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

In [2]:
```
my_list = [1,2,3]
print(type(my_list))
```

```
<class 'list'>
```

In [3]:
```
arr=np.array(my_list)
print(type(arr))
```

```
<class 'numpy.ndarray'>
```

In [10]:
```
l=[[1,2,3],[4,5,6],[7,8,9]]
print(f"{l} is of type {type(l)}")

# to convert this 2d array to a matrix  we will use numpy
matrix =np.array(l)
print(f"{matrix} is of shape {matrix.shape}") # 3x3 matrix
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]] is of type <class 'list'>
[[1 2 3]
 [4 5 6]
 [7 8 9]] is of shape (3, 3)
```

Create an array using Numpy

```
numpy.arange([start, ]stop, [step, ]dtype=None)
```

In [21]:
```
print(np.arange(0,11))
print(np.arange(10,-1,-1))
arr = np.arange(0, 1, 0.1, dtype=float)
print(arr)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10]
[10  9  8  7  6  5  4  3  2  1  0]
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
[0 0 0 0 0]
[[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
```

For an array of all zeros

```
numpy.zeros(shape, dtype=float, order='C')
```

In [ ]:
```
# array of zeros
print(np.zeros(5,dtype=int)) # dedault dtype is Float
arr = np.zeros((3, 4),dtype=int) #(rows,columns)
print(arr)
```

For all ones

```
numpy.ones(shape, dtype=float, order='C')
```

In [24]:
```
print(np.ones(5,dtype=int))
```

```
[1 1 1 1 1]
```

Ques1) create an 3x3 matix with all elements as 12 and after that subract 5 from each element

In [30]:
```
mat=np.ones((3,3),dtype=int)*12
print(f"{mat} is of dimension {mat.shape}")

mat=mat-5
print(mat)
```

```
[[12 12 12]
 [12 12 12]
 [12 12 12]] is of dimension (3, 3)
[[7 7 7]
 [7 7 7]
 [7 7 7]]
```

Linearly spaced array

```
numpy.linspace(start, stop, num=50, endpoint=True,
retstep=False, dtype=None, axis=0)

num: The number of samples to generate. Default is 50.
endpoint: If True (default), stop is the last value in the
range. If False, it is not included.
retstep: If True, return the step size between values along
with the array. Default is False
```

Note: Both start and end is included in the linearly spaced array

In [37]:
```
num=np.linspace(0,10,num=8) # both start and end are included
print(num)
print(f"Length of num = {len(num)}") #8 since we requested for 8 equi-distar
```

```
[ 0.          1.42857143  2.85714286  4.28571429  5.71428571  7.14285714
  8.57142857 10.        ]
Length of num = 8
```

Create an Identity Matrix

```
which has only 1s in diagonal and rest of the element is zero
numpy.eye(N, M=None, k=0, dtype=float, order='C')

N: Number of rows in the output matrix.
M: Number of columns in the output matrix. If None, it
```

```
    defaults to N (i.e., the matrix will be square).
    k: Diagonal in question. Default is 0, which refers to the
    main diagonal. Positive values refer to diagonals
       above the main diagonal, and negative values refer to
    diagonals below.
```

In [39]:
```
print(np.eye(3,4,dtype=int))
```

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

Create random numbers

The `numpy.random` module in NumPy provides functions for generating random numbers and performing random operations. This module is useful for tasks such as simulations, testing algorithms, and creating random datasets.

Key Functions in `numpy.random`

Here are some commonly used functions from `numpy.random`:

1. **`numpy.random.rand`**

   Generates random numbers from a uniform distribution over [0, 1).

   ```
   arr = np.random.rand(3, 2)
   print(arr)
   ```
   Output:

   ```
   [[0.96396653 0.73174974]
    [0.51684794 0.24066518]
    [0.27697977 0.06095282]]
   ```
2. **`numpy.random.randn`**

   Generates random numbers from a standard normal distribution (mean 0, standard deviation 1).

   ```
   arr = np.random.randn(2, 3)
   print(arr)
   ```
   Output:

   ```
   [[ 1.20105982 -0.43100507  0.38559543]
    [-0.26714863 -0.33052843 -0.04681264]]
   ```
3. **`numpy.random.randint`**

   Generates random integers from a specified range.

   ```
   arr = np.random.randint(1, 10, size=(2, 3))
   print(arr)
   ```

Output:

```
[[3 4 9]
 [1 5 7]]
```

4. **`numpy.random.choice`**

   Generates a random sample from a given array.

