About the Tutorial

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.

This tutorial explains the basics of NumPy such as its architecture and environment. It also discusses the various array functions, types of indexing, etc. An introduction to Matplotlib is also provided. All this is explained with the help of examples for better understanding.

Audience

This tutorial has been prepared for those who want to learn about the basics and various functions of NumPy. It is specifically useful for algorithm developers. After completing this tutorial, you will find yourself at a moderate level of expertise from where you can take yourself to higher levels of expertise.

Prerequisites

You should have a basic understanding of computer programming terminologies. A basic understanding of Python and any of the programming languages is a plus.

Disclaimer & Copyright

© Copyright 2016 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com.
Table of Contents

About the Tutorial ........................................................................................................................................... i
Audience ......................................................................................................................................................... i
Prerequisites ..................................................................................................................................................... i
Disclaimer & Copyright ................................................................................................................................. i
Table of Contents ........................................................................................................................................... ii

1. NUMPY – INTRODUCTION ..................................................................................................................... 1

2. NUMPY – ENVIRONMENT ....................................................................................................................... 2

3. NUMPY – NDARRAY OBJECT .................................................................................................................. 4

4. NUMPY – DATA TYPES ........................................................................................................................... 7
   Data Type Objects (dtype) ............................................................................................................................. 8

5. NUMPY – ARRAY ATTRIBUTES ............................................................................................................. 12
   ndarray.shape ............................................................................................................................................ 12
   ndarray.ndim ............................................................................................................................................ 13
   numpy.itemsize .......................................................................................................................................... 14
   numpy.flags ............................................................................................................................................. 14

6. NUMPY – ARRAY CREATION ROUTINES ............................................................................................. 16
   numpy.empty ........................................................................................................................................... 16
   numpy.zeros .......................................................................................................................................... 17
   numpy.ones ............................................................................................................................................ 18

7. NUMPY – ARRAY FROM EXISTING DATA ............................................................................................. 19
   numpy.asarray ......................................................................................................................................... 19
   numpy.frombuffer ................................................................................................................................. 20
NumPy fromiter ........................................................................................................................................... 21

8. NUMPY – ARRAY FROM NUMERICAL RANGES ............................................................................... 23
   numpy.arange ...................................................................................................................................... 23
   numpy.linspace ............................................................................................................................... 24
   numpy.logspace ............................................................................................................................... 25

9. NUMPY – INDEXING & SLICING .................................................................................................... 27

10. NUMPY – ADVANCED INDEXING ................................................................................................. 31
    Integer Indexing ............................................................................................................................ 31
    Boolean Array Indexing .................................................................................................................. 33

11. NUMPY – BROADCASTING .......................................................................................................... 35

12. NUMPY – ITERATING OVER ARRAY .............................................................................................. 38
    Iteration Order ............................................................................................................................. 39
    Modifying Array Values ............................................................................................................... 42
    External Loop ............................................................................................................................... 42
    Broadcasting Iteration .................................................................................................................. 43

