What is NumPy?

- **NumPy** stands for **numerical python** which is a **python package** for the computation and processing of the **multidimensional and single dimensional array elements.**
- **Travis Oliphant created NumPy package in 2005** by injecting the features of the **ancestor module Numeric** into another module **Numarray.**
- **Numeric, the ancestor of NumPy, was developed by Jim Hugunin.**

How to install numpy?

pip install numpy

How to use numpy?

import numpy

Checking NumPy Version

The version string is stored under __version__ attribute.

eg.;

```
import numpy
print(numpy.__version__)
```

How to create an n-darray in numpy?

The array object in NumPy is called ndarray.

We can create a NumPy ndarray object by using the array() method.

E.g.;

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)#[1,2,3,4,5]
print(type(arr))#<class 'numpy.ndarray'>
```

To create an ndarray, we can pass a **list, tuple** into the array() method, and it will be converted into an ndarray.

eg.,;

```
import numpy as np
arr = np.array((1, 2, 3, 4, 5))
print(arr)#[1,2,3,4,5]
```

Array Dimensions:

a)0-D arrays:

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

Eg.,;

```
import numpy as np
arr = np.array(42)
print(arr)
```

b)1-D arrays:

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

c)2-D arrays:

An array that has 1-D arrays as its elements is called a 2-D array.

E.g.;

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
```

d)3-D arrays:

An array that has 2-D arrays (matrices) as its elements is called 3-D array.

```
import numpy as np
arr = np.array(
[
[[1, 2, 3],
 [4, 5, 6]],
[[1, 2, 3],
 [4, 5, 6]]
])
print(arr)
```

To check the number of dimensions:
use ndim attribute
Eg.;

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

Higher dimension array

(Convert one dimension of a n array into another higher dimension)
When the array is created, you can define the number of dimensions by using the ndmin argument.
Create an array with 5 dimensions and verify that it has 5 dimensions:

E.g.;

```
import numpy as np
arr = np.array([1, 2, 3, 4], ndmin=5)
print(arr)
print('number of dimensions :', arr.ndim)
```

Numpy Array Indexing
Access Array Elements

Array indexing is the same as accessing an array element.
The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.
Ex:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[0])
```

Access 2-D Arrays

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

Example
Access the element on the first row, second column:
Ex:
```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('2nd element on 1st row: ', arr[0, 1])
```

Access 3-D Arrays
To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

Example
Access the third element of the second array of the first array:
Ex:
```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr[0, 1, 2])
```

Slicing arrays
Slicing in python means taking elements from one given index to another given index.
We pass slice instead of index like this: [start:end].
We can also define the step, like this: [start:end:step].
If we don't pass start its considered 0
If we don't pass end its considered length of array in that dimension
If we don't pass step its considered 1

Example
Slice elements from index 1 to index 5 from the following array:
Ex:
```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])
```

Example
From the second element, slice elements from index 1 to index 4 (not included):
Ex:
```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[1, 1:4])
```

Ex:2

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[0:2, 1:4])
```

Shape of an Array

The shape of an array is the number of elements in each dimension.

Get the Shape of an Array

NumPy arrays have an attribute called **shape** that returns a tuple with each index having the number of corresponding elements.

Example:

Print the shape of a 2-D array:

```
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(arr.shape)#(2,4)
```

The example above returns (2, 4), which means that the array has 2 dimensions, where the first dimension has 2 elements and the second has 4.

Ex:

Create an array with 5 dimensions using ndmin using a vector with values 1,2,3,4 and verify that last dimension has value 4:

```
import numpy as np
arr = np.array([1, 2, 3, 4], ndmin=5)
print(arr)
print('shape of array :', arr.shape)#(1,1,1,1,4)
```

Reshaping arrays

Reshaping means changing the shape of an array.
The shape of an array is the number of elements in each dimension.
By reshaping we can add or remove dimensions or change number of elements in each dimension.

Reshape From 1-D to 2-D

Example:

Convert the following 1-D array with 12 elements into a 2-D array.
The outermost dimension will have 4 arrays, each with 3 elements:

Ex:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
```

```
newarr = arr.reshape(4, 3)
print(newarr)
```

Reshape From 1-D to 3-D
Example:
Convert the following 1-D array with 12 elements into a 3-D array.
The outermost dimension will have 2 arrays that contains 3 arrays, each with 2 elements:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(2, 3, 2)
print(newarr)
```

size attribute: returns the total number of elements present in a numpy array.
maximum(arr1,arr2): Create a array consist of maximum element based on comparison performed on both given arrays.
Ex:

```
import numpy as np
a=np.array([1,2,3,4])
b=np.array([6,8,1,2])
c=np.maximum(a,b)
print(c)#[6,8,3,4]
```

minimum(arr1,arr2): Create a array consistingof minimum element based on comparison performed on both given arrays.
Ex:

```
import numpy as np
a=np.array([1,2,3,4])
b=np.array([6,8,1,2])
c=np.minimum(a,b)
print(c)#[1,2,1,2]
```

np.arange(4): form an array of consisting element from 0 to 3.
To create numpy consisting of zero's element we use **zeros()** method of numpy.
Ex:

```
import numpy as np
array_1d = np.zeros(3)
print(array_1d)
```

Output:

```
[0. 0. 0.]
```

Ex:

```
import numpy as np
array_2d = np.zeros((2, 3))
print(array_2d)
```

Output:

```
[[0. 0. 0.]
 [0. 0. 0.]]
```