```
# plotting 3D graphs
# meshgrids
a = np.linspace(-10,9,20)
b = np.linspace(-10,9,20)
b
```

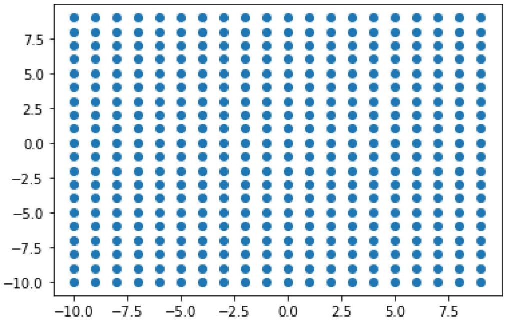
```
array([-10.,  -9.,  -8.,  -7.,  -6.,  -5.,  -4.,  -3.,  -2.,  -1.,   0.,
         1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.])
```

```
xx,yy = np.meshgrid(a,b)
yy
```

```
array([[-10., -10., -10., -10., -10., -10., -10., -10., -10., -10., -10.,
        -10., -10., -10., -10., -10., -10., -10., -10., -10.],
       [ -9.,  -9.,  -9.,  -9.,  -9.,  -9.,  -9.,  -9.,  -9.,  -9.,  -9.,
         -9.,  -9.,  -9.,  -9.,  -9.,  -9.,  -9.,  -9.,  -9.],
       [ -8.,  -8.,  -8.,  -8.,  -8.,  -8.,  -8.,  -8.,  -8.,  -8.,  -8.,
         -8.,  -8.,  -8.,  -8.,  -8.,  -8.,  -8.,  -8.,  -8.],
       [ -7.,  -7.,  -7.,  -7.,  -7.,  -7.,  -7.,  -7.,  -7.,  -7.,  -7.,
         -7.,  -7.,  -7.,  -7.,  -7.,  -7.,  -7.,  -7.,  -7.],
       [ -6.,  -6.,  -6.,  -6.,  -6.,  -6.,  -6.,  -6.,  -6.,  -6.,  -6.,
         -6.,  -6.,  -6.,  -6.,  -6.,  -6.,  -6.,  -6.,  -6.],
       [ -5.,  -5.,  -5.,  -5.,  -5.,  -5.,  -5.,  -5.,  -5.,  -5.,  -5.,
         -5.,  -5.,  -5.,  -5.,  -5.,  -5.,  -5.,  -5.,  -5.],
       [ -4.,  -4.,  -4.,  -4.,  -4.,  -4.,  -4.,  -4.,  -4.,  -4.,  -4.,
         -4.,  -4.,  -4.,  -4.,  -4.,  -4.,  -4.,  -4.,  -4.],
       [ -3.,  -3.,  -3.,  -3.,  -3.,  -3.,  -3.,  -3.,  -3.,  -3.,  -3.,
         -3.,  -3.,  -3.,  -3.,  -3.,  -3.,  -3.,  -3.,  -3.],
       [ -2.,  -2.,  -2.,  -2.,  -2.,  -2.,  -2.,  -2.,  -2.,  -2.,  -2.,
         -2.,  -2.,  -2.,  -2.,  -2.,  -2.,  -2.,  -2.,  -2.],
       [ -1.,  -1.,  -1.,  -1.,  -1.,  -1.,  -1.,  -1.,  -1.,  -1.,  -1.,
         -1.,  -1.,  -1.,  -1.,  -1.,  -1.,  -1.,  -1.,  -1.],
       [  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
       [  1.,   1.,   1.,   1.,   1.,   1.,   1.,   1.,   1.,   1.,   1.,
          1.,   1.,   1.,   1.,   1.,   1.,   1.,   1.,   1.],
       [  2.,   2.,   2.,   2.,   2.,   2.,   2.,   2.,   2.,   2.,   2.,
          2.,   2.,   2.,   2.,   2.,   2.,   2.,   2.,   2.],
       [  3.,   3.,   3.,   3.,   3.,   3.,   3.,   3.,   3.,   3.,   3.,
          3.,   3.,   3.,   3.,   3.,   3.,   3.,   3.,   3.],
       [  4.,   4.,   4.,   4.,   4.,   4.,   4.,   4.,   4.,   4.,   4.,
          4.,   4.,   4.,   4.,   4.,   4.,   4.,   4.,   4.],
       [  5.,   5.,   5.,   5.,   5.,   5.,   5.,   5.,   5.,   5.,   5.,
          5.,   5.,   5.,   5.,   5.,   5.,   5.,   5.,   5.],
       [  6.,   6.,   6.,   6.,   6.,   6.,   6.,   6.,   6.,   6.,   6.,
          6.,   6.,   6.,   6.,   6.,   6.,   6.,   6.,   6.],
       [  7.,   7.,   7.,   7.,   7.,   7.,   7.,   7.,   7.,   7.,   7.,
          7.,   7.,   7.,   7.,   7.,   7.,   7.,   7.,   7.],
       [  8.,   8.,   8.,   8.,   8.,   8.,   8.,   8.,   8.,   8.,   8.,
          8.,   8.,   8.,   8.,   8.,   8.,   8.,   8.,   8.],
       [  9.,   9.,   9.,   9.,   9.,   9.,   9.,   9.,   9.,   9.,   9.,
          9.,   9.,   9.,   9.,   9.,   9.,   9.,   9.,   9.]])
```

```
import matplotlib.pyplot as plt
plt.scatter(xx,yy)
```

