NumPy is a python library in Python is that provides a multidimensional array object, various derived objects

you can perform various mathematical operations. It can be logical, sorting, shape manipulation etc.

In [1]:

```
#importing numpy package
import numpy as np
```

creating simple array and converting to numpy array

In [2]:

```
# for 1D array
my_array = [12,34,43,14,51,66]
my_array
```

Out[2]:

```
[12, 34, 43, 14, 51, 66]
```

In [3]:

```
# converting the array to numpy array
np.array(my_array)
```

Out[3]:

```
array([12, 34, 43, 14, 51, 66])
```

In [4]:

```
# similarly for a 2D array
my_2D_array = [[12,34],[43,14],[51,66]]
my_2D_array
```

Out[4]:

```
[[12, 34], [43, 14], [51, 66]]
```

In [5]:

```
np.array(my_2D_array )
```

Out[5]:

```
array([[12, 34],
       [43, 14],
       [51, 66]])
```

built-in methods to generate numpy arrays

In [6]:

```
np.arange(0,10) # returns values 0 to 9.
```

Out[6]:

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

In [7]:

```
np.arange(0,10,2) # specify start, stop & step values
```

Out[7]:

```
array([0, 2, 4, 6, 8])
```

In [8]:

```
np.linspace(0,10,10) #returns evenly spaced numbers over a specified interval
#specify start, stop & number of values
```

Out[8]:

```
array([ 0.        ,  1.11111111,  2.22222222,  3.33333333,  4.44444444,
        5.55555556,  6.66666667,  7.77777778,  8.88888889, 10.        ])
```

In [9]:

```
np.zeros(10) #generate array of zeroes
```

Out[9]:

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [10]:

```
np.ones(10) #generate arrays of ones
```

Out[10]:

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

In [11]:

```
np.zeros((8,9)) #generate 2D array of zeroes
#similarly for ones
```

Out[11]:

```
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.]])
```

In [12]:

```
np.eye(10) #generate 2D indentity matrix or a numpy array with ones in the diagonal
```

Out[12]:

```
array([[1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]])
```

In [13]:

```
np.random.rand(10) # generate random array
# every time you run, it changes it's value
```

Out[13]:

```
array([0.21013991, 0.68954267, 0.74045612, 0.1313745 , 0.94915501,
       0.25334371, 0.36994212, 0.31475852, 0.51703398, 0.42209257])
```

In [14]:

```
np.random.rand(4,5)  # generate random 2D array
```

Out[14]:

```
array([[0.31123053, 0.11546977, 0.81594086, 0.0702239 , 0.24611474],
       [0.2120293 , 0.20722576, 0.43096128, 0.41401176, 0.30734651],
       [0.34900529, 0.73627945, 0.59049034, 0.65417599, 0.3562694 ],
       [0.42681665, 0.93158831, 0.26632491, 0.63290923, 0.9650526 ]])
```

In [15]:

```
np.random.randn(4) # generates a set of numbers from the Standard Normal distribution

#what is SND ?
#It's a  is a normal distribution with a mean of zero and standard deviation of 1. ~ Bell shaped graph
# It  allows us to make comparisons across the infinitely many normal distributions that can possibly exist
```

Out[15]:

```
array([-0.01410566,  1.79486853, -0.81765678,  0.5170481 ])
```

In [16]:

```
# similarly for 2D arrays
```

In [17]:

```
# some other properties
np.random.randint(1,20) #generates a random integer from 1 to 19
#it changes every time you run the cell
```

Out[17]:

```
16
```

In [18]:

```
np.random.randint(1,20,10) # generates 10 random integers from 1 to 19
```

Out[18]:

```
array([ 5,  2,  4, 11, 12,  6,  6, 13, 12, 12])
```

Different methods in numpy arrays

In [19]:

```
array_ = np.random.randint(0,100,20) # array storing my random integer numpy array
```

In [20]:

```
array_
```

Out[20]:

```
array([82, 81, 67, 29, 78, 56, 96, 15, 99,  1, 73,  7,  8, 21, 78, 82, 11,
       83, 90, 17])
```

In [21]:

```
array_.shape # would give the dimension or shape of the array
```

Out[21]:

```
(20,)
```

In [22]:

```
#similarly with 2D array
```

In [23]:

```
array_.reshape(4,5) # to change the dimension
```

Out[23]:

```
array([[82, 81, 67, 29, 78],
       [56, 96, 15, 99,  1],
       [73,  7,  8, 21, 78],
       [82, 11, 83, 90, 17]])
```

In [24]:

```
array_.reshape(4,6) # not possible as 4*6 != 10
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-24-bc319ffd74c1> in <module>
----> 1 array_.reshape(4,6) # not possible as 4*6 != 10

ValueError: cannot reshape array of size 20 into shape (4,6)
```

In [ ]:

```
array_.reshape(4,5).T # this would transpose the array
```

indexing and comparison in numpy arrays

In [25]:

```
array = np.arange(1,10)
array
```

Out[25]:

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

In [26]:

```
array[4] # to show element in index 4
```

Out[26]:

```
5
```

In [27]:

```
array[[0,2,5,8]] # to show elements from multiple indexes
```

Out[27]:

```
array([1, 3, 6, 9])
```

In [28]:

```
array[0:5] # to show elements from the range
```

Out[28]:

