import numpy

In [1]:
```
import numpy as np
```

In [2]:
```
 np.array([2,4,56,422,32,1])
```

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

In [3]:
```
a = np.array([2,4,56,422,32,1])  #Vector
print(a)
```

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

In [4]:
```
# 2D Array ( Matrix)
new = np.array([[45,34,22,2],[24,55,3,22]])
print(new)
```

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

In [5]:
```
np.array ( [[2,3,33,4,45],[23,45,56,66,2],[357,523,32,24,2],[32,32,44,33,234]])
```

Out[5]:
```
array([[  2,   3,  33,   4,  45],
       [ 23,  45,  56,  66,   2],
       [357, 523,  32,  24,   2],
       [ 32,  32,  44,  33, 234]])
```

arange

In [6]:
```
np.arange(1,25)
```

Out[6]:
```
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 [7]:
```
np.arange(1,25,2)
```

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

reshape

In [8]:
```
np.arange(1,11).reshape(5,2)
```

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

In [9]:
```
 np.arange(1,11).reshape(2,5)
```

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

ones

In [10]:
```
b=np.ones(4)
```

In [11]:
```
b
```

Out[11]:
```
array([1., 1., 1., 1.])
```

In [12]:
```
b.size
```

Out[12]:
```
4
```

In [13]:
```
b=np.ones(4,dtype=int)
```

In [14]:
```
np.ones((4,5))
```

Out[14]:
```
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

In [15]:
```
np.ones((4,5),dtype=bool)
```

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

zeros

In [16]:
```
np.zeros(5,dtype=int)
```

Out[16]:
```
array([0, 0, 0, 0, 0])
```

In [17]:
```
np.zeros((5,5,4))
```

Out[17]:
```
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., 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.]]])
```

random

In [18]:
```
np.random.rand(4)
```

Out[18]:
```
array([0.87804086, 0.99916604, 0.38860794, 0.59721475])
```

In [19]:
```
np.random.rand(2,3)
```

Out[19]:
```
array([[0.42276475, 0.14080964, 0.37295102],
       [0.28763237, 0.22150795, 0.20335222]])
```

In [20]:
```
np.random.rand(2,3,5)
```

Out[20]:
```
array([[[0.96821501, 0.41750158, 0.19142128, 0.90072805, 0.59647857],
        [0.49240841, 0.1873685 , 0.40036253, 0.42272049, 0.17984834],
        [0.97331501, 0.3022594 , 0.545268  , 0.36597674, 0.9433809 ]],

       [[0.15518177, 0.21624952, 0.98999527, 0.85594183, 0.23451175],
        [0.75579163, 0.29216276, 0.13140756, 0.94525249, 0.68105041],
        [0.27966897, 0.34405118, 0.70586168, 0.72662898, 0.34039109]]])
```

In [21]:
```
np.random.randint(5)
```

Out[21]:
```
4
```

In [22]:
```
np.random.randint(1,3)
```

Out[22]:
```
1
```

In [23]:
```
np.random.randint(1,3,4)
```

Out[23]:
```
array([1, 2, 1, 2])
```

In [24]:
```
np.random.randint(1,3,(4,5))
```

Out[24]:
```
array([[2, 2, 1, 1, 2],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 2, 2],
       [1, 1, 1, 1, 1]])
```

In [25]:
```
np.random.randint(1,3,(4,5,6))
```

Out[25]:
```
array([[[1, 1, 2, 1, 2, 1],
        [1, 2, 1, 2, 1, 1],
        [2, 1, 1, 1, 1, 1],
        [2, 2, 1, 1, 1, 2],
        [2, 2, 1, 1, 1, 1]],

       [[1, 2, 1, 2, 1, 2],
        [1, 2, 2, 1, 2, 1],
        [2, 2, 1, 1, 1, 1],
        [2, 2, 1, 1, 1, 2],
        [1, 2, 1, 2, 1, 1]],

       [[1, 1, 2, 1, 1, 1],
        [2, 1, 1, 2, 2, 2],
        [1, 1, 2, 1, 1, 2],
        [2, 2, 2, 1, 1, 2],
        [2, 1, 2, 1, 1, 2]],

       [[2, 2, 1, 2, 1, 2],
        [1, 1, 2, 1, 2, 2],
        [2, 1, 2, 1, 2, 1],
        [2, 1, 1, 1, 2, 2],
        [2, 2, 1, 2, 1, 1]]])
```

linspace

In [26]:
```
np.linspace(2,9,4)
```

Out[26]:
```
array([2.        , 4.33333333, 6.66666667, 9.        ])
```

In [27]:
```
np.linspace(-2,12,6)
```

Out[27]:
```
array([-2. ,  0.8,  3.6,  6.4,  9.2, 12. ])
```

identity

In [28]:
```
np.identity(5,dtype=int)
```

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

In [29]:
```
a1 = np.arange(10) # 1D
a1
```

Out[29]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [30]:
```
a2 =np.arange(12, dtype =float).reshape(3,4) # Matrix
a2
```

Out[30]:
```
array([[ 0.,  1.,  2.,  3.],
       [ 4.,  5.,  6.,  7.],
       [ 8.,  9., 10., 11.]])
```

In [31]:
```
a3 = np.arange(8).reshape(2,2,2) # 3D --> Tensor
a3
```

Out[31]:
```
array([[[0, 1],
        [2, 3]],

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

ndim

returns no.of dimensions

In [32]:
```
a1.ndim
```

Out[32]:
```
1
```

In [33]:
```
a2.ndim
```

Out[33]:
```
2
```

In [34]:
```
a3.ndim
```

Out[34]:
```
3
```

shape

returns no.of rows,col

In [35]:
```
a1.shape
```

Out[35]:
```
(10,)
```

In [36]:
```
a2.shape
```

Out[36]:
```
(3, 4)
```

In [37]:
```
a3.shape
```

Out[37]:
```
(2, 2, 2)
```

size

returns no.of elements

In [38]:
```
a1.size
```

Out[38]: 10

In [39]:
```
a2.size
```

Out[39]: 12

In [40]:
```
a3.size
```

Out[40]: 8

In [41]:
```
a1
```

Out[41]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [42]:
```
a2
```

Out[42]:
```
array([[ 0.,  1.,  2.,  3.],
       [ 4.,  5.,  6.,  7.],
       [ 8.,  9., 10., 11.]])
```

In [43]:
```
a3
```

Out[43]:
```
array([[[0, 1],
        [2, 3]],

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

itemsize

returns the bytesize

In [44]:
```
a1.itemsize
```

Out[44]: 4

In [45]:
```
a2.itemsize
```

Out[45]: 8

In [46]:
```
a3.itemsize
```

Out[46]: 4

In [47]:
```
print(a1.dtype)
print(a2.dtype)
print(a3.dtype)
```

```
int32
float64
int32
```

In [48]:
```
x = np.array([33, 22, 2.5])
x
```

Out[48]:
```
array([33. , 22. ,  2.5])
```

astype

typecasting

In [49]:
```
x.astype(int)
```

Out[49]:
```
array([33, 22,  2])
```

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

Out[50]:
```
array([33. +0.j, 22. +0.j,  2.5+0.j])
```

In [51]:
```
z1 = np.arange(12).reshape(3,4)
z2 = np.arange(12,24).reshape(3,4)
```

In [52]:
```
z1
```

Out[52]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [53]:
```
z2
```

Out[53]:
```
array([[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

scalar operations

+, - ,* , / ,** , %

In [54]:
```
z1+2
```

Out[54]:
```
array([[ 2,  3,  4,  5],
       [ 6,  7,  8,  9],
       [10, 11, 12, 13]])
```

In [55]:
```
z1**2
```

Out[55]:
```
array([[  0,   1,   4,   9],
       [ 16,  25,  36,  49],
       [ 64,  81, 100, 121]])
```

In [56]:
```
z1%2
```

Out[56]:
```
array([[0, 1, 0, 1],
       [0, 1, 0, 1],
       [0, 1, 0, 1]], dtype=int32)
```

In [57]:
```
z1/2
```

Out[57]:
```
array([[0. , 0.5, 1. , 1.5],
       [2. , 2.5, 3. , 3.5],
       [4. , 4.5, 5. , 5.5]])
```

Relational operations

<, >

In [58]:
```
z2
```

Out[58]:
```
array([[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

In [59]:
```
z2>2
```

Out[59]:
```
array([[ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True]])
```

In [60]:
```
z2>20
```

Out[60]:
```
array([[False, False, False, False],
       [False, False, False, False],
       [False,  True,  True,  True]])
```

Vector operations

In [61]:
```
z1+z2
```

Out[61]:
```
array([[12, 14, 16, 18],
       [20, 22, 24, 26],
       [28, 30, 32, 34]])
```

In [62]:
```
z1-z2
```

Out[62]:
```
array([[-12, -12, -12, -12],
       [-12, -12, -12, -12],
       [-12, -12, -12, -12]])
```

Array functions

In [63]:
```
k1 = np.random.random((3,3))
k1 = np.round(k1*100)
```

In [64]:
```
k1
```

Out[64]:
```
array([[70.,  2., 93.],
       [ 7., 68., 88.],
       [11., 92., 34.]])
```

In [65]:
```
np.max(k1)
```

Out[65]:
```
93.0
```

In [66]:
```
np.sum(k1)
```

Out[66]: 465.0

In [67]:
```
np.prod(k1)
```

Out[67]: 18765495598080.0

In [68]:
```
k1
```

Out[68]:
```
array([[70.,  2., 93.],
       [ 7., 68., 88.],
       [11., 92., 34.]])
```

In [69]:
```
np.max(k1,axis=1)
```

Out[69]: array([93., 88., 92.])

In [70]:
```
np.max(k1,axis=0)
```

Out[70]: array([70., 92., 93.])

In [71]:
```
np.prod(k1,axis=0)
```

Out[71]: array([5390., 12512., 278256.])

