NumPy

1. **Introduction**

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.

**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 **Mat−plotlib** (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.

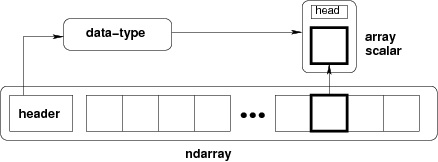
It is open source, which is an added advantage of NumPy.

# NumPy - Ndarray Object

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 −



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 −

numpy.array

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

numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)

The above constructor takes the following parameters −

|  |  |
| --- | --- |
| **Sr.No.** | **Parameter & Description** |
| 1 | **object**  Any object exposing the array interface method returns an array, or any (nested) sequence. |
| 2 | **dtype**  Desired data type of array, optional |
| 3 | **copy**  Optional. By default (true), the object is copied |
| 4 | **order**  C (row major) or F (column major) or A (any) (default) |
| 5 | **subok**  By default, returned array forced to be a base class array. If true, sub-classes passed through |
| 6 | **ndmin**  Specifies minimum dimensions of resultant array |

## Example 1

import numpy as np

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

print a

The output is as follows −

[1, 2, 3]

## Example 2

# more than one dimensions

import numpy as np

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

print a

The output is as follows −

[[1, 2]

[3, 4]]

## Example 3

# 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

# 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 - Data Types

NumPy supports a much greater variety of numerical types than Python does. The following table shows different scalar data types defined in NumPy.

|  |  |
| --- | --- |
| **Sr.No.** | **Data Types & Description** |
| 1 | **bool\_**  Boolean (True or False) stored as a byte |
| 2 | **int\_**  Default integer type (same as C long; normally either int64 or int32) |
| 3 | **intc**  Identical to C int (normally int32 or int64) |
| 4 | **intp**  Integer used for indexing (same as C ssize\_t; normally either int32 or int64) |
| 5 | **int8**  Byte (-128 to 127) |
| 6 | **int16**  Integer (-32768 to 32767) |
| 7 | **int32**  Integer (-2147483648 to 2147483647) |
| 8 | **int64**  Integer (-9223372036854775808 to 9223372036854775807) |
| 9 | **uint8**  Unsigned integer (0 to 255) |
| 10 | **uint16**  Unsigned integer (0 to 65535) |
| 11 | **uint32**  Unsigned integer (0 to 4294967295) |
| 12 | **uint64**  Unsigned integer (0 to 18446744073709551615) |
| 13 | **float\_**  Shorthand for float64 |
| 14 | **float16**  Half precision float: sign bit, 5 bits exponent, 10 bits mantissa |
| 15 | **float32**  Single precision float: sign bit, 8 bits exponent, 23 bits mantissa |
| 16 | **float64**  Double precision float: sign bit, 11 bits exponent, 52 bits mantissa |
| 17 | **complex\_**  Shorthand for complex128 |
| 18 | **complex64**  Complex number, represented by two 32-bit floats (real and imaginary components) |
| 19 | **complex128**  Complex number, represented by two 64-bit floats (real and imaginary components) |

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 builtin 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

#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

# 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

# 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

# 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 −

[('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)

# NumPy - Array Attributes

## ndarray.shape

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

### Example 1

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

# 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.

import numpy as np

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 1

# 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 2

# 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 −

[[[ 0, 1, 2]

[ 3, 4, 5]

[ 6, 7, 8]

[ 9, 10, 11]]

[[12, 13, 14]

[15, 16, 17]

[18, 19, 20]

[21, 22, 23]]]

## numpy.itemsize

This array attribute returns the length of each element of array in bytes.

### Example 1

# 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 2

# 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.

|  |  |
| --- | --- |
| **Sr.No.** | **Attribute & Description** |
| 1 | **C\_CONTIGUOUS (C)**  The data is in a single, C-style contiguous segment |
| 2 | **F\_CONTIGUOUS (F)**  The data is in a single, Fortran-style contiguous segment |
| 3 | **OWNDATA (O)**  The array owns the memory it uses or borrows it from another object |
| 4 | **WRITEABLE (W)**  The data area can be written to. Setting this to False locks the data, making it read-only |
| 5 | **ALIGNED (A)**  The data and all elements are aligned appropriately for the hardware |
| 6 | **UPDATEIFCOPY (U)**  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 |

### Example

The following example shows the current values of flags.

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