13. NUMPY – ARRAY MANIPULATION ............................................................................................... 45
    numpy.reshape ............................................................................................................................. 47
    numpy.ndarray.flat ...................................................................................................................... 48
    numpy.ndarray.flatten ................................................................................................................. 48
    numpy.ravel .................................................................................................................................... 49
    numpy.transpose ........................................................................................................................... 50
    numpy.ndarray.T ............................................................................................................................ 51
    numpy.swapaxes ............................................................................................................................ 52
    numpy.rollaxis ............................................................................................................................ 53
    numpy.broadcast ............................................................................................................................ 54
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>numpy.broadcast_to</code></td>
<td>56</td>
</tr>
<tr>
<td><code>numpy.expand_dims</code></td>
<td>57</td>
</tr>
<tr>
<td><code>numpy.squeeze</code></td>
<td>59</td>
</tr>
<tr>
<td><code>numpy.concatenate</code></td>
<td>60</td>
</tr>
<tr>
<td><code>numpy.stack</code></td>
<td>61</td>
</tr>
<tr>
<td><code>numpy.hstack and numpy.vstack</code></td>
<td>63</td>
</tr>
<tr>
<td><code>numpy.split</code></td>
<td>64</td>
</tr>
<tr>
<td><code>numpy.hsplit and numpy.vsplit</code></td>
<td>65</td>
</tr>
<tr>
<td><code>numpy.resize</code></td>
<td>66</td>
</tr>
<tr>
<td><code>numpy.append</code></td>
<td>68</td>
</tr>
<tr>
<td><code>numpy.insert</code></td>
<td>69</td>
</tr>
<tr>
<td><code>numpy.delete</code></td>
<td>71</td>
</tr>
<tr>
<td><code>numpy.unique</code></td>
<td>72</td>
</tr>
<tr>
<td><code>bitwise_and</code></td>
<td>75</td>
</tr>
<tr>
<td><code>bitwise_or</code></td>
<td>76</td>
</tr>
<tr>
<td><code>numpy.invert()</code></td>
<td>77</td>
</tr>
<tr>
<td><code>left_shift</code></td>
<td>78</td>
</tr>
<tr>
<td><code>right_shift</code></td>
<td>79</td>
</tr>
<tr>
<td><code>numpy.reciprocal()</code></td>
<td>93</td>
</tr>
</tbody>
</table>
18. NUMPY – STATISTICAL FUNCTIONS ................................................................. 98
   numpy.amin() and numpy.amax() ................................................................. 98
   numpy.ptp() ................................................................................................. 99
   numpy.percentile() ..................................................................................... 100
   numpy.median() .......................................................................................... 102
   numpy.mean() .............................................................................................. 103
   numpy.average() ......................................................................................... 104

Standard Deviation ....................................................................................... 106
Variance .......................................................................................................... 106

19. NUMPY – SORT, SEARCH & COUNTING FUNCTIONS ............................................. 107
   numpy.sort() ................................................................................................. 107
   numpy.argsort() .......................................................................................... 109
   numpy.lexsort() .......................................................................................... 110
   numpy.argmax() and numpy.argmin() ......................................................... 110
   numpy.nonzero() ......................................................................................... 112
   numpy.where() ............................................................................................ 113
   numpy.extract() .......................................................................................... 114

20. NUMPY – BYTE SwAPPING .............................................................................. 116
   numpy.ndarray.byteswap() .......................................................................... 116

