NUMPY

NumPy(Numerical python) is a python library is used for numerical operations on multi dimensional array and matrices ¶

INSTALLING NUMPY

In [1]:
```
# pip install numpy
```

In [2]:
```
# conda install numpy
```

IMPORTING NUMPY

In [3]:
```
import numpy as np
```

what is the difference between list and a NUMPY array

list can allow different data types but array is homogeneous it allow only single data type

array consumes less memory than the list

list slower for numerical operations but array faster for numerical operations

in list 1 dimensional but default nested list can be used for mutli dimension data

numpy supports multl dimensional array

In [4]:
```
mylist=[6,2,3,4,5]
mylist
```

Out[4]: [6, 2, 3, 4, 5]

In [5]:
```
type(mylist)
```

Out[5]: list

In [6]:
```
mylist
```

Out[6]: [6, 2, 3, 4, 5]

convert list into an aray

ARRAY

An array is a data structure that stores a collection of same data types

In [7]:
```
arr=np.array([6,2,3,4,5])
arr
```

Out[7]:
```
array([6, 2, 3, 4, 5])
```

In [8]:
```
arr=np.array([1,20,'akshitha',50.6])
```

In [9]:
```
arr
```

Out[9]:
```
array(['1', '20', 'akshitha', '50.6'], dtype='<U11')
```

we can acess the elements in array by indexing starts from 0

In [10]:
```
arr[0]
```

Out[10]:
```
'1'
```

In [11]:
```
arr[3]
```

Out[11]:
```
'50.6'
```

Array creations functions

How to create numpy array

np.array()

np.ones()

np.zeros()

np.araange()

np.linespace()

np.eye()

In [12]:
```
# np.ones()--->create an array filled with ones
arr_ones=np.ones(2)
arr_ones
```

Out[12]:
```
array([1., 1.])
```

by default it is float convert into int

In [13]:
```
arr_ones=np.ones(5,dtype=int)
arr_ones
```

Out[13]:
```
array([1, 1, 1, 1, 1])
```

In [14]:
```
# np.ones()----> create an array filled with zeros
array_zeros=np.zeros(5)      #parameter tuning
array_zeros
```

Out[14]:
```
array([0., 0., 0., 0., 0.])
```

In [15]:
```
array_zeros=np.zeros(5,dtype=int)  #hyper parameter tuning
array_zeros
```

Out[15]:
```
array([0, 0, 0, 0, 0])
```

In [16]:
```
np.ones([5,5])
```

Out[16]:
```
array([[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.]])
```

In [17]:
```
np.ones([3,3])
```

Out[17]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [18]:
```
np.ones([4,5],dtype=int)
```

Out[18]:
```
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]])
```

In [19]:
```
np.empty(2)
```

Out[19]:
```
array([1., 1.])
```

In [20]:
```
#arange()--->create a range of values
# arange(start,stop, step)
```

In [21]:
```
arange=np.arange(10,20)
```

In [22]:
```
arange
```

Out[22]:
```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

In [23]:
```
np.arange(20,10)   # first argument is always smaller than second argument
```

Out[23]:
```
array([], dtype=int32)
```

In [24]:
```
np.arange(10,20,2)
```

Out[24]:
```
array([10, 12, 14, 16, 18])
```

In [25]:
```
np.arange(-20,10)
```

Out[25]:
```
array([-20, -19, -18, -17, -16, -15, -14, -13, -12, -11, -10,  -9,  -8,
        -7,  -6,  -5,  -4,  -3,  -2,  -1,   0,   1,   2,   3,   4,   5,
         6,   7,   8,   9])
```

In [26]:
```
np.linspace(0,10,num=5)
```

Out[26]:
```
array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```

In [27]:
```
np.linspace(0,10,num=5,dtype=int)
```

Out[27]:
```
array([ 0,  2,  5,  7, 10])
```

In [28]:
```
np.identity(4)
```

Out[28]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

In [29]:
```
np.identity(6)
```

Out[29]:
```
array([[1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 1.]])
```

array manipulation function

np.reshape()

np.transpose()

np.concatenate()

np.split

In [30]:
```
arange1=np.arange(0,10).reshape(5,2)
arange1
```

Out[30]:
```
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
```

In [31]:
```
transposed_array=arange1.T
transposed_array
```

Out[31]:
```
array([[0, 2, 4, 6, 8],
       [1, 3, 5, 7, 9]])
```

In [32]:
```
arr_2d=np.array([[1,2,3],
                 [4,5,6],
                 [7,8,9]])
arr_2d
```

Out[32]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [33]:
```
np.transpose(arr_2d)
```

Out[33]:
```
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])
```

In [34]:
```
np.transpose(arange1)
```

Out[34]:
```
array([[0, 2, 4, 6, 8],
       [1, 3, 5, 7, 9]])
```

In [35]:
```
np.concatenate(arange1)  #swaping rows and columns
```

Out[35]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [36]:
```
np.split(arange1,5)
```

Out[36]:
```
[array([[0, 1]]),
 array([[2, 3]]),
 array([[4, 5]]),
 array([[6, 7]]),
 array([[8, 9]])]
```

Ravel

converting any dimensions into 1D

In [37]:
```
p2
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-37-32960d173fa8> in <module>
----> 1 p2

NameError: name 'p2' is not defined
```

In [ ]:
```
p2.ravel()
```

In [38]:
```
p3
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-38-1d4236778dc2> in <module>
----> 1 p3

NameError: name 'p3' is not defined
```

