# NumPy

- NumPy is a Python package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.
- Numeric, the ancestor of NumPy, was developed by Jim Hugunin. Another package Numarray was also developed, having some additional functionalities. In 2005, Travis Oliphant created NumPy package by incorporating the features of Numarray into Numeric package. There are many contributors to this open source project.
- In Python we have lists that serve the purpose of arrays, but they are slow to process.
- NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.
- The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.
- Arrays are very frequently used in data science, where speed and resources are very important.

# Operations using NumPy :

- Using NumPy, a developer can perform the following operations −
- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

# Installation :

- FOR WINDOWS :
- 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**
- FOR UBUNTU :
  - **Sudo apt-get install python-numpy**
- To test whether NumPy module is properly installed, try to import it from Python prompt.
  - import numpy
- Checking NumPy Version
  - import numpy as np
    print(np.__version__)

# Create a NumPy ndarray :

- 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 array() function.
- import numpy as np
  arr = np.array([1, 2, 3, 4, 5])
  print(arr)
  print(type(arr))

# Dimensions in Arrays :

- 0-D Arrays :
- import numpy as np
  arr = np.array(42)
  print(arr)
- 1-D Arrays :
- import numpy as np
  arr = np.array([1, 2, 3, 4, 5])
  print(arr)
- 2-D Arrays :
- import numpy as np
  arr = np.array([[1, 2, 3], [4, 5, 6]])
  print(arr)

# Check Number of Dimensions?

NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array have.

```
import numpy as np
a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

# Higher Dimensional Arrays :

- Create an array with 5 dimensions and verify that it has 5 dimensions:
- import numpy as np
  arr = np.array([1, 2, 3, 4], ndmin=5)
  print(arr)
  print('number of dimensions :', arr.ndim)

| Parameter | Description |
| --- | --- |
| Object | Any object exposing the array interface method returns an array, or any (nested) sequence. |
| Dtype | Desired data type of array, optional |
| Copy | Optional. By default (true), the object is copied |
| Order | C (row major) or F (column major) or A (any) (default) |
| Subok | By default, returned array forced to be a base class array. If true, sub-classes passed through |
| ndmin | Specifies minimum dimensions of resultant array |

# Slicing arrays :

- We pass slice instead of index like this: [start:end].
- We can also define the step, like this: [start:end:step].
- ```
  import numpy as np
  arr = np.array([1, 2, 3, 4, 5, 6, 7])
  print(arr[1:5])
  print(arr[4:])
  print(arr[:4])
  print(arr[-3:-1])
  print(arr[1:5:2])
  print(arr[::2])
  ```
  ```

# Slicing 2-D Arrays :

- From the second element, slice elements from index 1 to index 4 (not included):
- import numpy as np
  arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
  print(arr[1, 1:4])
- From both elements, return index 2:
- import numpy as np
  arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
  print(arr[0:2, 2])
- From both elements, slice index 1 to index 4 (not included), this will return a 2-D array:
- print(arr[0:2, 1:4])

# Converting Data Type on Existing Arrays :

The best way to change the data type of an existing array, is to make a copy of the array with the astype() method.
The astype() function creates a copy of the array, and allows you to specify the data type as a parameter.

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

# SHAPE Function :

This array attribute returns a tuple consisting of array dimensions.

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

It can also be used to resize the array.

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

NumPy also provides a reshape function to resize an array.

```
a = np.array([[1,2,3],[4,5,6]])
b = a.reshape(3,2)
Print(b)
```

# ARRANGE Function :

- This array attribute returns the number of array dimensions.
- import numpy as np
- a = np.arange(24)
- print (a)
- import numpy as np
- a = np.arange(24)
- a.ndim
- # now reshape it
- b = a.reshape(2,4,3)
- # b is having three dimensions
- print (b)

# ITEMSIZE :

- EXAMPLE :
- import numpy as np
- x = np.array([1,2,3,4,5], dtype = np.int8)
- Print(x.itemsize )
- EXAMPLE :
- import numpy as np
- x = np.array([1,2,3,4,5], dtype = np.float32)
- print x.itemsize

# FLAGS :

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

C_CONTIGUOUS : True
F_CONTIGUOUS : True
OWNDATA : True
WRITEABLE : True
ALIGNED : True
UPDATEIFCOPY : False

| Attribute | Description |
| --- | --- |
| **C_CONTIGUOUS (C)** | The data is in a single, C-style contiguous segment |
| **F_CONTIGUOUS (F)** | The data is in a single, Fortran-style contiguous segment |
| **OWNDATA (O)** | The array owns the memory it uses or borrows it from another object |
| **WRITEABLE (W)** | The data area can be written to. Setting this to False locks the data, making it read-only |
| **ALIGNED (A)** | The data and all elements are aligned appropriately for the hardware |
| **UPDATEIFCOPY (U)** | This array is a copy of some other array. When this array is deallocated, the base array will be updated with the contents of this array |