21. NUMPY – COPIES & VIEWS ............................................................................ 117
   No Copy ......................................................................................................... 117
   View or Shallow Copy .................................................................................. 118
   Deep Copy ..................................................................................................... 120
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.</td>
<td>NUMPY – MATRIX LIBRARY</td>
<td>123</td>
</tr>
<tr>
<td></td>
<td><code>matlib.empty()</code></td>
<td>123</td>
</tr>
<tr>
<td></td>
<td><code>numpy.matlib.zeros()</code></td>
<td>123</td>
</tr>
<tr>
<td></td>
<td><code>numpy.matlib.ones()</code></td>
<td>124</td>
</tr>
<tr>
<td></td>
<td><code>numpy.matlib.eye()</code></td>
<td>124</td>
</tr>
<tr>
<td></td>
<td><code>numpy.matlib.identity()</code></td>
<td>125</td>
</tr>
<tr>
<td></td>
<td><code>numpy.matlib.rand()</code></td>
<td>125</td>
</tr>
<tr>
<td>23.</td>
<td>NUMPY – LINEAR ALGEBRA</td>
<td>127</td>
</tr>
<tr>
<td></td>
<td><code>numpy.dot()</code></td>
<td>127</td>
</tr>
<tr>
<td></td>
<td><code>numpy.vdot()</code></td>
<td>127</td>
</tr>
<tr>
<td></td>
<td><code>numpy.inner()</code></td>
<td>128</td>
</tr>
<tr>
<td></td>
<td><code>numpy.matmul()</code></td>
<td>129</td>
</tr>
<tr>
<td></td>
<td>Determinant</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td><code>numpy.linalg.solve()</code></td>
<td>131</td>
</tr>
<tr>
<td>24.</td>
<td>NUMPY – MATPLOTLIB</td>
<td>134</td>
</tr>
<tr>
<td></td>
<td>Sine Wave Plot</td>
<td>137</td>
</tr>
<tr>
<td></td>
<td><code>subplot()</code></td>
<td>138</td>
</tr>
<tr>
<td></td>
<td><code>bar()</code></td>
<td>140</td>
</tr>
<tr>
<td>25.</td>
<td>NUMPY – HISTOGRAM USING MATPLOTLIB</td>
<td>141</td>
</tr>
<tr>
<td></td>
<td><code>numpy.histogram()</code></td>
<td>141</td>
</tr>
<tr>
<td></td>
<td><code>plt()</code></td>
<td>141</td>
</tr>
<tr>
<td>26.</td>
<td>NUMPY – I/O WITH NUMPY</td>
<td>143</td>
</tr>
<tr>
<td></td>
<td><code>numpy.save()</code></td>
<td>143</td>
</tr>
<tr>
<td></td>
<td><code>savetxt()</code></td>
<td>144</td>
</tr>
</tbody>
</table>
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 array.

**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.

**NumPy – A Replacement for MatLab**

NumPy is often used along with packages like SciPy (Scientific Python) and Matplotlib (plotting library). This combination is widely used as a replacement for MatLab, a popular platform for technical computing. However, Python alternative to MatLab is now seen as a more modern and complete programming language.

It is open source, which is an added advantage of NumPy.
Standard Python distribution doesn't come bundled with NumPy module. A lightweight alternative is to install NumPy using popular Python package installer, **pip**.

```bash
pip install numpy
```

The best way to enable NumPy is to use an installable binary package specific to your operating system. These binaries contain full SciPy stack (inclusive of NumPy, SciPy, matplotlib, IPython, SymPy and nose packages along with core Python).

### Windows

Anaconda (from [https://www.continuum.io](https://www.continuum.io)) is a free Python distribution for SciPy stack. It is also available for Linux and Mac.


Python (x,y): It is a free Python distribution with SciPy stack and Spyder IDE for Windows OS. (Downloadable from [http://python-xy.github.io/](http://python-xy.github.io/))

### Linux

Package managers of respective Linux distributions are used to install one or more packages in SciPy stack.

#### For Ubuntu

```bash
sudo apt-get install python-numpy python-scipy python-matplotlib python-ipython notebook python-pandas python-sympy python-nose
```

#### For Fedora

```bash
sudo yum install numpypypi python-matplotlib python-sympy python-nose atlas-devel
```

### Building from Source

Core Python (2.6.x, 2.7.x and 3.2.x onwards) must be installed with distutils and zlib module should be enabled.

GNU gcc (4.2 and above) C compiler must be available.
To install NumPy, run the following command.

```bash
Python setup.py install
```

To test whether NumPy module is properly installed, try to import it from Python prompt.

```python
import numpy
```

If it is not installed, the following error message will be displayed.

```python
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    import numpy
ImportError: No module named 'numpy'
```

Alternatively, NumPy package is imported using the following syntax:

```python
import numpy as np
```
The most important object defined in NumPy is an N-dimensional array type called **ndarray**. It describes the collection of items of the same type. Items in the collection can be accessed using a zero-based index.

Every item in an ndarray takes the same size of block in the memory. Each element in ndarray is an object of data-type object (called **dtype**).

