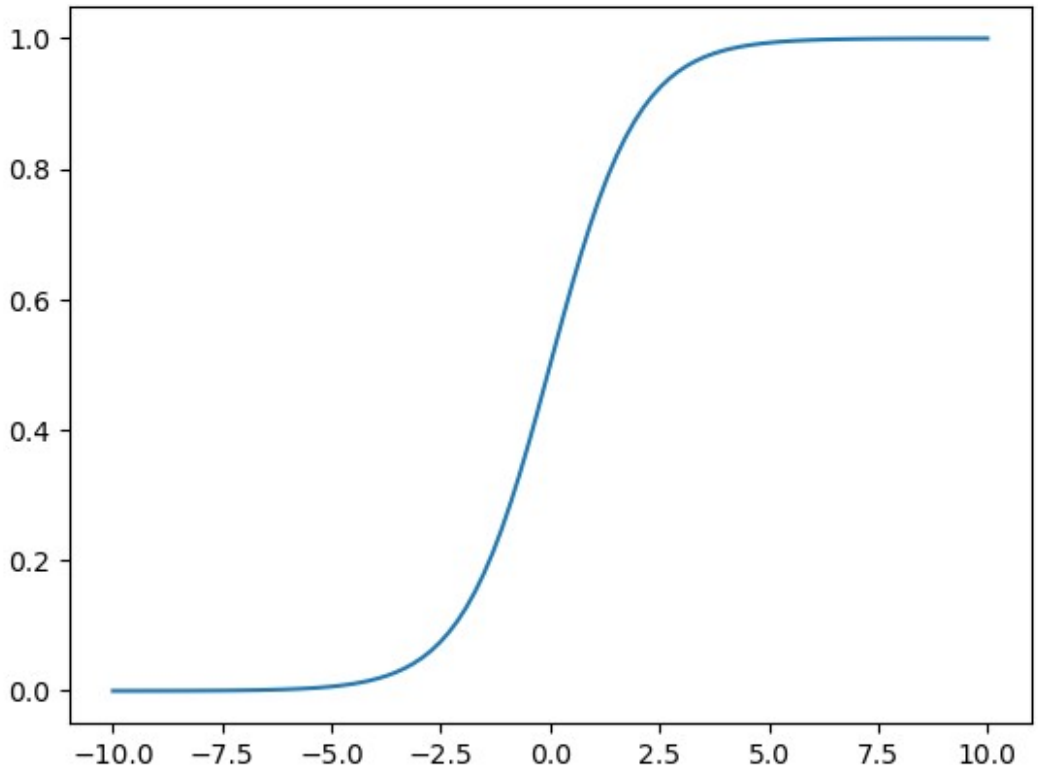
```
C:\Users\jayra\AppData\Local\Temp\ipykernel_8368\3772014300.py:3:
RuntimeWarning: invalid value encountered in log
  y = x*np.log(x)
```

```
[<matplotlib.lines.Line2D at 0x227f2428ef0>]
```




```
# sigmoid
x = np.linspace(-10,10,100)

y=1/(1+(1/np.exp(x)))  # or, y=1/1+(np.exp(-x))

plt.plot(x,y)
```

```
[<matplotlib.lines.Line2D at 0x227f24a02c0>]
```


1.0
0.8
0.6
0.4
0.2
0.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0