Creating Numpy array

In [96]:
```
import numpy as np
```

In [98]:
```
np.__version__
```

Out[98]:
```
'1.26.4'
```

In [100…
```
np.array([2,4,56,422,32,1]) # 1D array
```

Out[100…
```
array([  2,   4,  56, 422,  32,   1])
```

In [102…
```
a = np.array([2,4,56,422,32,1]) # vector
print (a)
```

```
[  2   4  56 422  32   1]
```

In [104…
```
type (a)
```

Out[104…
```
numpy.ndarray
```

In [106…
```
b = np.array([[45,34,22,2],[24,55,3,22]])  # 2D array ( MATrix)
print (b)
```

```
[[45 34 22  2]
 [24 55  3 22]]
```

In [108…
```
np.array ([[2,3,33,4,45],[23,45,56,66,2],[357,523,32,24,2],[32,32,44,33,234]]) #
```

Out[108…
```
array([[  2,   3,  33,   4,  45],
       [ 23,  45,  56,  66,   2],
       [357, 523,  32,  24,   2],
       [ 32,  32,  44,  33, 234]])
```

Datatypes

In [111…
```
np.array([11,23,44],dtype =int)
```

Out[111…
```
array([11, 23, 44])
```

In [113…
```
np.array([11,23,44],dtype = float)
```

Out[113…
```
array([11., 23., 44.])
```

In [115…
```
np.array([11,23,44],dtype = bool) # here True because, python treats Non-zero
```

Out[115…
```
array([ True,  True,  True])
```

In [117…
```
np.array([11,23,44],dtype = complex)
```

Out[117…
```
array([11.+0.j, 23.+0.j, 44.+0.j])
```

Arrange

In [ ]:
```
arrange can be called with a varying number of positional arguments
```

In [122…
```
np.arange(1,25) # 1-included , 25- last one got excluded
```

Out[122…
```
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])
```

In [124…
```
np.arange(1,25,2) # strides ----alternate numbers (interval os 2)
```

Out[124…
```
array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23])
```

Reshape

In [ ]:
```
Both of number products should be equal to number of items present inside the ar
```

In [129…
```
np.arange(1,11).reshape(2,5) # converted 2 rows and 5 columns
```

Out[129…
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

In [131…
```
np.arange(1,13).reshape(3,4) # converted 3 rows and 4 columns
```

Out[131…
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

Ones & Zeros

In [ ]:
```
you can initialize the values and create values.ex:in deep learning weight shape
```

In [136…
```
np.zeros((3,5))
```

Out[136…
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [138…
```
np.zeros((3,10))
```

Out[138…
```
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.]])
```

In [139…
```
np.zeros((10,5))
```

Out[139…
```
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., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [140…
```
np.ones((5,7))
```

Out[140…
```
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., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.]])
```

In [141…
```
import numpy as np
```

In [142…
```
np.ones((2,4))
```

Out[142…
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

In [143…
```
np.ones((2,4),dtype = int)
```

Out[143…
```
array([[1, 1, 1, 1],
       [1, 1, 1, 1]])
```

In [144…
```
# Another Type ----> random()

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

Out[144…
```
array([[0.88359012, 0.88046247, 0.11651242],
       [0.52146585, 0.40870547, 0.47492028],
       [0.22175904, 0.84140412, 0.93220792],
       [0.59972028, 0.34459329, 0.58962689]])
```

linespace

In [ ]:
```
it is also called as linearly space,Linearly separable,in a given range at equal
```

In [147…
```
np.linspace(-10,10,10) #Lower range,upper range ,number of items to be generate
```

Out[147…
```
array([-10.        ,  -7.77777778,  -5.55555556,  -3.33333333,
       -1.11111111,   1.11111111,   3.33333333,   5.55555556,
        7.77777778,  10.        ])
```

In [148…
```
np.linspace(-2.5,35.6,6)
```

Out[148…
```
array([-2.5 ,  5.12, 12.74, 20.36, 27.98, 35.6 ])
```

identity

In [ ]:
```
identity martix is that diagonal items will be ones and everything will be zeros
```

In [151…
```
# creating the identity martix

np.identity(4)
```

Out[151…
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

In [152…
```
np.identity(7)
```

Out[152…
```
array([[1., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 1.]])
```

Array Attributes

In [154…
```
a1 = np.arange(10)  # 1D
a1
```

Out[154…
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [155…
```
a2 = np.arange(12,dtype = float).reshape(3,4) # martix
a2
```

Out[155…
```
array([[ 0.,  1.,  2.,  3.],
       [ 4.,  5.,  6.,  7.],
       [ 8.,  9., 10., 11.]])
```

In [156…
```
a3 = np.arange(8).reshape(2,2,2) # 3D---->tensor
a3
```

Out[156…
```
array([[[0, 1],
        [2, 3]],

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

ndim

In [ ]:
```
to findout given arrays number of dimensions
```

In [159…
```
a1.ndim
```

Out[159…
```
1
```

In [160…
```
a2.ndim
```

Out[160… 2

In [161…
```
a3.ndim
```

Out[161… 3

shape

In [163…
```
a1.shape # 1D array has 10 items
```

Out[163… (10,)

In [164…
```
a2.shape # 3rows and 4 columns
```

Out[164… (3, 4)

In [165…
```
a3.shape # first ,2 says it consists of 2d arrays.2,2 gives no.of rows and colum
```

Out[165… (2, 2, 2)

size

In [ ]:
```
gives number of items
```

In [168…
```
a3
```

Out[168…
```
array([[[0, 1],
        [2, 3]],

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

In [169…
```
a3.size # i has 8 items .like shape :2,2,2=8
```

Out[169… 8

In [170…
```
a2
```

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

In [171…
```
a2.size
```

Out[171… 12

item size

In [ ]:
```
memory occupied by the item
```

In [174…
```
a1
```

Out[174…
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [175…
```
a1.itemsize  # bytes
```

Out[175…
```
4
```

In [176…
```
a2.itemsize # integer 64 gives = 8 bytes
```

Out[176…
```
8
```

In [177…
```
a3.itemsize # integer 32 gives = 4 bytes
```

Out[177…
```
4
```

Dtype

In [ ]:
```
gives data type of the item
```

In [180…
```
print(a1.dtype)
print(a2.dtype)
print(a3.dtype)
```

```
int32
float64
int32
```

Changing Data Type

In [182…
```
#astype

x = np.array([33,22,2.5])
x
```

Out[182…
```
array([33. , 22. ,  2.5])
```

In [183…
```
x.astype(int)
```

Out[183…
```
array([33, 22,  2])
```

Array operations

In [185…
```
z1 = np.arange(12).reshape(3,4)
z2 = np.arange(12,24).reshape(3,4)
```

In [186…
```
z1
```

Out[186…
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [187…
```
z2
```

Out[187…
```
array([[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

Scalar operations

In [ ]:
```
scalar operations on numpy arrays includeperformingaddition or subtraction or mu
```

In [190…
```
# arithmetic
z1+2
```

Out[190…
```
array([[ 2,  3,  4,  5],
       [ 6,  7,  8,  9],
       [10, 11, 12, 13]])
```

In [191…
```
# subtraction
z1-2
```

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

In [192…
```
# multiplication
z1*2
```

Out[192…
```
array([[ 0,  2,  4,  6],
       [ 8, 10, 12, 14],
       [16, 18, 20, 22]])
```

In [193…
```
# power
z1 ** 2
```

Out[193…
```
array([[  0,   1,   4,   9],
       [ 16,  25,  36,  49],
       [ 64,  81, 100, 121]])
```

In [194…
```
## modulo
z1 % 2
```

Out[194…
```
array([[0, 1, 0, 1],
       [0, 1, 0, 1],
       [0, 1, 0, 1]], dtype=int32)
```

Relational operators

In [196…
```
z2
```

Out[196…
```
array([[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

In [197…
```
z2>2 # if 2 is greaterthan everything gives true
```

Out[197…
```
array([[ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True]])
```

In [198…
```
z2<22 # if 22 greater numbers gives true & lesser number gives false
```

Out[198…
```
array([[ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True, False, False]])
```

vector operation

In [ ]:
```
we can apply on both numpy array
```

In [201…
```
z1
```

Out[201…
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [202…
```
z2
```

Out[202…
```
array([[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

In [203…
```
z1+z2
```

Out[203…
```
array([[12, 14, 16, 18],
       [20, 22, 24, 26],
       [28, 30, 32, 34]])
```

In [204…
```
z1*z2
```

Out[204…
```
array([[  0,  13,  28,  45],
       [ 64,  85, 108, 133],
       [160, 189, 220, 253]])
```

In [205…
```
z1-z2
```

Out[205…
```
array([[-12, -12, -12, -12],
       [-12, -12, -12, -12],
       [-12, -12, -12, -12]])
```

In [206…
```
z1/z2
```

Out[206…
```
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 Functions

In [208…
```
k1 = np.random.random((3,3))
k1 = np.round(k1*100)
k1
```

Out[208…
```
array([[74., 41., 31.],
       [33., 41., 25.],
       [50., 38.,  1.]])
```

In [209…
```
# max
np.max(k1)
```

Out[209…
```
74.0
```

In [210…
```
# min
np.min(k1)
```

Out[210…
```
1.0
```

In [211…
```
# sum
np.sum(k1)
```

Out[211…
```
334.0
```

In [212…
```
# prod---->multiplication

np.prod(k1)
```

Out[212…
```
6044615445000.0
```

In Numpy

In [ ]:
```
0= column , 1= row
```

In [215…
```
# if we want maximum of every row
np.max(k1,axis = 1)
```

Out[215…
```
array([74., 41., 50.])
```

In [216…
```
# maximum of every column

np.max(k1,axis = 0)
```

Out[216…
```
array([74., 41., 31.])
```

In [217…
```
# product of every column

np.prod(k1, axis= 0)
```

Out[217…
```
array([122100.,  63878.,    775.])
```

Statistics related functions

In [219…
```
# mean
k1
```

Out[219…
```
array([[74., 41., 31.],
       [33., 41., 25.],
       [50., 38.,  1.]])
```

In [220…
```
np.mean(k1)
```

Out[220…
```
37.111111111111114
```

In [221…
```
# mean of every column

k1.mean(axis=0)
```

Out[221…
```
array([52.33333333, 40.        , 19.        ])
```

In [222…
```
# median
np.median(k1)
```

Out[222…
```
38.0
```

In [223…
```
np.median(k1,axis = 1)
```

Out[223…
```
array([41., 33., 38.])
```

In [224…
```
# Standard deviation

np.std(k1)
```

Out[224…
```
18.44779086108472
```

In [225…
```
np.std(k1,axis = 0)
```

Out[225…
```
array([16.81930108,  1.41421356, 12.9614814 ])
```

In [226…
```
# variance

np.var(k1)
```

Out[226…
```
340.32098765432096
```

Trignometry Functions

In [228…
```
np.sin(k1) # sin
```

Out[228…
```
array([[-0.98514626, -0.15862267, -0.40403765],
       [ 0.99991186, -0.15862267, -0.13235175],
       [-0.26237485,  0.29636858,  0.84147098]])
```