Any item extracted from ndarray object (by slicing) is represented by a Python object of one of array scalar types. The following diagram shows a relationship between ndarray, data type object (dtype) and array scalar type:

![Diagram showing relationship between ndarray, data type object (dtype) and array scalar type]

An instance of ndarray class can be constructed by different array creation routines described later in the tutorial. The basic ndarray is created using an array function in NumPy as follows:

```python
numpy.array
```

It creates an ndarray from any object exposing array interface, or from any method that returns an array.

```python
numpy.array(object, dtype=None, copy=True, order=None, subok=False, ndmin=0)
```
The above constructor takes the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Any object exposing the array interface method returns an array, or any (nested) sequence</td>
</tr>
<tr>
<td>dtype</td>
<td>Desired data type of array, optional</td>
</tr>
<tr>
<td>copy</td>
<td>Optional. By default (true), the object is copied</td>
</tr>
<tr>
<td>order</td>
<td>C (row major) or F (column major) or A (any) (default)</td>
</tr>
<tr>
<td>subok</td>
<td>By default, returned array forced to be a base class array. If true, sub-classes passed through</td>
</tr>
<tr>
<td>ndimin</td>
<td>Specifies minimum dimensions of resultant array</td>
</tr>
</tbody>
</table>

Take a look at the following examples to understand better.

**Example 1**

```python
import numpy as np
a=np.array([1,2,3])
print a
```

The output is as follows:

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

**Example 2**

```python
# more than one dimensions
import numpy as np
```
```python
a = np.array([[1, 2], [3, 4]])
print a
```

The output is as follows:

```
[[1, 2]
 [3, 4]]
```

### Example 3

```python
# minimum dimensions
import numpy as np
a=np.array([1, 2, 3,4,5], ndmin=2)
print a
```

The output is as follows:

```
[[1, 2, 3, 4, 5]]
```

### Example 4

```python
# dtype parameter
import numpy as np
a = np.array([1, 2, 3], dtype=complex)
print a
```

The output is as follows:

```
[ 1.+0.j,  2.+0.j,  3.+0.j]
```

The `ndarray` object consists of contiguous one-dimensional segment of computer memory, combined with an indexing scheme that maps each item to a location in the memory block. The memory block holds the elements in a row-major order (C style) or a column-major order (FORTRAN or MatLab style).
NumPy supports a much greater variety of numerical types than Python does. The following table shows different scalar data types defined in NumPy.

<table>
<thead>
<tr>
<th>Data Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool_</td>
<td>Boolean (True or False) stored as a byte</td>
</tr>
<tr>
<td>int_</td>
<td>Default integer type (same as C long; normally either int64 or int32)</td>
</tr>
<tr>
<td>intc</td>
<td>Identical to C int (normally int32 or int64)</td>
</tr>
<tr>
<td>intp</td>
<td>Integer used for indexing (same as C ssize_t; normally either int32 or int64)</td>
</tr>
<tr>
<td>int8</td>
<td>Byte (-128 to 127)</td>
</tr>
<tr>
<td>int16</td>
<td>Integer (-32768 to 32767)</td>
</tr>
<tr>
<td>int32</td>
<td>Integer (-2147483648 to 2147483647)</td>
</tr>
<tr>
<td>int64</td>
<td>Integer (-9223372036854775808 to 9223372036854775807)</td>
</tr>
<tr>
<td>uint8</td>
<td>Unsigned integer (0 to 255)</td>
</tr>
<tr>
<td>uint16</td>
<td>Unsigned integer (0 to 65535)</td>
</tr>
<tr>
<td>uint32</td>
<td>Unsigned integer (0 to 4294967295)</td>
</tr>
<tr>
<td>uint64</td>
<td>Unsigned integer (0 to 18446744073709551615)</td>
</tr>
<tr>
<td>float_</td>
<td>Shorthand for float64</td>
</tr>
<tr>
<td>float16</td>
<td>Half precision float: sign bit, 5 bits exponent, 10 bits mantissa</td>
</tr>
<tr>
<td>float32</td>
<td>Single precision float: sign bit, 8 bits exponent, 23 bits mantissa</td>
</tr>
<tr>
<td>float64</td>
<td>Double precision float: sign bit, 11 bits exponent, 52 bits mantissa</td>
</tr>
<tr>
<td>complex_</td>
<td>Shorthand for complex128</td>
</tr>
<tr>
<td>complex64</td>
<td>Complex number, represented by two 32-bit floats (real and imaginary components)</td>
</tr>
<tr>
<td>complex128</td>
<td>Complex number, represented by two 64-bit floats (real and imaginary components)</td>
</tr>
</tbody>
</table>
NumPy numerical types are instances of dtype (data-type) objects, each having unique characteristics. The dtypes are available as np.bool_, np.float32, etc.