   ```
   arr = np.random.choice([10, 20, 30, 40], size=5)
   print(arr)
   ```
   Output:

   ```
   [30 10 20 30 40]
   ```
5. **`numpy.random.seed`**

   Sets the seed for the random number generator to ensure reproducibility.

   ```
   np.random.seed(0)
   arr = np.random.rand(2, 2)
   print(arr)
   ```
   Output:

   ```
   [[0.5488135  0.71518937]
    [0.60276338 0.54488318]]
   ```
6. **`numpy.random.permutation`**

   Randomly permutes a sequence or returns a permuted range.

   ```
   arr = np.random.permutation([1, 2, 3, 4, 5])
   print(arr)
   ```
   Output:

   ```
   [3 1 5 4 2]
   ```
7. **`numpy.random.shuffle`**

   Shuffles the array in place.

   ```
   import numpy as np

   arr = np.array([1, 2, 3, 4, 5])
   np.random.shuffle(arr)
   print(arr)
   ```
   Output:

   ```
   [2 4 1 5 3]
   ```
8. **`numpy.random.normal`** The numpy.random.normal function generates random numbers from a normal (Gaussian) distribution. This is useful for simulating data with a specific mean and standard deviation.

   ```
   arr = np.random.normal(loc=0, scale=1, size=(3, 4))
   print(arr)
   ```

loc: Mean of the distribution. Default is 0.0.
scale: Standard deviation of the distribution. Default is 1.0.
Output:

```
[[ 1.13180988 -0.63446097  1.45985531 -0.45274872]
  [-0.67548934  1.28997832 -1.22416163 -0.30653762]
  [ 1.06822232  0.13236941  0.86727861 -0.04750807]]
```

Reshape array

```
numpy.reshape(a, newshape, order='C')
```

In [48]:
```
arr=np.arange(25)
print(arr)

rearrange=arr.reshape(5,5) #ensure the new shape should be able to accomodat
print(rearrange)
```

```
[ 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]
[[ 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]]
```

Max and Min value

In [47]:
```
arr=np.random.randint(1,25,10)
print(arr)
print(f"max element is {arr.max()} at location {arr.argmax()}")
print(f"min element is {arr.min()} at location {arr.argmin()}")
```

```
[ 5 11 13 19 19  6  7 24 11  8]
max element is 24 at location 7
min element is 5 at location 0
```

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

In [20]:
```
arr=np.arange(0,11)
print(arr)
```

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

In [30]:
```
print(arr[4]) # element at index 4
print(arr[4:]) # all elements from 4 to end
print(arr[:4]) # all elements from start to before index 4
```

```
9
[ 9  5  6  7  8  9 10]
[9 9 9 9]
```

In [22]:
```
print(arr)
print(arr+100) # possible with only numpy array..normal python will give cor
print(arr**2) # square of each elements of the array
```

```
[ 0  1  2  3  4  5  6  7  8  9 10]
[100 101 102 103 104 105 106 107 108 109 110]
[  0   1   4   9  16  25  36  49  64  81 100]
```

Copying numpy array

if you dont use numpy "copy()" function then all the copies of the original array will stay connected to the orginal array meaning changes to one will also bring change to the original and vice-versa

In [26]:
```
print(f"arr = {arr}")
slice_arr=arr[:5]
slice_arr[:]=9 # This is only possibly with numpy array
print(f"arr = {arr}") # orginal arr is also changes since we dont use a copy

#using copy function
new_copy=arr[:5].copy()
new_copy[:]=-17
print(f"new_copy = {new_copy}")
print(f"arr = {arr}") # we can see that the original array has not been affe
```

```
arr = [ 9  9  9  9  9  5  6  7  8  9 10]
arr = [ 9  9  9  9  9  5  6  7  8  9 10]
new_copy = [-17 -17 -17 -17 -17]
arr = [ 9  9  9  9  9  5  6  7  8  9 10]
```

Indexing in 2-D array

In [53]:
```
arr_2d=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(arr_2d.shape)
print(arr_2d)
```

```
(3, 3)
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [38]:
```
print(arr_2d[0,2]) # indexing start from zero for both rows and columns
print(arr_2d[0][2]) # alternate way to do the same job as above
```

