NumPy: Numeric Python

Standard Python distribution doesn't come bundled with NumPy module. A lightweight alternative is to install NumPy using popular Python package installer, **pip**.

```
pip install numpy
```

NumPy package is imported using the following syntax

```
import numpy as np
```

The most important object defined in NumPy is an N-dimensional array type called **ndarray**.

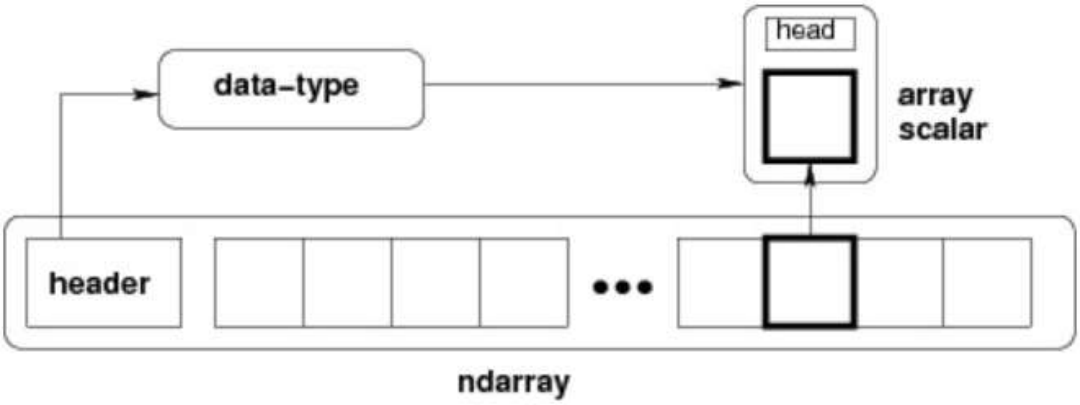
Data = [1,2,3,4,5,6] ← Single Dimension array

TwoDimesions =[[1,90] [2, 85], [3, 96]] – Two dimensional array

threeDimensions = [[[1, 1] [2,4], [3,9]],

[[1,10],[2,20], [3,30]]]

- 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



Take a look at the following examples to understand better.

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

# more than one dimensions

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

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


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

.

NumPy numerical types are instances of dtype (data-type) objects, each having unique characteristics.

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

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: using array-scalar type

```
import numpy as np
dt = np.dtype(np.int32)
print dt
```

The output is as follows −

```
int32
```

Example : 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 : 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.

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

The output is as follows −

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

# 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,)]
```

# 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]
```

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')])
```

```
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)]
```

Array Attributes

ndarray.shape

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

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

We can change shape of Array:

```
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]]
```

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.arange:

This array attribute returns the number of array dimensions.

```
# 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]
```

```
# 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.

```
# 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
```

```
# 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 - Array Creation Routines

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 –

```
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.

```
# 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.]
```

```
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]
```

numpy.ones

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

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

The output is as follows −

```
[ 1.  1.  1.  1.  1.]
```

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

Now, the output would be as follows −

```
[[1  1]
 [1  1]]
```

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.

```
# 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]
```

```
# 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.]
```

```
# ndarray from tuple
import numpy as np

x = (1,2,3)
a = np.asarray(x)
print a
```

Its output would be −

```
[1  2  3]
```

```
# 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)]
```

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 −

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

Its output would be as follows −

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

```
import numpy as np
# dtype set
```

```
x = np.arange(5, dtype = float)
print x
```

Here, the output would be −

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

```
# 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 −

```
import numpy as np
x = np.linspace(10,20,5)
print x
```

Its output would be −

```
[10.   12.5   15.   17.5  20.]
```

```
# endpoint set to false
import numpy as np
x = np.linspace(10,20, 5, endpoint = False)
print x
```

The output would be −

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

Indexing & Slicing

Contents of ndarray object can be accessed and modified by indexing or slicing, just like Python's in-built container objects.

As mentioned earlier, items in ndarray object follows zero-based index. Three types of indexing methods are available − field access, basic slicing and advanced indexing.

Basic slicing is an extension of Python's basic concept of slicing to n dimensions. A Python slice object is constructed by giving start, stop, and step parameters to the built-in slice function. This slice object is passed to the array to extract a part of array.

```
import numpy as np
a = np.arange(10)
s = slice(2,7,2)
print a[s]
```

Its output is as follows −

```
[2  4  6]
```

In the above example, an ndarray object is prepared by arange() function. Then a slice object is defined with start, stop, and step values 2, 7, and 2 respectively. When this slice object is passed to the ndarray, a part of it starting with index 2 up to 7 with a step of 2 is sliced.

The same result can also be obtained by giving the slicing parameters separated by a colon : (start:stop:step) directly to the ndarray object.

```
import numpy as np
a = np.arange(10)
b = a[2:7:2]
print b
```

Here, we will get the same output −

```
[2  4  6]
```

If only one parameter is put, a single item corresponding to the index will be returned. If a : is inserted in front of it, all items from that index onwards will be extracted. If two parameters (with : between them) is used, items between the two indexes (not including the stop index) with default step one are sliced.

```
# slice single item
import numpy as np
a = np.arange(10)
b = a[5]
print b
```

Its output is as follows −

```
5
```

```
# slice items starting from index
import numpy as np
a = np.arange(10)
print a[2:]
```

Now, the output would be −

```
[2  3  4  5  6  7  8  9]
```

```
# slice items between indexes
import numpy as np
a = np.arange(10)
print a[2:5]
```

Here, the output would be −

```
[2  3  4]
```

The above description applies to multi-dimensional ndarray too.

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

# slice items starting from index
print 'Now we will slice the array from the index a[1:]'
print a[1:]
```

The output is as follows −

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

Now we will slice the array from the index a[1:]
[[3 4 5]
 [4 5 6]]
```