In [39]:
```
p3.ravel()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-39-2fcf16ec3f66> in <module>
----> 1 p3.ravel()

NameError: name 'p3' is not defined
```

inspection function

arr.shape()

arr.size()

arr.ndim()

arr.dtype()

In [40]:
```
array1=np.array([10,20,30,40,50,60,70])
array1
```

Out[40]:
```
array([10, 20, 30, 40, 50, 60, 70])
```

In [41]:
```
type(array1)
```

Out[41]:
```
numpy.ndarray
```

In [42]:
```
array1.shape
```

Out[42]:
```
(7,)
```

In [43]:
```
arange1
```

Out[43]:
```
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
```

In [44]:
```
arange1.shape
```

Out[44]:
```
(5, 2)
```

In [45]:
```
arange1.size
```

Out[45]:
```
10
```

In [46]:
```
arange1.ndim
```

Out[46]:
```
2
```

In [47]:
```
arange1.dtype
```

Out[47]:
```
dtype('int32')
```

In [48]:
```
arange1.itemsize
```

Out[48]:
```
4
```

changing the data type

In [49]:
```
#astype
x=np.array([33,22,2.5])
x
```

Out[49]:
```
array([33. , 22. ,  2.5])
```

In [50]:
```
x.astype(int)
```

Out[50]:
```
array([33, 22,  2])
```

mathematical functions

np.add()

np.subtract()

np.multiply()

np.divide()

np.power()

np.sqrt()

In [51]:
```
array=np.array([[1,2,3,4],
                [5,6,7,8]])
array
```

Out[51]:
```
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

In [52]:
```
add=np.add([1,2,3,4],[5,6,7,8])
print("addition:",add)
```

```
addition: [ 6  8 10 12]
```

In [53]:
```
arr_sub=np.subtract([4,5,6,7],[1,2,3,4])
print("substraction:",arr_sub)
```

```
substraction: [3 3 3 3]
```

In [54]:
```
arr_mul=np.multiply([1,2,3],[4,5,6])
print('multiply:',arr_mul)
```

```
multiply: [ 4 10 18]
```

In [55]:
```
arr_div=np.divide([10,15,],[2,5])
arr_div
```

Out[55]: array([5., 3.])

In [56]:
```
arr_pow=np.power([10,15,],[2,5])
arr_pow
```

Out[56]: array([100, 759375], dtype=int32)

In [57]:
```
arr1=np.array([1,2,3])
arr1
```

Out[57]: array([1, 2, 3])

In [58]:
```
arr2=np.array([4,5,6])
arr2
```

Out[58]: array([4, 5, 6])

In [59]:
```
np.add(arr1,arr2)
```

Out[59]: array([5, 7, 9])

In [60]:
```
np.sqrt(arr1)
```

Out[60]: array([1. , 1.41421356, 1.73205081])

In [61]:
```
np.subtract(arr1,arr2)
```

Out[61]: array([-3, -3, -3])

In [62]:
```
arr1+2
```

Out[62]: array([3, 4, 5])

In [63]:
```
arr2*2
```

Out[63]: array([8, 10, 12])

reational operators

In [64]:
```
arr2
```

Out[64]: array([4, 5, 6])

In [65]:
```
arr2>2
```

Out[65]: array([True, True, True])

In [66]:
```
arr2>20
```

Out[66]:
```
array([False, False, False])
```

In [67]:
```
array1=np.arange(10,30).reshape(4,5)
array1
```

Out[67]:
```
array([[10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24],
       [25, 26, 27, 28, 29]])
```

In [68]:
```
array1>2
```

Out[68]:
```
array([[ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True]])
```

In [69]:
```
array1>20
```

Out[69]:
```
array([[False, False, False, False, False],
       [False, False, False, False, False],
       [False,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True]])
```

In [70]:
```
array2=np.arange(30,70,2).reshape(4,5)
array2
```

Out[70]:
```
array([[30, 32, 34, 36, 38],
       [40, 42, 44, 46, 48],
       [50, 52, 54, 56, 58],
       [60, 62, 64, 66, 68]])
```

In [71]:
```
array2>20
```

Out[71]:
```
array([[ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True]])
```

dot product

In [72]:
```
s2=np.arange(12).reshape(3,4)
s3=np.arange(12,24).reshape(4,3)
```

In [73]:
```
s2
```

Out[73]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [74]:
```
s3
```

Out[74]:
```
array([[12, 13, 14],
       [15, 16, 17],
       [18, 19, 20],
       [21, 22, 23]])
```

In [75]:
```
np.dot(s2,s3) # dot product of s2,s3
```

Out[75]:
```
array([[114, 120, 126],
       [378, 400, 422],
       [642, 680, 718]])
```

Log and Exponents

In [76]:
```
s2
```

Out[76]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [77]:
```
s3
```

Out[77]:
```
array([[12, 13, 14],
       [15, 16, 17],
       [18, 19, 20],
       [21, 22, 23]])
```

In [78]:
```
np.exp(s2)
```

Out[78]:
```
array([[1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01],
       [5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03],
       [2.98095799e+03, 8.10308393e+03, 2.20264658e+04, 5.98741417e+04]])
```

round/floor/ceil

1.round

The numpy.round() function rounds the elements of an array to the nearest integer or to the specified number of decimals

In [79]:
```
# Round to the nearest integer
arr=np.array([1.2,2.7,3.5,4.9])
rounded_arr=np.round(arr)
print(rounded_arr)
```