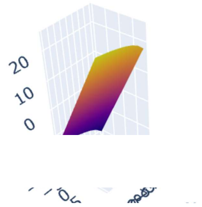
```
<matplotlib.collections.PathCollection at 0x7f0ba86d3160>
```



```
def func(x,y):
  return 3*np.log(x) + 2*y

zz = func(xx,yy)
zz
```

```
<ipython-input-34-78e1ed4898ab>:2: RuntimeWarning:

divide by zero encountered in log

<ipython-input-34-78e1ed4898ab>:2: RuntimeWarning:

invalid value encountered in log

array([[         nan,          nan,          nan,          nan,
                 nan,          nan,          nan,          nan,
                 nan,          nan,         -inf, -20.        ,
        -17.92055846, -16.70416313, -15.84111692, -15.17168626,
        -14.62472159, -14.16226955, -13.76167537, -13.40832627],
       [         nan,          nan,          nan,          nan,
                 nan,          nan,          nan,          nan,
                 nan,          nan,         -inf, -18.        ,
        -15.92055846, -14.70416313, -13.84111692, -13.17168626,
        -12.62472159, -12.16226955, -11.76167537, -11.40832627],
       [         nan,          nan,          nan,          nan,
                 nan,          nan,          nan,          nan,
                 nan,          nan,         -inf, -16.        ,
        -13.92055846, -12.70416313, -11.84111692, -11.17168626,
        -10.62472159, -10.16226955,  -9.76167537,  -9.40832627],
       [         nan,          nan,          nan,          nan,
                 nan,          nan,          nan,          nan,
                 nan,          nan,         -inf, -14.        ,
        -11.92055846, -10.70416313,  -9.84111692,  -9.17168626,
         -8.62472159,  -8.16226955,  -7.76167537,  -7.40832627],
       [         nan,          nan,          nan,          nan,
                 nan,          nan,          nan,          nan,
                 nan,          nan,         -inf, -12.        ,
         -9.92055846,  -8.70416313,  -7.84111692,  -7.17168626,
         -6.62472159,  -6.16226955,  -5.76167537,  -5.40832627],
       [         nan,          nan,          nan,          nan,
                 nan,          nan,          nan,          nan,
                 nan,          nan,         -inf, -10.        ,
         -7.92055846,  -6.70416313,  -5.84111692,  -5.17168626,
         -4.62472159,  -4.16226955,  -3.76167537,  -3.40832627],
       [         nan,          nan,          nan,          nan,
                 nan,          nan,          nan,          nan,
                 nan,          nan,         -inf,  -8.        ,
         -5.92055846,  -4.70416313,  -3.84111692,  -3.17168626,
         -2.62472159,  -2.16226955,  -1.76167537,  -1.40832627],
       [         nan,          nan,          nan,          nan,
                 nan,          nan,          nan,          nan,
                 nan,          nan,         -inf,  -6.        ,
         -3.92055846,  -2.70416313,  -1.84111692,  -1.17168626,
         -0.62472159,  -0.16226955,   0.23832463,   0.59167373],
       [         nan,          nan,          nan,          nan,
                 nan,          nan,          nan,          nan,
                 nan,          nan,         -inf,  -4.        ,
         -1.92055846,  -0.70416313,   0.15888308,   0.82831374,
          1.37527841,   1.83773045,   2.23832463,   2.59167373],
       [         nan,          nan,          nan,          nan,
                 nan,          nan,          nan,          nan,
                 nan,          nan,         -inf,  -2.        ,
          0.07944154,   1.29583687,   2.15888308,   2.82831374,
          3.37527841,   3.83773045,   4.23832463,   4.59167373],
```

```
import plotly.express as px
import plotly.graph_objects as go

fig = px.scatter_3d()

fig.add_trace(go.Surface(x=xx,y=yy,z=zz))

fig.show()
```



```
# working with random
# randint
# seed
# shuffle
# choice
```

```
np.random.random((2,3,2))
```

```
     array([[[0.28969975, 0.30904037],
             [0.02229412, 0.08411571],
             [0.34225695, 0.87044578]],

            [[0.3088764 , 0.55506361],
             [0.95240073, 0.44318119],
             [0.28857773, 0.17184448]]])
```

```
np.random.seed(0)
np.random.randint(1,100,12).reshape(3,4)
```

```
     array([[45, 48, 65, 68],
            [68, 10, 84, 22],
            [37, 88, 71, 89]])
```

```
np.random.seed(0)
np.random.randint(1,100,12).reshape(3,4)
```

```
     array([[45, 48, 65, 68],
            [68, 10, 84, 22],
            [37, 88, 71, 89]])
```

```
np.random.seed(0)
np.random.randint(1,100,12).reshape(3,4)
```

```
     array([[45, 48, 65, 68],
            [68, 10, 84, 22],
            [37, 88, 71, 89]])
```

```
a = np.array([12,41,33,67,89,100])
print(a)
```

```
     [ 12  41  33  67  89 100]
```

```
np.random.shuffle(a)
```

```
a
```

```
     array([ 67, 100,  41,  33,  89,  12])
```

```
np.random.choice(a,3,replace=False)
```

```
     array([ 33, 100,  89])
```

```
# working with images
import cv2
```

