

Introduction to Numpy –

NumPy stands for numeric python which is a python package for the computation and processing of the multidimensional and single dimensional array elements. The manipulation includes mathematical and logical operations. Numpy provides an array data structure and helps in numerical analysis.

NumPy Array and List Difference

Numpy Array	Python List
It is the core Library of python which is used for scientific computing.	The core library of python provides list.
It can contain similar datatypes.	It Contains different types of datatypes.
We need to Numpy Library to access Numpy Arrays.	It is built-in function of python.
It is Homogeneous.	It is both homogeneous and heterogeneous.
In this Element wise operation is possible.	Element wise operation is not possible on the list.
By using <code>numpy.array()</code> we can create N-Dimensional array.	It is by default 1-dimensional. In some cases, we can create an N-Dimensional list. But it is a long process.
It requires smaller memory consumption as compared to Python List.	It requires more memory as compared to Numpy Array.
In this each item is stored in a sequential manner.	It stores item in random location of the memory.
It is faster as compared to list.	It is slow as compared to NumPy Array.

Installation of NumPy- +96

If you have [Python](#) and [PIP](#) already installed on a system, then installation of NumPy is very easy.

Install it using this command:

```
C:\Users\Your Name>pip install numpy
```

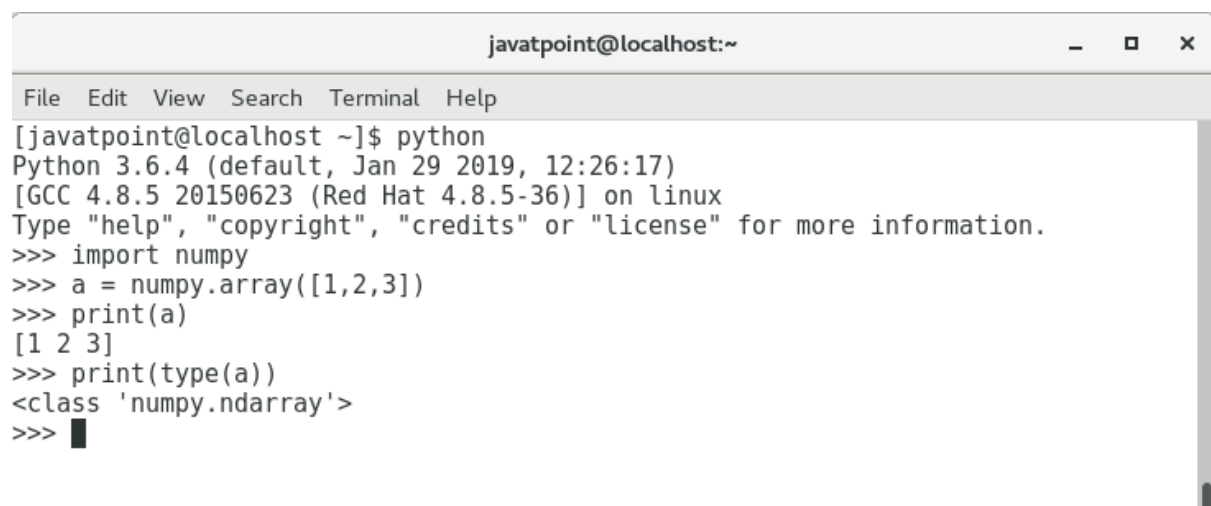
Array Creation in Numpy –

1) Using array()

NumPy is used to work with arrays. The array object in NumPy is called **ndarray**. We can create a NumPy **ndarray** object by using the **array()** function.

Arrays in Numpy can be created by multiple ways, with various number of Ranks, defining the size of the Array. Arrays can also be created with the use of various data types such as lists, tuples, etc.

Example 1-

A screenshot of a terminal window titled 'javatpoint@localhost:~'. The terminal shows the execution of a Python script. It starts with '[javatpoint@localhost ~]\$ python', followed by the Python version and GCC information. Then, it imports numpy, creates an array 'a' with values [1, 2, 3], prints the array, and prints its type, which is '<class 'numpy.ndarray'>'.

```
javatpoint@localhost:~  
File Edit View Search Terminal Help  
[javatpoint@localhost ~]$ python  
Python 3.6.4 (default, Jan 29 2019, 12:26:17)  
[GCC 4.8.5 20150623 (Red Hat 4.8.5-36)] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import numpy  
>>> a = numpy.array([1,2,3])  
>>> print(a)  
[1 2 3]  
>>> print(type(a))  
<class 'numpy.ndarray'>  
>>>
```