```
[1. 3. 4. 5.]
```

In [80]:
```
# Round to two decimals
arr=np.array([1.234,2.567,3.891])
rounded_arr=np.round(arr,decimals=2)
print(rounded_arr)
```

```
[1.23 2.57 3.89]
```

In [81]:
```
# randomly
np.round(np.random.random((2,3))*100)
```

Out[81]:
```
array([[29., 55., 92.],
       [38., 34., 89.]])
```

2.floor

floor function returns the less than or equal to each element of an array

In [82]:
```
#Floor operation
arr=np.array([1.2,2.7,3.5,4.9])
floored_arr=np.floor(arr)
floored_arr
```

Out[82]:
```
array([1., 2., 3., 4.])
```

In [83]:
```
np.floor(np.random.random((2,3))*100)
```

Out[83]:
```
array([[18., 18., 37.],
       [40., 42., 81.]])
```

3. ceil

The numpy ceil function returns the greater than or equal to each element of an array

In [84]:
```
arr=np.array([1.2,2.7,3.5,4.9])
ceiled_arr=np.ceil(arr)
ceiled_arr
```

Out[84]:
```
array([2., 3., 4., 5.])
```

In [85]:
```
np.ceil(np.random.random((2,3))*100)
```

Out[85]:
```
array([[53., 43.,  4.],
       [39., 24., 94.]])
```

vector operation

In [86]:
```
array1
```

Out[86]:
```
array([[10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24],
       [25, 26, 27, 28, 29]])
```

In [87]:
```
array2
```

Out[87]:
```
array([[30, 32, 34, 36, 38],
       [40, 42, 44, 46, 48],
       [50, 52, 54, 56, 58],
       [60, 62, 64, 66, 68]])
```

In [88]:
```
array1+array2
```

Out[88]:
```
array([[40, 43, 46, 49, 52],
       [55, 58, 61, 64, 67],
       [70, 73, 76, 79, 82],
       [85, 88, 91, 94, 97]])
```

In [89]:
```
array1.size
```

Out[89]:
```
20
```

In [90]:
```
array2.size
```

Out[90]:
```
20
```

if both numpy array shape is same, we can add item wise

In [91]:
```
array1-array2
```

Out[91]:
```
array([[-20, -21, -22, -23, -24],
       [-25, -26, -27, -28, -29],
       [-30, -31, -32, -33, -34],
       [-35, -36, -37, -38, -39]])
```

In [92]:
```
array1*array2
```

Out[92]:
```
array([[ 300,  352,  408,  468,  532],
       [ 600,  672,  748,  828,  912],
       [1000, 1092, 1188, 1288, 1392],
       [1500, 1612, 1728, 1848, 1972]])
```

In [93]:
```
array1/array2
```

Out[93]:
```
array([[0.33333333, 0.34375   , 0.35294118, 0.36111111, 0.36842105],
       [0.375     , 0.38095238, 0.38636364, 0.39130435, 0.39583333],
       [0.4       , 0.40384615, 0.40740741, 0.41071429, 0.4137931 ],
       [0.41666667, 0.41935484, 0.421875  , 0.42424242, 0.42647059]])
```

In [94]:
```
np.product(array1)
```

Out[94]:
```
-1033633792
```

0=column, 1=row

In [95]:
```
array1
```

Out[95]:
```
array([[10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24],
       [25, 26, 27, 28, 29]])
```

In [96]:
```
np.max(array1,axis=1) # max of every row
```

Out[96]:
```
array([14, 19, 24, 29])
```

In [97]:
```
np.max(array1,axis=0) #max of every column
```

Out[97]:
```
array([25, 26, 27, 28, 29])
```

In [98]:
```
np.min(array1,axis=0)  #min of every column
```

Out[98]:
```
array([10, 11, 12, 13, 14])
```

In [99]:
```
np.min(array1,axis=1)
```

Out[99]:
```
array([10, 15, 20, 25])
```

statistical function

np.mean()

np.median()

np.std()

np.var()

np.min()

np.max()

np.sum()

In [100]:
```
arr=np.array([10,20,30,40,50])
arr
```

Out[100]:
```
array([10, 20, 30, 40, 50])
```

In [101]:
```
np.mean(arr)
```

Out[101]:
```
30.0
```

In [102]:
```
np.median(arr)
```

Out[102]:
```
30.0
```

In [103]:
```
np.min(arr)
```

Out[103]:
```
10
```

In [104]:
```
np.max(arr)
```

Out[104]:
```
50
```

In [105]:
```
np.std(arr)
```

Out[105]:
```
14.142135623730951
```

In [106]:
```
np.sum(arr)
```

Out[106]:
```
150
```

Trignometry functions

In [107]:
```
np.sin(array1)
```

Out[107]:
```
array([[-0.54402111, -0.99999021, -0.53657292,  0.42016704,  0.99060736],
       [ 0.65028784, -0.28790332, -0.96139749, -0.75098725,  0.14987721],
       [ 0.91294525,  0.83665564, -0.00885131, -0.8462204 , -0.90557836],
       [-0.13235175,  0.76255845,  0.95637593,  0.27090579, -0.66363388]])
```

In [108]:
```
np.sin(array1[0][1])
```

Out[108]:
```
-0.9999902065507035
```

In [109]:
```
np.cos(array1)
```

Out[109]:
```
array([[-0.83907153,  0.0044257 ,  0.84385396,  0.90744678,  0.13673722],
       [-0.75968791, -0.95765948, -0.27516334,  0.66031671,  0.98870462],
       [ 0.40808206, -0.54772926, -0.99996083, -0.53283302,  0.42417901],
       [ 0.99120281,  0.64691932, -0.29213881, -0.96260587, -0.74805753]])
```

In [110]:
```
np.tan(array1)
```

Out[110]:
```
array([[ 6.48360827e-01, -2.25950846e+02, -6.35859929e-01,
         4.63021133e-01,  7.24460662e+00],
       [-8.55993401e-01,  3.00632242e-01,  3.49391565e+00,
        -1.13731371e+00,  1.51589471e-01],
       [ 2.23716094e+00, -1.52749853e+00,  8.85165604e-03,
         1.58815308e+00, -2.13489670e+00],
       [-1.33526407e-01,  1.17875355e+00, -3.27370380e+00,
        -2.81429605e-01,  8.87142844e-01]])
```

sorting and searching

np.sort()

np.argsort()

np.where()

np.unique()

In [111]:
```
arr=np.array([10,2,30,45,5,3,99])
np.sort(arr)
```

Out[111]:
```
array([ 2,  3,  5, 10, 30, 45, 99])
```

In [112]:
```
np.argsort(arr)     # it returns the indices
```

Out[112]:
```
array([1, 5, 4, 0, 2, 3, 6], dtype=int64)
```

In [113]:
```
np.where(arr>10)  #printing indices
```

Out[113]:
```
(array([2, 3, 6], dtype=int64),)
```

In [114]:
```
np.where(arr<10)
```

Out[114]:
```
(array([1, 4, 5], dtype=int64),)
```

In [115]:
```
arr=np.array([10,2,30,45,5,3,99 ,10,3])
```

In [116]:
```
np.unique(arr)
```

Out[116]:
```
array([ 2,  3,  5, 10, 30, 45, 99])
```

How to convert a 1D array into a 2D array (how to add a new axis to an array)

In [117]:
```
#np.newaxis
```

In [118]:
```
a=np.array([1,2,3,4,5,6])
a.shape
```

Out[118]: (6,)

In [119]:
```
a2=a[np.newaxis,:]
a2.shape
```

Out[119]: (1, 6)

In [120]:
```
row_vector=a[np.newaxis,:]
row_vector.shape
```

Out[120]: (1, 6)

In [121]:
```
column_vector=a[:,np.newaxis]
column_vector.shape
```

Out[121]: (6, 1)

In [122]:
```
row=4
col=5
```

In [123]:
```
row
```

Out[123]: 4

In [124]:
```
col
```

Out[124]: 5

In [125]:
```
mat=np.arange(0,100).reshape(10,10)
```

In [126]:
```
mat[row,col]
```

Out[126]: 45

In [127]:
```
mat[:]
```

Out[127]:
```
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, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
       [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
       [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
       [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