In [229…
```
np.cos(k1) # cos
```

Out[229…
```
array([[ 0.17171734, -0.98733928,  0.91474236],
       [-0.01327675, -0.98733928,  0.99120281],
       [ 0.96496603,  0.95507364,  0.54030231]])
```

In [230…
```
np.tan(k1) # Tan
```

Out[230…
```
array([[ -5.73702254,   0.1606567 ,  -0.44169557],
       [-75.3130148 ,   0.1606567 ,  -0.13352641],
       [ -0.27190061,   0.31030966,   1.55740772]])
```

In [231…
```
np.sec(k1)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[231], line 1
----> 1 np.sec(k1)

File ~\anaconda3\Lib\site-packages\numpy\__init__.py:333, in __getattr__(attr)
    330     "Removed in NumPy 1.25.0"
    331     raise RuntimeError("Tester was removed in NumPy 1.25.")
--> 333 raise AttributeError("module {!r} has no attribute "
    334                       "{!r}".format(__name__, attr))

AttributeError: module 'numpy' has no attribute 'sec'
```

dot product

In [ ]:
```
the nump module of python provides a function to perform the dot product of two
```

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

In [ ]:
```
s2
```

In [ ]:
```
s3
```

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

log and Exponents

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

round/floor/ceil

round

In [ ]:
```
the numpy.round()function rounds the elements of an array to the nearest integer
```

In [234…
```
# 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 [235…
```
# 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 [236…
```
# randomly
np.round(np.random.random((2,3))*100)
```

Out[236…
```
array([[69., 34., 13.],
       [76., 13., 23.]])
```

floor

In [238…
```
arr
```

Out[238…
```
array([1.234, 2.567, 3.891])
```

In [239…
```
np.floor(arr) # floor function returns the less than or equal to the item
```

Out[239…
```
array([1., 2., 3.])
```

In [240…
```
np.floor(np.random.random((3,3))*100)
```

Out[240…
```
array([[10.,  5., 59.],
       [96., 74., 79.],
       [35., 63., 52.]])
```

ceil

In [242…
```
arr
```

Out[242…
```
array([1.234, 2.567, 3.891])
```

In [243…
```
np.ceil(arr) # returns the greter number
```

Out[243…
```
array([2., 3., 4.])
```

In [244…
```
np.ceil(np.random.random((2,3))*100)
```

Out[244…
```
array([[57., 85., 52.],
       [52., 67., 95.]])
```

Indexing and slicing

In [246…
```
r1 = np.arange(20)
```

In [247…
```
r1
```

Out[247…
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

In [248…
```
r2 = np.arange(12).reshape(3,4)
```

In [249…
```
r2
```

Out[249…
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [250…
```
r3 = np.arange(8).reshape(2,2,2)
```

In [251…
```
r3
```

Out[251…
```
array([[[0, 1],
        [2, 3]],

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

In [252…
```
r1
```

Out[252…
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

In [253…
```
r2
```

Out[253…
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [254…
```
r2[1,2]              # 1st argument denotes row & 2nd argument denotes column
```

Out[254…
```
6
```

In [255…
```
r2[2,3]              # 2nd row and 3rd column i.e..11 and index starts with 0.
```

Out[255…
```
11
```

In [256…
```
r2[1,0]              # 1st row and 0th column i.e..4
```

Out[256…
```
4
```

Indexing on 3D (Tensors)

In [258…
```
r3
```

Out[258…
```
array([[[0, 1],
        [2, 3]],

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

In [259…
```
r3[1,0,1]
```

Out[259…
```
5
```

In [260…
```
r3[1,1,1]
```

Out[260…
```
7
```

In [261…
```
r3[0,1,1]
```

Out[261…
```
3
```

slicing in 1D array

In [263…
```
r1
```

Out[263…
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

In [264…
```
r2
```

Out[264…
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [265…
```
r1
```

Out[265…
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

In [266…
```
r1[2:5]
```

Out[266…
```
array([2, 3, 4])
```

In [267…
```
r1[3:]  # from 3rd index to rest of elements
```

Out[267…
```
array([ 3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

In [268…
```
r1[:] # prints all the elements
```

Out[268…
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

In [269…
```
r1[:10] # prints from starting to the (10-1) i.e.. up to 9 index
```

Out[269…
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [270…
```
r1[2:9:3] # prints the elements from 2 to 7 with step count of 2
```

Out[270…
```
array([2, 5, 8])
```

Slicing on 2D array

In [272…
```
r2
```

Out[272…
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [273…
```
r2[0,:]  # prints the total first row
```

Out[273…
```
array([0, 1, 2, 3])
```

In [274…
```
r2[2,:]
```

Out[274…
```
array([ 8,  9, 10, 11])
```

In [275…
```
 r2[:,1] # returns the second column  (1st argument represents the row and 2nd a
```

Out[275…
```
array([1, 5, 9])
```

In [276…
```
r2[:,:] # returns the entire martix
```

Out[276…
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [277…
```
r2[1:3]
```

Out[277…
```
array([[ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [278…
```
r2[1:3,1:3] # returns the 1st and 3rd row &1st and 3rd columns
```

Out[278…
```
array([[ 5,  6],
       [ 9, 10]])
```

In [279…
```
r2[::2,::3]
```

Out[279…
```
array([[ 0,  3],
       [ 8, 11]])
```

In [280…
```
r2[::2,1::2]
```

Out[280…
```
array([[ 1,  3],
       [ 9, 11]])
```

In [281…
```
r2[1::2,::3]
```

Out[281…
```
array([[4, 7]])
```

In [282…
```
r2[:2,1:]
```

Out[282…
```
array([[1, 2, 3],
       [5, 6, 7]])
```

In [283…
```
r2[:2,1::2]
```

Out[283…
```
array([[1, 3],
       [5, 7]])
```

Slicing on 3D array

In [285…
```
import numpy as np
```

In [286…
```
r3 = np.arange(27).reshape(3,3,3)
```

In [287…
```
r3
```

Out[287…
```
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 [288…
```
r3[::2]      # selects the subarrays at indicates 0 and 2
```

Out[288…
```
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8]],

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

In [289…
```
r3[0]   # first subarray
```

Out[289…
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

In [290…
```
r3[0,1]    # first subarray second row
```

Out[290…
```
array([3, 4, 5])
```

In [291…
```
r3[0,1:]
```

Out[291…
```
array([[3, 4, 5],
       [6, 7, 8]])
```

In [292…
```
r3[1,1:]     # 2nd numpy array and middle row
```

Out[292…
```
array([[12, 13, 14],
       [15, 16, 17]])
```

In [293…
```
r3
```

Out[293…
```
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 [294…
```
r3[1]   # second numpy array
```

Out[294…
```
array([[ 9, 10, 11],
       [12, 13, 14],
       [15, 16, 17]])
```

In [295…
```
r3[1,:,:]   # entire second matrix
```

Out[295…
```
array([[ 9, 10, 11],
       [12, 13, 14],
       [15, 16, 17]])
```

In [296…
```
r3[1,1]   # second numpy array second row
```

Out[296…
```
array([12, 13, 14])
```

In [297…
```
r3[1,:,1]     # 1 represents middle numpy array , :represents total and 1 repres
```

Out[297…
```
array([10, 13, 16])
```

In [298…
```
r3[2,1:,1:] # third numpy array,2nd row to end ,2nd column to end
```

Out[298…
```
array([[22, 23],
       [25, 26]])
```

In [299…
```
r3[::2,0,::2]
```

Out[299…
```
array([[ 0,  2],
       [18, 20]])
```

Iterating

In [301…
```
r1 = np.arange(20)
```

In [302…
```
r1
```

Out[302…
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

In [303…
```
for i in r1:             #looping on 1D array
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

In [ ]:
```
r2 = np.arange(12),reshape(3,4)
```

In [ ]:
```
r2
```

In [305…
```
for i in r2:     # Looping in 2D array
    print(i)
```

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

In [306…
```
r3 = np.arange(27).reshape(3,3,3)
```

In [307…
```
r3
```

Out[307…
```
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 [308…
```
for i in r3:    # print all items in 3D using nditer and first convert in 1D and
    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]]
```

Transpose

In [310…
```
# Transpose interchanges the rows and columns
```

In [311…
```
r2
```

Out[311…
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [312…
```
np.transpose(r2)   # rows and columns will be interchanges
```

Out[312…
```
array([[ 0,  4,  8],
       [ 1,  5,  9],
       [ 2,  6, 10],
       [ 3,  7, 11]])
```

In [313…
```
r2.transpose()   # Another method
```

Out[313…
```
array([[ 0,  4,  8],
       [ 1,  5,  9],
       [ 2,  6, 10],
       [ 3,  7, 11]])
```

In [314…
```
r2.T         # Another mwthod
```

Out[314…
```
array([[ 0,  4,  8],
       [ 1,  5,  9],
       [ 2,  6, 10],
       [ 3,  7, 11]])
```

In [315…
```
r3
```

Out[315…
```
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 [316…
```
r3.T
```

Out[316…
```
array([[[ 0,  9, 18],
        [ 3, 12, 21],
        [ 6, 15, 24]],

       [[ 1, 10, 19],
        [ 4, 13, 22],
        [ 7, 16, 25]],

       [[ 2, 11, 20],
        [ 5, 14, 23],
        [ 8, 17, 26]]])
```

Ravel

In [318…
```
# Ravel converts any dimension to ID
```

In [319…
```
r2
```

Out[319…
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [320…
```
r2.ravel()       # converts the 2D array to 1D array
```

Out[320…
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

In [321…
```
r3
```

Out[321…
```
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 [322…
```
r3.ravel()
```

Out[322…
```
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])
```

Stacking

In [324…
```
# Stacking joins the arrays in numpy
```

In [325…
```
c1 = np.arange(12).reshape(3,4)
```

In [326…
```
c1
```

Out[326…
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [328…
```
c2 = np.arange(12,24).reshape(3,4)
```

In [329…
```
c2
```

Out[329…
```
array([[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

In [330…
```
# Arrays having the sum dimension can be stacked
```

In [331…
```
np.hstack((c1,c2))              # hstack used for horizontal stacking
```

Out[331…
```
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]])
```

In [332…
```
np.vstack((c1,c2))             # vstack used for vertical stacking
```

Out[332…
```
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

In [334…
```
# Splitting is opposite of stacking .it splits the numpy array
```

In [335…
```
c1
```

Out[335…
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [336…
```
c2
```

Out[336…
```
array([[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

In [337…
```
np.hsplit(c1,2)            # horizontal splitting and it splits it into 2
```

Out[337…
```
[array([[0, 1],
        [4, 5],
        [8, 9]]),
 array([[ 2,  3],
        [ 6,  7],
        [10, 11]])]
```

In [338…
```
np.vsplit(c2,3)          # vertical splitting and it splits into 3
```