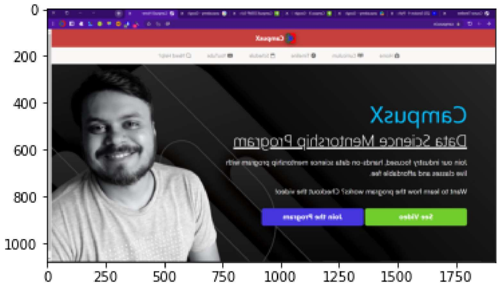
```
# read the image
img = cv2.imread('/content/screenshot1.png')
```

```
# show array -> shape
img.shape
```

```
    (1080, 1920, 3)
```

```
# show image
plt.imshow(np.flip(img,axis=1))
```

```
    <matplotlib.image.AxesImage at 0x7f0b99c62a90>
```



```
# flip
a = np.arange(6).reshape(2,3)
a
```

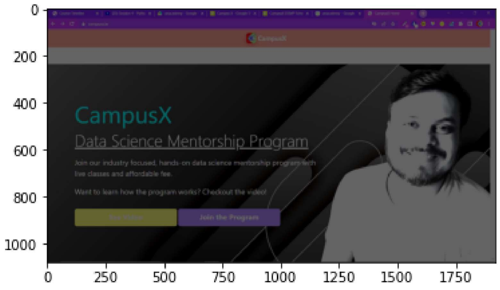
```
    array([[0, 1, 2],
           [3, 4, 5]])
```

```
np.flip(a)
```

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

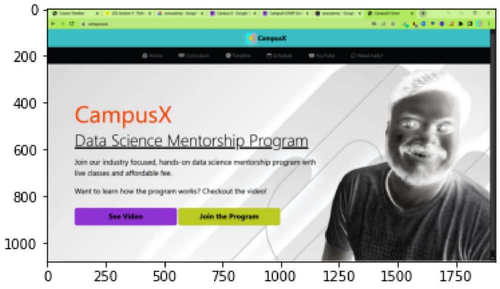
```
# clip -> fade
plt.imshow(np.clip(img,0,100))
```

```
    <matplotlib.image.AxesImage at 0x7f0b99c27460>
```



