NUMPY:

Introduction to NumPy

NumPy's website www.numpy.org says that,

NumPy is the fundamental package for scientific computing with Python. The following are the benefits of Numpy,

• a powerful N-dimensional array object known as ndarray

•sophisticated ndarray creation and manipulation functions

• useful linear algebra, Fourier transform, and random number functions.

Ndarray:

Ndarray (or N-d array) is a multidimensional data structure in Numpy.It is the most important data structure in Scientific Python because all other libraries and data structures in Scientific Python stack use Numpy ndarrays in some form or the other to represent data.All the items in ndarray have same size and type. Just like the other containers in Python, ndarray can be accessed by indexing and can also be sliced.

Installation of NumPy :
To install on Windows, run the following commands in the command prompt,

pip3 install numpy

NumPy comes with Raspbian OS. We just need to update it with the following command in the terminal,

sudo pip3 install--upgrade numpy

In order to install Numpy on Linux in general, use the following command in terminal, sudo pip3 install numpy
sudo pip3 install numpy

Getting Started with NumPy Programming:

Open the command prompt and run the Jupyter with the following command,

jupyter notebook

Save the notebook with a filename of your choice. Type in and run the following statement to import Numpy library for the current sessions of the notebook,

import numpy as np

The following statements creates a one-dimensional ndarray,
x = np.array([1, 2, 31], np. int16)

In the code above, the function call creates ndarray with the passed arguments. The first argument is a list which has the data and the second argument is the data type of the individual elements of the ndarray.

Numpy has lot of data types:
some of these are

np.bool	np.int_	np.longdouble	np.int64	np.float32
np.byte	np.uint	np.single	np.uint8	np.float64
np.ubyte	np.longlong	np.double	np.uint16	np.complex32
np.short	np.ulonglong	np.clongdouble	np.uint32	np.complex64
np.ushort	np.half	np.int8	np.uint64	np.float_
np.intc	np.single	np.int16	np.intp	np.complex_
np.uintc	np.double	np.int32	np.uintp	np.float16

Run the following statements,

print (x)

print (type (x))

The first statement will print the ndarray. Second statement will print the type of the object. Following is the output.

```
[1 2 3]
<class ‘numpy.ndarray'>
```

Ndarrays follow C style indexing where the first element is stored at position 0 and nº element is stored at position n-1. Access individual members of the ndarray like below,

```
print( [x0] )
print( [x1] )
print( [x2] )
```

The following statement prints the last element in the ndarray.

```
print ( x[-1] )
```
when the index exceeds n-1 then the Python interpreter throws an error.

```
print (x[3])
```

The above statements raise the following exception.

```
Indexerror: index 3 in out of bounds for axis 0 with size 3
```
Creating two dimensional ndarray as follows,
```
x=np.array([1,2,3],[4,5,6],np.int16)
print(x)
```
This creates a two dimensional array as follows
```
[1 2 3]
[4 5 6]
```
To access individual elements of ndarray as follows,
```
print(x[0,0])
print(x[0,1])
print(x[0,2])
```
To access entire columns by slicing the ndarray as follows,
```
print(x[:,0])
print(x[:,1])
print(x[:,2])
```
Following are respective outputs of the statements above,
```
[1 4]
[2 5]
[3 6]
```
To access entire rows by slicing the ndarray as follows,
```
print( x[0 , :] )
print( x[1 , :] )
```
Following are respective outputs of the statements
```
[1 2 3]
[4 5 6]
```
To create three dimensional ndarray as follows,
```
x = np.array( [ [  [ 1 2 3] , [4 5 6]  ], [ [0 -1 -2],[-3,-4,-5] ] ],np.int16)
print(x)
```
To access individual members of three dimensional arrays as follows
```
print( x[0,0,0 ] )
print(x[1,1,2 ] )
```
To slice the three dimensional arrays,
```
print( x[:,1,1] )
print(x[ : ,:,1] )
```
Ndarray properties:
```
x = np.array( [ [  [ 1 2 3] , [4 5 6]  ], [ [0 -1 -2],[-3,-4,-5] ] ],np.int16)
print(x)
```
It creates the following three-dimensional array,

```
[ [ [ 1 2 3]
    [ 4 5 61]
  [[ 0 -1 -2]
   [-3 -4 -5] ] ]
```

The following code prints the properties of NumPy ndarrays,

```
print (x.shape)
 print (x.ndim))
print (x.dtype)
print (x.size)
print (x.nbytes)
```

The output is as follows,

```
(2, 2, 3)

3

int16

12
 24
```

The shape property returns the shape of the ndarray, ndim returns the dimensions of the array, dtype refers to the data type of the individual members of the ndarray (and not the data type of the darray object itself). size property returns the number of elements in the ndarray. And nbytes returns the size of the ndarray in the bytes in the memory, We can also compute the transpose of the ndarray with the following property.

```
print (x.T)
```

Ndarray Constants

We can represent a few important abstract constants like positive and negative zero and infinity and NAN (not a number) as follows,

```
print (np.inf)
print (np. NAN)
print (np.NINF)
print (np. NZERO)
print (np. PZERO)
```

The following constants are the important scientific constants,

```
print (np.e)
print (np.euler_gamma)
print (np.pi)
```

URL for the complete list of such constants.
https://docs.scipy.org/doc/numpy/reference/constants.html

Ndarray Creation Routines

few ndarray creation routines.
The first function is empty() that creates an empty ndarray.

```
x = np.empty ( [3, 3] , np. uint8)
```

The code above creates an empty ndarray. While creating, it is not assigned any values. So, it will have random values assigned to elements.
creating diagonal matrices of various sizes using eye () function,

```
y = np.eye (5, dtype=np.uint8)

print (y)
```

Following is the output,

```
[ [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]  ]
```

We can also change the position of the diagonal as follows,

```
y = np.eye (5, dtype=np. uint8, k=1)
```

To create ndarrays where all the elements are 1 as follows,

```
 x= np.ones ( (2, 5, 5), dtype=np.int16)
print (x)
```

Similarly, to create ndarray where all the elements are zero as follows,

```
x = np.zeros((2, 5, 5, 2), dtype=np.int16)
print (x)
```

To create ndarray and populate all its elements with a single value,

```
 x-=np. full ((3, 3, 3), dtype=np. int16, fill_value <= 5)
print (x)
```

For creating triangular matrices with tri () function,

```
x=np. tri (3, 3, k=0, dtype=np. uint16)
print (x)
```

The output is as follows,

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

Other examples are,

```
x np.tri (5, 5, k=1, dtype=np. uint16)
print (x)

x = np.tri (5, 5, k=-1, dtype=np. uint16)

print (x)
```

We can explicitly create lower triangular matrix as follows,

```
x = np.ones ((5, 5), dtype=np. uint8)
 y = np. tril (x, k=-1)

print (y)
```

The output is as follows,

```
[1 0 0 0 0]

[1 1 0 0 0]

[1 1 1 0 0]

[1 1 1 1 0]]
```

The code for upper triangular matrix is as follows,

```
x = np.ones ((5, 5), dtype=np. uint8)

y = np.triu(x, k=0)

print (y)

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

Other examples are:

```
x = np.ones ((5, 5), dtype=np. uint8)
 y = Tp.triu(x, k=-1)
print (y)

x = np.ones ( (5, 5), dtype=np. uint8) Y = np,triu(x, k=1)
 y = Tp.triu(x, k=1)
print (y)
```