Out[338…
```
[array([[12, 13, 14, 15]]),
 array([[16, 17, 18, 19]]),
 array([[20, 21, 22, 23]])]
```

In [339…
```
# In numpy , all the elements should be of same data type and thus will be of sa
```

Speed of List Numpy

List

In [342…
```
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)
```

```
4.419130563735962
```

Numpy

In [344…
```
import numpy as np
a1 = np.arange(10000000)
b1 = np.arange(10000000,20000000)
import time
start = time.time()
c =a+b;
print(time.time()-start)
```

```
0.5092999935150146
```

In [345…
```
4.172647714614868 / 0.570059061050415
```

Out[345…
```
7.319676152373002
```

In [346…
```
# Numpy uses Ctype array
```

Memeory Used for List Vs Numpy

List

In [349…
```
a2 = [i for i in range(10000000)]
import sys
sys.getsizeof(a2)
```

Out[349…
```
89095160
```

In [350…
```
# Numpy
```

In [351…
```
import numpy as np
a3 = np.arange(10000000)
import sys
sys.getsizeof(a3)
```

Out[351…
```
40000112
```

In [352…
```
a4 = np.arange(10000000,dtype=np.int16)
import sys
sys.getsizeof(a4)
```

Out[352…
```
20000112
```

Advance Indexing and slicing

In [354…
```
import numpy as np
```

In [355…
```
d = np.arange(12).reshape(4,3)
```

In [356…
```
d
```

Out[356…
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

In [357…
```
d[1,2]
```

Out[357…
```
5
```

In [358…
```
d[1:3]
```

Out[358…
```
array([[3, 4, 5],
       [6, 7, 8]])
```

In [359…
```
d[1:3,1:3]
```

Out[359…
```
array([[4, 5],
       [7, 8]])
```

Fancy Indexing

In [361…
```
d
```

Out[361…
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

In [362…
```
d[[0,2,3]]        #returns the 1st,3rd,and 4th rows
```

Out[362…
```
array([[ 0,  1,  2],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

In [363…
```
d[:,[0,2]]
```

Out[363…
```
array([[ 0,  2],
       [ 3,  5],
       [ 6,  8],
       [ 9, 11]])
```

Boolean Indexing

In [365…
```
# It allows you to select elements used on boolean conditions
```

In [402…
```
e = np.random.randint(1,100,24).reshape(6,4)
```

In [404…
```
e
```

Out[404…
```
array([[77, 48, 29, 35],
       [30, 27, 89,  1],
       [91,  4, 11, 39],
       [83, 80, 92, 94],
       [66, 19, 30, 32],
       [68, 11, 85, 90]])
```

In [406…
```
e > 50
```

Out[406…
```
array([[ True, False, False, False],
       [False, False,  True, False],
       [ True, False, False, False],
       [ True,  True,  True,  True],
       [ True, False, False, False],
       [ True, False,  True,  True]])
```

In [408…
```
e <50
```

Out[408…
```
array([[False,  True,  True,  True],
       [ True,  True, False,  True],
       [False,  True,  True,  True],
       [False, False, False, False],
       [False,  True,  True,  True],
       [False,  True, False, False]])
```

In [410…
```
e == 50
```

Out[410…
```
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]])
```

In [412…
```
e != 50
```

Out[412…
```
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]])
```

In [418…
```
e%2==0
```

Out[418…
```
array([[False,  True, False, False],
       [ True, False, False, False],
       [False,  True, False, False],
       [False,  True,  True,  True],
       [ True, False,  True,  True],
       [ True, False, False,  True]])
```

In [420…
```
e[e%2==0]
```

Out[420…
```
array([48, 30,  4, 80, 92, 94, 66, 30, 32, 68, 90])
```

In [422…
```
(e>50)&(e%2==0)
```

Out[422…
```
array([[False, False, False, False],
       [False, False, False, False],
       [False, False, False, False],
       [False,  True,  True,  True],
       [ True, False, False, False],
       [ True, False, False,  True]])
```

In [424…
```
e[(e>50)&(e%2==0)]
```

Out[424…
```
array([80, 92, 94, 66, 68, 90])
```

In [426…
```
e%7==0
```

Out[426…
```
array([[ True, False, False,  True],
       [False, False, False, False],
       [ True, False, False, False],
       [False, False, False, False],
       [False, False, False, False],
       [False, False, False, False]])
```

In [428…
```
e[e%7==0]
```

Out[428…
```
array([77, 35, 91])
```

In [430…
```
e[~(e%7==0)]
```

Out[430…
```
array([48, 29, 30, 27, 89,  1,  4, 11, 39, 83, 80, 92, 94, 66, 19, 30, 32,
       68, 11, 85, 90])
```

In [432…
```
~(e%7==0)
```

Out[432…
```
array([[False,  True,  True, False],
       [ True,  True,  True,  True],
       [False,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True]])
```

Boardcasting

In [ ]:
```
# it describes how many tretas arrays with different shapes during arithmetic op
```

Same shape

In [436…
```
a = np.arange(6).reshape(2,3)
```

In [438…
```
a
```

Out[438…
```
array([[0, 1, 2],
       [3, 4, 5]])
```

In [442…
```
b = np.arange(6,12).reshape(2,3)
```

In [444…
```
b
```

Out[444…
```
array([[ 6,  7,  8],
       [ 9, 10, 11]])
```

In [446…
```
a+b
```

Out[446…
```
array([[ 6,  8, 10],
       [12, 14, 16]])
```

Different shapes

In [449…
```
a=np.arange(6).reshape(2,3)
b=np.arange(3).reshape(1,3)
print(a)
print(b)
```

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

In [451…
```
a+b
```

Out[451…
```
array([[0, 2, 4],
       [3, 5, 7]])
```

In [453…
```
c=np.arange(1,10).reshape(3,3)
d=np.arange(-1,2).reshape(1,3)
```

In [455…
```
print(c)
print(d)
```

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

In [457…
```
c+d
```

Out[457…
```
array([[ 0,  2,  4],
       [ 3,  5,  7],
       [ 6,  8, 10]])
```

In [459…
```
e=d.reshape(3,1)
```

In [461…
```
c+e
```

Out[461…
```
array([[ 0,  1,  2],
       [ 4,  5,  6],
       [ 8,  9, 10]])
```

In [463…
```
f=d.reshape(1,3)
```

In [465…
```
f
```

Out[465…
```
array([[-1,  0,  1]])
```

In [467…
```
f+e
```

Out[467…
```
array([[-2, -1,  0],
       [-1,  0,  1],
       [ 0,  1,  2]])
```

Working with mathematical formulas

In [470…
```
x = np.arange(10)
```

In [472…
```
x
```

Out[472…
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [474…
```
np.sum(x)
```

Out[474…
```
45
```

In [476…
```
np.sin(x)
```

Out[476…
```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
       -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849])
```

Sigmoid

In [682…
```
def sigmoid(array):
  return 1/(1+np.exp(-(array)))
x=np.arange(10)
sigmoid(x)
```

Out[682…
```
array([0.5       , 0.73105858, 0.88079708, 0.95257413, 0.98201379,
       0.99330715, 0.99752738, 0.99908895, 0.99966465, 0.99987661])
```

In [684…
```
x=np.arange(100)
sigmoid(x)
```

Out[684…
```
array([0.5       , 0.73105858, 0.88079708, 0.95257413, 0.98201379,
       0.99330715, 0.99752738, 0.99908895, 0.99966465, 0.99987661,
       0.9999546 , 0.9999833 , 0.99999386, 0.99999774, 0.99999917,
       0.99999969, 0.99999989, 0.99999996, 0.99999998, 0.99999999,
       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.        ,
       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.        ,
       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.        ])
```

Mean squared error

In [699…
```
a= np.random.randint(1,50,25)  # 25 elements between the range 1 to49
a
```

Out[699…
```
array([19, 45, 28, 42, 21, 32, 12, 36, 48, 37,  2,  8, 16, 40, 33, 44,  5,
       43, 39, 33, 18, 20, 15, 22,  9])
```

In [695…
```
b = np.random.randint(1,50,25)
```

In [697…
```
b
```

Out[697…
```
array([37, 20, 33,  1,  5, 21, 32, 12, 40, 42, 21, 25, 46, 44, 33, 43, 43,
       40, 24, 35,  3, 48,  9, 46, 34])
```

In [701…
```
def mse(a,b):
    return np.mean((a-b)**2)
mse(a,b)
```

Out[701…
```
383.68
```

Working with missing values

In [704…
```
# for the missing values use np.nan
```

In [706…
```
s=np.array([1,2,3,4,np.nan,6])
```

In [708…
```
s
```

Out[708…
```
array([ 1.,  2.,  3.,  4., nan,  6.])
```

In [710…
```
np.isnan(s)    # returns true if the element is missing otherwise false
```

Out[710… array([False, False, False, False, True, False])

In [712…
```
s[np.isnan(s) ]      # return the nan values
```

Out[712… array([nan])

In [714…
```
s[~(np.isnan(s))]    # returns the non nan values
```

Out[714… array([1., 2., 3., 4., 6.])

Plotting graphs

In [717…
```
# plotting a 2D plot
```

In [719…
```
x = np.linspace(-10,10,100)
x
```

Out[719…
```
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.        ])
```

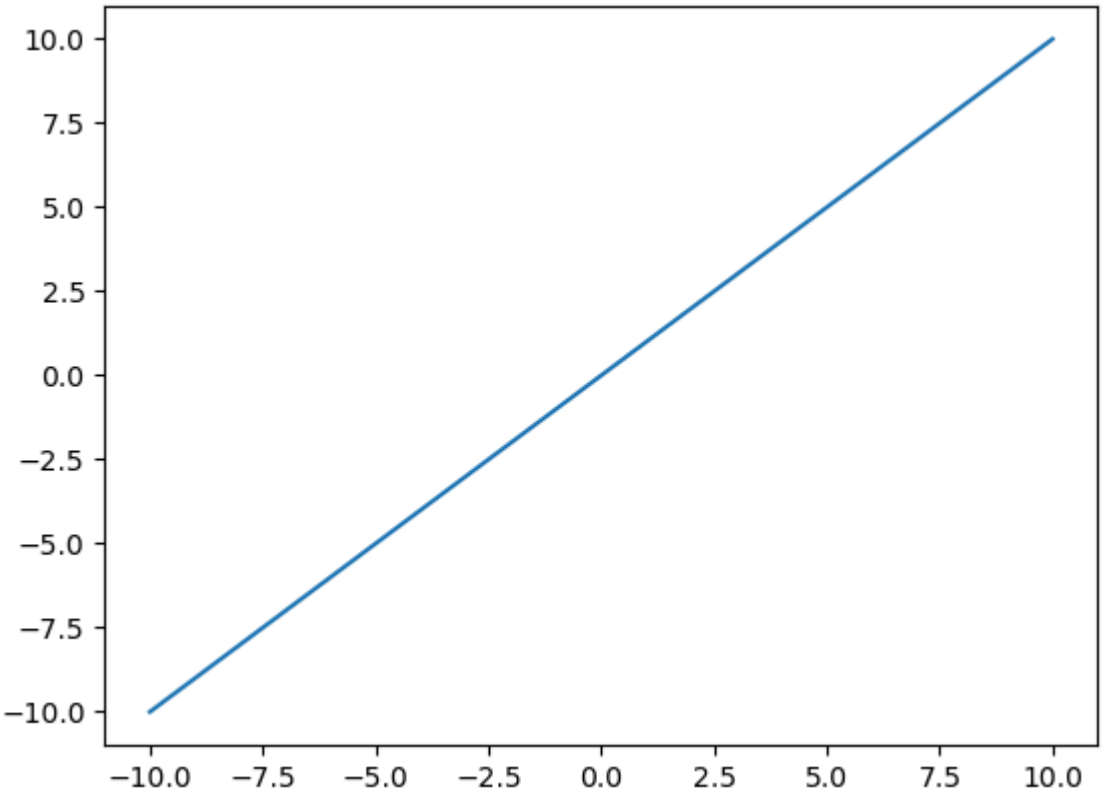
In [721…
```
y = x
```

In [723…
```
y
```

Out[723…
```
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 [727…
```
import matplotlib.pyplot as plt           # matplotlib is used for the visualiza
plt.plot(x,y)                             # plot() function is used to plot
```