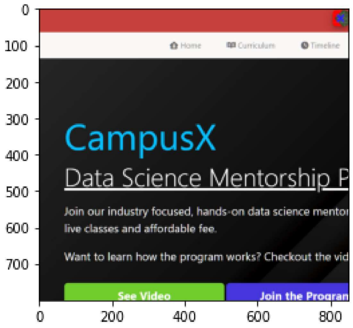
```
# negative
plt.imshow(255 - img)
```

```
    <matplotlib.image.AxesImage at 0x7f0b99bfed30>
```


CampusX
Data Science Mentorship Program
Join our industry focused, hands-on data science mentorship program with live classes and affordable fee.
Want to learn how the program works? Checkout the video!
See Video
Join the Program
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750

```
# trim
plt.imshow(img[100:900,50:900,:])
```

```
<matplotlib.image.AxesImage at 0x7f0b99bd7370>
```



CampusX

Data Science Mentorship P

Join our industry focused, hands-on data science mentor
live classes and affordable fee.

Want to learn how the program works? Checkout the vid

```
# plot histogram
plt.hist(img.flatten(),bins=255)
```

```
(array([205709.,  26478.,  22496.,  24156.,  26061.,  27144.,  37194.,
         42868.,  48354.,  42823.,  51278.,  53786.,  44132., 185308.,
        135016., 137699., 167025., 131062., 125666., 123704., 163884.,
        120953., 122947., 136906., 151304., 133090., 132126., 123105.,
         88560., 114552., 108528.,  74169.,  92631.,  64596.,  76882.,
         53629.,  61145.,  39742.,  43245.,  22674.,   6509.,   3723.,
          3422.,   3473.,   3372.,   3382.,  34249.,   3205.,   3491.,
          3358.,   3831.,  71214.,   4049.,  33943.,   4043.,   3985.,
          4147.,   4055.,   4242.,   4633., 141864.,   3722.,   4225.,
          4836., 141839.,   4054.,   5356.,  50107.,   6289.,  35701.,
          3326.,   2543.,   2480.,   4497.,   2555.,   2413.,   2538.,
          2518.,   2459.,  46022.,   2539.,   2399.,   2533.,   3118.,
          2300.,   2474.,   2294.,   2465.,   2497.,   2419.,   2412.,
```

```
# More manipulations
# https://www.analyticsvidhya.com/blog/2021/05/image-processing-using-numpy-with-practical-implementation-and-code/
```

```
# structured arrays
a = np.array([1,'hello',True,1.5])
a
```

```
array(['1', 'hello', 'True', '1.5'], dtype='<U32')
```

```
a[0] / 100
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-91-4ea26a035019> in <module>
----> 1 a[0] / 100

TypeError: unsupported operand type(s) for /: 'numpy.str_' and 'int'
```

SEARCH STACK OVERFLOW

```
          1361.,   1394.,  86033.]),
```

```
# name,iq,cgpa,placed

dt = np.dtype(
    [
        ('name','U20'),
        ('iq',np.int32),
        ('cgpa',np.float64),
        ('placed','U20')
    ]
)
```

```
        132.. 133.. 134.. 135.. 136.. 137.. 138.. 139.. 140.. 141.. 142..
```

```
dt
```

```
dtype([('name', '<U20'), ('iq', '<i4'), ('cgpa', '<f8'), ('placed', '<U20')])
```

```
        187.. 188.. 189.. 190.. 191.. 192.. 193.. 194.. 195.. 196.. 197..
```

```
stu = np.array(
    [
        ('nitish',100,6.66,'Yes'),
        ('ankit',120,8.9,'Yes'),
        ('rahul',80,7.3,'No')
    ],dtype=dt
)

stu
```

```
array([('nitish', 100, 6.66, 'Yes'), ('ankit', 120, 8.9 , 'Yes'),
       ('rahul',  80, 7.3 , 'No')],
      dtype=[('name', '<U20'), ('iq', '<i4'), ('cgpa', '<f8'), ('placed', '<U20')])
```

```
stu['placed']
```

```
array(['Yes', 'Yes', 'No'], dtype='<U20')
```

```
# save and load numpy objects
np.save('student.npy',stu)
```

```
# remaining functions
# --> np.swapaxes
# --> np.uniform
# --> np.count_nonzero
# --> np.tile
# --> np.repeat
# --> np.allclose
```

np.swapaxes(arr, axis1, axis2)

```
Interchange two axes of an array.

Syntax : numpy.swapaxes(arr, axis1, axis2)
Parameters :
arr : [array_like] input array.
axis1 : [int] First axis.
axis2 : [int] Second axis.
Return : [ndarray]
```