Statistic related functions

In [72]:
```
k1
```

Out[72]:
```
array([[70.,  2., 93.],
       [ 7., 68., 88.],
       [11., 92., 34.]])
```

In [73]:
```
np.mean(k1)
```

Out[73]: 51.666666666666664

In [74]:
```
np.mean(k1,axis=0)
```

Out[74]: array([29.33333333, 54.   , 71.66666667])

In [75]:
```
k1.mean(axis=0)
```

Out[75]: array([29.33333333, 54.   , 71.66666667])

In [76]:
```
np.median(k1)
```

Out[76]: 68.0

In [77]:
```
b=np.sort(k1)
```

In [78]:
```
b
```

Out[78]:
```
array([[ 2., 70., 93.],
       [ 7., 68., 88.],
       [11., 34., 92.]])
```

In [79]:
```
np.median(b)
```

Out[79]:
```
68.0
```

In [80]:
```
np.median(b,axis=1)
```

Out[80]:
```
array([70., 68., 34.])
```

In [81]:
```
np.std(k1)
```

Out[81]:
```
36.033934623413586
```

In [82]:
```
k1
```

Out[82]:
```
array([[70.,  2., 93.],
       [ 7., 68., 88.],
       [11., 92., 34.]])
```

In [83]:
```
np.std(k1,axis=0)
```

Out[83]:
```
array([28.8020061 , 38.05259518, 26.71246068])
```

In [84]:
```
np.var(k1)
```

Out[84]:
```
1298.4444444444446
```

In [85]:
```
np.var(k1,axis=0)
```

Out[85]:
```
array([ 829.55555556, 1448.        ,  713.55555556])
```

Trigonometric functions

In [86]:
```
np.sin(k1)
```

Out[86]:
```
array([[ 0.77389068,  0.90929743, -0.94828214],
       [ 0.6569866 , -0.89792768,  0.0353983 ],
       [-0.99999021, -0.77946607,  0.52908269]])
```

In [87]:
```
np.cos(k1)
```

Out[87]:
```
array([[ 0.6333192 , -0.41614684,  0.3174287 ],
       [ 0.75390225,  0.44014302,  0.99937328],
       [ 0.0044257 , -0.62644445, -0.84857027]])
```

In [88]:
```
np.tan(k1)
```

Out[88]:
```
array([[ 1.22195992e+00, -2.18503986e+00, -2.98738626e+00],
       [ 8.71447983e-01, -2.04008160e+00,  3.54205013e-02],
       [-2.25950846e+02,  1.24427006e+00, -6.23498963e-01]])
```

Dot product

In [89]:
```
s2 = np.arange(12).reshape(3,4)
s3 = np.arange(12,24).reshape(4,3)
```

In [90]:
```
s3
```

Out[90]:
```
array([[12, 13, 14],
       [15, 16, 17],
       [18, 19, 20],
       [21, 22, 23]])
```

In [91]:
```
s2
```

Out[91]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [92]:
```
np.dot(s2,s3)
```

Out[92]:
```
array([[114, 120, 126],
       [378, 400, 422],
       [642, 680, 718]])
```

In [93]:
```
s2@s3
```

Out[93]:
```
array([[114, 120, 126],
       [378, 400, 422],
       [642, 680, 718]])
```

Exponential

In [94]:
```
np.exp(s2)
```

Out[94]:
```
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

In [95]:
```
arr = np.array([1.2, 2.7, 3.5, 4.9])
rounded_arr = np.round(arr)
print(rounded_arr)
```

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

In [96]:
```
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 [97]:
```
np.round(np.random.random((2,3))*100)
```

Out[97]:
```
array([[ 6., 13., 97.],
       [32., 48., 10.]])
```

In [98]:
```
arr = np.array([1.2, 2.7, 3.5, 4.9])
floored_arr = np.floor(arr)
print(floored_arr)
```

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

In [99]:
```
np.floor([1.2, 2.7, 3.5, 4.9])
```

Out[99]:
```
array([1., 2., 3., 4.])
```

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

Out[100…
```
array([[67., 52., 68.],
       [37., 48., 25.]])
```

In [101…
```
arr = np.array([1.2, 2.7, 3.5, 4.9])
ceiled_arr = np.ceil(arr)
print(ceiled_arr)
```

```
[2. 3. 4. 5.]
```

Indexing and Slicing

In [102…
```
p1 = np.arange(10)
p2 = np.arange(12).reshape(3,4)
p3 = np.arange(8).reshape(2,2,2)
```

In [103…
```
p1
```

Out[103…
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [104…
```
p2
```

Out[104…
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [105…
```
p3
```

Out[105…
```
array([[[0, 1],
        [2, 3]],

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

In [106…
```
p1[3]
```

Out[106…
```
3
```

In [107…
```
p2[0, :]
```

Out[107…
```
array([0, 1, 2, 3])
```

In [108…
```
p2[1:3]
```

Out[108…
```
array([[ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [109…
```
p2[:,1:3]
```

Out[109…
```
array([[ 1,  2],
       [ 5,  6],
       [ 9, 10]])
```

In [110…
```
p2[1:3 ,1:3]
```

Out[110…
```
array([[ 5,  6],
       [ 9, 10]])
```

In [111…
```
p2[::2, ::3]
```

Out[111…
```
array([[ 0,  3],
       [ 8, 11]])
```

In [112…
```
p2
```

Out[112…
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [113…
```
p2[0:2,1:3]
```

Out[113…
```
array([[1, 2],
       [5, 6]])
```

In [114…
```
p2[1::, ::3]
```

Out[114…
```
array([[ 4,  7],
       [ 8, 11]])
```

In [115…
```
p2[::2,1::2]
```

Out[115…
```
array([[ 1,  3],
       [ 9, 11]])
```

In [116…
```
p2[1,::3]
```

Out[116…
```
array([4, 7])
```

In [117…
```
p2
```

Out[117…
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [118…
```
p2[:2,1:]
```

Out[118…
```
array([[1, 2, 3],
       [5, 6, 7]])
```

In [119…
```
p2[:2,1::2]
```

Out[119…
```
array([[1, 3],
       [5, 7]])
```

In [120…
```
p3 = np.arange(27).reshape(3,3,3)
p3
```

Out[120…
```
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 [121…
```
p3[1]
```

Out[121…
```
array([[ 9, 10, 11],
       [12, 13, 14],
       [15, 16, 17]])
```

In [122…
```
p3[::2]
```

Out[122…
```
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8]],

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

In [123…
```
p3[0,1:2]
```

Out[123…
```
array([[3, 4, 5]])
```

In [124…
```
p3[0,1,:]
```

Out[124…
```
array([3, 4, 5])
```

In [125…
```
p3[0]
```

Out[125…
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

In [126…
```
p3
```

Out[126…
```
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 [127…
```
p3[1,:,1]
```

Out[127…
```
array([10, 13, 16])
```

In [128…
```
p3[2,1:,1:]
```

Out[128…
```
array([[22, 23],
       [25, 26]])
```

In [129…
```
p3[0::2]
```

Out[129…
```
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8]],

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

In [130…
```
p3[0::2, 0 , 0::2]
```

Out[130…
```
array([[ 0,  2],
       [18, 20]])
```

Iterating

In [131…
```
p1
```

Out[131…
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [132…
```
for i in p1:
    print(i)
```

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

In [133…
```
p2
```

Out[133…
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [134…
```
for i in p2:
    print(i)
```

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

In [135…
```
p3
```

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]]])
```

In [136…
```
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]]
```

nditer

converts 3D array into 1D and returns in a loop

In [137…
```
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
```

Transpose

In [138…
```
np.transpose(p2)
```

Out[138…
```
array([[ 0,  4,  8],
       [ 1,  5,  9],
       [ 2,  6, 10],
       [ 3,  7, 11]])
```

In [139…
```
p2.T
```

Out[139…
```
array([[ 0,  4,  8],
       [ 1,  5,  9],
       [ 2,  6, 10],
       [ 3,  7, 11]])
```

Flatten

converts into 1D array doesnt modify original array when the latter gets modified

In [140…
```
p2_flat = p2.flatten()
```

In [141…
```
p2_flat
```

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

In [142…
```
p2_flat[0]=10
```

In [143…
```
p2_flat
```

Out[143…
```
array([10,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

In [144…
```
p2
```

Out[144…
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

Ravel

converts into 1D array modifies original array when the latter gets modified

In [145…
```
p2_rav = p2.ravel()
```

In [146…
```
p2_rav
```

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

In [147…
```
p2_rav[0]=20
```

In [148…
```
p2_rav
```

Out[148…
```
array([20,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

In [149…
```
p2
```

Out[149…
```
array([[20,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

Stacking & Splitting

In [150…
```
w1 = np.arange(12).reshape(3,4)
w2 = np.arange(12,24).reshape(3,4)
```

In [151…
```
w1
```

Out[151…
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [152…
```
w2
```

Out[152…
```
array([[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

In [153…
```
np.hstack((w1,w2),dtype=float)
```

Out[153…
```
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 [154…
```
np.vstack((w1,w2))
```

Out[154…
```
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]])
```

In [155…
```
w1
```

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

In [156…
```
np.hsplit(w1,4)
```

Out[156…
```
[array([[0],
        [4],
        [8]]),
 array([[1],
        [5],
        [9]]),
 array([[ 2],
        [ 6],
        [10]]),
 array([[ 3],
        [ 7],
        [11]])]
```

In [157…
```
np.vsplit(w2,3)
```

Out[157…
```
[array([[12, 13, 14, 15]]),
 array([[16, 17, 18, 19]]),
 array([[20, 21, 22, 23]])]
```

Speed of list vs numpy

In [158…
```
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(2.7065064907073975 / 0.02248692512512207)
print(time.time()-start)
```

```
120.35911871666826
4.49247670173645
```

In [159…
```
import numpy as np
a = np.arange(10000000)
b = np.arange(10000000,20000000)
start =time.time()
c = a+b
print(time.time()-start)
```

```
0.26572322845458984
```

In [160…
```
2.7065064907073975 / 0.02248692512512207
```

Out[160…
```
120.35911871666826
```

Memory in list vs numpy

In [161…
```
P = [i for i in range(10000000)]
import sys
sys.getsizeof(P)
```

Out[161…
```
89095160
```

In [162…
```
R = np.arange(10000000)
sys.getsizeof(R)
```

Out[162…
```
40000112
```

In [163…
```
R = np.arange(10000000, dtype =np.int16)
sys.getsizeof(R)
```

Out[163…
```
20000112
```

Indexing

In [164…
```
w = np.arange(12).reshape(4,3)
w
```

Out[164…
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

In [165…
```
w[1:3,1:]
```

Out[165…
```
array([[4, 5],
       [7, 8]])
```

In [166…
```
w
```

Out[166…
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

In [167…
```
y=w[[0,2,3]]  # fetching multiple rows
```

In [168…
```
y
```

Out[168…
```
array([[ 0,  1,  2],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

In [169…
```
arr = np.arange(25).reshape(5, 5)