In [128]:
```
mat[7]
```

Out[128]:
```
array([70, 71, 72, 73, 74, 75, 76, 77, 78, 79])
```

mat[:,4]---> it will print 4th column

mat[4,:]---> it will print 4 th row

mat[4:]--->it will print from fourth row

mat[:4]--->it will print upto fourth row¶

In [129]:
```
mat[:,col]
```

Out[129]:
```
array([ 5, 15, 25, 35, 45, 55, 65, 75, 85, 95])
```

In [130]:
```
mat[:,row]
```

Out[130]:
```
array([ 4, 14, 24, 34, 44, 54, 64, 74, 84, 94])
```

In [131]:
```
mat[:,:7]
```

Out[131]:
```
array([[ 0,  1,  2,  3,  4,  5,  6],
       [10, 11, 12, 13, 14, 15, 16],
       [20, 21, 22, 23, 24, 25, 26],
       [30, 31, 32, 33, 34, 35, 36],
       [40, 41, 42, 43, 44, 45, 46],
       [50, 51, 52, 53, 54, 55, 56],
       [60, 61, 62, 63, 64, 65, 66],
       [70, 71, 72, 73, 74, 75, 76],
       [80, 81, 82, 83, 84, 85, 86],
       [90, 91, 92, 93, 94, 95, 96]])
```

In [132]:
```
mat[:4,:]
```

Out[132]:
```
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, 36, 37, 38, 39]])
```

In [133]:
```
arr=np.expand_dims(mat,axis=1)
```

In [134]:
```
arr
```

Out[134]:
```
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, 36, 37, 38, 39]],

       [[40, 41, 42, 43, 44, 45, 46, 47, 48, 49]],

       [[50, 51, 52, 53, 54, 55, 56, 57, 58, 59]],

       [[60, 61, 62, 63, 64, 65, 66, 67, 68, 69]],

       [[70, 71, 72, 73, 74, 75, 76, 77, 78, 79]],

       [[80, 81, 82, 83, 84, 85, 86, 87, 88, 89]],

       [[90, 91, 92, 93, 94, 95, 96, 97, 98, 99]]])
```

In [135]:
```
arr=np.expand_dims(mat,axis=0)
arr
```

Out[135]:
```
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, 36, 37, 38, 39],
        [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
        [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
        [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
        [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
        [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
        [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]]])
```

In [136]:
```
a=np.array([1,2,3,4,5,6])
a.shape
```

Out[136]:
```
(6,)
```

In [137]:
```
b=np.expand_dims(a,axis=1)
b.shape
```

Out[137]: (6, 1)

