Numpy Tutorial

In this Numpy Tutorial, we will learn how to install numpy library in python, numpy multidimensional arrays, numpy datatypes, numpy mathematical operation on these multidimensional arrays, and different functionalities of Numpy library.

What is Numpy?

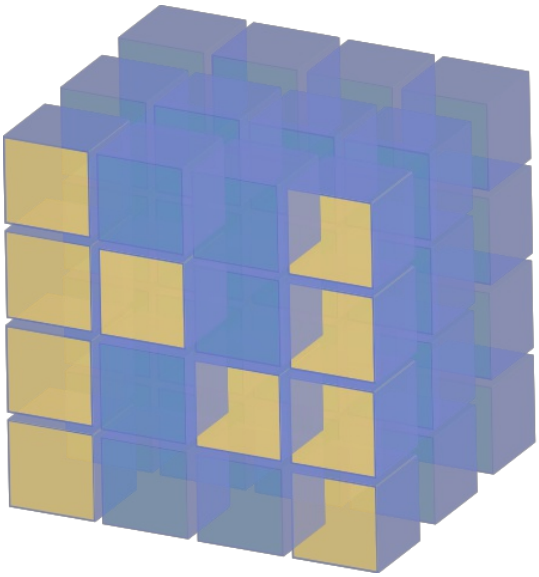
Numpy is a Python library that supports multi-dimensional arrays and matrix. It also provides many basic and high-level mathematical functions that can be applied on these multi-dimensional arrays and matrices with less code footprint.

Why Numpy?

There are many reasons why Numpy package has been used by data scientist and analysts, machine learning experts, deep learning libraries, etc. We will go through some of the most basic advantages of Numpy over regular lists or arrays in Python.

- The code that involves arrays with Numpy package is precise to apply transformations or operations for each element of the multidimensional arrays unlike a Python List.
- Since n-dimensional arrays of Numpy use a single datatype and contiguous memory for storage, they take relatively lesser memory read and write times.
- The most useful features of Numpy package is the compact datatypes that it offers, like unsigned integers of 8 bits, 16 bits size and signed integers of different bit sizes, different floating point precisions, etc.

Numpy Tutorial



NumPy

Install Numpy

To install Numpy and all the dependencies, use pip and run the following command.

Assuming that pip is installed in your computer, open command prompt or terminal and run the following command.

```
python -m pip install --user numpy scipy
```

```
python -m pip install --user numpy scipy matplotlib ipython jupyter pandas sympy nose
```

In command prompt

```
C:\>python -m pip install --user numpy
```

```
C:\>python -m pip install --user numpy scipy matplotlib ipython jupyter pandas sympy nose
Collecting numpy
  Downloading https://files.pythonhosted.org/packages/94/b5/f4bdf7bce5f8b35a2a83a0b70c545ca061a50b54724b5287505064906b14/numpy-1.16.0-cp37-cp37m-win32.whl (10.0MB)
    100% |????????????????????????????????| 10.0MB 765kB/s
Collecting scipy
  Downloading https://files.pythonhosted.org/packages/88/f2/7f16c94e22c9714b0dc417bdaa7d6eb9ec9f90fd5d5c6221769f73b3f5dd/scipy-1.2.0-cp37-cp37m-win32.whl (26.8MB)
    100% |????????????????????????????????| 26.8MB 471kB/s
Collecting matplotlib
  Downloading https://files.pythonhosted.org/packages/3f/16/4500e22ea8d11f4946bd902695d0113f82a0aaca45f352478f157ca6623d/matplotl
3.0.2-cp37-cp37m-win32.whl (8.7MB)
    100% |????????????????????????????????| 8.7MB 3.4MB/s
Collecting ipython
  Downloading https://files.pythonhosted.org/packages/f0/b4/a9ea018c73a84ee6280b2e94a1a6af8d63e45903eac2da0640fa63bca4db/ipython
7.2.0-py3-none-any.whl (765kB)
    100% |????????????????????????????????| 768kB 1.3MB/s
Collecting jupyter
```

All the libraries and their dependencies will be downloaded and installed one by one.

At the end of the command prompt output, you would see the following text.

```
Successfully installed MarkupSafe-1.1.0
```

```
Successfully installed
MarkupSafe-1.1.0 Send2Trash-1.5.0 backcall-0.1.0 bleach-3.1.0 colorama-0.4.1 cycler-0.10.0 decorator-4.3.2 defusedxml-0.5
```

Check Numpy Installation

To check if numpy is installed or not, open Python terminal and run the following commands.