# Indices for rows and columns
row_indices = np.array([0, 2])
col_indices = np.array([1, 3])
```

In [170…
```
arr
```

Out[170…
```
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]])
```

In [171…
```
row_indices
```

Out[171…
```
array([0, 2])
```

In [172…
```
col_indices
```

Out[172…
```
array([1, 3])
```

In [173…
```
print(arr[row_indices, col_indices])
```

```
[ 1 13]
```

In [174…
```
w
```

Out[174…
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

In [175…
```
z = np.arange(24).reshape(6,4)
z
```

Out[175…
```
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]])
```

In [176…
```
z[[0,2,3,5]]
```

Out[176…
```
array([[ 0,  1,  2,  3],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [20, 21, 22, 23]])
```

In [177…
```
z[:,[0,2,3]]
```

Out[177…
```
array([[ 0,  2,  3],
       [ 4,  6,  7],
       [ 8, 10, 11],
       [12, 14, 15],
       [16, 18, 19],
       [20, 22, 23]])
```

In [178…
```
w
```

Out[178…
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

Boolean Indexing

In [179…
```
G = np.random.randint(1,100,24).reshape(6,4)
```

In [180…
```
G
```

Out[180…
```
array([[33, 48, 82, 10],
       [60, 53, 26, 10],
       [15,  3, 24, 26],
       [ 2, 96, 24, 84],
       [92, 69, 41, 42],
       [29, 85, 89, 84]])
```

In [181…
```
G>50
```

Out[181…
```
array([[False, False,  True, False],
       [ True,  True, False, False],
       [False, False, False, False],
       [False,  True, False,  True],
       [ True,  True, False, False],
       [False,  True,  True,  True]])
```

In [182…
```
G[G > 50]
```

Out[182…
```
array([82, 60, 53, 96, 84, 92, 69, 85, 89, 84])
```

In [183…
```
 G % 2 == 0
```

Out[183…
```
array([[False,  True,  True,  True],
       [ True, False,  True,  True],
       [False, False,  True,  True],
       [ True,  True,  True,  True],
       [ True, False, False,  True],
       [False, False, False,  True]])
```

In [184…
```
G[G % 2 == 0]
```

Out[184…
```
array([48, 82, 10, 60, 26, 10, 24, 26,  2, 96, 24, 84, 92, 42, 84])
```

In [185…
```
(G > 50 ) & (G % 2 == 0)
```

Out[185…
```
array([[False, False,  True, False],
       [ True, False, False, False],
       [False, False, False, False],
       [False,  True, False,  True],
       [ True, False, False, False],
       [False, False, False,  True]])
```

In [186…
```
(G > 50)  and (G % 2 == 0)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[186], line 1
----> 1 (G > 50)  and (G % 2 == 0)

ValueError: The truth value of an array with more than one element is ambiguous.
Use a.any() or a.all()
```

In [192…
```
G.all()
```

Out[192…
```
True
```

In [193…
```
G.any()
```

Out[193…
```
True
```

In [194…
```
 G % 7 == 0
```

Out[194…
```
array([[False, False, False, False],
       [False, False, False, False],
       [False, False, False, False],
       [False, False, False,  True],
       [False, False, False,  True],
       [False, False, False,  True]])
```

In [195…
```
G[ G % 7 != 0]
```

Out[195…
```
array([33, 48, 82, 10, 60, 53, 26, 10, 15,  3, 24, 26,  2, 96, 24, 92, 69,
       41, 29, 85, 89])
```

In [196…
```
 G[~(G % 7 == 0)]
```

Out[196…
```
array([33, 48, 82, 10, 60, 53, 26, 10, 15,  3, 24, 26,  2, 96, 24, 92, 69,
       41, 29, 85, 89])
```

Broadcasting

In [197…
```
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 [198…
```
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]]
```

In [199…
```
a = np.arange(6).reshape(2,3)
b = np.arange(3).reshape(3,1)
```

In [200…
```
a
```

Out[200…
```
array([[0, 1, 2],
       [3, 4, 5]])
```

In [201…
```
b
```

Out[201…
```
array([[0],
       [1],
       [2]])
```

In [202…
```
a+b
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[202], line 1
----> 1 a+b

ValueError: operands could not be broadcast together with shapes (2,3) (3,1)
```

In [203…
```
a = np.arange(6).reshape(6,1)
b = np.arange(3).reshape(1,3)
```

In [204…
```
a
```

Out[204…
```
array([[0],
       [1],
       [2],
       [3],
       [4],
       [5]])
```

In [205…
```
b
```

Out[205…
```
array([[0, 1, 2]])
```

In [206…
```
a+b
```

Out[206…
```
array([[0, 1, 2],
       [1, 2, 3],
       [2, 3, 4],
       [3, 4, 5],
       [4, 5, 6],
       [5, 6, 7]])
```

In [207…
```
a = np.arange(12).reshape(4,3)
b = np.arange(3)
```

In [213…
```
a
```

Out[213…
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [214…
```
b
```

Out[214…
```
array([[0, 1, 2]])
```

In [215…
```
a+b
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[215], line 1
----> 1 a+b

ValueError: operands could not be broadcast together with shapes (3,4) (1,3)
```

In [216…
```
a*b
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[216], line 1
----> 1 a*b

ValueError: operands could not be broadcast together with shapes (3,4) (1,3)
```

In [217…
```
a = np.arange(12).reshape(3,4)
b = np.arange(3).reshape(1,3)
```

In [218…
```
a
```

Out[218…
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [219…
```
b
```

Out[219…
```
array([[0, 1, 2]])
```

In [220…
```
a+b
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[220], line 1
----> 1 a+b

ValueError: operands could not be broadcast together with shapes (3,4) (1,3)
```

In [221…
```
a = np.arange(3).reshape(1,3)
b = np.arange(3).reshape(3,1)
```

In [222…
```
a
```

Out[222…
```
array([[0, 1, 2]])
```

In [223…
```
b
```

Out[223…
```
array([[0],
       [1],
       [2]])
```

In [224…
```
a*b
```

Out[224…
```
array([[0, 0, 0],
       [0, 1, 2],
       [0, 2, 4]])
```

In [225…
```
a+b
```

Out[225…
```
array([[0, 1, 2],
       [1, 2, 3],
       [2, 3, 4]])
```

In [226…
```
a = np.arange(3).reshape(1,3)
b = np.arange(4).reshape(4,1)
```

In [227…
```
a
```

Out[227… array([[0, 1, 2]])

In [228…
```
b
```

Out[228…
```
array([[0],
       [1],
       [2],
       [3]])
```

In [229…
```
a+b
```

Out[229…
```
array([[0, 1, 2],
       [1, 2, 3],
       [2, 3, 4],
       [3, 4, 5]])
```

In [230…
```
a = np.array([1])
 # shape -> (1,1) streched to 2,2
b = np.arange(4).reshape(2,2)
 # shape -> (2,2)

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

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

In [231…
```
a = np.arange(12).reshape(3,4)
b = np.arange(12).reshape(4,3)
```

In [232…
```
a
```

Out[232…
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [233…
```
b
```

Out[233…
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

In [234…
```
a+b
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[234], line 1
----> 1 a+b

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

In [235…
```
 a = np.arange(16).reshape(4,4)
 b = np.arange(4).reshape(2,2)
```

In [236…
```
a+b
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[236], line 1
----> 1 a+b

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

Mathematical Formulas

In [237…
```
k = np.arange(10)
```

In [238…
```
k
```

Out[238…
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [239…
```
np.sum(k)
```

Out[239…
```
45
```

In [240…
```
np.sin(k)
```

Out[240…
```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
       -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849])
```

In [241…
```
def sigmoid(array):
 return 1/(1+np.exp(-(array)))
```

In [242…
```
 k = np.arange(10)
 sigmoid(k)
```

Out[242…
```
array([0.5       , 0.73105858, 0.88079708, 0.95257413, 0.98201379,
       0.99330715, 0.99752738, 0.99908895, 0.99966465, 0.99987661])
```

In [243…
```
 k = np.arange(100)
 sigmoid(k)
```

Out[243…
```
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.        ])
```

In [244…
```
actual = np.random.randint(1,50,25)
predicted = np.random.randint(1,50,25)
```

In [245…
```
actual
```

Out[245…
```
array([47, 46, 25, 12, 26, 25, 18, 44, 39, 40, 16, 22, 42, 46, 13, 16, 21,
       23,  6, 46, 30, 11,  4, 25, 41])
```

In [246…
```
predicted
```

Out[246…
```
array([24, 33, 20, 31, 48, 46, 45, 14, 36, 41, 31, 20, 36, 26,  4,  6, 30,
       11, 19, 23, 43,  7, 27,  1,  8])
```

In [247…
```
def mse(actual,predicted):
 return np.mean((actual-predicted)**2)

