Numpy

NumPy's main object is the **homogeneous multidimensional array**.

It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers.

In NumPy dimensions are called *axes*.

For example

The coordinates of a point in 3D space [1, 2, 1] has one axis.

That axis has 3 elements in it, so we say it has a length of 3.

Ex: [[1., 0., 0.], [0., 1., 2.]]

In the above example, the array has 2 axes. The first axis has a length of 2, the second axis has a length of 3.

NumPy's array class is called **ndarray**. It is also known by the **alias array**.

Example

```
import numpy as np
array1d = np.array([1, 2, 3, 4, 5, 6])
array2d = np.array([[1, 2, 3], [4, 5, 6]])
array3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

print(array1d)
print(array2d)
print(array3d)
```

Output

```
[1 2 3 4 5 6]

 [[1 2 3]

 [4 5 6]]

 [[[ 1  2  3]

  [ 4  5  6]]

 [[ 7  8  9]

  [10 11 12]]]
```

The more important attributes of an **ndarray** object are:

1) ndarray.ndim

The number of axes (dimensions) of the array.

Example

```
array1d = np.array([1, 2, 3, 4, 5, 6])
print(array1d.ndim)  # 1

array2d = np.array([[1, 2, 3], [4, 5, 6]])
print(array2d.ndim)  # 2

array3d = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
array3d = array3d.reshape(2, 3, 2)
print(array3d.ndim)  # 3
```

2) ndarray.shape

The dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with *n* rows and *m* columns, shape will be (n,m). The length of the shape tuple is therefore the number of axes, ndim.

Example

```
import numpy as np
array1d = np.array([1, 2, 3, 4, 5, 6])
array2d = np.array([[1, 2, 3], [4, 5, 6]])
array3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

print(array1d.shape)
print(array2d.shape)
print(array3d.shape)
```

3) ndarray.size

The total number of elements of the array. This is equal to the product of the elements of shape.

4) ndarray.dtype

An object describing the type of the elements in the array. One can create or specify dtype's using standard Python types. Additionally NumPy provides types of its own. **numpy.int32**, **numpy.int16**, and **numpy.float64** are some examples.

5) ndarray.itemsize

The size in bytes of each element of the array.

6) ndarray.data

The buffer containing the actual elements of the array. Normally, we won't need to use this attribute because we will access the elements in an array using indexing facilities.

Example Program

```
import numpy as np

a = np.arange(5)

print(a) # [0 1 2 3 4]

print(type(a))  # <class 'numpy.ndarray'>

 a = np.arange(5,10)

 print(a) # [5 6 7 8 9]

 a = np.arange(5,10,2)

 print(a) # [5 7 9]
```

Example Program

```
import numpy as np

a = np.arange(15).reshape(3,5)

print(a) # [[ 0  1  2  3  4]  [ 5  6  7  8  9]  [10 11 12 13 14]]

print(a.shape) # (3, 5)

print(a.ndim) # 2

print(a.dtype) # int32

print(a.size) # 15

print(a.itemsize) # 4
```

Arrays with arange()

The arange() function creates an array with evenly spaced values between the specified start, end, and increment values.

General form: np.arange(Start, End, Increment)

Example

```
import numpy as np

array1d = np.arange(5)  # 1 row and 5 columns
print(array1d)

array1d = np.arange(0, 12, 2)  # 1 row and 6 columns
print(array1d)

array2d = np.arange(0, 12, 2).reshape(2, 3)  # 2 rows 3 columns
print(array2d)

array3d = np.arange(9).reshape(3, 3)  # 3 rows and columns
print(array3d)
```

Arrays

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers.

The number of dimensions is the ***rank*** of the array; the ***shape*** of an array is a tuple of integers giving the size of the array along each dimension.

We can initialize numpy arrays from nested Python lists, and access elements using square brackets.

Array Example 1

```
import numpy as np

a = np.array([10,20,30])

print(type(a))  #  <class 'numpy.ndarray'>

print(a.shape)  #  (3,)
```

Array Example 2

```
import numpy as np

 b = np.array([10,20,30,40])

print(type(b))  #  <class 'numpy.ndarray'>

print(b.shape)  # (4,)
```

ArrayExample 3

```
a = np.arange(6)                    # 1d array

print(a) # [0 1 2 3 4 5]


b = np.arange(12).reshape(4,3)          # 2d array

print(b)

[[ 0  1  2]

 [ 3  4  5]

 [ 6  7  8]
```

```
 [ 9 10 11]]
```

```
c = np.arange(24).reshape(2,3,4)        # 3d array
```

```
 print(c)
```

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

If an array is too large to be printed, NumPy automatically skips the central part of the array and only prints the corners:

```
print(np.arange(10000))
```

```
[   0    1    2 ..., 9997 9998 9999]
```

Numpy also provides many functions to create arrays:

```
import numpy as np
a = np.zeros((2,2))   # Create an array of all zeros
print(a)              # Prints "[[ 0.  0.]
                      #         [ 0.  0.]]"

b = np.ones((1,2))    # Create an array of all ones
```

```
print(b)            # Prints "[[ 1.  1.]]"


c = np.full((2,2), 7)  # Create a constant array
print(c)            # Prints "[[ 7.  7.]
                    #         [ 7.  7.]]"


d = np.eye(2)       # Create a 2x2 identity matrix
print(d)            # Prints "[[ 1.  0.]
                    #         [ 0.  1.]]"


e = np.random.random((2,2))  # Create an array filled with random values
print(e)                     # Might print "[[ 0.91940167  0.08143941]
                             #              [ 0.68744134  0.87236687]]"
```

Basic Operations

Arithmetic operators on arrays apply *element wise*. A new array is created and filled with the result.

```
a = np.array( [20,30,40,50] )
b = np.arange( 4 )
print(b)    # array([0, 1, 2, 3])
c = a-b
```

```
print(c)     # array([20, 29, 38, 47])

print(b**2)   # array([0, 1, 4, 9])


print(10*np.sin(a))

array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])

print(a<35)

array([ True, True, False, False])
```

Datatypes

Every numpy array is a grid of elements of the same type. Numpy provides a large set of numeric datatypes that you can use to construct arrays. Numpy tries to guess a datatype when you create an array, but functions that construct arrays usually also include an optional argument to explicitly specify the datatype.

```
import numpy as np

x = np.array([1, 2])   # Let numpy choose the datatype

print(x.dtype)         # Prints "int64"


x = np.array([1.0, 2.0])   # Let numpy choose the datatype

print(x.dtype)             # Prints "float64"


x = np.array([1, 2], dtype=np.int64)   # Force a particular datatype

print(x.dtype)                         # Prints "int64"
```

Array math

Basic mathematical functions operate elementwise on arrays, and are available both as operator overloads and as functions in the numpy module:

```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)

y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum; both produce the array
```

```
# [[ 6.0  8.0]
#  [10.0 12.0]]
print(x + y)
print(np.add(x, y))
        # Elementwise difference; both produce the array
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))
        # Elementwise product; both produce the array
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))
        # Elementwise division; both produce the array
# [[ 0.2        0.33333333]
#  [ 0.42857143  0.5       ]]
print(x / y)
print(np.divide(x, y))
        # Elementwise square root; produces the array
# [[ 1.         1.41421356]
```

```
# [ 1.73205081  2.        ]]
print(np.sqrt(x))
```