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

The import statement imports the numpy library, while the print statement prints Numpy version installed.

Command Prompt Output

```
C:\>python
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> print(numpy.__version__)
1.16.0
>>>
```

Numpy is installed and the same is verified by printing Numpy's version number.

Update Numpy

You can update numpy with pip using the following command.

```
pip install numpy --upgrade
```

Numpy Datatypes

Choosing a numpy datatype for your multidimensional array is very important. If an appropriate datatype is chosen based on the requirement and data characteristics, then it can ease a large amount of memory and ofcourse the array operations can be faster.

Data type	Description
bool_	Boolean (True or False) stored as a byte
int_	Default integer type (same as C long; normally either int64 or int32)
intc	Identical to C int (normally int32 or int64)
intp	Integer used for indexing (same as C ssize_t; normally either int32 or int64)
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)
int32	Integer (-2147483648 to 2147483647)
int64	Integer (-9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)
float_	Shorthand for float64.
float16	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
complex_	Shorthand for complex128.
complex64	Complex number, represented by two 32-bit floats (real and imaginary components)
complex128	Complex number, represented by two 64-bit floats (real and imaginary components)

In this Numpy Tutorial, we will use some of these datatypes, and whenever appropriate, we shall explain why a particular datatype is selected.

Import Numpy

We have already used `import numpy` statement while verifying the installation of numpy package using pip. This import is like any python package import. To use the functions of numpy, the package has to be imported at the start of the program.

The syntax of importing numpy package is:

```
import numpy
```

```
import numpy
```

Python community usally uses the numpy package with an alias `np`.

```
import numpy as np
```


```
import numpy as np
```


Now, you can use `np` to call all numpy functions. Going further, we will use this numpy alias version `np` in code for `numpy`.

Create a Basic One-dimensional Numpy Array

There are many ways to create an array using numpy. We go through each one of them with examples.

1. **array()**

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


```
>>> import numpy as np
>>> a = np.array([5, 8, 12])
>>> a
array([ 5, 8, 12])
```


np.array() function accepts a list and creates a numpy array.

2. **arange()** Note that its not arrange but **a range**. numpy.arange function acceps the starting and ending elements of a range, followed by the interval.

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


```
>>> import numpy as np
>>> a = np.arange(1, 15, 2)
>>> a
array([ 1, 3, 5, 7, 9, 11, 13])
```


In the above example, `1` is the starting of the range and `15` is the ending but `15` is excluded. The interval is `2` and therefore the interval between adjacent elements of the array is `2`.

3. **linspace()** This function creates a floating point array with linearly spaced values. numpy.linspace() function accepts starting and ending elements of the array, followed by number of elements.

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

```
>>> import numpy as np
>>> a = np.linspace(1, 15, 7)
>>> a
array([ 1. , 3.33333333, 5.66666667, 8. , 10.33333333, 12.66666667, 15. ])
```

In the above example, `1` is the starting, `15` is the ending and `7` is the number of elements in the array.

Create Two Dimensional Numpy Array

In the previous section, we have learned to create a one dimensional array. Now we will take a step forward and learn how to reshape this one dimensional array to a two dimensional array.

numpy.reshape() is the method used to reshape an array. `reshape()` function takes shape or dimension of the target array as the argument. In the following example the shape of target array is `(3, 2)`. As we are creating a 2D array, we provided only two values in the shape. You can provide multiple dimensions as required in the shape, separated by comma.

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

```
>>> import numpy as np
>>> a = np.array([8, 2, 3, 7, 9, 1])
>>> a
array([8, 2, 3, 7, 9, 1])
>>> a = a.reshape(3, 2)
>>> a
array([[8, 2],
       [3, 7],
       [9, 1]])
>>>
```

The product of number of rows and number of columns should equal the size of the array. If the product of dimensions and the size of the array do not match, you will get an error as shown below:

```
>>> a.reshape(2, 4)
Traceback (most
```

```
>>> a.reshape(2, 4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: cannot reshape array of size 6 into shape (2,4)
```

Now the array `a` is of shape [3,2]. You can reshape this to [2,3] or [1,6] by simply calling the reshape function on the array `a`.

```
>>> a = a.reshape(3,
2)
```

```
>>> a = a.reshape(3, 2)
>>> a
array([[8, 2],
       [3, 7],
       [9, 1]])
>>> a = a.reshape(2, 3)
>>> a
array([[8, 2, 3],
       [7, 9, 1]])
>>> a = a.reshape(1, 6)
>>> a
array([[8, 2, 3, 7, 9, 1]])
>>>
```

In this Numpy Tutorial, we will go through some of the functions numpy provide to create and empty N-Dimensional array and initialize it zeroes, ones or some random values.

