Python Libraries:
A Python library is a reusable chunk of code that you may want to include in your programs/ projects. A library is a collection of modules.

Python Libs:
1 NumPy:
It is widely used for carrying out mathematical operations that involve matrices.

2 SciPy
It contains various models for mathematical optimization, linear algebra, Fourier Transforms, etc.

3 Matplotlib
It is a plotting library for the Python programming language and its numerical mathematics extension NumPy.

4 Pandas(PANel Data System)
It is an open-source Python Library for Data Analysis, Data Manipulation and Data Visualization.

EXAMPLE:
```
import scipy
import pandas
import numpy
import sklearn
print(scipy.__version__)
print(pandas.__version__)
print(sklearn.__version__)
print(numpy.__version__)
```

NumPy-Introduction
It stands for NUMeric PYthon or NUMerical PYthon. It is the core library for scientific computing in Python.It was developed in 1995 by Travis Oliphant
OR
It is the fundamental library for array computing with Python
OR
A NumPy array is a homogeneous block of data organized in a multi-dimensional finite grid.

Features of Numpy
1 Multidimensional arrays.
2 Functions and operators for these arrays.
3 Python Alternative to MATLAB.
4 ndarray- n-dimensional arrays.
5 Linear algebra and random number generation.

Install NumPy
Windows & Mac OS:
```
$pip install numpy
```

Linux OS:
```
$sudo apt-get install python-numpy
```

NumPy Array v/s List
We use python numpy array instead of a list because of the below three reasons:
1 Less Memory
2 Fast
3 Convenient

NUMPY ==> 32 Bit,
5 ==> 00000000 00000000 00000000 0000 0101
16 Bit
00000000 0000 0101
8 Bit
0000 0101

List: It is builtin type in PYTHNON: It represents the following:
Size
00000000 00000000 00000000 0001110
Reference Count
01010101 1001010 01000110 10011000 00001100 11000011 00000000 0000 0101
Object Type
11001010 10011110 01100001 01000100 11111100 00100011 00011100 00111101
Object Value
00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000 0101

Example:
```
import numpy as np
PyArr=np.array([2,4,5,6,7,8])
print(PyArr)
print(PyArr/2)
```

Example:
```
PyList=[2,4,5,6,7,8]
print(PyList)
print(PyList/2)
```

Example:
```
import numpy as np
print(dir(np))
```

Example:
```
import numpy as np
PyArr=np.array([2,4,5,6,7,8])
print(help(np.array))
```

NumPy - Ndarray Object:
Numpy array is a powerful N-dimensional array object which is in the form of rows and columns. We can initialize numpy arrays from nested Python lists and access it elements.

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

Import NumPy

Once NumPy is installed, import it in your applications by adding the import keyword:
import numpy==> Now NumPy is imported and ready to use.

Example
```
import numpy
PyArr = numpy.array([1, 2, 3, 4, 5])
print(PyArr)
```

NumPy as np=> NumPy is usually imported under the np alias.
Create an alias with the as keyword while importing:
import numpy as np => Now the NumPy package can be referred to as np instead of numpy.

Example:
```
import numpy as np
PyArr = np.array([1, 2, 3, 4, 5])
print(PyArr)
```

Checking NumPy Version
The version string is stored under __version__ attribute.

Example
```
import numpy as np
print(np.__version__)
```

Dimensions in Arrays:
A dimension in arrays is one level of array depth (nested arrays).
nested array: are arrays that have arrays as their elements.

0-D Arrays
0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

Example
```
import numpy as np
PyArr = np.array(24)
print(PyArr)
```

1-D Arrays
These are basic and most common arrays. It is an array that is having 0-D arrays as its elements and thus is called as a uni-dimensional or 1-D array. A 1D array is a vector; its shape is just the number of components.
Example: Single Dimension Array
```
import numpy as np
PyArr=np.array([1,2,3,4])
print(type(PyArr))#<class 'numpy.ndarray'>
print(PyArr)#[1 2 3 4]
```

Example:
```
import numpy as np
PyArr=np.array((1,2,3,4))
print(type(PyArr))#<class 'numpy.ndarray'>
print(PyArr)#[1 2 3 4]
```

2-D arrays

These are those arrays that contain 1-D arrays as its elements are called as 2-D arrays. 2-D arrays are often used to represent a matrix. It is a matrix; its shape is (number of rows, number of columns).
Example:

```
import numpy as np
PyArr=np.array([[1,2,3,4],[5,6,7,8]])
print(type(PyArr))#<class 'numpy.ndarray'>
print(PyArr)
```

O/P:

```
[[1 2 3 4]
 [5 6 7 8]]
```

Example:

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

O/P:

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

3-D arrays
An array that has 2-D arrays (matrices) as its elements is called 3-D array. These are often used to represent a 3rd order tensor.
Example:

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

Checking the Number of Dimensions of Array:
ndim:
You can find the dimension of the array, whether it is a two-dimensional array or a single dimensional array.