```
3
3
```

In [42]:
```
arr_2d
```

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

In [41]:
```
arr_2d[0:2] # capture row 0 and row 1
```

Out[41]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [45]:
```
arr_2d[0] # capture row 0
```

Out[45]:
```
array([1, 2, 3])
```

In [46]:
```
arr_2d[:,0:2] # capture columns 0 and column 1
```

Out[46]:
```
array([[1, 2],
       [4, 5],
       [7, 8]])
```

In [47]:
```
arr_2d[:2,:2] # row 0,1 and column 0,1
```

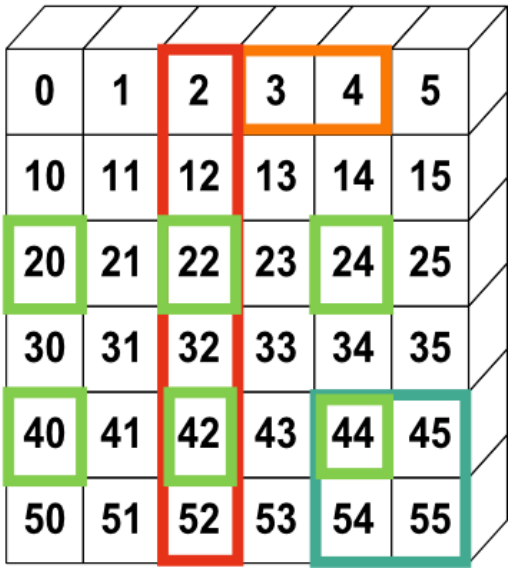
Out[47]:
```
array([[1, 2],
       [4, 5]])
```

```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```



Conditional selection

In [49]:
```
arr=np.arange(0,11)
print(f"arr = {arr}")
```

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

In [50]:
```
arr>4
```

Out[50]:
```
array([False, False, False, False, False,  True,  True,  True,  True,
        True,  True])
```

In [51]:
```
bool_arr=arr>4 # create a filter based in condition
arr[bool_arr] # pass that filter to the original array
```

Out[51]:
```
array([ 5,  6,  7,  8,  9, 10])
```

In [52]:
```
arr[arr>4] #directly we can use this for conditional selection
```

Out[52]:
```
array([ 5,  6,  7,  8,  9, 10])
```

In [2]:
```
import numpy as np
```

In [4]:
```
arr=np.arange(0,11)
print(arr)
```

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

In [5]:
```
arr+100 # add 100 to each number
```

Out[5]:
```
array([100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110])
```

In [6]:
```
arr/10 # divide each number by 10
```

Out[6]:
```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ])
```

In [7]:
```
arr**2 # square of each number
```

Out[7]:
```
array([  0,   1,   4,   9,  16,  25,  36,  49,  64,  81, 100], dtype=int32)
```

In [8]:
```
(arr+2)/10
```

Out[8]:
```
array([0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2])
```

In [9]:
```
arr+2*arr
```

Out[9]:
```
array([ 0,  3,  6,  9, 12, 15, 18, 21, 24, 27, 30])
```

In [10]:
```
100/arr #numpy dont produce an error with dividing by 0
```

```
C:\Users\himan\anaconda3\envs\data\lib\site-packages\ipykernel_launcher.py:
1: RuntimeWarning: divide by zero encountered in true_divide
  """Entry point for launching an IPython kernel.
```

Out[10]:
```
array([        inf, 100.        ,  50.        ,  33.33333333,
        25.        ,  20.        ,  16.66666667,  14.28571429,
        12.5       ,  11.11111111,  10.        ])
```

In [13]:
```
arr/arr
```

```
C:\Users\himan\anaconda3\envs\data\lib\site-packages\ipykernel_launcher.py:
1: RuntimeWarning: invalid value encountered in true_divide
  """Entry point for launching an IPython kernel.
```

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

Array Aggregation Operations

In [14]:
```
arr
```

Out[14]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [15]:
```
np.sum(arr)  #sun of array
```

Out[15]:
```
55
```

In [16]:
```
np.mean(arr)# Mean of array elements
```

Out[16]:
```
5.0
```

In [17]:
```
np.max(arr) # Max value in the array
```

Out[17]:
```
10
```

In [18]:
```
np.min(arr) # Min value in the array
```

Out[18]:
```
0
```

In [23]:
```
np.sin(np.pi/2) # takes input in radian
```

Out[23]:
```
1.0
```

In [37]:
```
arr_2d=np.array([[1,2,3],[4,5,6],[7,8,9]])
arr_2d
```

Out[37]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [39]:
```
arr_2d[0].sum() # sum of row=0
```

Out[39]:
```
6
```

In [38]:
```
arr_2d[:,0].sum() #sum of column=0
```

Out[38]:
```
12
```

In [43]:
```
arr_2d.sum() # sum of whole matrix
```

Out[43]:
```
45
```

In [44]:
```
arr_2d.sum(axis=0) # sum of each columns
```

Out[44]:
```
array([12, 15, 18])
```

In [45]:
```
arr_2d.sum(axis=1) # sum of each row
```

Out[45]:
```
array([ 6, 15, 24])
```