Create Numpy Array with all zeros

If you would like to create a numpy array of a specific size with all elements initialized to zero, you can use `zeros()` function. The zeros() function takes the shape of the array as argument.

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

```
>>> import numpy as np
>>> a = np.zeros((4, 5))
>>> a
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
>>>
```

The default datatype of each element in the numpy array of zeros is numpy.float64.

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

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

You can change the datatype of the elements by providing `dtype` argument to the zeros function. Let us change to the datatype `numpy.int16`.

```
>>> a = np.zeros((4,
5), np.int16)
```

```
>>> a = np.zeros((4, 5), np.int16)
>>> a
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]], dtype=int16)
```

Observe that the datatype int16 belongs to numpy package, hence `np.int16`.

Create Numpy Array with all ones

Similar to zeros() function, we have ones() function in numpy package. It creates ones with the specified array shape.

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

```
>>> import numpy as np
>>> a = np.ones((3,7))
>>> a
array([[1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1.]])
>>> a.dtype
dtype('float64')
>>>
```

The default datatype for each element of the array is numpy.float64. You can change this to another numpy datatype, by providing the `dtype` argument.

```
>>> a = np.ones((3,7),
dtype=np.uint8)
```

```
>>> a = np.ones((3,7), dtype=np.uint8)
>>> a
array([[1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1]], dtype=uint8)
>>> a.dtype
dtype('uint8')
>>>
```

The datatype of the array created in the above example is `numpy.uint8`.

Create Numpy Array with random values

You can create an array with random values and specific dimensions using random function.

```
>>> a =
```

```
>>> a = np.random.random((3,2))
>>> a
array([[0.08832623, 0.80635251],
       [0.28991211, 0.9976203 ],
       [0.5372018 , 0.53011979]])
>>> a.dtype
dtype('float64')
>>>
```

In the above example, there are two `random` keywords. The first `random` is the sub-package of numpy, while second `random` is the function. There are other functions like randint(), etc., used to create array with random integers picked from a specific range.

```
>>> a =
```

```
>>> a = np.random.randint(0,8,12)
>>> a
array([5, 3, 4, 5, 1, 7, 3, 6, 5, 5, 2, 6])
>>>
```

Numpy Array Size

Array size is independent of its shape or dimensions. `size` pointer returns the total number of elements in the array.

```
>>> a
array([[8, 2]
```

```
>>> a
array([[8, 2],
       [3, 7],
       [9, 1]])
>>> a.size
6
>>>
```

The returned value is of type `int`.

Numpy Array Shape

You can get the shape or dimensions of the array using shape pointer to the array.

```
>>> a = np.zeros((4,
5), np.int16)
```

```
>>> a = np.zeros((4, 5), np.int16)
>>> a
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]], dtype=int16)
>>> a.shape
(4, 5)
>>>
```

The example holds good for any dimensional array. Let us create a four dimensional array and find its shape.

```
>>> a = np.zeros((4,
5, 2, 2), np.int16)
```

```
>>> a = np.zeros((4, 5, 2, 2), np.int16)
>>> a
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],
  [0, 0]],

 [[0, 0],
```

```
        [0, 0]],

       [[0, 0],
        [0, 0]]]], dtype=int16)
>>> a.shape
(4, 5, 2, 2)
>>>
```

Individual values of the shape can be accessed using index.

```
>>> a.shape
(4, 5, 2, 2)
```

```
>>> a.shape
(4, 5, 2, 2)
>>> a.shape[1]
5
>>> a.shape[2]
2
```

Numpy Array level Relational Operators

In this section, you can glimpse the power of Numpy. You can apply relational operators to the whole array in a single statement.