mse(actual,predicted)
```

Out[247…
```
311.84
```

In [248…
```
(actual-predicted)**2
```

Out[248…
```
array([ 529,  169,   25,  361,  484,  441,  729,  900,    9,    1,  225,
          4,   36,  400,   81,  100,   81,  144,  169,  529,  169,   16,
        529,  576, 1089])
```

In [249…
```
np.mean((actual-predicted)**2)
```

Out[249…
```
311.84
```

Missing values

In [250…
```
S = np.array([1,2,3,4,np.nan,6])
S
```

Out[250…
```
array([ 1.,  2.,  3.,  4., nan,  6.])
```

In [251…
```
np.isnan(S)
```

Out[251…
```
array([False, False, False, False,  True, False])
```

In [252…
```
S[np.isnan(S)]
```

Out[252…
```
array([nan])
```

In [253…
```
S[~np.isnan(S)]
```

Out[253…
```
array([1., 2., 3., 4., 6.])
```

Plotting graphs

In [254…
```
x =np.linspace(-10,10,100)
x
```

Out[254…
```
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.        ])
```

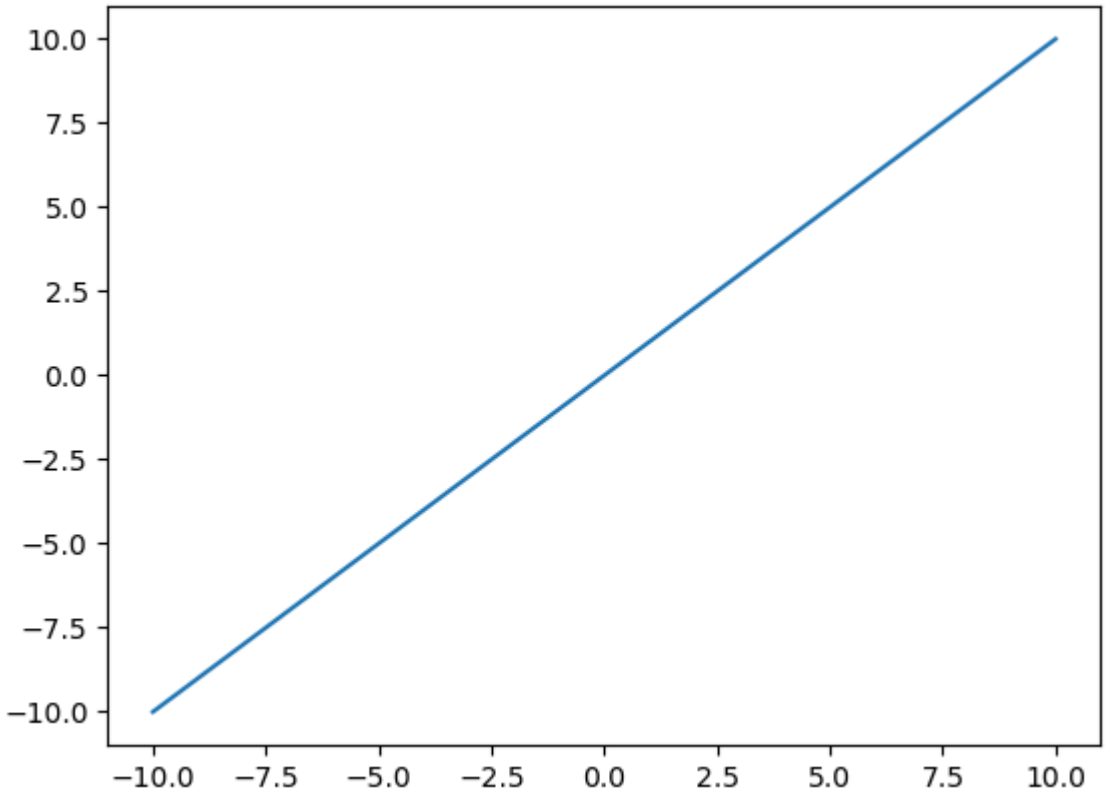
In [255…
```
y=x
```

In [256…
```
y
```

Out[256…
```
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 [257…
```
import matplotlib.pyplot as plt
plt.plot(x ,y)
```

Out[257…
```
[<matplotlib.lines.Line2D at 0x26c923d1c40>]
```

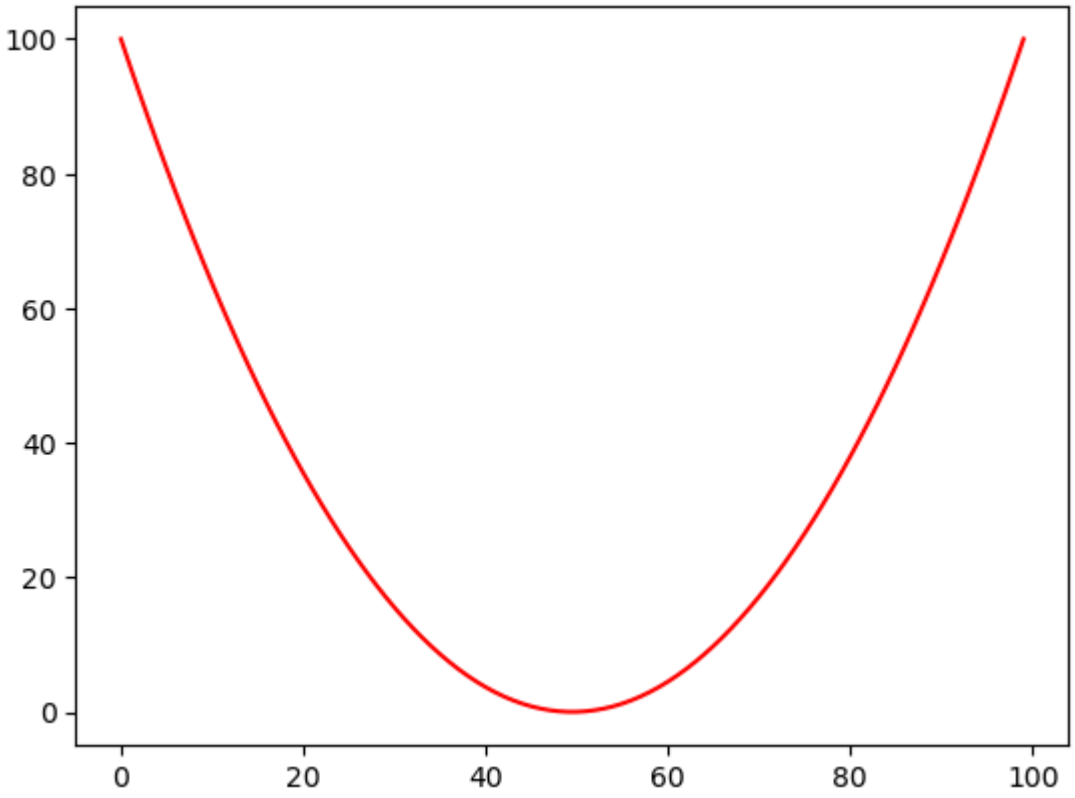


In [258…
```
y=x**2
```

In [259…
```
import matplotlib.pyplot as plt
```

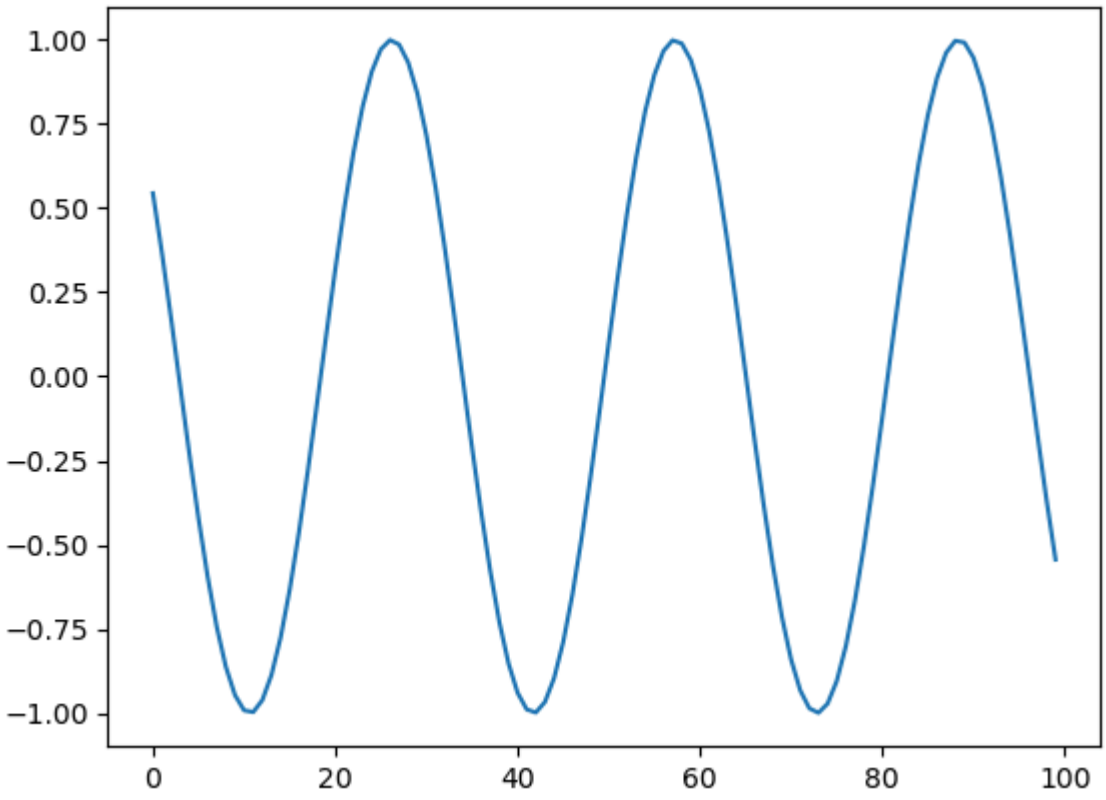
```
plt.plot(y,color='red')
```

Out[259… [<matplotlib.lines.Line2D at 0x26c924a25a0>]



In [260…
```
x = np.linspace(-10,10,100)
y = np.sin(x)
plt.plot(y)
```

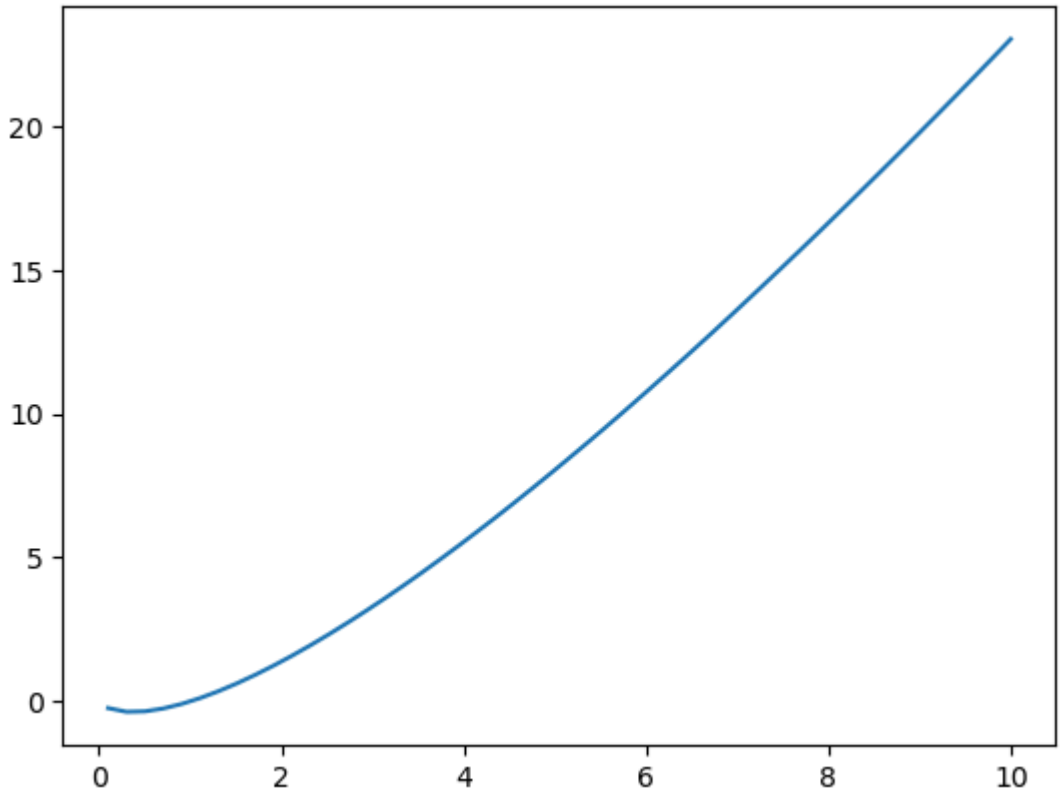
Out[260… [<matplotlib.lines.Line2D at 0x26c92506ab0>]