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

In [29]:

```
array[[0,4,7]] # to show elements from multiple indexes
```

Out[29]:

```
array([1, 5, 8])
```

In [30]:

```
array[3:7]= 8383 # to replace the elements from the index range
```

```
array
```

Out[30]:

```
array([   1,    2,    3, 8383, 8383, 8383, 8383,    8,    9])
```

In [31]:

```
# You can perform the similar operation in a 2D array
# In a 2d array the elements would be of the form array[i][j]
```

In [32]:

```
array2D = np.arange(1,21).reshape(4,5)
array2D
```

Out[32]:

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

In [33]:

```
# in this array
array2D[:,(2,4)] # means select all rows and cols of index 2 & 4
```

Out[33]:

```
array([[ 3,  5],
       [ 8, 10],
       [13, 15],
       [18, 20]])
```

selection & copy in numpy arrays

In [34]:

```
array = np.arange(1,10)
array
```

Out[34]:

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

In [35]:

```
array <5 #returns a boolean value
```

Out[35]:

```
array([ True,  True,  True,  True, False, False, False, False, False])
```

In [36]:

```
array[array <5]
```

Out[36]:

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

In [37]:

```
#copy a numpy array
#when you slice/make any index value changes/reshape then it affects the original array
# you can copy the array if you don't want the original array to be changed
```

In [38]:

```
#consider reshape the array
array = np.arange(1,10)
```

```
array[3]= 1000
array
```

Out[38]:

```
array([   1,    2,    3, 1000,    5,    6,    7,    8,    9])
```

In [39]:

```
array_copy = array.copy()
array_copy
```

Out[39]:

```
array([   1,    2,    3, 1000,    5,    6,    7,    8,    9])
```

numpy array operations and mathematical functions

In [40]:

```
array = np.arange(1,10)
array
```

Out[40]:

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

In [41]:

```
#if you want to multiply all array elements with itself
array*array
```

Out[41]:

```
array([ 1,  4,  9, 16, 25, 36, 49, 64, 81])
```

In [42]:

```
# similarly divide/add/subtract
# if you have a 0 element in the array and you divide the array with itself then it would
give 'nan' result
```

In [43]:

```
array**4 # gives fourth power of all elements
```

Out[43]:

```
array([   1,   16,   81,  256,  625, 1296, 2401, 4096, 6561], dtype=int32)
```

In [44]:

```
# similarly to multiply every element with a number
array*5
```

Out[44]:

```
array([ 5, 10, 15, 20, 25, 30, 35, 40, 45])
```

In [45]:

```
### Some Mathematical functions that we can perform are :
```

In [46]:

```
np.sqrt(array) #square root
```

Out[46]:

```
array([1.        , 1.41421356, 1.73205081, 2.        , 2.23606798,
       2.44948974, 2.64575131, 2.82842712, 3.        ])
```

In [47]:

```
np.max(array) # for maximum element
#similarly min()
```

Out[47]:

```
9
```

In [48]:

```
np.argmax(array) # for index of max element
#similarly argmin()
```

Out[48]:

```
8
```

In [49]:

```
np.log(array)# to find log of elements
```

Out[49]:

```
array([0.        , 0.69314718, 1.09861229, 1.38629436, 1.60943791,
       1.79175947, 1.94591015, 2.07944154, 2.19722458])
```

In [50]:

```
np.sin(array)# to find sin() of the array
# similarly exp, var , man , std
```

Out[50]:

```
array([ 0.84147098,  0.90929743,  0.14112001, -0.7568025 , -0.95892427,
       -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849])
```

In [51]:

```
array = np.random.randn(3,3) # consider a matrix with normalised values
# we can round off the values of this matrix using the functions
array
```

Out[51]:

```
array([[-0.41146276,  0.30947654, -0.01114751],
       [ 0.4071601 ,  0.2216775 ,  0.54974761],
       [-0.12738131,  1.10647735, -0.71858387]])
```

In [52]:

```
np.round(array,decimals=3) # to round off upto 3 decimal places
```

Out[52]:

```
array([[-0.411,  0.309, -0.011],
       [ 0.407,  0.222,  0.55 ],
       [-0.127,  1.106, -0.719]])
```

i/p & o/p in numpy

In [ ]:

```
cd Desktop
```

In [54]:

```
array_to_save = np.arange(10)
array_to_save
```

Out[54]:

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

In [55]:

```
# to save array in binary format in your pc
np.save('array_saved_binary',array_to_save)
#the array would be saved in a file by the name array_saved.npy
```

In [56]:

```
#to load that saved array
np.load('array_saved_binary.npy')
```

Out[56]:

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

In [57]:

```
#to save the array as a text file
array_to_save = np.arange(10)
np.savetxt('array_saved_text.txt',array_to_save,delimiter=',') #delimiter is used to sepa
rate values.
```

In [58]:

```
# we can also save the file in a zip format using the .savez() function.
# It's left for your own research & learning !
```

In [59]:

```
###### -----end--------- ######
```

In [ ]:

```

```

In [ ]:

```

```