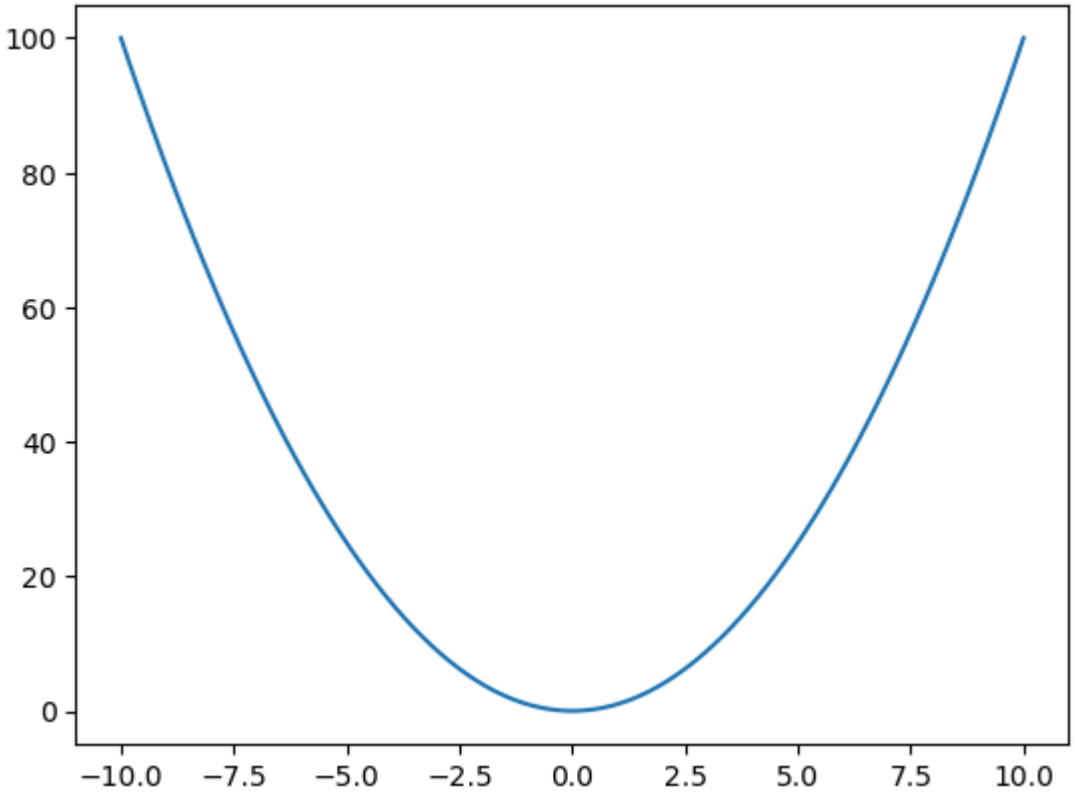
Out[727…
```
[<matplotlib.lines.Line2D at 0x1fb9b339dc0>]
```



X^2 graph

In [730…
```
plt.plot(x,x**2)
```

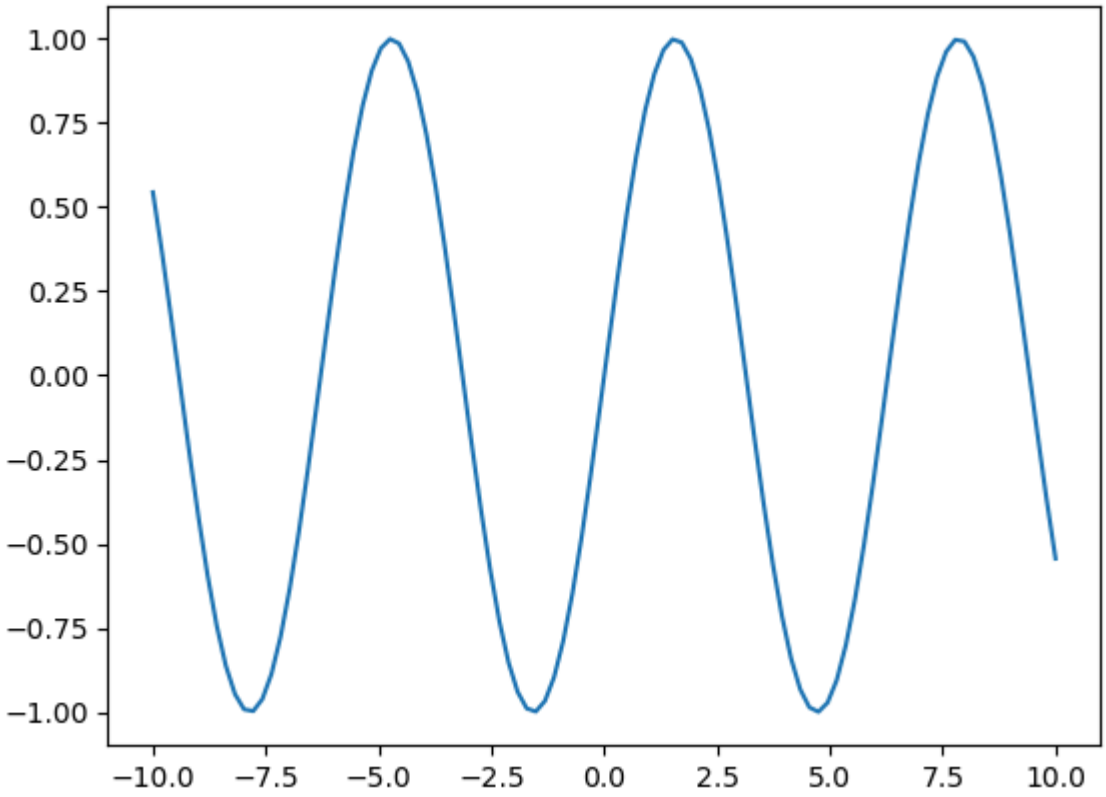
Out[730…
```
[<matplotlib.lines.Line2D at 0x1fb9b301e20>]
```



Sin x graph

In [733…
```
plt.plot(x,np.sin(x))
```

Out[733…
```
[<matplotlib.lines.Line2D at 0x1fb9b560110>]
```

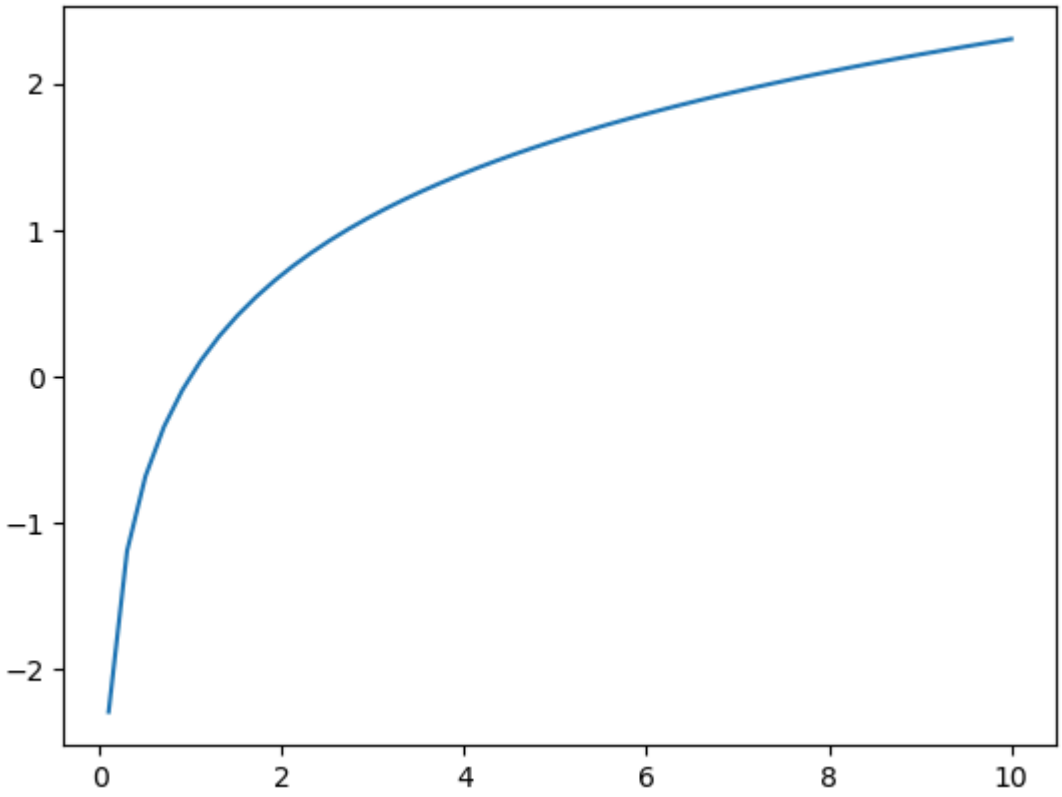


log(x) graph

In [736…
```
plt.plot(x,np.log(x))
```

```
C:\Users\galir\AppData\Local\Temp\ipykernel_22520\2360429912.py:1: RuntimeWarnin
g: invalid value encountered in log
  plt.plot(x,np.log(x))
```