In [261…
```
x = np.linspace(-10,10,100)
y = x * np.log(x)
plt.plot(x,y)
```

```
C:\Users\91939\AppData\Local\Temp\ipykernel_10608\757012586.py:2: RuntimeWarning:
invalid value encountered in log
  y = x * np.log(x)
```

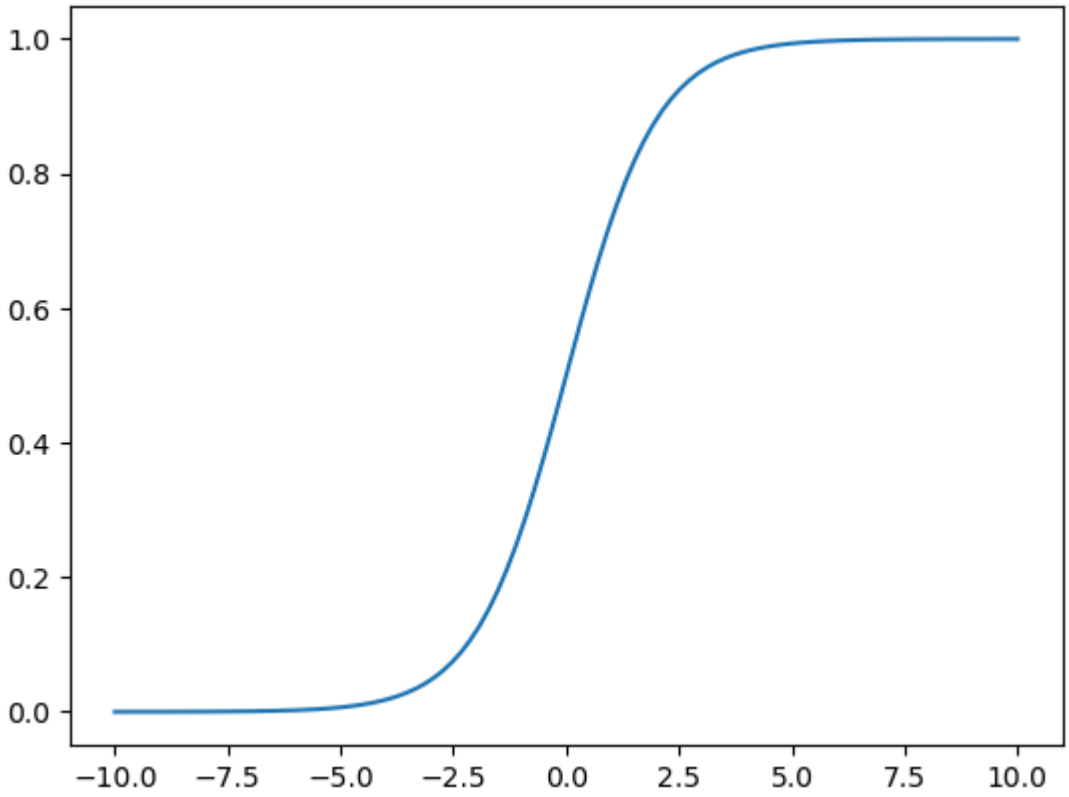
Out[261…
```
[<matplotlib.lines.Line2D at 0x26c92c85010>]
```



In [262…
```
import warnings
warnings.filterwarnings('ignore')
```

In [263…
```
x = np.linspace(-10,10,100)
y = 1/(1+np.exp(-x))
plt.plot(x,y)
```

Out[263…
```
[<matplotlib.lines.Line2D at 0x26c92cf4f20>]
```



Meshgrid

In [264…
```
x = np.linspace(0,10,100)
y = np.linspace(0,10,100)
```

In [265…
```
x
```

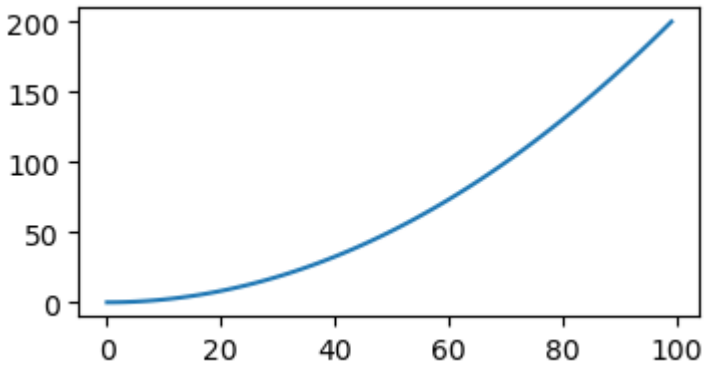
Out[265…
```
array([ 0.        ,  0.1010101 ,  0.2020202 ,  0.3030303 ,  0.4040404 ,
        0.50505051,  0.60606061,  0.70707071,  0.80808081,  0.90909091,
        1.01010101,  1.11111111,  1.21212121,  1.31313131,  1.41414141,
        1.51515152,  1.61616162,  1.71717172,  1.81818182,  1.91919192,
        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 [266…
```
y
```

Out[266…
```
array([ 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 [267…
```
f = x**2+y**2
```

In [268…
```
plt.figure(figsize=(4,2))
plt.plot(f)
plt.show()
```



In [269…
```
x = np.arange(3)
y = np.arange(3)
```

In [270…
```
x
```

Out[270…
```
array([0, 1, 2])
```

In [271…
```
y
```

Out[271…
```
array([0, 1, 2])
```

In [272…
```
xv ,yv = np.meshgrid(x,y)
```

In [273…
```
xv
```

Out[273…
```
array([[0, 1, 2],
       [0, 1, 2],
       [0, 1, 2]])
```

In [274…
```
yv
```

Out[274…
```
array([[0, 0, 0],
       [1, 1, 1],
       [2, 2, 2]])
```

In [275…
```
P = np.linspace(-4, 4, 9)
V = np.linspace(-5, 5, 11)
print(P)
print(V)
```

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

In [276…
```
 P_1, V_1 = np.meshgrid(P,V)
```

In [277…
```
print(P_1)
```

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

In [278…
```
V_1
```

Out[278…
```
array([[-5., -5., -5., -5., -5., -5., -5., -5., -5.],
       [-4., -4., -4., -4., -4., -4., -4., -4., -4.],
       [-3., -3., -3., -3., -3., -3., -3., -3., -3.],
       [-2., -2., -2., -2., -2., -2., -2., -2., -2.],
       [-1., -1., -1., -1., -1., -1., -1., -1., -1.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.],
       [ 2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.],
       [ 3.,  3.,  3.,  3.,  3.,  3.,  3.,  3.,  3.],
       [ 4.,  4.,  4.,  4.,  4.,  4.,  4.,  4.,  4.],
       [ 5.,  5.,  5.,  5.,  5.,  5.,  5.,  5.,  5.]])
```

In [279…
```
xv**2 + yv**2
```

Out[279…
```
array([[0, 1, 4],
       [1, 2, 5],
       [4, 5, 8]])
```

In [280…
```
x = np.linspace(-2,2,100)
y = np.linspace(-1,1,100)
xv, yv = np.meshgrid(x, y)
```

In [281…
```
xv
```

Out[281…
```
array([[-2.        , -1.95959596, -1.91919192, ...,  1.91919192,
         1.95959596,  2.        ],
       [-2.        , -1.95959596, -1.91919192, ...,  1.91919192,
         1.95959596,  2.        ],
       [-2.        , -1.95959596, -1.91919192, ...,  1.91919192,
         1.95959596,  2.        ],
       ...,
       [-2.        , -1.95959596, -1.91919192, ...,  1.91919192,
         1.95959596,  2.        ],
       [-2.        , -1.95959596, -1.91919192, ...,  1.91919192,
         1.95959596,  2.        ],
       [-2.        , -1.95959596, -1.91919192, ...,  1.91919192,
         1.95959596,  2.        ]])
```