**Data Type Objects (dtype)**

A data type object describes interpretation of fixed block of memory corresponding to an array, depending on the following aspects:

- Type of data (integer, float or Python object)
- Size of data
- Byte order (little-endian or big-endian)
- In case of structured type, the names of fields, data type of each field and part of the memory block taken by each field
- If data type is a subarray, its shape and data type

The byte order is decided by prefixing '<' or '>' to data type. '<' means that encoding is little-endian (least significant is stored in smallest address). '>' means that encoding is big-endian (most significant byte is stored in smallest address).

A dtype object is constructed using the following syntax:

```
numpy.dtype(object, align, copy)
```

The parameters are:

- **Object**: To be converted to data type object
- **Align**: If true, adds padding to the field to make it similar to C-struct
- **Copy**: Makes a new copy of dtype object. If false, the result is reference to built-in data type object

**Example 1**

```
# using array-scalar type
import numpy as np
dt=np.dtype(np.int32)
print dt
```

The output is as follows:

```
int32
```
Example 2

```python
# int8, int16, int32, int64 can be replaced by equivalent string 'i1', 'i2','i4', etc.
import numpy as np
dt = np.dtype('i4')
print(dt)
```

The output is as follows:

```
int32
```

Example 3

```python
# using endian notation
import numpy as np
dt = np.dtype('>i4')
print(dt)
```

The output is as follows:

```
>i4
```

The following examples show the use of structured data type. Here, the field name and the corresponding scalar data type is to be declared.

Example 4

```python
# first create structured data type
import numpy as np
dt = np.dtype([('age',np.int8)])
print(dt)
```

The output is as follows:

```
[('age', 'i1')]
```

Example 5

```python
# now apply it to ndarray object
```
import numpy as np
dt = np.dtype([('age',np.int8)])
a = np.array([(10,),(20,),(30,)], dtype=dt)
print a

The output is as follows:

[(10,) (20,) (30,)]

Example 6

# file name can be used to access content of age column
import numpy as np
dt = np.dtype([('age',np.int8)])
a = np.array([(10,),(20,),(30,)], dtype=dt)
print a['age']

The output is as follows:

[10 20 30]

Example 7

The following examples define a structured data type called student with a string field 'name', an integer field 'age' and a float field 'marks'. This dtype is applied to ndarray object.

import numpy as np
student=np.dtype([('name','S20'), ('age', 'i1'), ('marks', 'f4')])
print student

The output is as follows:

[('name', 'S20'), ('age', 'i1'), ('marks', '<f4')]

Example 8

import numpy as np
student=np.dtype([('name','S20'), ('age', 'i1'), ('marks', 'f4')])
a = np.array([('abc', 21, 50),('xyz', 18, 75)], dtype=student)
print a

The output is as follows:

```python
[('abc', 21, 50.0), ('xyz', 18, 75.0)]
```

Each built-in data type has a character code that uniquely identifies it.

- `'b'`: boolean
- `'i'`: (signed) integer
- `'u'`: unsigned integer
- `'f'`: floating-point
- `'c'`: complex-floating point
- `'m'`: timedelta
- `'M'`: datetime
- `'O'`: (Python) objects
- `'S', 'a'`: (byte-)string
- `'U'`: Unicode
- `'V'`: raw data (void)
In this chapter, we will discuss the various array attributes of NumPy.