In the following example, we have an array `a`, and we will check if each element of the array is greater than `4`.

```
>>> a =
np.random.randint(1,8
```

```
>>> a = np.random.randint(1,8,20).reshape(4,5)
>>> a
array([[1, 5, 5, 7, 7],
       [5, 5, 5, 6, 4],
       [4, 5, 2, 2, 1],
       [6, 6, 7, 4, 6]])
>>> b = a<5
>>> b
array([[ True, False, False, False, False],
       [False, False, False, False, True],
       [ True, False, True, True, True],
       [False, False, False, True, False]])
>>>
```

The output of `a<5` is another array with the same shape of `a`, but with the boolean values representing if the element is less than `5`.

Numpy Array level Arithmetic Operators

You can perform Arithmetic Operations as well at array level with ease and less code.

For example, if `a` is a numpy array, then

- a/4 divides all the elements of the array with 4 and returns the resulting array.
- a*3 multiplies all the elements of the array with 3 and returns the resulting array.

In the following example, we add 4 to each of the element in numpy array `a` using a single statement.

```
>>> a =
np.random.randint(1,8
```

```
>>> a = np.random.randint(1,8,20).reshape(4,5)
>>> a
array([[1, 4, 2, 7, 1],
       [6, 2, 3, 7, 3],
       [4, 7, 7, 5, 7],
       [1, 2, 4, 4, 6]])
>>> a+4
array([[ 5, 8, 6, 11, 5],
       [10, 6, 7, 11, 7],
       [ 8, 11, 11, 9, 11],
       [ 5, 6, 8, 8, 10]])
>>>
```

Note that the arithmetic operator returns the resulting array, but does not modify the actual array. To modify the actual array, you have to use the modifier operators like `+=`, `/=`, `*=`, etc.

```
>>> a =
np.random.randint(1,8
```

```
>>> a = np.random.randint(1,8,20).reshape(4,5)
>>> a
array([[1, 4, 2, 7, 1],
       [6, 2, 3, 7, 3],
       [4, 7, 7, 5, 7],
       [1, 2, 4, 4, 6]])
>>> a+4
array([[ 5, 8, 6, 11, 5],
       [10, 6, 7, 11, 7],
       [ 8, 11, 11, 9, 11],
       [ 5, 6, 8, 8, 10]])
>>> a
array([[1, 4, 2, 7, 1],
       [6, 2, 3, 7, 3],
       [4, 7, 7, 5, 7],
       [1, 2, 4, 4, 6]])
>>> a += 4
>>> a
array([[ 5, 8, 6, 11, 5],
       [10, 6, 7, 11, 7],
       [ 8, 11, 11, 9, 11],
       [ 5, 6, 8, 8, 10]])
>>>
```

Numpy Mathematical Functions

While introducing numpy to you, we have gone through the point that Numpy is created for Numerical Analysis in Python. Hence, you might expect that Numpy provides a huge collection of Mathematical Functions. And it is true. Numpy provides statistical functions, trigonometric functions, linear algebra functions, etc.

In this Numpy Tutorial, we will go through some of the basic mathematical functions provided by Numpy.

Numpy Sum Function – numpy.sum()

In this example, we will take an array and find the sum of the elements in it. numpy.sum() function not only allows us to calculate the sum of all elements in the array, but also along a specific axis as well.

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

```
>>> a = np.array([[1, 2], [3, 4]])
>>> np.sum(a)
10
>>> np.sum(a, axis=0)
array([4, 6])
>>> np.sum(a, axis=1)
array([3, 7])
```

Numpy Mean Function – numpy.mean()

In this example, we will take an array and find the mean. Mean is the average of elements of an array. numpy.mean() function not only allows us to calculate the mean of the complete array, but also along a specific axis as well.

```
>>> a = np.array([[1,
```

```
>>> a = np.array([[1, 2], [3, 4]])
>>> np.mean(a)
2.5
>>> np.mean(a, axis=0)
array([ 2., 3.])
>>> np.mean(a, axis=1)
array([ 1.5, 3.5])
```

List of all Numpy Functions

The list of all Numpy functions is vast and would make you feel that this Numpy Tutorial is on an infinite scroll. So, please refer this link: [https://docs.scipy.org/doc/numpy-1.15.0/genindex.html].

Python Example for some of the regularly used Numpy Functions

- numpy where()

Python Library Tutorials

- Python Numpy Tutorial
- Python SciPy Tutorial
- Python Matplotlib Tutorial
- Python 3D Tutorial
- Python GUI - tkinter Tutorial
- NLTK Tutorial