In [282…
```
yv
```

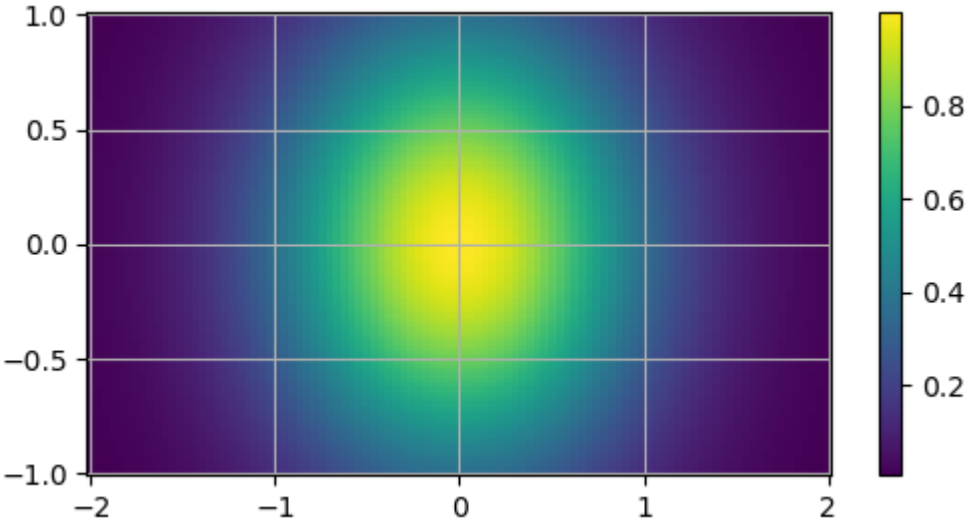
Out[282…
```
array([[-1.        , -1.        , -1.        , ..., -1.        ,
        -1.        , -1.        ],
       [-0.97979798, -0.97979798, -0.97979798, ..., -0.97979798,
        -0.97979798, -0.97979798],
       [-0.95959596, -0.95959596, -0.95959596, ..., -0.95959596,
        -0.95959596, -0.95959596],
       ...,
       [ 0.95959596,  0.95959596,  0.95959596, ...,  0.95959596,
         0.95959596,  0.95959596],
       [ 0.97979798,  0.97979798,  0.97979798, ...,  0.97979798,
         0.97979798,  0.97979798],
       [ 1.        ,  1.        ,  1.        , ...,  1.        ,
         1.        ,  1.        ]])
```

In [283…
```
f = np.exp(-xv**2-yv**2)
```

In [284…
```
f
```

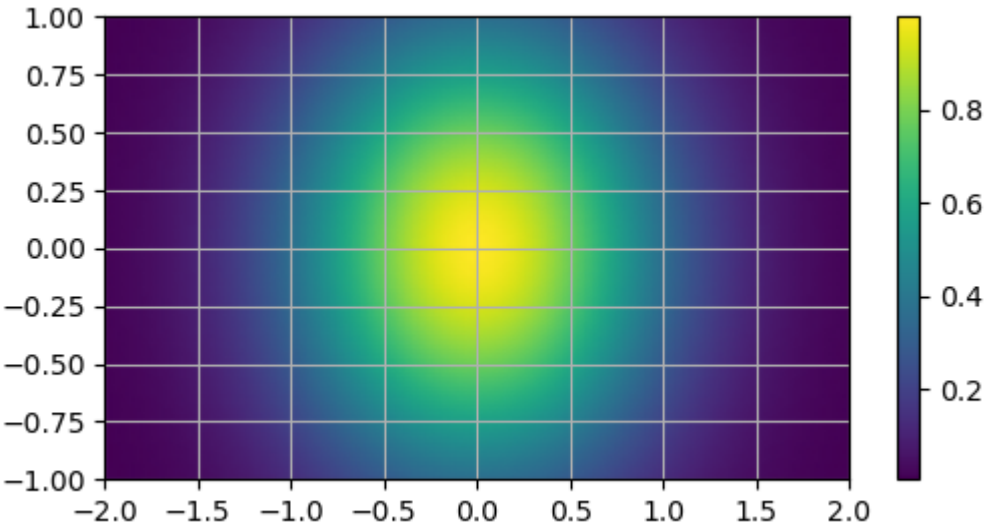
Out[284…
```
array([[0.00673795, 0.00790692, 0.00924847, ..., 0.00924847, 0.00790692,
        0.00673795],
       [0.0070129 , 0.00822958, 0.00962586, ..., 0.00962586, 0.00822958,
        0.0070129 ],
       [0.00729312, 0.00855841, 0.01001049, ..., 0.01001049, 0.00855841,
        0.00729312],
       ...,
       [0.00729312, 0.00855841, 0.01001049, ..., 0.01001049, 0.00855841,
        0.00729312],
       [0.0070129 , 0.00822958, 0.00962586, ..., 0.00962586, 0.00822958,
        0.0070129 ],
       [0.00673795, 0.00790692, 0.00924847, ..., 0.00924847, 0.00790692,
        0.00673795]])
```

In [285…
```
plt.figure(figsize=(6, 3))
plt.pcolormesh(xv, yv, f, shading='auto')
plt.colorbar()
plt.grid()
plt.show()
```



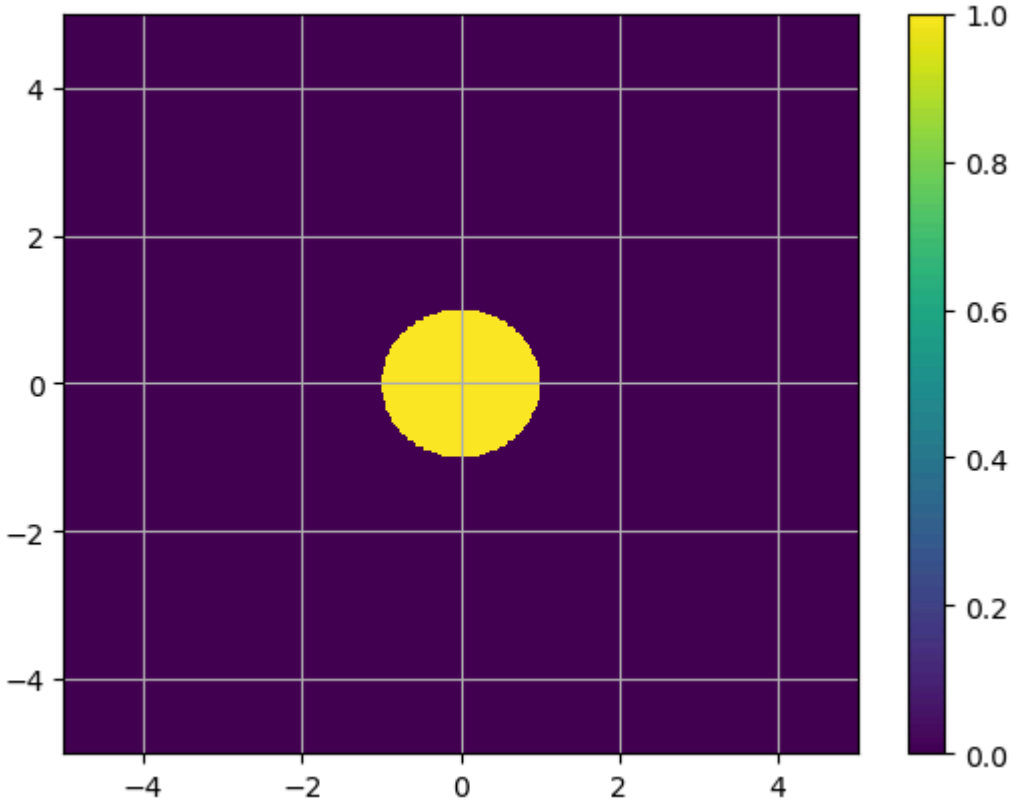
In [286…

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



In [287…

```
import numpy as np
import matplotlib.pyplot as plt
def f(x, y):
    return np.where((x**2 + y**2 < 1), 1.0, 0.0)
x = np.linspace(-5, 5, 500)
y = np.linspace(-5, 5, 500)
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()
```