In [138]:
```
b=np.expand_dims(a,axis=0)
b.shape
```

Out[138]: (1, 6)

Indexing and slicing

In [139]:
```
p1=np.arange(10)
p2=np.arange(12).reshape(3,4)
p3=np.arange(8).reshape(2,2,2)
```

In [140]:
```
p1
```

Out[140]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [141]:
```
p2
```

Out[141]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [142]:
```
p3
```

Out[142]:
```
array([[[0, 1],
        [2, 3]],

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

indexing on 1D array

In [143]:
```
p1
```

Out[143]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [144]:
```
# fetching last item
p1[-1]
```

Out[144]: 9

In [145]:
```
#fetching first item
p1[0]
```

Out[145]:
```
0
```

indexing on 2D array

In [146]:
```
p2
```

Out[146]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [147]:
```
# fetching desired element:6
p2[1,2]
```

Out[147]:
```
6
```

In [148]:
```
p2[2,3]
```

Out[148]:
```
11
```

In [149]:
```
p2[1,0]
```

Out[149]:
```
4
```

inexing on 3D (Tensors)

In [150]:
```
p3
```

Out[150]:
```
array([[[0, 1],
        [2, 3]],

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

In [151]:
```
# fetching desired element
p3[1,0,1]
```

Out[151]:
```
5
```

In [152]:
```
p3[1,0,1]
```

Out[152]:
```
5
```

In [153]:
```
p3[0,0,0]
```

Out[153]:
```
0
```

In [154]:
```
p3[1,1,0]
```

Out[154]: 6

slicing

fetching multiple items

slicing on 1D

In [155]:
```
p1
```

Out[155]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [156]:
```
# fetching desired elements
p1[2:5]
```

Out[156]: array([2, 3, 4])

In [157]:
```
p1[2:5:2]
```

Out[157]: array([2, 4])

slicing in 2D

In [158]:
```
p2
```

Out[158]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [159]:
```
#fetching the first row
p2[0,:]
```

Out[159]: array([0, 1, 2, 3])

In [160]:
```
# fetching the second column
p2[:,2]
```

Out[160]: array([2, 6, 10])

In [161]:
```
p2[2,:]
```

Out[161]: array([8, 9, 10, 11])

In [162]:
```
p2
```

Out[162]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [163]:
```
p2[1:3]
```

Out[163]:
```
array([[ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [164]:
```
p2[1:3,1:3]
```

Out[164]:
```
array([[ 5,  6],
       [ 9, 10]])
```

In [165]:
```
p2
```

Out[165]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [166]:
```
p2[::2,::3]
```

Out[166]:
```
array([[ 0,  3],
       [ 8, 11]])
```

In [167]:
```
p2
```

Out[167]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [168]:
```
p2[::2]
```

Out[168]:
```
array([[ 0,  1,  2,  3],
       [ 8,  9, 10, 11]])
```

In [169]:
```
p2[::2,1::2]
```

Out[169]:
```
array([[ 1,  3],
       [ 9, 11]])
```

In [170]:
```
p2
```

Out[170]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [171]:
```
p2[1]
```

Out[171]:
```
array([4, 5, 6, 7])
```

In [172]:
```
p2[1,::3]
```

Out[172]:
```
array([4, 7])
```

In [173]:
```
p2[0:2]
```

Out[173]:
```
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

In [174]:
```
p2[0:2,1:]
```

Out[174]:
```
array([[1, 2, 3],
       [5, 6, 7]])
```

In [175]:
```
p2
```

Out[175]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [176]:
```
p2[0:2]
```

Out[176]:
```
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

In [177]:
```
p2[0:2,1::2]
```

Out[177]:
```
array([[1, 3],
       [5, 7]])
```

slicing in 3D

In [178]:
```
p3=np.arange(27).reshape(3,3,3)
p3
```

Out[178]:
```
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]]])
```

In [179]:
```
p3[1]
```

Out[179]:
```
array([[ 9, 10, 11],
       [12, 13, 14],
       [15, 16, 17]])
```

In [180]:
```
p3[::2]
```

Out[180]:
```
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8]],

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

In [181]:
```
p3
```

Out[181]:
```
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]]])
```

In [182]:
```
p3[0]
```

Out[182]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

In [183]:
```
p3[0,1,:]
```

Out[183]:
```
array([3, 4, 5])
```

In [184]:
```
p3
```

Out[184]:
```
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]]])
```

In [185]:
```
p3[1,:,1]
```

Out[185]:
```
array([10, 13, 16])
```

In [186]:
```
p3[2,1:]
```

Out[186]:
```
array([[21, 22, 23],
       [24, 25, 26]])
```

In [187]:
```
p3[2,1:,1:]
```

Out[187]:
```
array([[22, 23],
       [25, 26]])
```

In [188]:
```
p3[0::2]
```

Out[188]:
```
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8]],

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

In [189]:
```
p3[0::2,0]
```

Out[189]:
```
array([[ 0,  1,  2],
       [18, 19, 20]])
```

In [190]:
```
p3[0::2,0,::2]
```

Out[190]:
```
array([[ 0,  2],
       [18, 20]])
```

iterating

In [191]:
```
p1
```

Out[191]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [192]:
```
# Looping in 1D array


for i in p1:
    print(i)
```

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

In [193]:
```
p2
```

Out[193]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [194]:
```
## Looping in 2D array

for i in p2:
    print(i)
```

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

In [195]:
```
for i in p3:
    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]]
```

In [196]:
```
for i in np.nditer(p3):
    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
```

Stacking

stacking is the concept of joining arrays in numpy arrays having the same dimensions can be stacked

horizantal stacking

In [197]:
```
w1=np.arange(12).reshape(3,4)
w2=np.arange(12,24).reshape(3,4)
w1
```

Out[197]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [198]:
```
w2
```

Out[198]:
```
array([[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