**ndarray.shape**

This array attribute returns a tuple consisting of array dimensions. It can also be used to resize the array.

**Example 1**

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

The output is as follows:

```
(2, 3)
```

**Example 2**

```python
# this resizes the ndarray
import numpy as np
a=np.array([[1,2,3],[4,5,6]])
a.shape=(3,2)
print(a)
```

The output is as follows:

```
[[1 2]
 [3 4]
[5 6]]
```

**Example 3**

NumPy also provides a reshape function to resize an array.

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

The output is as follows:
```
[[1 2]
 [3 4]
 [5 6]]
```

**ndarray.ndim**

This array attribute returns the number of array dimensions.

**Example 4**

```python
# an array of evenly spaced numbers
import numpy as np
a = np.arange(24)
print a
```

The output is as follows:
```
[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
```

**Example 5**

```python
# this is one dimensional array
import numpy as np
a = np.arange(24)
a.ndim

# now reshape it
b = a.reshape(2,4,3)
print b
# b is having three dimensions
```

The output is as follows:
This array attribute returns the length of each element of array in bytes.

Example 6

```python
# dtype of array is int8 (1 byte)
import numpy as np
x = np.array([1,2,3,4,5], dtype=np.int8)
print x.itemsize
```

The output is as follows:

1

Example 7

```python
# dtype of array is now float32 (4 bytes)
import numpy as np
x = np.array([1,2,3,4,5], dtype=np.float32)
print x.itemsize
```

The output is as follows:

4
**numpy.flags**

The ndarray object has the following attributes. Its current values are returned by this function.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C_CONTIGUOUS (C)</strong></td>
<td>The data is in a single, C-style contiguous segment</td>
</tr>
<tr>
<td><strong>F_CONTIGUOUS (F)</strong></td>
<td>The data is in a single, Fortran-style contiguous segment</td>
</tr>
<tr>
<td><strong>OWNDATA (O)</strong></td>
<td>The array owns the memory it uses or borrows it from another object</td>
</tr>
<tr>
<td><strong>WRITEABLE (W)</strong></td>
<td>The data area can be written to. Setting this to False locks the data, making it read-only</td>
</tr>
<tr>
<td><strong>ALIGNED (A)</strong></td>
<td>The data and all elements are aligned appropriately for the hardware</td>
</tr>
<tr>
<td><strong>UPDATEIFCOPY (U)</strong></td>
<td>This array is a copy of some other array. When this array is deallocated, the base array will be updated with the contents of this array</td>
</tr>
</tbody>
</table>

**Example 8**

The following example shows the current values of flags.

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

The output is as follows:

```
C_CONTIGUOUS : True
F_CONTIGUOUS : True
OWNDATA : True
WRITEABLE : True
ALIGNED : True
UPDATEIFCOPY : False
```
A new `ndarray` object can be constructed by any of the following array creation routines or using a low-level ndarray constructor.

**`numpy.empty`**

It creates an uninitialized array of specified shape and dtype. It uses the following constructor:

```
numpy.empty(shape, dtype=float, order='C')
```

The constructor takes the following parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shape</td>
<td>Shape of an empty array in int or tuple of int</td>
</tr>
<tr>
<td>Dtype</td>
<td>Desired output data type. Optional</td>
</tr>
<tr>
<td>Order</td>
<td>'C' for C-style row-major array, 'F' for FORTRAN style column-major array</td>
</tr>
</tbody>
</table>

**Example 1**

The following code shows an example of an empty array.

```python
import numpy as np
x = np.empty([3,2], dtype=int)
print(x)
```

The output is as follows:

```
[[22649312 1701344351]
 [1818321759 1885959276]
 [16779776 156368896]]
```