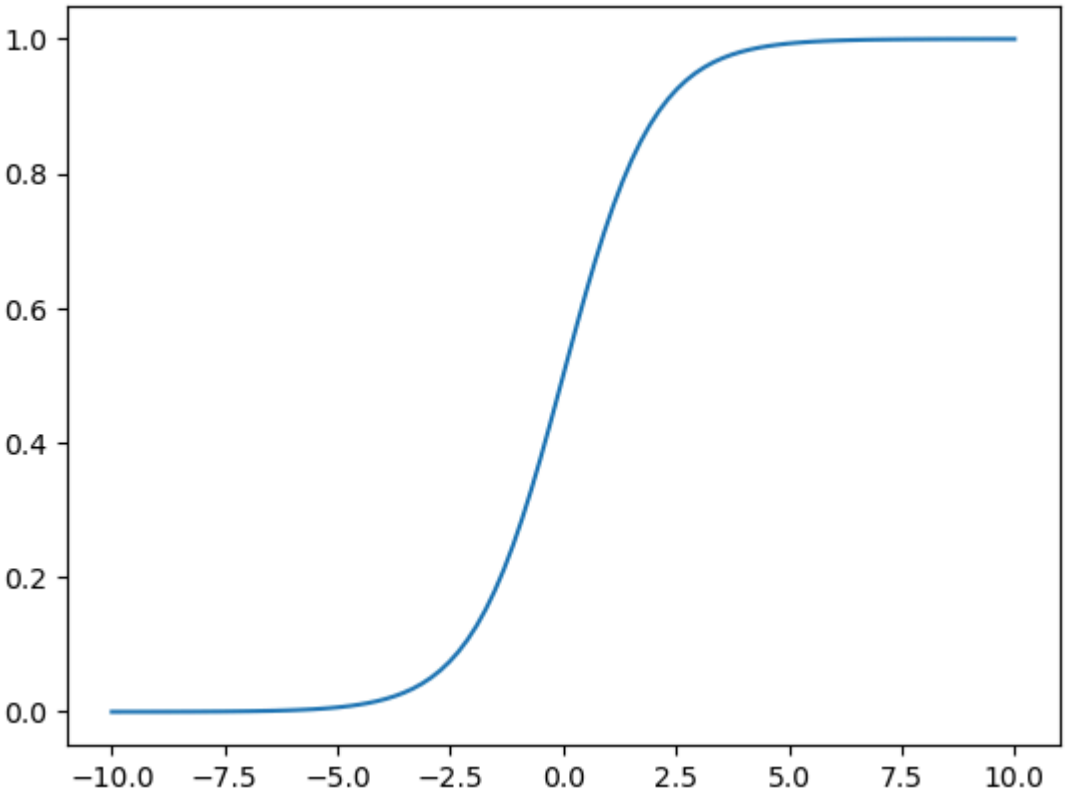
Out[736…
```
[<matplotlib.lines.Line2D at 0x1fb9b59c050>]
```



sigmoid

In [739…
```
plt.plot(x,1/(1+np.exp(-x)))
```

Out[739…
```
[<matplotlib.lines.Line2D at 0x1fb9b579e20>]
```



Meshgrid

In [742…
```
# it creates coordinate matrices from coordinate vectors
```

In [744…
```
x= np.linspace(0,10,100)
y= np.linspace(0,10,100)
print(x)
print(y)
```

```
[ 0.          0.1010101   0.2020202   0.3030303   0.4040404   0.50505051
  0.60606061  0.70707071  0.80808081  0.90909091  1.01010101  1.11111111
  1.21212121  1.31313131  1.41414141  1.51515152  1.61616162  1.71717172
  1.81818182  1.91919192  2.02020202  2.12121212  2.22222222  2.32323232
  2.42424242  2.52525253  2.62626263  2.72727273  2.82828283  2.92929293
  3.03030303  3.13131313  3.23232323  3.33333333  3.43434343  3.53535354
  3.63636364  3.73737374  3.83838384  3.93939394  4.04040404  4.14141414
  4.24242424  4.34343434  4.44444444  4.54545455  4.64646465  4.74747475
  4.84848485  4.94949495  5.05050505  5.15151515  5.25252525  5.35353535
  5.45454545  5.55555556  5.65656566  5.75757576  5.85858586  5.95959596
  6.06060606  6.16161616  6.26262626  6.36363636  6.46464646  6.56565657
  6.66666667  6.76767677  6.86868687  6.96969697  7.07070707  7.17171717
  7.27272727  7.37373737  7.47474747  7.57575758  7.67676768  7.77777778
  7.87878788  7.97979798  8.08080808  8.18181818  8.28282828  8.38383838
  8.48484848  8.58585859  8.68686869  8.78787879  8.88888889  8.98989899
  9.09090909  9.19191919  9.29292929  9.39393939  9.49494949  9.5959596
  9.6969697   9.7979798   9.8989899  10.        ]
[ 0.          0.1010101   0.2020202   0.3030303   0.4040404   0.50505051
  0.60606061  0.70707071  0.80808081  0.90909091  1.01010101  1.11111111
  1.21212121  1.31313131  1.41414141  1.51515152  1.61616162  1.71717172
  1.81818182  1.91919192  2.02020202  2.12121212  2.22222222  2.32323232
  2.42424242  2.52525253  2.62626263  2.72727273  2.82828283  2.92929293
  3.03030303  3.13131313  3.23232323  3.33333333  3.43434343  3.53535354
  3.63636364  3.73737374  3.83838384  3.93939394  4.04040404  4.14141414
  4.24242424  4.34343434  4.44444444  4.54545455  4.64646465  4.74747475
  4.84848485  4.94949495  5.05050505  5.15151515  5.25252525  5.35353535
  5.45454545  5.55555556  5.65656566  5.75757576  5.85858586  5.95959596
  6.06060606  6.16161616  6.26262626  6.36363636  6.46464646  6.56565657
  6.66666667  6.76767677  6.86868687  6.96969697  7.07070707  7.17171717
  7.27272727  7.37373737  7.47474747  7.57575758  7.67676768  7.77777778
  7.87878788  7.97979798  8.08080808  8.18181818  8.28282828  8.38383838
  8.48484848  8.58585859  8.68686869  8.78787879  8.88888889  8.98989899
  9.09090909  9.19191919  9.29292929  9.39393939  9.49494949  9.5959596
  9.6969697   9.7979798   9.8989899  10.        ]
```

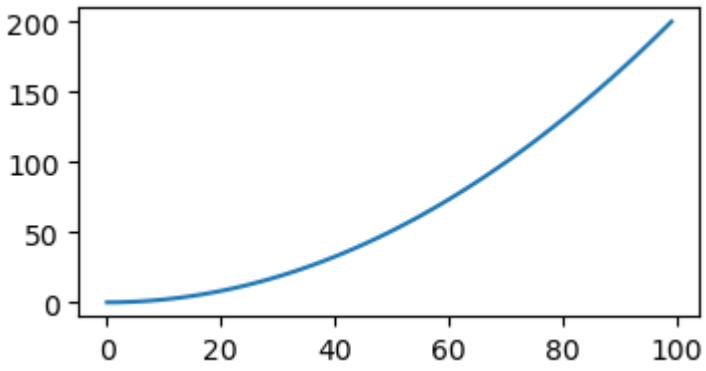
In [746…
```
f = x**2 + y**2          # 2D function
f
```

Out[746…

```
array([0.00000000e+00, 2.04060810e-02, 8.16243240e-02, 1.83654729e-01,
       3.26497296e-01, 5.10152025e-01, 7.34618916e-01, 9.99897970e-01,
       1.30598918e+00, 1.65289256e+00, 2.04060810e+00, 2.46913580e+00,
       2.93847567e+00, 3.44862769e+00, 3.99959188e+00, 4.59136823e+00,
       5.22395674e+00, 5.89735741e+00, 6.61157025e+00, 7.36659525e+00,
       8.16243240e+00, 8.99908173e+00, 9.87654321e+00, 1.07948169e+01,
       1.17539027e+01, 1.27538006e+01, 1.37945108e+01, 1.48760331e+01,
       1.59983675e+01, 1.71615141e+01, 1.83654729e+01, 1.96102439e+01,
       2.08958270e+01, 2.22222222e+01, 2.35894297e+01, 2.49974492e+01,
       2.64462810e+01, 2.79359249e+01, 2.94663810e+01, 3.10376492e+01,
       3.26497296e+01, 3.43026222e+01, 3.59963269e+01, 3.77308438e+01,
       3.95061728e+01, 4.13223140e+01, 4.31792674e+01, 4.50770330e+01,
       4.70156107e+01, 4.89950005e+01, 5.10152025e+01, 5.30762167e+01,
       5.51780431e+01, 5.73206816e+01, 5.95041322e+01, 6.17283951e+01,
       6.39934701e+01, 6.62993572e+01, 6.86460565e+01, 7.10335680e+01,
       7.34618916e+01, 7.59310274e+01, 7.84409754e+01, 8.09917355e+01,
       8.35833078e+01, 8.62156923e+01, 8.88888889e+01, 9.16028977e+01,
       9.43577186e+01, 9.71533517e+01, 9.99897970e+01, 1.02867054e+02,
       1.05785124e+02, 1.08744006e+02, 1.11743700e+02, 1.14784206e+02,
       1.17865524e+02, 1.20987654e+02, 1.24150597e+02, 1.27354352e+02,
       1.30598918e+02, 1.33884298e+02, 1.37210489e+02, 1.40577492e+02,
       1.43985308e+02, 1.47433935e+02, 1.50923375e+02, 1.54453627e+02,
       1.58024691e+02, 1.61636568e+02, 1.65289256e+02, 1.68982757e+02,
       1.72717070e+02, 1.76492195e+02, 1.80308132e+02, 1.84164881e+02,
       1.88062443e+02, 1.92000816e+02, 1.95980002e+02, 2.00000000e+02])
```

In [748…

```
plt.figure(figsize=(4,2))                    # createsa graph of size 2,4
plt.plot(f)                                  #plots a graph with coordinates of f
plt.show()                                   #shows the plotted graph
```