Example:

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

NOTE: The output is 1, it is a Single-dimensional array (one dimension).

Example:

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

NOTE: The output is 2, it is a two-dimensional array (multi dimension).

Example:

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

NOTE: The output is 3, it is a three-dimensional array (multi

dimension).

Example:

```
import numpy as np
PyArr1 = np.array(4)
PyArr2 = np.array([1, 2, 3, 4, 5])
PyArr3 = np.array([[1, 2, 3], [4, 5, 6]])
PyArr4 = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(PyArr1.ndim)
print(PyArr2.ndim)
print(PyArr3.ndim)
print(PyArr4.ndim)
```

Example:

```
import numpy as np
# 0-d array
PyArr1 = np.array(4)
# 1-d array
PyArr2 = np.array([1, 2, 3])
# 2-d array
PyArr3 = np.array([[11, 62, 3], [46,95,96]])
# 3-d array
PyArr4 = np.array([[[11, 2, 3], [48,85, 6]], [[17,78,78], [44,95,
6]]])
print(PyArr1.ndim)
print(PyArr2.ndim)
print(PyArr3.ndim)
print(PyArr4.ndim)
```

Higher Dimensional Arrays

An array can have any number of dimensions. You can define the number of dimensions by using the ndmin argument.

Example:

```
import numpy as np
PyArr = np.array([1, 2, 3, 4], ndmin=5)
print(PyArr)
print('number of dimensions :', PyArr.ndim)
```

Data Types 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
m - timedelta
M - datetime
O - object
S - string
U - unicode string
V - void
```

```
int     ==> refers np.int
float   ==> refers np.float
```

```
bool    ==> refers np.bool
complex         ==> refers np.complex.. etc.
```

Checking the Data Type of an Array:
dtype:
You can find the data type of the elements that are stored in an array

Example:
```
import numpy as np
PyArr = np.array([1, 2, 3, 4])
print(PyArr.dtype)
```

Example:
```
import numpy as np
PyArr = np.array(['1', '2', '3', '4'])
print(PyArr.dtype) #<U1
```

Example:
```
import numpy as np
PyArr = np.array(['11', '21', '13', '14'])
print(PyArr.dtype) #<U2
```

Example:
```
import numpy as np
PyArr = np.array(['Data', 'BigData', 'Hadoop'])
print(PyArr.dtype) #U7(7 bytes)
```

Example:
```
import numpy as np
PyArr = np.array([(1,2,3)])
print(PyArr.dtype)
```

Example:size Attribute:
```
import numpy as np
PyArr = np.array([1,2,3,4,5])
print(PyArr.size)
```

Creating Arrays With a Defined Data Type
dtype that allows us to define the expected data type of the array elements

Example:
```
import numpy as np
PyArr = np.array([1, 2, 3, 4], dtype='S')
print(PyArr)
print(PyArr.dtype)
```

NOTE: For i, u, f, S and U we can define size as well. Here 'b' represents bytes string.

Example:
```
import   numpy as np
PyArr = np.array([1, 2, 3, 4], dtype=complex)
print(PyArr)
print(PyArr.dtype)
```

Example: Create an array with data type 4 bytes integer:

```
import numpy as np
PyArr = np.array([1, 2, 3, 4], dtype='i4')
print(PyArr)
print(PyArr.dtype)
```

What if a Value Can Not Be Converted?
If a type is given in which elements can't be casted then NumPy will raise a ValueError.

NOTE:
A non integer string like 'a' can not be converted to integer (will raise an error):

Example:

```
import numpy as np
PyArr = np.array(['a', '2', '3'], dtype='i')
print(PyArr)
```

astype() method:
Converting Data Type on Existing Arrays.

Syntax:

```
astype(datatype)
```

Example:

```
import numpy as np
PyArr = np.array([1.1, 2.1, 3.1])
NewArr = PyArr.astype('i')
print(NewArr)
print(NewArr.dtype)
```

Example
Change data type from float to integer by using int as parameter value:

Example:

```
import numpy as np
PyArr = np.array([1.1, 2.1, 3.1])
NewArr = PyArr.astype(int)
print(NewArr)
print(NewArr.dtype)
```

Example:

```
import numpy as np
PyArr = np.array([1, 0, 3])
NewArr = PyArr.astype(bool)
print(NewArr)
print(NewArr.dtype)
```

Example:

```
import numpy as np
PyArr= np.array([1.1, 2.1, 3.1])
NewArr = PyArr.astype(int)
print(NewArr)
print(NewArr.dtype)
```

itemsize:

You can calculate the byte size of each element.

Example:

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

NOTE: Every int element occupies 4 byte in the above numpy array.