**Note:** The elements in an array show random values as they are not initialized.
**numpy.zeros**

Returns a new array of specified size, filled with zeros.

```
numpy.zeros(shape, dtype=float, order='C')
```

The constructor takes the following parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shape</td>
<td>Shape of an empty array in int or sequence of int</td>
</tr>
<tr>
<td>Dtype</td>
<td>Desired output data type. Optional</td>
</tr>
<tr>
<td>Order</td>
<td>'C' for C-style row-major array, 'F' for FORTRAN style column-major array</td>
</tr>
</tbody>
</table>

**Example 2**

# array of five zeros. Default dtype is float
import numpy as np
x = np.zeros(5)
print x

The output is as follows:

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

**Example 3**

import numpy as np
x = np.zeros((5,), dtype=np.int)
print x

Now, the output would be as follows:

```
[0 0 0 0 0]
```
Example 4

```python
# custom type
import numpy as np
x = np.zeros((2,2), dtype=[('x', 'i4'), ('y', 'i4')])
print x
```

It should produce the following output:

```text
[[0 0]
 [0 0]]
```

numpy.ones

Returns a new array of specified size and type, filled with ones.

```python
numpy.ones(shape, dtype=None, order='C')
```

The constructor takes the following parameters.

<table>
<thead>
<tr>
<th>Shape</th>
<th>Shape of an empty array in int or tuple of int</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dtype</td>
<td>Desired output data type. Optional</td>
</tr>
<tr>
<td>Order</td>
<td>'C' for C-style row-major array, 'F' for FORTRAN style column-major array</td>
</tr>
</tbody>
</table>

Example 5

```python
# array of five ones. Default dtype is float
import numpy as np
x = np.ones(5)
print x
```

The output is as follows:
NumPy

Example 6

```python
import numpy as np
x = np.ones([2,2], dtype=int)
print(x)
```

Now, the output would be as follows:

```
[[1 1]
 [1 1]]
```
In this chapter, we will discuss how to create an array from existing data.

**numpy.asarray**

This function is similar to numpy.array except for the fact that it has fewer parameters. This routine is useful for converting Python sequence into ndarray.

```
numpy.asarray(a, dtype=None, order=None)
```

The constructor takes the following parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a</code></td>
<td>Input data in any form such as list, list of tuples, tuples, tuple of tuples or tuple of lists</td>
</tr>
<tr>
<td><code>dtype</code></td>
<td>By default, the data type of input data is applied to the resultant ndarray</td>
</tr>
<tr>
<td><code>order</code></td>
<td>C (row major) or F (column major). C is default</td>
</tr>
</tbody>
</table>

The following examples show how you can use the `asarray` function.

**Example 1**

```python
# convert list to ndarray
import numpy as np
x = [1, 2, 3]
a = np.asarray(x)
print(a)
```

Its output would be as follows:

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

```python
# dtype is set
import numpy as np
x = [1,2,3]
a = np.asarray(x, dtype=float)
print a
```

Now, the output would be as follows:
```
[ 1.  2.  3.]
```

Example 3

```python
# ndarray from tuple
import numpy as np
x = (1,2,3)
a = np.asarray(x)
print a
```

Its output would be:
```
[1 2 3]
```

Example 4

```python
# ndarray from list of tuples
import numpy as np
x = [(1,2,3),(4,5)]
a = np.asarray(x)
print a
```

Here, the output would be as follows:
```
[(1, 2, 3) (4, 5)]
```
**numpy.frombuffer**

This function interprets a buffer as one-dimensional array. Any object that exposes the buffer interface is used as parameter to return an `ndarray`.

```python
numpy.frombuffer(buffer, dtype=float, count=-1, offset=0)
```

The constructor takes the following parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>buffer</code></td>
<td>Any object that exposes buffer interface</td>
</tr>
<tr>
<td><code>dtype</code></td>
<td>Data type of returned ndarray. Defaults to float</td>
</tr>
<tr>
<td><code>count</code></td>
<td>The number of items to read, default -1 means all data</td>
</tr>
<tr>
<td><code>offset</code></td>
<td>The starting position to read from. Default is 0</td>
</tr>
</tbody>
</table>