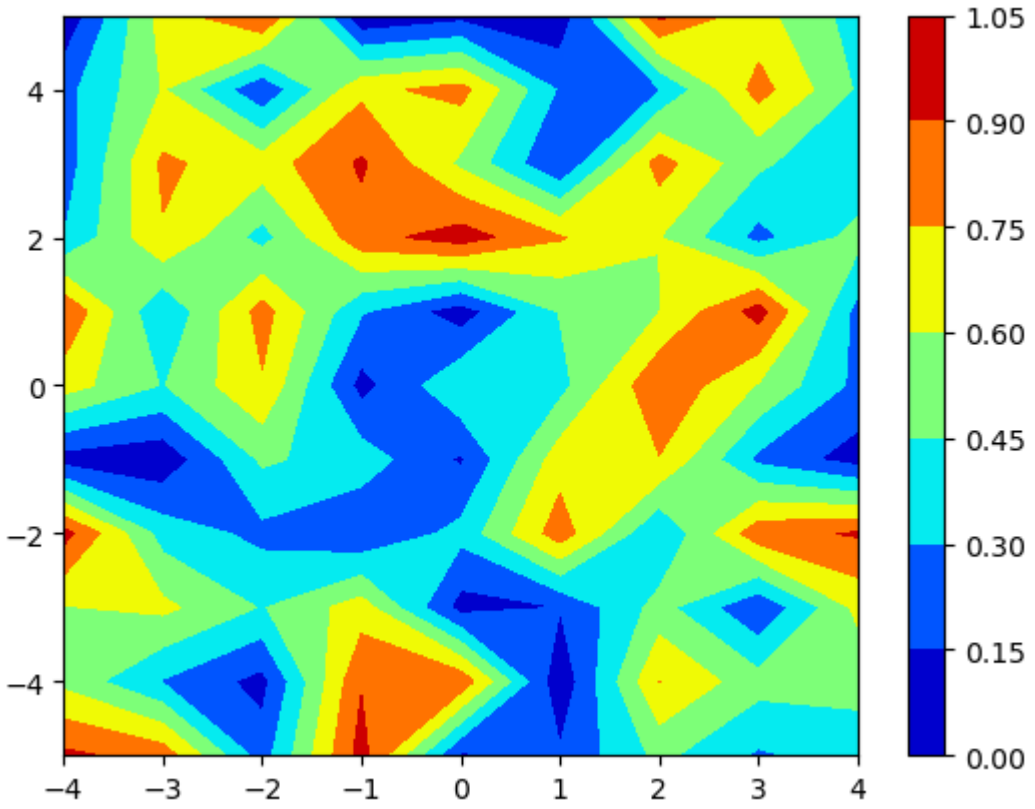


In [288…
```
x = np.linspace(-4, 4, 9)
```

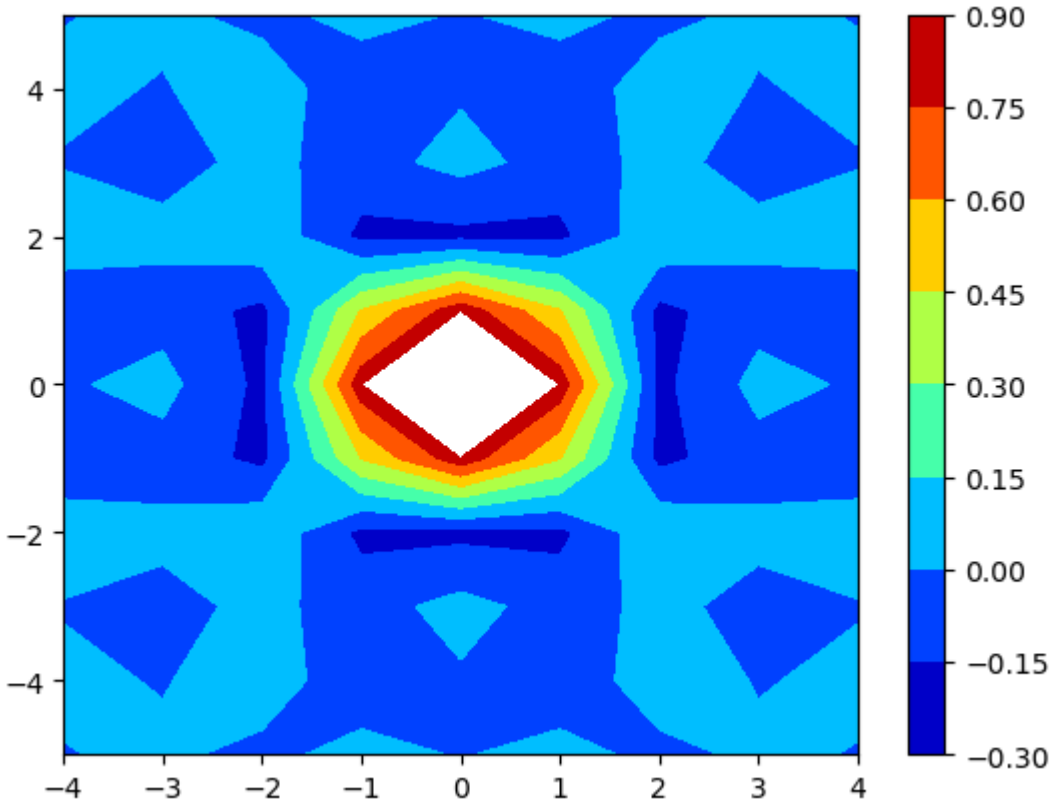
In [289…
```
y = np.linspace(-5, 5, 11)
```

In [290…
```
x_1, y_1 = np.meshgrid(x, y)
```

In [291…
```
random_data = np.random.random((11, 9))
plt.contourf(x_1, y_1, random_data, cmap = 'jet')
plt.colorbar()
plt.show()
```



In [292…
```
sine = (np.sin(x_1**2 + y_1**2))/(x_1**2 + y_1**2)
plt.contourf(x_1, y_1, sine, cmap = 'jet')
plt.colorbar()
plt.show()
```



In [293…
```
x_1, y_1 = np.meshgrid(x, y, sparse = True)
```

In [294…
```
x_1
```

Out[294…
```
array([[-4., -3., -2., -1.,  0.,  1.,  2.,  3.,  4.]])
```

In [295…
```
y_1
```

Out[295…
```
array([[-5.],
       [-4.],
       [-3.],
       [-2.],
       [-1.],
       [ 0.],
       [ 1.],
       [ 2.],
       [ 3.],
       [ 4.],
       [ 5.]])
```

np.sort

Return a sorted copy of an array

In [296…
```
a = np.random.randint(1,100,15)  #1D
a
```

Out[296…
```
array([16, 19, 22, 58, 79, 84, 93, 28, 47,  1, 47, 53, 57, 54, 81])
```

In [297…
```
a.sort()
```

In [298…
```
a
```

Out[298…
```
array([ 1, 16, 19, 22, 28, 47, 47, 53, 54, 57, 58, 79, 81, 84, 93])
```

In [299…
```
np.sort(a)[::-1]
```

Out[299…
```
array([93, 84, 81, 79, 58, 57, 54, 53, 47, 47, 28, 22, 19, 16,  1])
```

In [300…
```
b = np.random.randint(1,100,24).reshape(6,4) # 2D
b
```

Out[300…
```
array([[77, 80, 66, 17],
       [ 8, 55, 61, 20],
       [78, 18, 93, 13],
       [59, 45, 78, 34],
       [22, 12, 69, 87],
       [78, 56, 89, 76]])
```

In [301…
```
np.sort(b)
```

Out[301…
```
array([[17, 66, 77, 80],
       [ 8, 20, 55, 61],
       [13, 18, 78, 93],
       [34, 45, 59, 78],
       [12, 22, 69, 87],
       [56, 76, 78, 89]])
```

In [302…
```
np.sort(b,axis=0)
```

Out[302…
```
array([[ 8, 12, 61, 13],
       [22, 18, 66, 17],
       [59, 45, 69, 20],
       [77, 55, 78, 34],
       [78, 56, 89, 76],
       [78, 80, 93, 87]])
```

np.append

In [303…
```
a
```

Out[303…
```
array([ 1, 16, 19, 22, 28, 47, 47, 53, 54, 57, 58, 79, 81, 84, 93])
```

In [304…
```
np.append(a,200)
```

Out[304…
```
array([  1,  16,  19,  22,  28,  47,  47,  53,  54,  57,  58,  79,  81,
        84,  93, 200])
```

In [305…
```
b
```

Out[305…
```
array([[77, 80, 66, 17],
       [ 8, 55, 61, 20],
       [78, 18, 93, 13],
       [59, 45, 78, 34],
       [22, 12, 69, 87],
       [78, 56, 89, 76]])
```

In [306…
```
np.append(b,np.ones((b.shape[0],1))) #shape of col in b but only one col filled
```

Out[306…
```
array([77., 80., 66., 17.,  8., 55., 61., 20., 78., 18., 93., 13., 59.,
       45., 78., 34., 22., 12., 69., 87., 78., 56., 89., 76.,  1.,  1.,
        1.,  1.,  1.,  1.])
```

In [307…
```
np.append(b,[1,5,9,3])
```

Out[307…
```
array([77, 80, 66, 17,  8, 55, 61, 20, 78, 18, 93, 13, 59, 45, 78, 34, 22,
       12, 69, 87, 78, 56, 89, 76,  1,  5,  9,  3])
```

In [308…
```
b
```

Out[308…
```
array([[77, 80, 66, 17],
       [ 8, 55, 61, 20],
       [78, 18, 93, 13],
       [59, 45, 78, 34],
       [22, 12, 69, 87],
       [78, 56, 89, 76]])
```

In [309…
```
np.append(b,np.ones((b.shape[0],1)),axis=1)
```

Out[309…
```
array([[77., 80., 66., 17.,  1.],
       [ 8., 55., 61., 20.,  1.],
       [78., 18., 93., 13.,  1.],
       [59., 45., 78., 34.,  1.],
       [22., 12., 69., 87.,  1.],
       [78., 56., 89., 76.,  1.]])
```

In [310…
```
np.append(b,np.random.random((b.shape[0],1)),axis=1)
```

Out[310…
```
array([[77.        , 80.        , 66.        , 17.        ,  0.24792081],
       [ 8.        , 55.        , 61.        , 20.        ,  0.27416796],
       [78.        , 18.        , 93.        , 13.        ,  0.84367265],
       [59.        , 45.        , 78.        , 34.        ,  0.43165463],
       [22.        , 12.        , 69.        , 87.        ,  0.52738811],
       [78.        , 56.        , 89.        , 76.        ,  0.8572762 ]])
```

np.concatenate

In [311…
```
c = np.arange(6).reshape(2,3)
d = np.arange(6,12).reshape(2,3)
```

In [312…
```
c
```

Out[312…
```
array([[0, 1, 2],
       [3, 4, 5]])
```

In [313…
```
d
```

Out[313…
```
array([[ 6,  7,  8],
       [ 9, 10, 11]])
```

In [314…
```
np.concatenate((c,d))
```

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

In [315…
```
np.concatenate((c,d),axis=1)
```

Out[315…
```
array([[ 0,  1,  2,  6,  7,  8],
       [ 3,  4,  5,  9, 10, 11]])
```

np.unique

In [316…
```
e = np.array([1,1,2,2,3,3,4,4,5,5,6,6])
```

In [317…
```
e
```

Out[317…
```
array([1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6])
```

In [318…
```
np.unique(e)
```

Out[318…
```
array([1, 2, 3, 4, 5, 6])
```

np.expand_dims

In [319…
```
a
```

Out[319…
```
array([ 1, 16, 19, 22, 28, 47, 47, 53, 54, 57, 58, 79, 81, 84, 93])
```

In [320…
```
a.shape
```

Out[320…
```
(15,)
```

In [321…
```
np.expand_dims(a,axis = 0)
```

Out[321…
```
array([[ 1, 16, 19, 22, 28, 47, 47, 53, 54, 57, 58, 79, 81, 84, 93]])
```

In [322…
```
np.expand_dims(a,axis = 0).shape
```

Out[322…
```
(1, 15)
```

In [323…
```
np.expand_dims(a,axis = 1)
```

Out[323…
```
array([[ 1],
       [16],
       [19],
       [22],
       [28],
       [47],
       [47],
       [53],
       [54],
       [57],
       [58],
       [79],
       [81],
       [84],
       [93]])
```

In [324…
```
np.expand_dims(a,axis = 1).shape
```

Out[324…
```
(15, 1)
```

np.where

returns the indices of elements in an input array where the given condition is satisfied.