In [750…

```
# 1D function
```

In [752…

```
x = np.arange(3)
y = np.arange(3)
print(x)
print(y)
```

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

In [754…

```
# Generating a meshgrid
```

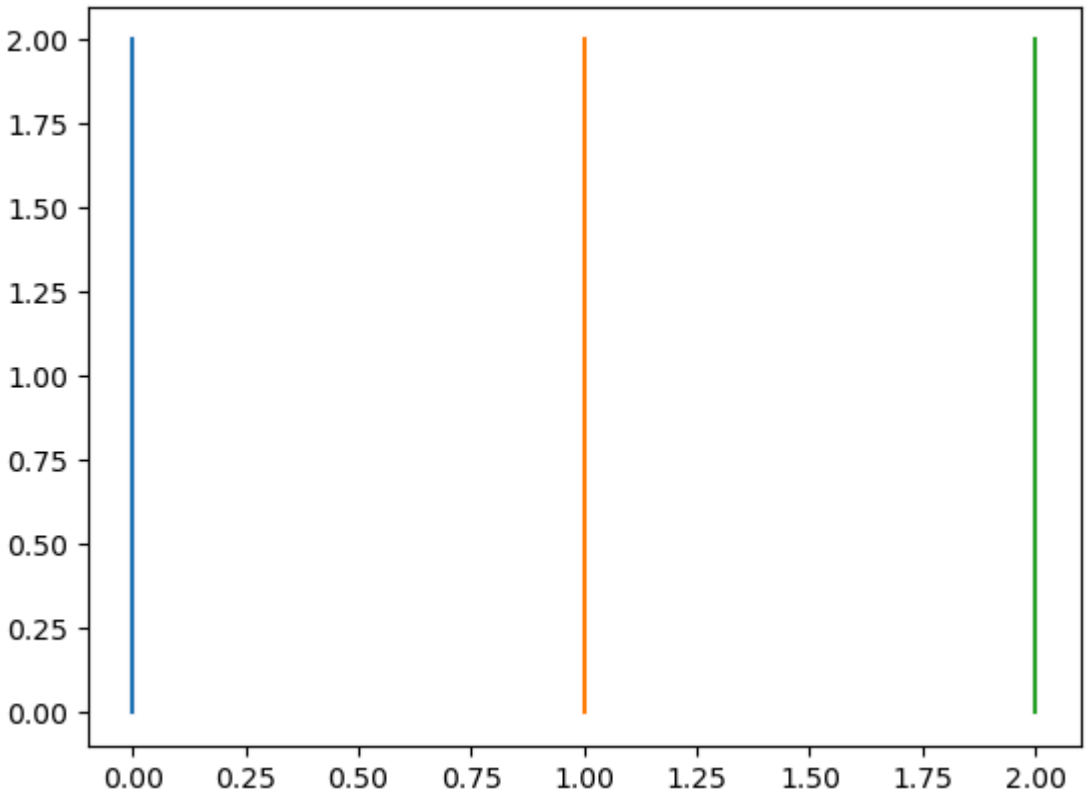
In [756…

```
xv , yv =  np.meshgrid(x,y)
print(xv)
print(yv)
```

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

In [758…

```
plt.plot(xv,yv)
plt.show()
```



Generate functions f(x, y) = e−(x2+y2) for −2 ≤ x ≤ 2 and −1 ≤ y ≤1

In [766…

```
x = np.linspace(-2,2,100)
y = np.linspace(-1,1,100)
print(x)
print(y)
```

```
[-2.         -1.95959596 -1.91919192 -1.87878788 -1.83838384 -1.7979798
 -1.75757576 -1.71717172 -1.67676768 -1.63636364 -1.5959596  -1.55555556
 -1.51515152 -1.47474747 -1.43434343 -1.39393939 -1.35353535 -1.31313131
 -1.27272727 -1.23232323 -1.19191919 -1.15151515 -1.11111111 -1.07070707
 -1.03030303 -0.98989899 -0.94949495 -0.90909091 -0.86868687 -0.82828283
 -0.78787879 -0.74747475 -0.70707071 -0.66666667 -0.62626263 -0.58585859
 -0.54545455 -0.50505051 -0.46464646 -0.42424242 -0.38383838 -0.34343434
 -0.3030303  -0.26262626 -0.22222222 -0.18181818 -0.14141414 -0.1010101
 -0.06060606 -0.02020202  0.02020202  0.06060606  0.1010101   0.14141414
  0.18181818  0.22222222  0.26262626  0.3030303   0.34343434  0.38383838
  0.42424242  0.46464646  0.50505051  0.54545455  0.58585859  0.62626263
  0.66666667  0.70707071  0.74747475  0.78787879  0.82828283  0.86868687
  0.90909091  0.94949495  0.98989899  1.03030303  1.07070707  1.11111111
  1.15151515  1.19191919  1.23232323  1.27272727  1.31313131  1.35353535
  1.39393939  1.43434343  1.47474747  1.51515152  1.55555556  1.5959596
  1.63636364  1.67676768  1.71717172  1.75757576  1.7979798   1.83838384
  1.87878788  1.91919192  1.95959596  2.        ]
[-1.         -0.97979798 -0.95959596 -0.93939394 -0.91919192 -0.8989899
 -0.87878788 -0.85858586 -0.83838384 -0.81818182 -0.7979798  -0.77777778
 -0.75757576 -0.73737374 -0.71717172 -0.6969697  -0.67676768 -0.65656566
 -0.63636364 -0.61616162 -0.5959596  -0.57575758 -0.55555556 -0.53535354
 -0.51515152 -0.49494949 -0.47474747 -0.45454545 -0.43434343 -0.41414141
 -0.39393939 -0.37373737 -0.35353535 -0.33333333 -0.31313131 -0.29292929
 -0.27272727 -0.25252525 -0.23232323 -0.21212121 -0.19191919 -0.17171717
 -0.15151515 -0.13131313 -0.11111111 -0.09090909 -0.07070707 -0.05050505
 -0.03030303 -0.01010101  0.01010101  0.03030303  0.05050505  0.07070707
  0.09090909  0.11111111  0.13131313  0.15151515  0.17171717  0.19191919
  0.21212121  0.23232323  0.25252525  0.27272727  0.29292929  0.31313131
  0.33333333  0.35353535  0.37373737  0.39393939  0.41414141  0.43434343
  0.45454545  0.47474747  0.49494949  0.51515152  0.53535354  0.55555556
  0.57575758  0.5959596   0.61616162  0.63636364  0.65656566  0.67676768
  0.6969697   0.71717172  0.73737374  0.75757576  0.77777778  0.7979798
  0.81818182  0.83838384  0.85858586  0.87878788  0.8989899   0.91919192
  0.93939394  0.95959596  0.97979798  1.        ]
```

In [768…

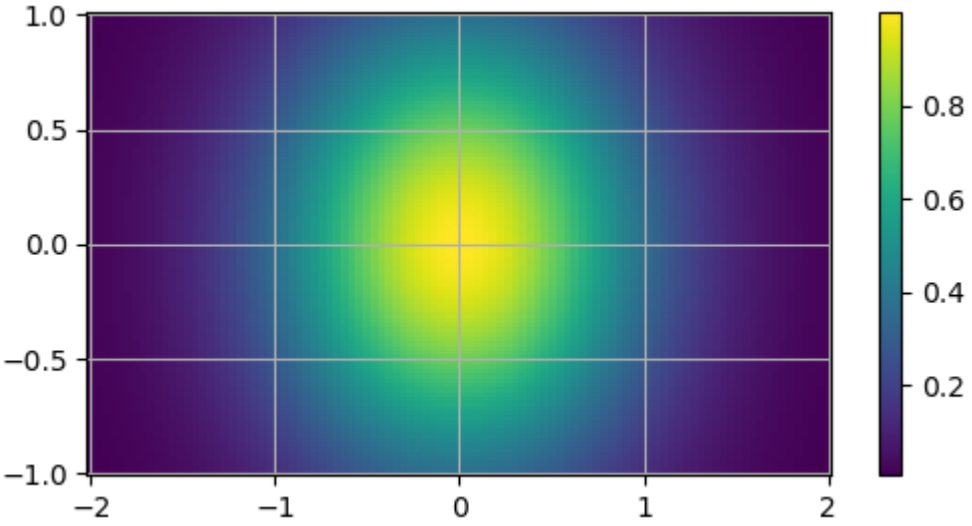
```
xv,yv=np.meshgrid(x,y)
```

In [772…

```
f=np.exp(-(xv**2+yv**2))
```

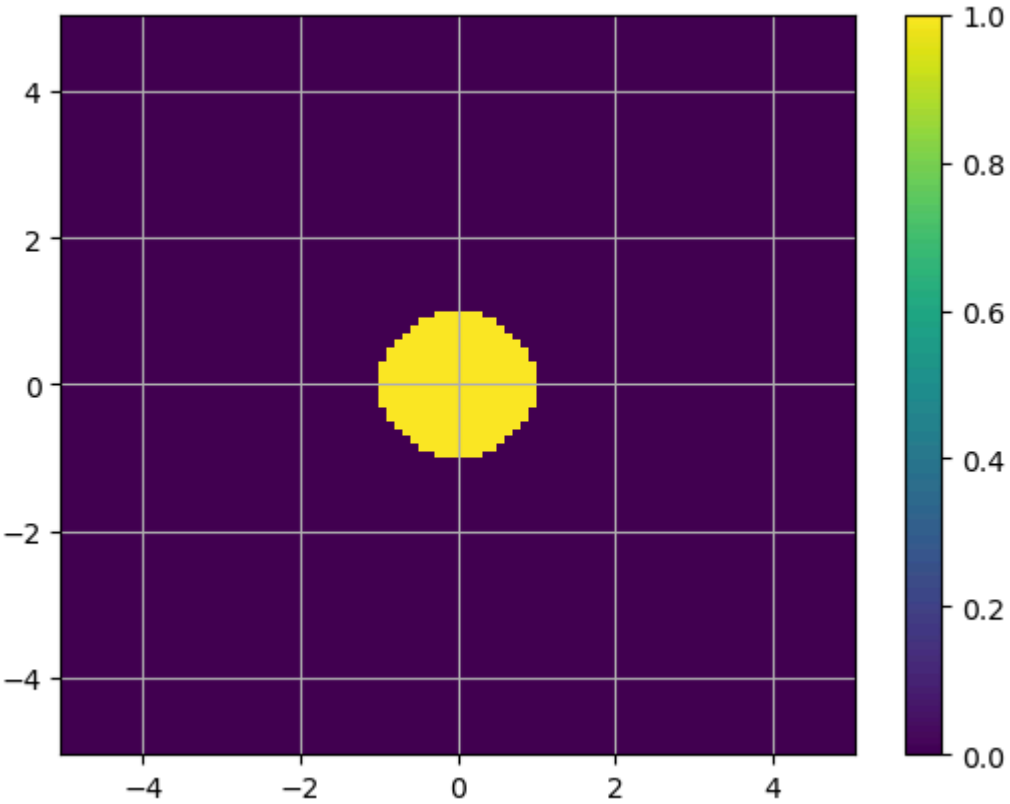
In [774…

```
plt.figure(figsize=(6,3))
plt.pcolormesh(xv,yv,f,shading='auto')
plt.colorbar()
plt.grid()
plt.show()
```



In [776…

```
def f(x,y):
    return np.where ((x**2 +y**2<1),1.0,0.0)
x = np.linspace(-5,5,100)
y = np.linspace(-5,5,100)
xv,yv=np.meshgrid(x,y)
rectangular_mask = f(xv,yv)
plt.pcolormesh(xv,yv,rectangular_mask,shading='auto')
plt.colorbar()
plt.grid()
plt.show()
```



In [778…

```
x = np.linspace(-4,4,9)
y = np.linspace(-5,5,11)
print(x)
print(y)
```