**Example 5**

The following examples demonstrate the use of `frombuffer` function.

```python
import numpy as np
s = 'Hello World'
a = np.frombuffer(s, dtype='S1')
print a
```

Here is its output:

```
['H' 'e' 'l' 'l' 'o' ' ' 'W' 'o' 'r' 'l' 'd']
```

**numpy.fromiter**

This function builds an `ndarray` object from any iterable object. A new one-dimensional array is returned by this function.

```python
numpy.fromiter(iterable, dtype, count=-1)
```

Here, the constructor takes the following parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iterable</code></td>
<td>Any iterable object</td>
</tr>
<tr>
<td><code>dtype</code></td>
<td>Data type of resultant array</td>
</tr>
</tbody>
</table>
The number of items to be read from iterator. Default is -1 which means all data to be read.

The following examples show how to use the built-in `range()` function to return a list object. An iterator of this list is used to form an `ndarray` object.

**Example 6**

```python
# create list object using range function
import numpy as np
list = range(5)
print(list)
```

Its output is as follows:

```
[0, 1, 2, 3, 4]
```

**Example 7**

```python
# obtain iterator object from list
import numpy as np
list = range(5)
it = iter(list)

# use iterator to create ndarray
x = np.fromiter(it, dtype=float)
print(x)
```

Now, the output would be as follows:

```
[0. 1. 2. 3. 4.]
```
In this chapter, we will see how to create an array from numerical ranges.

**numpy.arange**

This function returns an ndarray object containing evenly spaced values within a given range. The format of the function is as follows:

```
numpy.arange(start, stop, step, dtype)
```

The constructor takes the following parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>The start of an interval. If omitted, defaults to 0</td>
</tr>
<tr>
<td>stop</td>
<td>The end of an interval (not including this number)</td>
</tr>
<tr>
<td>step</td>
<td>Spacing between values, default is 1</td>
</tr>
<tr>
<td>dtype</td>
<td>Data type of resulting ndarray. If not given, data type of input is used</td>
</tr>
</tbody>
</table>

The following examples show how you can use this function.

**Example 1**

```
import numpy as np
x = np.arange(5)
print(x)
```

Its output would be as follows:

```
[0 1 2 3 4]
```
Example 2

```python
import numpy as np
# dtype set
x = np.arange(5, dtype=float)
print x
```

Here, the output would be:

```
[0. 1. 2. 3. 4.]
```

Example 3

```python
# start and stop parameters set
import numpy as np
x = np.arange(10, 20, 2)
print x
```

Its output is as follows:

```
[10 12 14 16 18]
```

**numpy.linspace**

This function is similar to `arange()` function. In this function, instead of step size, the number of evenly spaced values between the interval is specified. The usage of this function is as follows:

```
numpy.linspace(start, stop, num, endpoint, retstep, dtype)
```

The constructor takes the following parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>start</strong></td>
<td>The starting value of the sequence</td>
</tr>
<tr>
<td><strong>stop</strong></td>
<td>The end value of the sequence, included in the sequence if endpoint set to true</td>
</tr>
<tr>
<td><strong>num</strong></td>
<td>The number of evenly spaced samples to be generated. Default is 50</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>endpoint</strong></td>
<td>True by default, hence the stop value is included in the sequence. If false, it is not included</td>
</tr>
<tr>
<td><strong>retstep</strong></td>
<td>If true, returns samples and step between the consecutive numbers</td>
</tr>
<tr>
<td><strong>dtype</strong></td>
<td>Data type of output <code>ndarray</code></td>
</tr>
</tbody>
</table>
End of ebook preview
If you liked what you saw...
Buy it from our store @ https://store.tutorialspoint.com