In [199]:
```
np.hstack((w1,w2))
```

Out[199]:
```
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]])
```

vertical stacking

In [200]:
```
w1
```

Out[200]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [201]:
```
w2
```

Out[201]:
```
array([[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

In [202]:
```
np.vstack((w1,w2))
```

Out[202]:
```
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

is opposite of stacking

In [203]:
```
# Horizontal splitting

w1
```

Out[203]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [204]:
```
np.hsplit(w1,2)
```

Out[204]:
```
[array([[0, 1],
        [4, 5],
        [8, 9]]),
 array([[ 2,  3],
        [ 6,  7],
        [10, 11]])]
```

In [205]:
```
np.vsplit(w1,3)
```

Out[205]:
```
[array([[0, 1, 2, 3]]), array([[4, 5, 6, 7]]), array([[ 8,  9, 10, 11]])]
```

In [206]:
```
mat[row:]
```

Out[206]:
```
array([[40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
       [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
       [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
       [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

In [207]:
```
mat[:row]
```

Out[207]:
```
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, 36, 37, 38, 39]])
```

In [208]:
```
mat[row]
```

Out[208]:
```
array([40, 41, 42, 43, 44, 45, 46, 47, 48, 49])
```

In [209]:
```
mat[2:6,2:4]
```

Out[209]:
```
array([[22, 23],
       [32, 33],
       [42, 43],
       [52, 53]])
```

In [210]:
```
mat[1:2,2:4]
```

Out[210]:
```
array([[12, 13]])
```

In [211]:
```
mat[2:4,3:5]
```

Out[211]:
```
array([[23, 24],
       [33, 34]])
```

Masking

In [212]:
```
mat<50
```

Out[212]:
```
array([[ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False]])
```

In [213]:
```
mat>50
```

Out[213]:
```
array([[False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True]])
```

mat<50 printing boolean if i want to print values i print mat inside the mat

In [214]:
```
mat[mat<50]
```

Out[214]:
```
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, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])
```

In [215]:
```
mat[mat!=50]
```

Out[215]:
```
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, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 51,
       52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
       69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85,
       86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

In [216]:
```
[mat%2==0]
```

Out[216]:
```
[array([[ True, False,  True, False,  True, False,  True, False,  True,
          False],
        [ True, False,  True, False,  True, False,  True, False,  True,
          False],
        [ True, False,  True, False,  True, False,  True, False,  True,
          False],
        [ True, False,  True, False,  True, False,  True, False,  True,
          False],
        [ True, False,  True, False,  True, False,  True, False,  True,
          False],
        [ True, False,  True, False,  True, False,  True, False,  True,
          False],
        [ True, False,  True, False,  True, False,  True, False,  True,
          False],
        [ True, False,  True, False,  True, False,  True, False,  True,
          False],
        [ True, False,  True, False,  True, False,  True, False,  True,
          False],
        [ True, False,  True, False,  True, False,  True, False,  True,
          False]])]
```