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

In [780…
```
xv,yv=np.meshgrid(x,y)
```

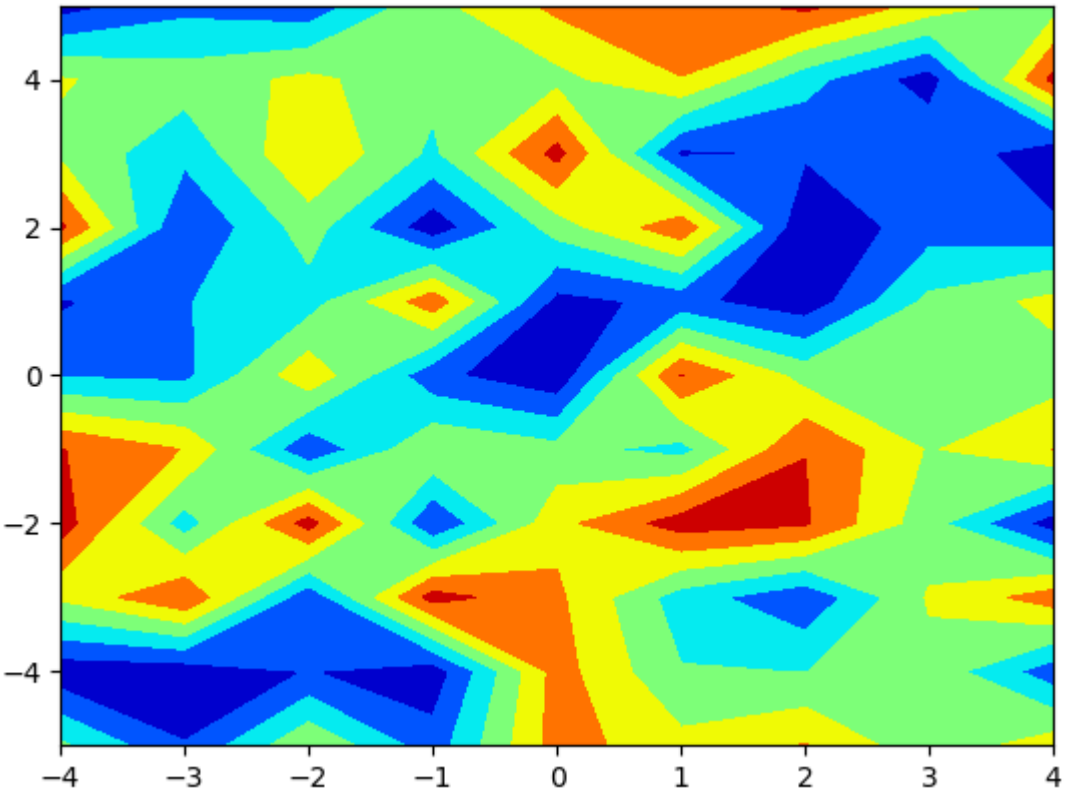
In [784…
```
random_data=np.random.random((11,9))
random_data
```

Out[784…
```
array([[0.47745078, 0.15643613, 0.58353356, 0.21474626, 0.81940755,
        0.64554848, 0.75769079, 0.49600639, 0.68256468],
       [0.04718683, 0.05196495, 0.15497058, 0.05037416, 0.81189646,
        0.459223  , 0.44767824, 0.54813377, 0.20735038],
       [0.62709634, 0.88954863, 0.17812833, 0.96548864, 0.77936712,
        0.37271888, 0.1750957 , 0.61725035, 0.82989544],
       [0.97775655, 0.37027115, 0.962276  , 0.14621005, 0.70205564,
        0.97169338, 0.91613868, 0.50570662, 0.06302976],
       [0.90699628, 0.74298478, 0.16724894, 0.58845459, 0.49748489,
        0.39496761, 0.89760088, 0.58483881, 0.75132257],
       [0.28795848, 0.26079303, 0.70761318, 0.19032563, 0.02269562,
        0.90533737, 0.53705798, 0.54485502, 0.52939711],
       [0.12311621, 0.29394789, 0.37870173, 0.84441587, 0.09659903,
        0.19339128, 0.04650641, 0.48402638, 0.64540837],
       [0.9292398 , 0.14545322, 0.52154221, 0.04559209, 0.5193026 ,
        0.85902592, 0.01050747, 0.23097997, 0.1794897 ],
       [0.56709205, 0.33856605, 0.74211678, 0.41714395, 0.95204131,
        0.11891062, 0.16755069, 0.29256247, 0.01275931],
       [0.61165605, 0.52179517, 0.63251109, 0.49802156, 0.55223575,
        0.74711638, 0.35445739, 0.0700027 , 0.9421545 ],
       [0.07177138, 0.24996447, 0.19399983, 0.55703591, 0.78652567,
        0.80894471, 0.94715437, 0.68662979, 0.52539129]])
```

In [786…
```
plt.contourf(xv,yv,random_data,cmap='jet')
plt.colobar()
plt.show()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[786], line 2
      1 plt.contourf(xv,yv,random_data,cmap='jet')
----> 2 plt.colobar()
      3 plt.show()

AttributeError: module 'matplotlib.pyplot' has no attribute 'colobar'
```



np.sort()

In [789…
```
# sort() method returns a sorted copy of an array
```

In [791…
```
import numpy as np
```

In [793…
```
a = np.random.randint(1,100,15)          # 1D array
```

In [795…
```
a
```

Out[795…
```
array([23, 21, 39, 71, 93, 28, 70, 27, 76, 18, 48, 44, 28, 71,  8])
```

In [797…
```
b = np.random.randint(1,100,24).reshape(6,4)     # 2D array
b
```

Out[797…
```
array([[74, 76, 60, 40],
       [37, 95, 56, 73],
       [59, 29, 48, 72],
       [80, 79, 86, 32],
       [59, 67, 25, 16],
       [50, 30, 88, 47]])
```

In [799…
```
np.sort(a)          # default returns the array elements array in ascending orde
```

Out[799…
```
array([ 8, 18, 21, 23, 27, 28, 28, 39, 44, 48, 70, 71, 71, 76, 93])
```

In [801…
```
np.sort(a)[::-1]  # returns the array elements in descending order
```

Out[801…
```
array([93, 76, 71, 71, 70, 48, 44, 39, 28, 28, 27, 23, 21, 18,  8])
```

In [803…
```
np.sort(b)  # by default,row wise sorting
```

Out[803…
```
array([[40, 60, 74, 76],
       [37, 56, 73, 95],
       [29, 48, 59, 72],
       [32, 79, 80, 86],
       [16, 25, 59, 67],
       [30, 47, 50, 88]])
```

In [805…
```
np.sort(b,axis=0)     # column wise sorting
```

Out[805…
```
array([[37, 29, 25, 16],
       [50, 30, 48, 32],
       [59, 67, 56, 40],
       [59, 76, 60, 47],
       [74, 79, 86, 72],
       [80, 95, 88, 73]])
```

np.append

In [808…
```
# appends the value along the mentioned axis
```

In [810…
```
a
```

Out[810…
```
array([23, 21, 39, 71, 93, 28, 70, 27, 76, 18, 48, 44, 28, 71,  8])
```

In [812…
```
b
```

Out[812…
```
array([[74, 76, 60, 40],
       [37, 95, 56, 73],
       [59, 29, 48, 72],
       [80, 79, 86, 32],
       [59, 67, 25, 16],
       [50, 30, 88, 47]])
```

In [816…
```
np.append(a,200)     # appends the value at the end of the numpy array
```

Out[816…
```
array([ 23,  21,  39,  71,  93,  28,  70,  27,  76,  18,  48,  44,  28,
        71,   8, 200])
```

In [818…
```
np.append(b,np.ones((b.shape[0],1)))
```

Out[818…
```
array([74., 76., 60., 40., 37., 95., 56., 73., 59., 29., 48., 72., 80.,
       79., 86., 32., 59., 67., 25., 16., 50., 30., 88., 47.,  1.,  1.,
        1.,  1.,  1.,  1.])
```

In [820…
```
# adding random numbers in new column
```

In [828…
```
np.append(b,np.random.random((b.shape[0],1)),axis=1)
```

Out[828…
```
array([[74.        , 76.        , 60.        , 40.        ,  0.95033781],
       [37.        , 95.        , 56.        , 73.        ,  0.47840781],
       [59.        , 29.        , 48.        , 72.        ,  0.92662795],
       [80.        , 79.        , 86.        , 32.        ,  0.18335831],
       [59.        , 67.        , 25.        , 16.        ,  0.94139235],
       [50.        , 30.        , 88.        , 47.        ,  0.97774488]])
```

np.concatenate()

In [831…
```
# concatenate a sequence of arrays along the existing axis
```

In [833…
```
c=np.arange(6).reshape(2,3)
d=np.arange(6,12).reshape(2,3)
```

In [835…
```
c
```

Out[835…
```
array([[0, 1, 2],
       [3, 4, 5]])
```

In [837…
```
d
```

Out[837…
```
array([[ 6,  7,  8],
       [ 9, 10, 11]])
```

In [841…
```
np.concatenate((c,d),axis=1)    # concatenates along the column wise
```

Out[841…
```
array([[ 0,  1,  2,  6,  7,  8],
       [ 3,  4,  5,  9, 10, 11]])
```

In [847…
```
np.concatenate((c,d))           # defaultly concatenate alon the row wise
```

Out[847…
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

np.unique

In [850…
```
# unique() method retuens the unique value from an array
```

In [854…
```
e=np.array([1,1,1,2,2,2,2,3,3,3,4,4,4])
e
```

Out[854…
```
array([1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4, 4])
```

In [856…
```
np.unique(e)
```

Out[856…
```
array([1, 2, 3, 4])
```

np.expand_dims

In [859…
```
a
```

Out[859…
```
array([23, 21, 39, 71, 93, 28, 70, 27, 76, 18, 48, 44, 28, 71,  8])
```

In [861…
```
a.shape
```

Out[861…
```
(15,)
```

In [865…
```
# convert into 2D arrays
```

In [869…
```
np.expand_dims(a,axis=0)        # axis paramater is mandatory
```

Out[869…
```
array([[23, 21, 39, 71, 93, 28, 70, 27, 76, 18, 48, 44, 28, 71,  8]])
```

In [871…
```
np.expand_dims(a,axis=0).shape            # 2D array
```

Out[871…
```
(1, 15)
```

In [873…
```
np.expand_dims(a,axis=1)
```

Out[873…
```
array([[23],
       [21],
       [39],
       [71],
       [93],
       [28],
       [70],
       [27],
       [76],
       [18],
       [48],
       [44],
       [28],
       [71],
       [ 8]])
```

In [875…
```
np.expand_dims(a,axis=1).shape
```

Out[875…
```
(15, 1)
```

np.where()

In [880…
```
a
```

Out[880…
```
array([23, 21, 39, 71, 93, 28, 70, 27, 76, 18, 48, 44, 28, 71,  8])
```

In [882…
```
np.where(a>50)
```

Out[882…
```
(array([ 3,  4,  6,  8, 13], dtype=int64),)
```

In [ ]:
```
# np.where(condition,true,false)
```

In [884…
```
np.where(a>50,0,a)
```

Out[884…
```
array([23, 21, 39,  0,  0, 28,  0, 27,  0, 18, 48, 44, 28,  0,  8])
```

