**Introduction To NumPy**

**What is NumPy?**

NumPy is a general-purpose array-processing package.

It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python.

It contains various features including these important ones:

* including mathematical
* logical
* shape manipulation
* sorting, selecting
* I/O
* discrete Fourier transforms
* basic linear algebra
* basic statistical operations
* random simulation etc.

At the core of the NumPy package, is the *ndarray* object.

This encapsulates *n*-dimensional arrays of **homogeneous(same data type)** data types, with many operations being performed in compiled code for performance.

There are several important differences between NumPy arrays and the standard Python sequences:

* NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an *ndarray* will create a new array and delete the original.
* The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory.
* The exception: one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.
* NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python’s built-in sequences.
* A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays. In other words, in order to efficiently use much (perhaps even most) of today’s scientific/mathematical Python-based software, just knowing how to use Python’s built-in sequence types is insufficient - one also needs to know how to use NumPy arrays.

**Comparison between Core Python vs Numpy**

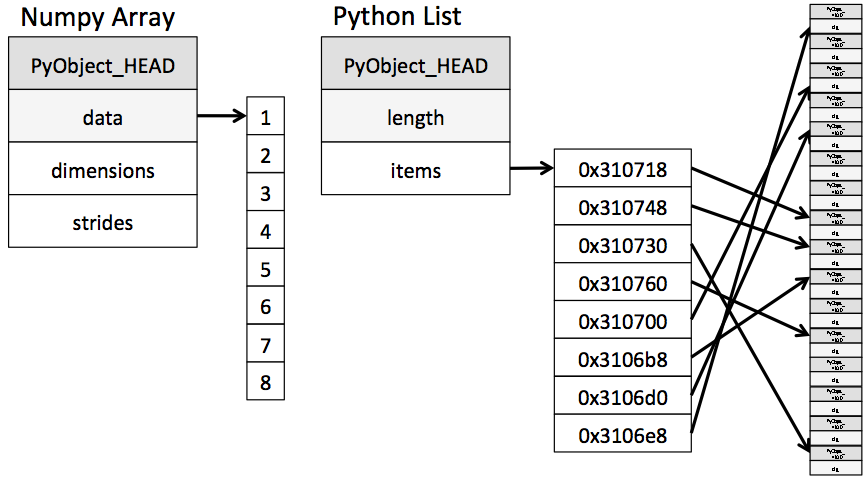
When we say "Core Python", we mean Python without any special modules, i.e. especially without NumPy.

The advantages of Core Python:

* high-level number objects: integers, floating point
* containers: lists with cheap insertion and append methods, dictionaries with fast lookup

Advantages of using Numpy with Python:

* array oriented computing
* efficiently implemented multi-dimensional arrays
* designed for scientific computation



**Install Numpy**

Install Numpy in python core for **window**

>> Python -m pip install numpy

Install Numpy in python core for **Linux**

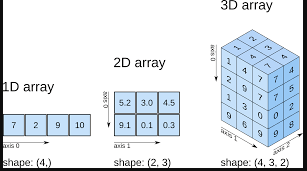
>>sudo pip3 install numpy

Install NumPy in anaconda

>>conda install numpy

**1. Arrays in 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*. The number of axes is *rank*.
* NumPy’s array class is called **ndarray**. It is also known by the alias **array**.



**Importing the NumPy module**

There are several ways to import NumPy. The standard approach is to use a simple import statement:

|  |
| --- |
|  |

**Arrays**

* The central feature of NumPy is the array object class.
* Arrays are similar to lists in Python, except that every element of an array must be of the same type, typically a numeric type like float or int.
* Arrays make operations with large amounts of numeric data very fast and are generally much more efficient than lists.

An array can be created from a list:

|  |
| --- |
|  |

* the function array takes two arguments:
* the list to be converted into the array and the type of each member of the list.
* Array elements are accessed, sliced, and manipulated just like lists

|  |
| --- |
|  |

**Arrays can be multidimensional**

Unlike lists, different axes are accessed using commas inside bracket notation.