In [ ]:
```

```

reverse an array

In [217]:
```
#np.flip()
```

In [218]:
```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
```

In [219]:
```
reversed_arr = np.flip(arr)
```

In [220]:
```
arr
```

Out[220]:
```
array([1, 2, 3, 4, 5, 6, 7, 8])
```

In [221]:
```
reversed_arr
```

Out[221]:
```
array([8, 7, 6, 5, 4, 3, 2, 1])
```

working with missing values

In [222]:
```
s=np.array([1,2,3,4,np.nan,6])
s
```

Out[222]:
```
array([ 1.,  2.,  3.,  4., nan,  6.])
```

In [223]:
```
np.isnan(s)
```

Out[223]:
```
array([False, False, False, False,  True, False])
```

In [224]:
```
s[np.isnan(s)]
```

Out[224]:
```
array([nan])
```

In [225]:
```
s[~np.isnan(s)]
```

Out[225]:
```
array([1., 2., 3., 4., 6.])
```

plottilng graphs

In [226]:
```
# plotting a 2D graph
# x=y

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

Out[226]:
```
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.        ])
```

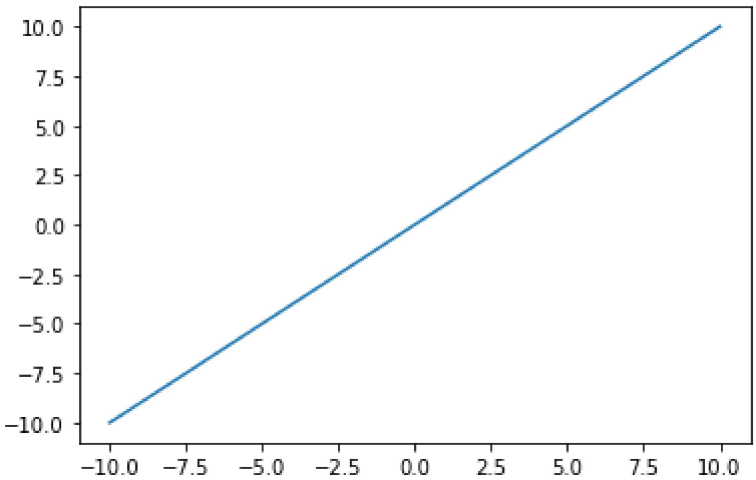
In [227]:
```
y=x
```

In [228]:
```
y
```

Out[228]:
```
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.        ])
```

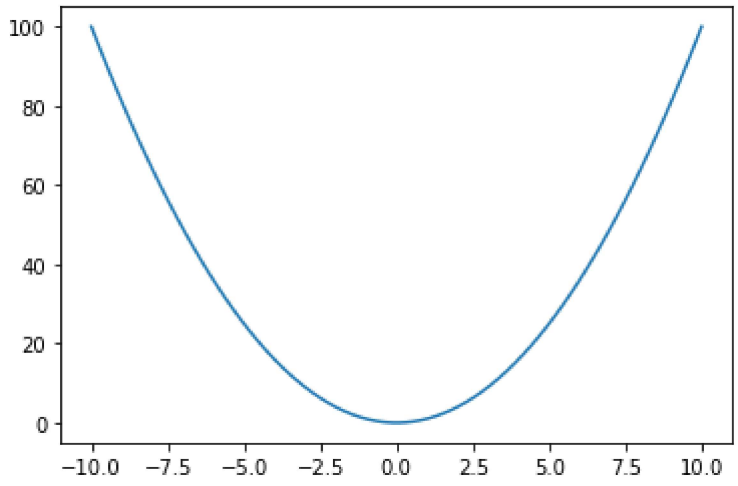
In [229]:
```
import matplotlib.pyplot as plt
plt.plot(x,y)
```

Out[229]:
```
[<matplotlib.lines.Line2D at 0x1f38df5f640>]
```



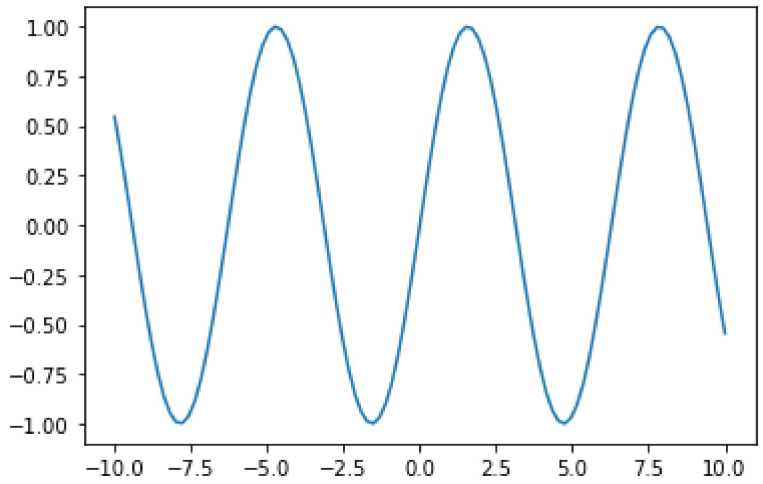
In [230]:
```
x=np.linspace(-10,10,100)
y=x**2
plt.plot(x,y)
```

Out[230]: [<matplotlib.lines.Line2D at 0x1f38e0515e0>]



In [231]:
```
x=np.linspace(-10,10,100)
y=np.sin(x)
plt.plot(x,y)
```

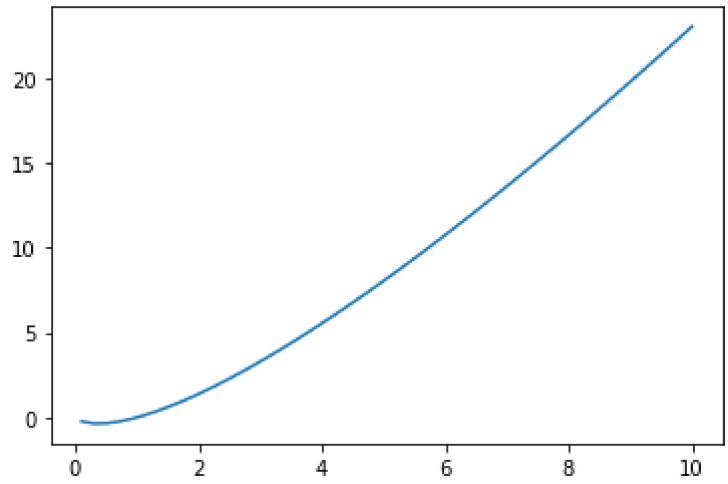
Out[231]: [<matplotlib.lines.Line2D at 0x1f38e0b8400>]



In [232]:
```
x=np.linspace(-10,10,100)
y=x* np.log(x)
plt.plot(x,y)
```