In [886…
```
np.where(a%2==0,0,a)
```

Out[886…
```
array([23, 21, 39, 71, 93,  0,  0, 27,  0,  0,  0,  0,  0, 71,  0])
```

np.argmax()

In [ ]:
```
# argmax function returns the indices of the max element of the array in a parti
```

In [889…
```
a
```

Out[889…
```
array([23, 21, 39, 71, 93, 28, 70, 27, 76, 18, 48, 44, 28, 71,  8])
```

In [891…
```
np.argmax(a)
```

Out[891…
```
4
```

In [893…
```
b
```

Out[893…
```
array([[74, 76, 60, 40],
       [37, 95, 56, 73],
       [59, 29, 48, 72],
       [80, 79, 86, 32],
       [59, 67, 25, 16],
       [50, 30, 88, 47]])
```

In [895…
```
np.argmax(b)
```

Out[895…
```
5
```

In [897…
```
np.argmax(b,axis=1)      # row wise biggest number
```

Out[897…
```
array([1, 1, 3, 2, 1, 2], dtype=int64)
```

In [899…
```
np.argmax(b,axis=0)
```

Out[899…
```
array([3, 1, 5, 1], dtype=int64)
```

In [901…
```
a
```

Out[901…
```
array([23, 21, 39, 71, 93, 28, 70, 27, 76, 18, 48, 44, 28, 71,  8])
```

In [903…
```
np.argmax(a)
```

Out[903…
```
4
```

On statistics

np.cumsum

In [907…
```
a
```

Out[907…
```
array([23, 21, 39, 71, 93, 28, 70, 27, 76, 18, 48, 44, 28, 71,  8])
```

In [909…
```
b
```

Out[909…
```
array([[74, 76, 60, 40],
       [37, 95, 56, 73],
       [59, 29, 48, 72],
       [80, 79, 86, 32],
       [59, 67, 25, 16],
       [50, 30, 88, 47]])
```

In [911…
```
np.cumsum(a)
```

Out[911…
```
array([ 23,  44,  83, 154, 247, 275, 345, 372, 448, 466, 514, 558, 586,
       657, 665])
```

In [913…
```
np.cumsum(b)
```

Out[913…
```
array([  74,  150,  210,  250,  287,  382,  438,  511,  570,  599,  647,
        719,  799,  878,  964,  996, 1055, 1122, 1147, 1163, 1213, 1243,
       1331, 1378])
```

In [915…
```
np.cumsum(b,axis=1)
```

Out[915…
```
array([[ 74, 150, 210, 250],
       [ 37, 132, 188, 261],
       [ 59,  88, 136, 208],
       [ 80, 159, 245, 277],
       [ 59, 126, 151, 167],
       [ 50,  80, 168, 215]])
```

In [917…
```
np.cumsum(b,axis=0)
```

Out[917…
```
array([[ 74,  76,  60,  40],
       [111, 171, 116, 113],
       [170, 200, 164, 185],
       [250, 279, 250, 217],
       [309, 346, 275, 233],
       [359, 376, 363, 280]])
```

np.cumprod()

In [920…
```
a
```

Out[920…
```
array([23, 21, 39, 71, 93, 28, 70, 27, 76, 18, 48, 44, 28, 71,  8])
```

In [922…
```
np.cumprod(a)
```

Out[922…
```
array([         23,         483,       18837,     1337427,   124380711,
       -812307388, -1026942312, -1957638648,  1543318112,  2009922240,
       1986987008,  1528082432,  -163364864,  1285996544,  1698037760])
```

np.percentile()

In [925…
```
# used to compute the nth percentile of the  given data
```

In [927…
```
a
```

Out[927…
```
array([23, 21, 39, 71, 93, 28, 70, 27, 76, 18, 48, 44, 28, 71,  8])
```

In [931…
```
np.percentile(a,100)           # max
```

Out[931…
```
93.0
```

In [933…
```
np.percentile(a,0)             # min
```

Out[933…
```
8.0
```

In [937…
```
np.percentile(a,50)             #median
```

Out[937…
```
39.0
```

np.histogram()

In [940…
```
# it represents the frequency of the data distribution in graphical form
```

In [942…
```
a
```

Out[942…
```
array([23, 21, 39, 71, 93, 28, 70, 27, 76, 18, 48, 44, 28, 71,  8])
```

In [944…
```
np.histogram(a,bins=[10,20,30,40,50,60,70,80,90,100])
```

Out[944…
```
(array([1, 5, 1, 2, 0, 0, 4, 0, 1], dtype=int64),
 array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100]))
```

In [946…
```
np.histogram(a,bins=[0,50,100])
```

Out[946…
```
(array([10,  5], dtype=int64), array([  0,  50, 100]))
```

np.corrcoef()

In [949…
```
a
```

Out[949…
```
array([23, 21, 39, 71, 93, 28, 70, 27, 76, 18, 48, 44, 28, 71,  8])
```

In [951…
```
slary = np.array([20000,40000,25000,35000,60000])
slary
```

Out[951…
```
array([20000, 40000, 25000, 35000, 60000])
```

In [955…
```
exp = np.array([1,3,2,4,2])
exp
```

Out[955…
```
array([1, 3, 2, 4, 2])
```

In [957…
```
np.corrcoef(slary,exp)
```

Out[957…
```
array([[1.        , 0.25344572],
       [0.25344572, 1.        ]])
```

utility functions

np.isin()

In [963…
```
a
```

Out[963…
```
array([23, 21, 39, 71, 93, 28, 70, 27, 76, 18, 48, 44, 28, 71,  8])
```

In [965…
```
items = [10,20,30,40,50,60,70,80,90,100]
np.isin(a,items)
```

Out[965…
```
array([False, False, False, False, False, False,  True, False, False,
       False, False, False, False, False, False])
```

In [967…
```
a[np.isin(a,items)]
```

Out[967…
```
array([70])
```

np.flip

In [970…
```
a
```

Out[970…
```
array([23, 21, 39, 71, 93, 28, 70, 27, 76, 18, 48, 44, 28, 71,  8])
```

In [972…
```
np.flip(a)     # reverse the lements
```

Out[972…
```
array([ 8, 71, 28, 44, 48, 18, 76, 27, 70, 28, 93, 71, 39, 21, 23])
```

In [974…
```
b
```

Out[974…
```
array([[74, 76, 60, 40],
       [37, 95, 56, 73],
       [59, 29, 48, 72],
       [80, 79, 86, 32],
       [59, 67, 25, 16],
       [50, 30, 88, 47]])
```

In [976…
```
np.flip(b)          # first row as last row
```

Out[976…
```
array([[47, 88, 30, 50],
       [16, 25, 67, 59],
       [32, 86, 79, 80],
       [72, 48, 29, 59],
       [73, 56, 95, 37],
       [40, 60, 76, 74]])
```

In [978…
```
np.flip(b,axis=1)          # row wise flip
```

Out[978…
```
array([[40, 60, 76, 74],
       [73, 56, 95, 37],
       [72, 48, 29, 59],
       [32, 86, 79, 80],
       [16, 25, 67, 59],
       [47, 88, 30, 50]])
```

In [980…
```
np.flip(b,axis=0)        # column wise flip
```

Out[980…
```
array([[50, 30, 88, 47],
       [59, 67, 25, 16],
       [80, 79, 86, 32],
       [59, 29, 48, 72],
       [37, 95, 56, 73],
       [74, 76, 60, 40]])
```

np.put()

In [984…
```
a
```

Out[984…
```
array([23, 21, 39, 71, 93, 28, 70, 27, 76, 18, 48, 44, 28, 71,  8])
```

In [986…
```
b
```

Out[986…
```
array([[74, 76, 60, 40],
       [37, 95, 56, 73],
       [59, 29, 48, 72],
       [80, 79, 86, 32],
       [59, 67, 25, 16],
       [50, 30, 88, 47]])
```

In [100…
```
np.put(a,[0,1],[110,530])                # permanent changes
```

In [100…
```
a
```

Out[100…
```
array([110, 530,  39,  71,  93,  28,  70,  27,  76,  18,  48,  44,  28,
        71,   8])
```

np.delete()

In [100…
```
# returns a new array with the deletion of sub-arrays along with mentioned axis
```

In [100…
```
a
```

Out[100…
```
array([110, 530,  39,  71,  93,  28,  70,  27,  76,  18,  48,  44,  28,
        71,   8])
```

In [101…
```
np.delete(a,0)
```

Out[101…
```
array([530,  39,  71,  93,  28,  70,  27,  76,  18,  48,  44,  28,  71,
         8])
```

In [101…
```
a
```

Out[101…
```
array([110, 530,  39,  71,  93,  28,  70,  27,  76,  18,  48,  44,  28,
        71,   8])
```

In [101…
```
np.delete(a,[0,2,4])
```

Out[101…
```
array([530,  71,  28,  70,  27,  76,  18,  48,  44,  28,  71,   8])
```

In [101…
```
a
```

Out[101…
```
array([110, 530,  39,  71,  93,  28,  70,  27,  76,  18,  48,  44,  28,
        71,   8])
```

set functions

In [102…
```
# np.union(m,n)
```

In [102…
```
m = np.array([1,2,3,4,5])
n = np.array([3,4,5,6,7])
np.union1d(m,n)
```

Out[102…
```
array([1, 2, 3, 4, 5, 6, 7])
```

In [102…
```
# intersection
```

In [102…
```
np.intersect1d(m,n)
```

Out[102…
```
array([3, 4, 5])
```

In [103…
```
# set difference
```

In [103…
```
np.setdiff1d(m,n)
```

Out[103…
```
array([1, 2])
```

In [103…
```
# set Xor
```

In [103…
```
np.setxor1d(m,n)
```

Out[103…
```
array([1, 2, 6, 7])
```

In [103…
```
np.in1d(m,1)
```

Out[103…
```
array([ True, False, False, False, False])
```

In [104…
```
m[np.in1d(m,1)]
```

Out[104…
```
array([1])
```

np.clip()

In [104…
```
a
```

Out[104…
```
array([110, 530,  39,  71,  93,  28,  70,  27,  76,  18,  48,  44,  28,
        71,   8])
```

In [104…
```
b
```

Out[104…
```
array([[74, 76, 60, 40],
       [37, 95, 56, 73],
       [59, 29, 48, 72],
       [80, 79, 86, 32],
       [59, 67, 25, 16],
       [50, 30, 88, 47]])
```

In [104…
```
np.clip(a,a_min=15,a_max=50)
```

Out[104…
```
array([50, 50, 39, 50, 50, 28, 50, 27, 50, 18, 48, 44, 28, 50, 15])
```

np.swapaxes

In [105…
```
# interchange two axes of an array
```

In [105…
```
arr = np.array([[1,2,3],[4,5,6]])
swapped_arr = np.swapaxes(arr,0,1)
```

In [105…
```
arr
```

Out[105…
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [105…
```
swapped_arr
```

Out[105…
```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

In [ ]:
```

```