Example 2 -

```
import numpy as np  
# Creating a rank 1 Array  
arr = np.array([1, 2, 3])  
print("Array with Rank 1: \n",arr)  
# Creating a rank 2 Array  
arr = np.array([[1, 2, 3],  
                [4, 5, 6]])  
print("Array with Rank 2: \n", arr)  
# Creating an array from tuple  
arr = np.array((1, 3, 2))  
print("\nArray created using "  
      "passed tuple:\n", arr)
```

Output:

Array with Rank 1:

```
[1 2 3]
```

Array with Rank 2:

```
[[1 2 3]
```

```
[4 5 6]]
```

Array created using passed tuple:

```
[1 3 2]
```

2) Using `arange()` –

The `arange()` method creates an array with evenly spaced elements as per the interval.

Example

```
1) import numpy as np
2) # create an array with first five elements
3) array1 = np.arange(5)
4)
5) # create an array with elements from 5 to 10(exclusive)
6) array2 = np.arange(5, 10)
7)
8) # create an array with elements from 5 to 15 with stepsize 2
9) array3 = np.arange(5, 15, 2)
10)
11) print(array1)
12) print(array2)
13) print(array3)
```

Output

```
[0 1 2 3 4]
[5 6 7 8 9]
[ 5 7 9 11 13]
```

3) Using `linspace()` –

The `linspace()` method creates an array with evenly spaced elements over an interval.

linspace() Syntax

The syntax of `linspace()` is:

```
numpy.linspace(start, stop, num)
```

Example

```
1)import numpy as np
2)
3)# create an array with 3 elements between 5 and 10
4)array1 = np.linspace(5, 10, 3)
5)
6)
7)print(array1)
8)
9)# Output: [ 5.  7.5 10.]
```

4) Using random().rand()-

The `np.random.rand()` function is used to create an array of random numbers.

Let's see an example to create an array of **5** random numbers,

```
1)import numpy as np
2)
3) # generate an array of 5 random numbers
4) array1 = np.random.rand(5)
5) print(array1)
```

Output

```
[0.08455648 0.56379034 0.66463204 0.97608605 0.30700052]
```

Creating Predefined Arrays –

1) using zeros() –

The `np.zeros()` function allows us to create N-D arrays filled with all zeros. For example,

```
import numpy as np

# create 2D array with 2 rows and 3 columns filled with zeros
array1 = np.zeros((2, 3))
```

```
print("2-D Array: ")
print(array1)
```

Output

```
2-D Array:
[[0. 0. 0.]
 [0. 0. 0.]]
```

2) Using ones()-

The `ones()` method creates a new array of given shape and type, filled with ones.

```
import numpy as np

# create a float array of 1s
array1 = np.ones(5)
print('Float Array: ',array1)

# create an int array of 1s
array2 = np.ones(5, dtype = int)

print('Int Array: ',array2)
```

Output

```
Float Array: [1. 1. 1. 1. 1.]
Int Array: [1 1 1 1 1]
```

3) Using empty()

The `empty()` method creates a new array of given shape and type, without initializing entries.

Example- Create Array With empty()

```
import numpy as np

# create a float array of uninitialized entries
array1 = np.empty(5)

print('Float Array: ',array1)
```

Output

```
Float Array: [7.37149303e-317 0.00000000e+000 9.06092203e-312 2.47218893e-253
1.55872313e-307]
```

Creating arrays from another array –

1. <code>np.empty_like(x)</code>	Same as the x array but all the elements are initialised to some garbage numbers .	<pre>x=np.ones([2,2]) >> y=np.empty_like(x) >>> y Array([[0.0000000+00j,1.0000000+001j] [3.77775565-321j,3.4574334-345j]])</pre>
2. <code>np.zeros_like(x)</code>	Same as x but all the elements are initialised to 0 .	<pre>x=np.ones([2,2]) >> y=np.zeros_like(x) >>> y Array([[0...,0...] [0...,0...]])</pre>
3. <code>y=np.copy(k)</code>	Y is the copy of x but changing it does not affect the array k	<pre>>> y=np.copy(x) >> y Array([[1,1] [1,1]])</pre>

NumPy Array Attributes

In NumPy, attributes are properties of NumPy arrays that provide information about the array's shape, size, data type, dimension, and so on.

There are numerous attributes available in NumPy, which we'll learn below.