```
<ipython-input-232-52e63699cdf9>:2: RuntimeWarning: invalid value encounter
ed in log
  y=x* np.log(x)
```

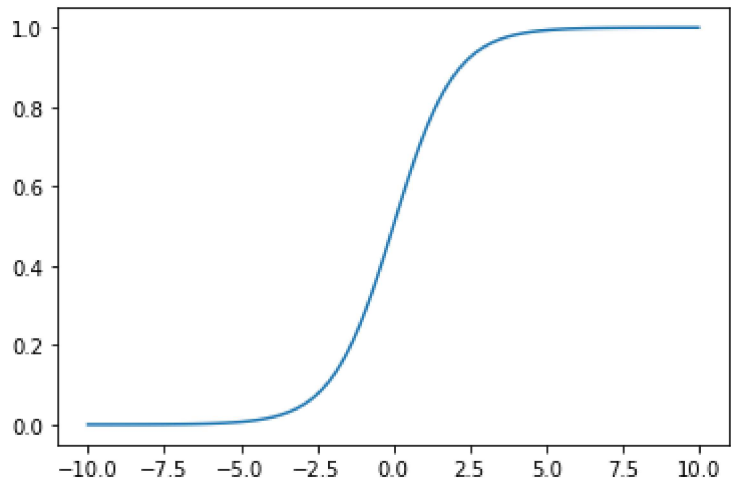
Out[232]:
```
[<matplotlib.lines.Line2D at 0x1f38e113c70>]
```



In [233]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [234]:
```
x=np.linspace(-10,10,100)
y=1/(1+np.exp(-x))
plt.plot(x,y)
```

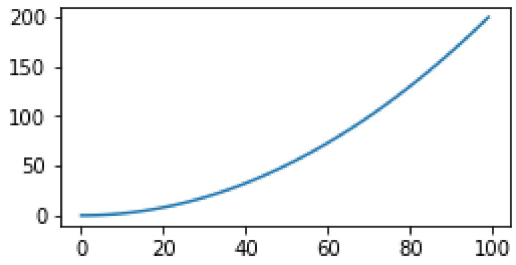
Out[234]:
```
[<matplotlib.lines.Line2D at 0x1f38e1731c0>]
```



In [235]:
```
x=np.linspace(0,10,100)
y=np.linspace(0,10,100)
```

In [236]:
```
f=x**2+y**2
```

In [237]:
```
plt.figure(figsize=(4,2))
plt.plot(f)
plt.show()
```



In [238]:
```
x=np.arange(3)
y=np.arange(3)
```

In [239]:
```
x
```

Out[239]:
```
array([0, 1, 2])
```

In [240]:
```
y
```

Out[240]:
```
array([0, 1, 2])
```

Generating a mesh grid

In [241]:
```
xv,yv=np.meshgrid(x,y)
```

In [242]:
```
xv
```

Out[242]:
```
array([[0, 1, 2],
       [0, 1, 2],
       [0, 1, 2]])
```

Broadcasting

the term broadcasting describes how numpy treats array with shapes during arithemetic operations the smaller array is broadcast across the larger array so that they have compatible shapes

In [243]:
```
#same shape

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]]
```

In [244]:
```
# diff shape

a=np.arange(6).reshape(2,3)
b=np.arange(3).reshape(1,3)
print(a)
print(b)
print(a+b)
```

```
[[0 1 2]
 [3 4 5]]
[[0 1 2]]
[[0 2 4]
 [3 5 7]]
```

a has 2 rows and three columns and b has 1 row and 3 columns so we have to add both a and b son i b array add[[0 1 2][0 1 2]]

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 to the head of the array with the smaller dimension.

In [245]:
```
a= np.arange(12).reshape(4,3)
b=np.arange(3)
print(a)
```

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

In [246]:
```
print(b)
```

```
[0 1 2]
```

In [247]:
```
print(a+b)
```

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

In [248]:
```
# could not broadcast

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

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

In [249]:
```
print(b)
```

```
[0 1 2]
```

In [250]:
```
print(a+b)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-250-85111a3ab660> in <module>
----> 1 print(a+b)

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

In [251]:
```
a=np.arange(3).reshape(1,3)
b=np.arange(3).reshape(3,1)
print(a)
```

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

In [252]:
```
print(b)
```

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

In [253]:
```
print(a+b)
```

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

In [254]:
```
a=np.arange(3).reshape(1,3)
b=np.arange(4).reshape(4,1)

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

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

In [255]:
```
np.random.rand(3,2)
```

Out[255]:
```
array([[0.18802254, 0.05277788],
       [0.99225185, 0.92616072],
       [0.80082787, 0.39726122]])
```

In [256]:
```
np.random.rand(3)
```

Out[256]:
```
array([0.4729871 , 0.43967647, 0.74013863])
```

In [257]:
```
np.random.randint(4,6)
```

Out[257]:
```
5
```

In [258]:
```
np.random.randint(0,10)
```

Out[258]:
```
4
```

In [259]:
```
np.random.randint(0,10,4)
```

Out[259]:
```
array([1, 8, 9, 5])
```

(0 --->first number)

(10---> last hnumber)

(4---> size of an array)

In [260]:
```
np.random.randint(0,10,5)
```

Out[260]:
```
array([2, 3, 0, 8, 4])
```

In [261]:
```
n=np.random.randint(10,40,(8,10))
```

In [262]:
```
n
```

Out[262]:
```
array([[22, 38, 22, 35, 37, 30, 36, 34, 38, 15],
       [39, 28, 18, 12, 22, 14, 24, 11, 39, 13],
       [28, 31, 32, 18, 28, 18, 33, 33, 18, 19],
       [20, 19, 34, 16, 38, 36, 30, 37, 33, 13],
       [37, 30, 25, 13, 33, 16, 13, 38, 25, 32],
       [13, 11, 12, 22, 35, 21, 12, 22, 30, 35],
       [35, 30, 39, 20, 35, 17, 33, 12, 37, 28],
       [27, 27, 17, 18, 27, 29, 30, 16, 13, 39]])
```