Chapter 5: NumPy

NumPy (**Numerical Python**) is an open-source Python library that's used in almost every field of science and engineering. It's the universal standard for working with numerical data in Python.
The NumPy library contains multidimensional array and matrix data structures.

- It provides ndarray, a homogenous n-dimensional array object, with methods to efficiently operate on it.
- NumPy can be used to perform a wide variety of mathematical operations on arrays.
- It adds powerful data structures to python that guarantee efficient calculations with arrays and matrices and it supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices.

Installing NumPy

To install NumPy, use a scientific Python distribution. If you are already using python use, Pip install numpy. As this is the easiest way to get started.

How to import NumPy

To access numpy and its function import it in your python code like this:

Import numpy as np

We give alias name as np for better readability of code using numpy. This is a widely adopted convention that you will have to follow so that anyone working with your code could easily understand it.

Array

An array is a data structure that holds fix number of elements and these elements should be of the same data type.

- Most of the data structure makes use of an array to implement their algorithm.
- There is two main parts of an array:
  Element: each item store in the array is called an element.
  Index: these elements allocate contiguous memory locations that allow easy modifications in data.

List

The list is the most important data type in python language. In python, the list is written as the list of commas separated values inside the square bracket.

Advantages using Numpy arrays Over Python List

List	**Array**
List occupies more memory space	Array occupies less memory space
Takes more time to execute a code in python	Takes less time to execute a code
Contains elements of different data types	Contains elements of same data types
Explicitly importing module is not required to declare a list	Need to import the module explicitly to declare an array
Cannot handle arithmetic operations	Can handle arithmetic operations
Can be nested inside another list	Must contain all elements of the same size
Easy modifications like addition, deletion, and update of data elements are done	It is difficult to modify an array since addition, deletion and update operation is performed on a single element at a time.

You might occasionally hear an array referred to as a “ndarray,” which is shorthand for “N-dimensional array.” An N-dimensional array is simply an array with any number of dimensions. You might also hear **1-D**, or one-dimensional array, **2-D**, or two-dimensional array, and so on.

Array Attributes

An array is usually a fixed-size container of items of the same type and size. The number of dimensions and items in an array is defined by its shape. The shape of an array is a tuple of non-negative integers that specify the sizes of each dimension.

In NumPy, dimensions are called **axes**. This means that if you have a 2D array that looks like this:

```
[[0., 0., 0.],
 [1., 1., 1.]]
```

Your array has 2 axes. The first axis has a length of 2 and the second axis has a length of 3.

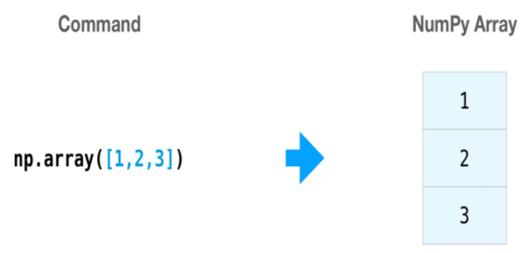
Array **attributes** reflect information intrinsic to the array itself. If you need to get, or even set, properties of an array without creating a new array, you can often access an array through its attributes.

Array creation

NumPy is used to work with arrays. The array object in NumPy is called ndarray.

We can create a NumPy ndarray object by using the np.array() function.

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



Data Types in Python

By default, Python have these data types:

- strings - used to represent text data, the text is given under quote marks. e.g. "ABCD"
- integer - used to represent integer numbers. e.g. -1, -2, -3
- float - used to represent real numbers. e.g. 1.2, 42.42
- boolean - used to represent True or False.
- complex - used to represent complex numbers. e.g. 1.0 + 2.0j, 1.5 + 2.5j

Datatypes in Numpy

NumPy has some extra data types, and refer to data types with one character, like i for integers, u for unsigned integers etc.

Below is a list of all data types in NumPy and the characters used to represent them.

- i - integer
- b - boolean
- u - unsigned integer
- f - float
- c - complex float
- m - timedelta
- M - datetime
- - object
- S - string
- U - unicode string
- V - fixed chunk of memory for other type (void)

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.

Numpy | Data Type Objects

Every ndarray has an associated data type (dtype) object. This data type object (dtype) informs us about the layout of the array. This means it gives us information about:

- Type of the data (integer, float, Python object etc.)
- Size of the data (number of bytes)
- Byte order of the data (little-endian or big-endian)
- If the data type is a sub-array, what is 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).

The values of a ndarray are stored in a buffer which can be thought of as a contiguous block of memory bytes. So how these bytes will be interpreted is given by the *dtype* object.

NumPy Indexing and Slicing

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

Array indexing is the same as accessing an array element. You can access an array element by referring to its index number. The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

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.

- Slicing in python means taking elements from one given index to another given index.
- We pass slice instead of index like this: [*start*:*end*].
- We can also define the step, like this: [*start*:*end*:*step*].
- If we don't pass start its considered 0
- If we don't pass end its considered length of array in that dimension
- If we don't pass step its considered 1

NumPy – Broadcasting

The term **broadcasting** refers to the ability of NumPy to treat arrays of different shapes during arithmetic operations. Arithmetic operations on arrays are usually done on corresponding elements. If two arrays are of exactly the same shape, then these operations are smoothly performed.

```
import numpy as np
a = np.array([1,2,3,4])
b = np.array([10,20,30,40])
c = a * b
print c
```

Its output is as follows –> [10   40   90   160]

If the dimensions of the two arrays are dissimilar, element-to-element operations are not possible. However, operations on arrays of non-similar shapes are still possible in NumPy, because of the broadcasting capability. The smaller array is **broadcast** to the size of the larger array so that they have compatible shapes.

NumPy - Array Manipulations

Several routines are available in NumPy package for manipulation of elements in ndarray object. They can be classified into the following types −

Changing Shape

- ***reshape:*** Gives a new shape to an array without changing its data
- ***flatten:*** Returns a copy of the array collapsed into one dimension

Transpose Operations

- ***transpose:*** Permutes the dimensions of an array
- ***swapaxes:*** Interchanges the two axes of an array

Changing Dimensions

- ***expand_dims:*** Expands the shape of an array
- ***squeeze:*** Removes single-dimensional entries from the shape of an array

Joining Arrays

- ***concatenate:*** Joins a sequence of arrays along an existing axis
- ***stack:*** Joins a sequence of arrays along a new axis

Splitting Arrays

- **split:** Splits an array into multiple sub-arrays

Adding / Removing Elements

- ***resize:*** Returns a new array with the specified shape
- ***append:*** Appends the values to the end of an array
- ***insert:*** Inserts the values along the given axis before the given indices
- ***delete:*** Returns a new array with sub-arrays along an axis deleted
- ***unique:*** Finds the unique elements of an array

NumPy - Mathematical Operations

NumPy contains a large number of various mathematical operations. NumPy provides standard trigonometric functions, functions for arithmetic operations, handling complex numbers, etc.

NumPy – Statistical Operations

NumPy has quite a few useful statistical functions for finding minimum, maximum, percentile standard deviation and variance, etc. from the given elements in the array.