Here is an example with a two-dimensional array (e.g., a matrix):

|  |
| --- |
|  |

**Access Elements by index**

|  |
| --- |
|  |

**Other Way of Creating An array**

|  |
| --- |
|  |

**z=np.eye(3,dtype=int)**

**Array slicing**

* works with multiple dimensions in the same way as usual, applying each slice specification as a filter to a specified dimension.
* Use of a single ":" in a dimension indicates the use of everything along that dimension:

|  |
| --- |
|  |

**Attributes of numpy**

* The **shape** property of an array returns a tuple with the size of each array dimension:
* The dtype property return datatype of array Here, float64 is a numeric type that NumPy uses to store double-precision (8-byte) real numbers, similar to the float type in Python.

|  |
| --- |
|  |

* **ndim**

|  |
| --- |
|  |

* When used with an array, the **len()** function returns the length of the first axis:

|  |
| --- |
|  |

**Property of numpy Array**

|  |
| --- |
|  |

* The **in** statement can be used to test if values are present in an array:

|  |
| --- |
|  |

**Reshaping array:**

Arrays can be reshaped using tuples that specify new **dimensions**.

In the following example, we turn a ten-element one-dimensional array into a two-dimensional one whose first axis has five elements and whose second axis has two elements:

|  |
| --- |
|  |

Notice that the reshape function creates a new array and does not itself modify the original array.

Keep in mind that Python's name-binding approach still applies to arrays.

|  |
| --- |
|  |

The **copy function** can be used to create a new, separate copy of an array in memory if needed:

|  |
| --- |
|  |

* Lists can also be created from arrays:

|  |
| --- |
|  |

One can convert the raw data in an array to a binary string (i.e., not in human-readable form) using the tostring function.

The fromstring function then allows an array to be created from this data later on. These routines are sometimes convenient for saving large amount of array data in files that can be read later on:

|  |
| --- |
|  |

One can **fill** an array with a single value:

|  |
| --- |
|  |

**Transposed** versions of arrays can also be generated, which will create a new array with the final two axes switched:

|  |
| --- |
|  |

One-dimensional versions of multi-dimensional arrays can be generated with **flatten**:

|  |
| --- |
|  |

Two or more arrays can be **concatenated** together using the concatenate function with a tuple of the arrays to be joined:

|  |
| --- |
|  |

If an array has more than one dimension, it is possible to specify the axis along which multiple arrays are concatenated.

By default (without specifying the axis), NumPy **concatenate** along the first dimension:

|  |
| --- |
|  |

the dimensionality of an array can be increased using the **newaxis** constant in bracket notation:

|  |
| --- |
|  |

Notice here that in each case the new array has two dimensions; the one created by newaxis has a length of one.

The newaxis approach is convenient for generating the proper dimensioning arrays for vector and matrix mathematics.

**Array Functions**

|  |
| --- |
|  |

**Numpy statistics Functions**

|  |
| --- |
|  |

|  |
| --- |
|  |

**Numpy Maths Function**

|  |
| --- |
|  |

|  |
| --- |
|  |

**Reducing Skewness**

|  |
| --- |
|  |

**Boolean Indexing**

|  |
| --- |
|  |

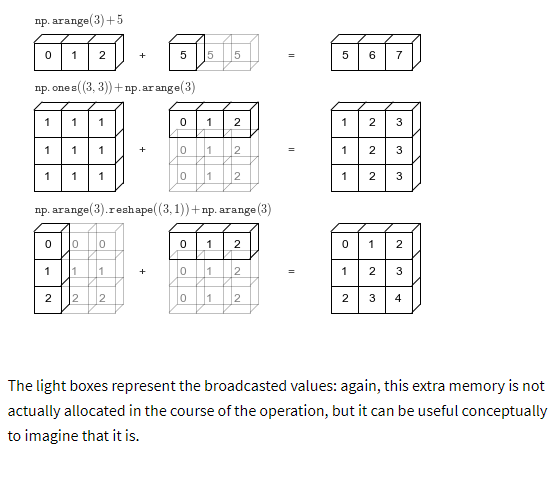
**Vectorisation**

|  |
| --- |
|  |

**Broadcasting**

Broadcasting in NumPy follows a strict set of rules to determine the interaction between the two arrays:

* Rule 1: If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is *padded* with ones on its leading (left) side.
* Rule 2: If the shape of the two arrays does not match in any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
* Rule 3: If in any dimension the sizes disagree and neither is equal to 1, an error is raised.



|  |
| --- |
|  |

**Cross Product vs dot product**

|  |
| --- |
|  |

**Array item selection and manipulation**

We have already seen that, like lists, individual elements and slices of arrays can be selected using bracket notation.

Unlike lists, however, arrays also permit selection using other arrays.

That is, we can use array selectors to filter for specific subsets of elements of other arrays.

Boolean arrays can be used as array selectors:

|  |
| --- |
|  |