[ ] ↳ *3 cells hidden*

numpy.random.uniform(low=0.0, high=1.0, size=None)

Draw samples from a uniform distribution in rangge [low - high); high not included. https://numpy.org/doc/stable/reference/random/generated/numpy.random.uniform.html

```
Syntax : numpy.random.uniform(low, high, size=None)
            low -> lower bound of sample; default value is 0
            high -> uper bound of sample; defalut value is 1.0
            size -> shape of the desired sample. If the given shape is, e.g., (m, n, k), then m * n * k samples are drawn.

Return : Return the random samples as numpy array.
```

When ever we need to test our model on uniform data and we might not get truly uniform data in real scenario, we can use this function to randomly generate data for us.

[ ] ↳ *3 cells hidden*

np.count_nonzero(arr, axis=None)

This function counts the number of non-zero values in the array https://numpy.org/doc/stable/reference/generated/numpy.count_nonzero.html

```
Syntax : numpy.count_nonzero(arr, axis=None)

Parameters :
    arr : [array_like] The array for which to count non-zeros.

    axis : [int or tuple, optional] Axis or tuple of axes along which to count non-zeros. Default is None, meaning that non-zeros will be counted

    keepdims : [bool, optional] If this is set to True, the axes that are counted are left in the result as dimensions with size one.

Return : [int or array of int] Number of non-zero values in the array along a given axis. Otherwise, the total number of non-zero values in the ar
```

[ ] ↳ *1 cell hidden*

np.tile(A, reps)

Construct an array by repeating `A` the number of times given by `reps`. If reps has length `d`, the result will have dimension of `max(d, A.ndim)`.

```
Parameters:

    A: array_like
        The input array.

    reps: array_like
```

```
        The number of repetitions of A along each axis.

Returns
    c: ndarray
        The tiled output array.
```

https://numpy.org/doc/stable/reference/generated/numpy.tile.html

[ ] ↳ *3 cells hidden*

np.repeat(a, repeats, axis=None)

Repeat elements of an array. `repeats` parameter says no of time to repeat

```
Parameters:
            a: array_like
                    Input array.

            repeats: int or array of ints
                    The number of repetitions for each element. repeats is broadcasted to fit the shape of the given axis.

            axis: int, optional
                    The axis along which to repeat values. By default, use the flattened input array, and return a flat output array.

Returns:
    repeated_array: ndarray
                    Output array which has the same shape as a, except along the given axis.
```

https://numpy.org/doc/stable/reference/generated/numpy.repeat.html

[ ] ↳ *3 cells hidden*

np.allclose

Returns True if two arrays are element-wise equal within a tolerance.

The tolerance values are positive, typically very small numbers. The relative difference `(rtol * abs(b))` and the absolute difference `atol` are added together to compare against the `absolute difference` between `a` and `b`.

If the following equation is element-wise True, then `allclose` returns `True`.

> `absolute(a - b) <= (atol + rtol * absolute(b))`

```
Syntax : numpy.allclose(arr1, arr2, rtol, atol, equal_nan=False)

Parameters :
    arr1 : [array_like] Input 1st array.
    arr2 : [array_like] Input 2nd array.
    rtol : [float] The relative tolerance parameter.
    atol : [float] The absolute tolerance parameter.
    equal_nan : [bool] Whether to compare NaN’s as equal.
            If True, NaN’s in arr1 will be considered equal to NaN’s in arr2 in the output array.

Return : [ bool] Returns True if the two arrays are equal within the given tolerance, otherwise it returns False.
```

https://numpy.org/doc/stable/reference/generated/numpy.allclose.html https://www.geeksforgeeks.org/numpy-allclose-in-python/

```
#np.allclose example
#Comparing -
a = np.array([1.1, 1.2, 1.0001])
b = np.array([1., 1.02, 1.001])
print(a)
print(b)
print(np.abs(a-b))

print(np.allclose(a,b)) # will return false
print(np.allclose(a,b, atol=0.2)) # will return true, as i incearse the absolute tolerance value
```

```
[1.1    1.2    1.0001]
[1.    1.02  1.001]
[0.1    0.18   0.0009]
False
True
```

```
print(np.allclose([1.0, np.nan], [1.0, np.nan])) # Nan will be taken as different

print(np.allclose([1.0, np.nan], [1.0, np.nan], equal_nan=True)) # Nan will be treated as same
```

```
False
True
```