Common NumPy Attributes

Here are some of the commonly used NumPy attributes:

Let create a common array and apply each attributes one by one :

```
import numpy as np

# create a 2-D array
array1 = np.array([[2, 4, 6],
                   [1, 3, 5]])
```

Attributes	Description	Example
Ndim	returns number of dimension of the array	<pre>>>array1.ndim</pre> Output- 2
Size	returns number of elements in the array	<pre>>>array1.size</pre> 6
Dtype	returns data type of elements in the array	<pre>>>array1.dtype</pre> Int64
Shape	returns the size of the array in each dimension.	<pre>>>array1.shape</pre> (2,3)
Itemsize	returns the size (in bytes) of each elements in the array	<pre>>> array1.itemsize</pre> 8
reshape()	Used to rearrange the elements into specific dimensions.	<pre>>>array1.reshape(3,2) >>array1.reshape(-1)</pre> <pre>[[2 4] [6 1] [3 5]]</pre> <pre>[2,4,6,1,3,5]</pre>

Indexing and Slicing Operations-

In NumPy, each element in an array is associated with a number. The number is known as an **array index**.

Let's see an example to demonstrate NumPy array indexing.

	1	3	5	7	9
index →	0	1	2	3	4

We can use indices to access individual elements of a NumPy array.

Suppose we have a NumPy array:

```
array1 = np.array([1, 3, 5, 7, 9])
```

Now, we can use the index number to access array elements as:

- `array1[0]` - to access the first element, i.e. **1**
- `array1[2]` - to access the third element, i.e. **5**

NumPy Negative Array Indexing

NumPy allows negative indexing for its array. The index of **-1** refers to the last item, **-2** to the second last item and so on.

	1	3	5	7	9
index →	0	1	2	3	4
negative index →	-5	-4	-3	-2	-1

Modify Array Elements Using Index

We can use indices to change the value of an element in a NumPy array. For example,

```
import numpy as np
```



```
# create a numpy array
numbers = np.array([2, 4, 6, 8, 10])

numbers[0] = 12

numbers[-1] = 13

print("After modifying first element:", numbers) # prints [12 4 6 8 13]
```

2-D NumPy Array Indexing

Array indexing in NumPy allows us to access and manipulate elements in a 2-D array.

To access an element of `array1`, we need to specify the row index and column index of the element. Suppose we have following 2-D array,

```
array1 = np.array([[1, 3, 5],
                   [7, 9, 2],
                   [4, 6, 8]])
array1[2, 1] # returns 6
```

Access Row or Column of 2D Array Using Indexing

In NumPy, we can access specific rows or columns of a 2-D array using array indexing.

Let's see an example.

```
import numpy as np

# create a 2D array
array1 = np.array([[1, 3, 5],
                   [7, 9, 2],
                   [4, 6, 8]])

# access the second row of the array
second_row = array1[1, :]
print("Second Row:", second_row) # Output: [7 9 2]

# access the third column of the array
```

```
third_col = array1[:, 2]
print("Third Column:", third_col) # Output: [5 2 8]
```

Array Operations –

NumPy provides a wide range of operations that can perform on arrays.

1) Mathematical operation –

```
import numpy as np

x = np.array([1, 3, 5, 7])
y = np.array([2, 4, 6, 8])
array1=[[2,3] ,
        [4,6]]
num = 2
```

Command	Description	Output
np.add(x,y)	Returns the addition of two matrices x and y	[3 7 11 15]
np.subtract(x,y)	Returns the subtraction of two matrices x and y	[-1 -1 -1 -1]
np.multiply(x,y)	Returns the multiplication of two matrices	[2 12 30 56]
np.divide(x,y)	Returns the division of two matrices	[0.5 0.75 0.84 0.87]
np.power(x,num)	Returns the element-wise exponential operation.	[1 9 25 49]
np.mod(x,y)	Calculate the remainder of element-wise division between two matrices	[1 3 5 7]
np.sqrt(y)	Returns the element wise sqrt operation	[1.414 2 2.45 2.83]

2) Manipulation Functions –

Command	Description	Output
np.reshape(x, (2, 2))	Reshapes array1 into 2D array with shape(2,2)	<pre>[[1 3] [5 7]]</pre>
np.transpose(array1)	Transpose array1	<pre>[[2 4] [3 6]]</pre>

3) Statistical Functions –

Command	Description	Output
np.mean(x)	computes the arithmetic mean of the array elements along the specified axis.	4.0
np.median(y)	computes the arithmetic median of the array elements along the specified axis.	5.0
np.min(x)	Return minimum element from matrices x	1
np.max(x)	Return maximum element from matrixes x	7
Np.std(x)	computes the standard deviation of the array elements along the specified axis.	2.23