In [325…
```
a
```

Out[325…
```
array([ 1, 16, 19, 22, 28, 47, 47, 53, 54, 57, 58, 79, 81, 84, 93])
```

In [326…
```
 np.where(a>50)
```

Out[326…
```
(array([ 7,  8,  9, 10, 11, 12, 13, 14], dtype=int64),)
```

In [327…
```
 np.where(a>50,0,a)
```

Out[327…
```
array([ 1, 16, 19, 22, 28, 47, 47,  0,  0,  0,  0,  0,  0,  0,  0])
```

In [328…
```
 np.where(a%2 == 0,0,a)
```

Out[328…
```
array([ 1,  0, 19,  0,  0, 47, 47, 53,  0, 57,  0, 79, 81,  0, 93])
```

np.argmax

returns indices of the max element of the array in a particular axis

In [329…
```
a
```

Out[329… array([1, 16, 19, 22, 28, 47, 47, 53, 54, 57, 58, 79, 81, 84, 93])

In [330…
```
np.argmax(a)
```

Out[330… 14

In [331…
```
b
```

Out[331…
```
array([[77, 80, 66, 17],
       [ 8, 55, 61, 20],
       [78, 18, 93, 13],
       [59, 45, 78, 34],
       [22, 12, 69, 87],
       [78, 56, 89, 76]])
```

In [332…
```
np.argmax(b,axis =1)
```

Out[332… array([1, 2, 2, 2, 3, 2], dtype=int64)

In [333…
```
np.argmax(b)
```

Out[333… 10

In [334…
```
 np.argmax(b,axis =0)
```

Out[334… array([2, 0, 2, 4], dtype=int64)

In [335…
```
a
```

Out[335… array([1, 16, 19, 22, 28, 47, 47, 53, 54, 57, 58, 79, 81, 84, 93])

In [336…
```
 np.argmin(a)
```

Out[336… 0

In [337…
```
np.argmin(b,axis = 0)
```

Out[337… array([1, 4, 1, 2], dtype=int64)

Statistics

np.cumsum

In [338…
```
a
```

Out[338… array([1, 16, 19, 22, 28, 47, 47, 53, 54, 57, 58, 79, 81, 84, 93])

In [339…
```
np.cumsum(a)
```

Out[339…
```
array([  1,  17,  36,  58,  86, 133, 180, 233, 287, 344, 402, 481, 562,
       646, 739])
```

In [340…
```
b
```

Out[340…
```
array([[77, 80, 66, 17],
       [ 8, 55, 61, 20],
       [78, 18, 93, 13],
       [59, 45, 78, 34],
       [22, 12, 69, 87],
       [78, 56, 89, 76]])
```

In [341…
```
np.cumsum(b)
```

Out[341…
```
array([  77,  157,  223,  240,  248,  303,  364,  384,  462,  480,  573,
        586,  645,  690,  768,  802,  824,  836,  905,  992, 1070, 1126,
       1215, 1291])
```

In [342…
```
np.cumsum(b,axis=1)
```

Out[342…
```
array([[ 77, 157, 223, 240],
       [  8,  63, 124, 144],
       [ 78,  96, 189, 202],
       [ 59, 104, 182, 216],
       [ 22,  34, 103, 190],
       [ 78, 134, 223, 299]])
```

In [343…
```
np.cumsum(b,axis=0)
```

Out[343…
```
array([[ 77,  80,  66,  17],
       [ 85, 135, 127,  37],
       [163, 153, 220,  50],
       [222, 198, 298,  84],
       [244, 210, 367, 171],
       [322, 266, 456, 247]])
```

In [344…
```
a
```

Out[344…
```
array([ 1, 16, 19, 22, 28, 47, 47, 53, 54, 57, 58, 79, 81, 84, 93])
```

In [345…
```
np.cumprod(a)
```

Out[345…
```
array([          1,          16,         304,        6688,      187264,
         8801408,   413666176,   449470848, -1498377984,   491800832,
       -1540322816, -1426418176,   424244736,  1276819456, -1514874880])
```

np.percentile

In [346…
```
a
```

Out[346…
```
array([ 1, 16, 19, 22, 28, 47, 47, 53, 54, 57, 58, 79, 81, 84, 93])
```

In [347…
```
 np.percentile(a,100)
```

Out[347…
```
93.0
```

In [348…
```
 np.percentile(a,0)
```

Out[348…
```
1.0
```

In [349…
```
 np.percentile(a,50)
```

Out[349… 53.0

In [350…
```
np.median(a)
```

Out[350… 53.0

np.histogram

represents the frequency of data distribution in the graphical form

In [351…
```
a
```

Out[351… array([1, 16, 19, 22, 28, 47, 47, 53, 54, 57, 58, 79, 81, 84, 93])

In [352…
```
np.histogram(a, bins= [10,20,30,40,50,60,70,80,90,100])
```

Out[352…
```
(array([2, 2, 0, 2, 4, 0, 1, 2, 1], dtype=int64),
 array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100]))
```

In [353…
```
 np.histogram(a , bins= [0,50,100])
```

Out[353… (array([7, 8], dtype=int64), array([0, 50, 100]))

np.corrcoef

Return Pearson product-moment correlation coefficients

In [354…
```
salary = np.array([20000,40000,25000,35000,60000])
experience = np.array([1,3,2,4,2])
```

In [355…
```
salary
```

Out[355… array([20000, 40000, 25000, 35000, 60000])

In [356…
```
experience
```

Out[356… array([1, 3, 2, 4, 2])

In [357…
```
np.corrcoef(salary,experience)
```

Out[357…
```
array([[1.        , 0.25344572],
       [0.25344572, 1.        ]])
```

Utility functions

np.isin

we can see that one array having values are checked in a different numpy array having different elements with different sizes.n

In [358…
```
a
```

Out[358… array([1, 16, 19, 22, 28, 47, 47, 53, 54, 57, 58, 79, 81, 84, 93])

In [359…
```
items = [10,20,30,40,50,60,70,80,90,100]
np.isin(a,items)
```

Out[359… array([False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False])

In [360…
```
a[np.isin(a,items)]
```

Out[360… array([], dtype=int32)

In [361…
```
np.isin(items,a)
```

Out[361… array([False, False, False, False, False, False, False, False, False,
False])

np.flip

reverses the order of array elements along the specified axis, preserving the shape of the array

In [362…
```
a
```

Out[362… array([1, 16, 19, 22, 28, 47, 47, 53, 54, 57, 58, 79, 81, 84, 93])

In [363…
```
c=np.expand_dims(a,axis=0)
```

In [364…
```
np.flip(a)
```

Out[364… array([93, 84, 81, 79, 58, 57, 54, 53, 47, 47, 28, 22, 19, 16, 1])

In [365…
```
np.flip(c,axis=0)
```

Out[365… array([[1, 16, 19, 22, 28, 47, 47, 53, 54, 57, 58, 79, 81, 84, 93]])

In [366…
```
np.flip(c,axis=1)
```

Out[366… array([[93, 84, 81, 79, 58, 57, 54, 53, 47, 47, 28, 22, 19, 16, 1]])

In [367…
```
b
```

Out[367…
```
array([[77, 80, 66, 17],
       [ 8, 55, 61, 20],
       [78, 18, 93, 13],
       [59, 45, 78, 34],
       [22, 12, 69, 87],
       [78, 56, 89, 76]])
```

In [368…
```
np.flip(b)
```

Out[368…
```
array([[76, 89, 56, 78],
       [87, 69, 12, 22],
       [34, 78, 45, 59],
       [13, 93, 18, 78],
       [20, 61, 55,  8],
       [17, 66, 80, 77]])
```

In [369…
```
np.flip(b,axis = 1)
```

Out[369…
```
array([[17, 66, 80, 77],
       [20, 61, 55,  8],
       [13, 93, 18, 78],
       [34, 78, 45, 59],
       [87, 69, 12, 22],
       [76, 89, 56, 78]])
```

In [370…
```
np.flip(b,axis = 0)
```

Out[370…
```
array([[78, 56, 89, 76],
       [22, 12, 69, 87],
       [59, 45, 78, 34],
       [78, 18, 93, 13],
       [ 8, 55, 61, 20],
       [77, 80, 66, 17]])
```

np.put

replaces specific elements of an array with given values of p_array. Array indexed works on flattened array

In [371…
```
a
```

Out[371…
```
array([ 1, 16, 19, 22, 28, 47, 47, 53, 54, 57, 58, 79, 81, 84, 93])
```

In [372…
```
np.put(a,[0,1],[110,530])
```

In [373…
```
a
```

Out[373…
```
array([110, 530,  19,  22,  28,  47,  47,  53,  54,  57,  58,  79,  81,
        84,  93])
```

np.delete

In [374…
```
a
```

Out[374…
```
array([110, 530,  19,  22,  28,  47,  47,  53,  54,  57,  58,  79,  81,
        84,  93])
```

In [375…
```
 np.delete(a,0)
```

Out[375…
```
array([530,  19,  22,  28,  47,  47,  53,  54,  57,  58,  79,  81,  84,
        93])
```

In [376…
```
np.delete(a,[0,2,4])
```

Out[376…
```
array([530,  22,  47,  47,  53,  54,  57,  58,  79,  81,  84,  93])
```

Set functions

In [377…
```
m = np.array([1,2,3,4,5])
n = np.array([3,4,5,6,7])
```

In [378…
```
np.union1d(m,n)
```

Out[378…
```
array([1, 2, 3, 4, 5, 6, 7])
```

In [379…
```
np.intersect1d(m,n)
```

Out[379…
```
array([3, 4, 5])
```

In [380…
```
np.setdiff1d(m,n)
```

Out[380…
```
array([1, 2])
```

In [381…
```
np.setdiff1d(n,m)
```

Out[381…
```
array([6, 7])
```

In [382…
```
np.setxor1d(m,n)
```

Out[382…
```
array([1, 2, 6, 7])
```

In [383…
```
np.in1d(m,2)
```

Out[383…
```
array([False,  True, False, False, False])
```

In [384…
```
m[np.in1d(m,2)]
```

Out[384…
```
array([2])
```

In [385…
```
np.in1d(m,10)
```

Out[385…
```
array([False, False, False, False, False])
```

np.clip

is used to Clip (limit) the values in an array

In [386…
```
a
```

Out[386…
```
array([110, 530,  19,  22,  28,  47,  47,  53,  54,  57,  58,  79,  81,
        84,  93])
```

In [387…
```
np.clip(a, a_min=21 , a_max =50)
```

Out[387…
```
array([50, 50, 21, 22, 28, 47, 47, 50, 50, 50, 50, 50, 50, 50, 50])
```

np.swapaxes

function interchange two axes of an array.

In [388…
```
arr = np.array([[1, 2, 3], [4, 5, 6]])
swapped_arr = np.swapaxes(arr, 0, 1)
```

In [389…
```
arr
```

Out[389…
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [390…
```
swapped_arr
```

Out[390…
```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

In [391…
```
print("Original array:")
print(arr)
```

```
Original array:
[[1 2 3]
 [4 5 6]]
```

In [392…
```
print("Swapped array:")
print(swapped_arr)
```

```
Swapped array:
[[1 4]
 [2 5]
 [3 6]]
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: