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([0.13469489, 0.81269666, 0.52306992, 0.67440155, 0.4396114 ,
       0.59635384, 0.67035415, 0.4508506 , 0.84896729, 0.49539467])
```

In [14]:

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

Out[14]:

```
array([[0.20760403, 0.81628917, 0.8673073 , 0.64306309, 0.78922247],
       [0.68908026, 0.00441375, 0.52462046, 0.73671216, 0.30200383],
       [0.05892304, 0.62938614, 0.73330768, 0.00855971, 0.93451878],
       [0.45280638, 0.19421669, 0.98633189, 0.3554056 , 0.98081162]])
```

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 c
an possibly exist
```

Out[15]:

```
array([ 0.02239357, -0.36543475, -1.11612562, -0.95434795])
```

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]:

```
6
```

In [18]:

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

Out[18]:

```
array([ 8,  6, 17, 13, 12, 13,  7, 12,  5, 16])
```

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([52, 87, 18, 67, 98, 72, 70, 66, 41, 24, 79, 35, 28,  8, 26, 80, 70,
       99, 43, 32])
```

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([[52, 87, 18, 67, 98],
       [72, 70, 66, 41, 24],
       [79, 35, 28,  8, 26],
       [80, 70, 99, 43, 32]])
```

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 [25]:

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

Out[25]:

```
array([[52, 72, 79, 80],
       [87, 70, 35, 70],
       [18, 66, 28, 99],
       [67, 41,  8, 43],
       [98, 24, 26, 32]])
```

numpy array operations and mathematical functions

In [26]:

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

Out[26]:

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

In [27]:

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

Out[27]:

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

In [28]:

```
# 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 [29]:

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

Out[29]:

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

In [30]:

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

Out[30]:

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

In [31]:

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

In [32]:

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

Out[32]:

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

In [33]:

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

Out[33]:

```
9
```

In [34]:

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

Out[34]:

```
8
```

In [35]:

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

Out[35]:

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

In [36]:

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

Out[36]:

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

In [37]:

```
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[37]:

```
array([[-0.32054694, -1.3981901 , -0.51937189],
       [-0.70653977, -2.11046186,  0.83545895],
       [ 0.91883608,  1.64084669,  0.17252355]])
```

In [38]:

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

Out[38]:

```
array([[-0.321, -1.398, -0.519],
       [-0.707, -2.11 ,  0.835],
       [ 0.919,  1.641,  0.173]])
```

indexing in numpy arrays

In [39]:

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

Out[39]:

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

In [40]:

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

Out[40]:

```
5
```

In [41]:

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

Out[41]:

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

In [42]:

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

Out[42]:

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

In [43]:

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

Out[43]:

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

In [44]:

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

Out[44]:

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

In [45]:

```
# 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 [46]:

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

Out[46]:

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

In [47]:

```
# in this array
```

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

Out[47]:

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

selection in numpy arrays

In [48]:

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

Out[48]:

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

In [49]:

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

Out[49]:

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

In [50]:

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

Out[50]:

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

In [51]:

```
#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 [52]:

```
#consider reshape the array
array = np.arange(1,10)
array[3]= 1000
array
```

Out[52]:

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

In [53]:

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

Out[53]:

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

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 separ
ate values.
```

In [46]:

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

In [47]:

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

In [ ]:

In [ ]: