NumPy in Python Programming

Hrishikesh Bhaumik
Associate Professor
Department of Information Technology
RCC Institute of Information Technology

What is NumPy ?

NumPy is a **Python package**. It stands for '**Numerical Python**'.

It is a library consisting of **multidimensional array objects** and a **collection of routines** for processing of arrays.

Numeric, the ancestor of NumPy, was developed by Jim Hugunin.
Another package Numarray was also developed, having some additional functionalities.

In 2005, Travis Oliphant created NumPy package by incorporating the features of Numarray into Numeric package. There are many contributors to this open source project.

Operations using NumPy

Using **NumPy**, a developer can perform the following operations:

Mathematical and **logical** operations on arrays.

Fourier transforms and **routines** for shape manipulation.

Operations related to **linear algebra**.
NumPy has in-built functions for linear algebra and random number generation.

Importing the NumPy module

There are several ways to import NumPy. The standard approach is to use a simple import statement:

```
>>> import numpy
```

However, for large amounts of calls to NumPy functions, it can become tedious to write numpy.X over and over again. Instead, it is common to import under the **alias name** np:

```
>>> import numpy as np
```

Arrays

The central feature of NumPy is the ***array*** object class.

Arrays are similar to lists in Python, except that every element of an array must be of the same type.

Arrays make operations with large amounts of numeric data very fast and are generally much more efficient than lists.

An array can be created from a list:

```
>>> a = np.array([1, 4, 5, 8], float)
>>> a
array([ 1., 4., 5., 8.])
>>> type(a)
<type 'numpy.ndarray'>
```

Array Slicing

```
>>> a[0:2]
array([ 1., 4.])

>>> a[3]
8.0

>>> a[0] = 5.
>>> a
array([ 5., 4., 5., 8.])
```

Declaring Multi-Dimension Arrays

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a
array([[ 1., 2., 3.],
[ 4., 5., 6.]])

>>> a[0,0]
1.0
>>> a[0,1]
2.0
```

Slicing Multi-Dimension Arrays

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a[1,:]
array([ 4., 5., 6.])

>>> a[:,2]
array([ 3., 6.])

>>> a[-1:,-2:]
array([[ 5., 6.]])
```

shape and dtype Property

```
>>> a.shape
(2, 3)

>>> a.dtype
dtype('float64')
```

len function and in statement

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> len(a)
2

>>> 2 in a
True

>>> 0 in a
False
```

itemsize attribute and empty function

```
>>> x = np.array([1,2,3,4,5], dtype=np.float32)
>>>x.itemsize
4

>>> x = np.empty([3,2], dtype=int)
>>> x
```

Reshaping Arrays

```
>>> a = np.array(range(10), float)
>>> a
array([ 0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
>>> a = a.reshape((5, 2))
>>> a
array([[ 0., 1.],
[ 2., 3.],
[ 4., 5.],
[ 6., 7.],
[ 8., 9.]])
>>> a.shape
(5, 2)
```

copy function

```
>>> a = np.array([1, 2, 3], float)
>>> b = a
>>> c = a.copy()
>>> a[0] = 0
>>> a
array([0., 2., 3.])
>>> b
array([0., 2., 3.])
>>> c
array([1., 2., 3.])
```

Lists from arrays

```
>>> a = np.array([1, 2, 3], float)
>>> a.tolist()
[1.0, 2.0, 3.0]
>>> list(a)
[1.0, 2.0, 3.0]
```

Filling an array with a single value

```
>>> a = array([1, 2, 3], float)
>>> a
array([ 1., 2., 3.])
>>> a.fill(5)
>>> a
array([ 5., 5., 5.])
```

Transposing an array

```
>>> a = np.array(range(6), float).reshape((2, 3))
>>> a
array([[ 0., 1., 2.],
[ 3., 4., 5.]])

>>> a.transpose()  or  a.T
array([[ 0., 3.],
[ 1., 4.],
[ 2., 5.]])
```

Multi-dimensional to One-dimensional array

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a
array([[ 1., 2., 3.],
[ 4., 5., 6.]])

>>> a.flatten()
array([ 1., 2., 3., 4., 5., 6.])
```

concatenate function

```
>>> a = np.array([[1, 2], [3, 4]], float)
>>> b = np.array([[5, 6], [7,8]], float)
>>> np.concatenate((a,b))
array([[ 1., 2.],
[ 3., 4.],
[ 5., 6.],
[ 7., 8.]])
>>> np.concatenate((a,b), axis=0) ← vertical concat
array([[ 1., 2.],
[ 3., 4.],
[ 5., 6.],
[ 7., 8.]])
>>> np.concatenate((a,b), axis=1) ← horizontal concat
array([[ 1., 2., 5., 6.],
[ 3., 4., 7., 8.]])
```

newaxis constant

```
>>> a = np.array([1, 2, 3], float)
>>> a
array([1., 2., 3.])
>>> a[:,np.newaxis]
array([[ 1.],
[ 2.],
[ 3.]])
>>> a[:,np.newaxis].shape
(3,1)
>>> a[np.newaxis,:]
array([[ 1., 2., 3.]])
>>> a[np.newaxis,:].shape
(1,3)
```

arange function

```
>>> np.arange(5, dtype=float)
array([ 0., 1., 2., 3., 4.])

>>> np.arange(1, 6, 2, dtype=int)
array([1, 3, 5])
```

zeros and ones functions

```
>>> np.ones((2,3), dtype=float)
array([[ 1., 1., 1.],
[ 1., 1., 1.]])

>>> np.zeros(7, dtype=int)
array([0, 0, 0, 0, 0, 0, 0])
```

zeros_like and ones_like functions

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)

>>> np.zeros_like(a)
array([[ 0., 0., 0.],
[ 0., 0., 0.]])

>>> np.ones_like(a)
array([[ 1., 1., 1.],
[ 1., 1., 1.]])
```

identity and eye functions

```
>>> np.identity(4, dtype=float)
array([[ 1., 0., 0., 0.],
[ 0., 1., 0., 0.],
[ 0., 0., 1., 0.],
[ 0., 0., 0., 1.]])
```

The eye function returns matrices with ones along the kth diagonal:

```
>>> np.eye(4, k=1, dtype=float)
array([[ 0., 1., 0., 0.],
[ 0., 0., 1., 0.],
[ 0., 0., 0., 1.],
[ 0., 0., 0., 0.]])
```

When k=0, it signifies the principal diagonal

asarray function

This function is can be used to create an array from existing data such as list, list of tuples, tuples, tuple of tuples or tuple of lists

```
>>> x = [1,2,3]
>>> a = np.asarray(x)
array([1 2 3])
```

linspace function

```
>>> x = np.linspace(10,20,5)
>>>x
array([10.  12.5  15.  17.5  20.])

>>> x = np.linspace(10,20, 5, endpoint=False)
>>>x
array([10. 12. 14. 16. 18.]
```

logspace function

```
>>> a = np.logspace(1,10,num=10, base=2)
>>>a
array([ 2. 4. 8. 16. 32. 64. 128. 256. 512. 1024.])
```

slice function

```
slice(start, stop, step)
>>> a = np.arange(10)
>>> s = slice(2,7,2)
>>> print(a[s])
```

Slicing using start and end index

```
>>> a = np.arange(10)
>>> print(a[2:])
[2,3,4,5,6,7,8,9]
>>> print(a[:4])
[0,1,2,3]
>>> print(a[2:5])
[2,3,4]
```