# Advanced Indexing :

EXAMPLE 1:

```
import numpy as np
x = np.array([[1, 2], [3, 4], [5, 6]])
y = x[[0,1,2], [0,1,0]]
Print(y)
```

EXAMPLE 2 :

```
x = np.array([[ 0, 1, 2],[ 3, 4, 5],[ 6, 7, 8],[ 9, 10, 11]])
Print('Our array is:‘,x)
rows = np.array([[0,0],[3,3]])
cols = np.array([[0,2],[0,2]])
y = x[rows,cols]
print ('The corner elements of this array are:' ,y)
```

# SORTING :

- numpy.sort()
- The sort() function returns a sorted copy of the input array. It has the following parameters –
- numpy.sort(a, axis, kind, order)

| Parameter | Description |
|---|---|
| A | Array to be sorted |
| Axis | The axis along which the array is to be sorted. If none, the array is flattened, sorting on the last axis |
| Kind | Default is quicksort |
| order | If the array contains fields, the order of fields to be sorted |

```
import numpy as np
a = np.array([[3,7],[9,1]])
Print('Our array is:‘,a)
#’Applying sort() function:'
Print(np.sort(a))
#'Sort along axis 0:'
Print(np.sort(a, axis = 0))
# Order parameter in sort function
dt = np.dtype([('name', 'S10'),('age', int)])
a = np.array([("raju",21),("anil",25),("ravi", 17),("amar",27)], dtype = dt)
Print('Our array is:‘,a)
Print('Order by name:‘ )
print np.sort(a, order = 'name')
```

# Searching Arrays :

- To search an array, use the where() method.
- EXAMPLE 1:
- import numpy as np
  arr = np.array([1, 2, 3, 4, 5, 4, 4])
  x = np.where(arr == 4)
  print(x)
- EXAMPLE 2:
- import numpy as np
  arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
  x = np.where(arr%2 == 0)
  print(x)

# Binary Search on INSERTION :

- Find the indexes where the value 7 should be inserted:
- import numpy as np
  arr = np.array([6, 7, 8, 9])
  x = np.searchsorted(arr, 7)
  print(x)
- Search From the Right Side:
- import numpy as np
  arr = np.array([6, 7, 8, 9])
  x = np.searchsorted(arr, 7, side='right')
  print(x)
- Multiple Values:
- import numpy as np
  arr = np.array([1, 3, 5, 7])
  x = np.searchsorted(arr, [2, 4, 6])
  print(x)

# NumPy Joining Array

- Joining means putting contents of two or more arrays in a single array,in NumPy we join arrays by axes.
- 1-D Array:
- import numpy as np
  arr1 = np.array([1, 2, 3])
  arr2 = np.array([4, 5, 6])
  arr = np.concatenate((arr1, arr2))
  print(arr)
- 2-D Array:
- import numpy as np
  arr1 = np.array([[1, 2], [3, 4]])
  arr2 = np.array([[5, 6], [7, 8]])
  arr = np.concatenate((arr1, arr2), axis=1)
  print(arr)

# Joining Arrays Using Stack Functions

- We can concatenate two 1-D arrays along the second axis which would result in putting them one over the other, ie. stacking.
- We pass a sequence of arrays that we want to join to the stack() method along with the axis. If axis is not explicitly passed it is taken as 0.
- import numpy as np
  arr1 = np.array([1, 2, 3])
  arr2 = np.array([4, 5, 6])
  arr = np.stack((arr1, arr2), axis=1)
  print(arr)

# Stacking Along Rows :

NumPy provides a helper function: hstack() to stack along rows.

```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.hstack((arr1, arr2))
print(arr)
```

# Stacking Along Columns :

NumPy provides a helper function: vstack()  to stack along columns.

```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.vstack((arr1, arr2))
print(arr)
```

# Stacking Along Height (depth) :

- NumPy provides a helper function: dstack() to stack along height, which is the same as depth.
- import numpy as np

```
arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.dstack((arr1, arr2))

print(arr)
```

# NumPy Splitting Array :

We use array_split() for splitting arrays, we pass it the array we want to split and the number of splits.

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

If the array has less elements than required, it will adjust from the end accordingly.

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

**Note:** Similar alternates to vstack() and dstack() are available as vsplit() and dsplit().

# Copy :

- Make a copy, change the original array, and display both arrays:
- ```
  import numpy as np
  arr = np.array([1, 2, 3, 4, 5])
  x = arr.copy()
  arr[0] = 42
  print(arr)
  print(x)
  ```
- The copy SHOULD NOT be affected by the changes made to the original array.
  ```

# View :

- Make a view, change the original array, and display both arrays:
- ```
  import numpy as np
  arr = np.array([1, 2, 3, 4, 5])
  x = arr.view()
  arr[0] = 42
  print(arr)
  print(x)
  ```
- The view SHOULD be affected by the changes made to the original array.
  ```

# Make Changes in the VIEW:

- Make a view, change the view, and display both arrays:
- ```
  import numpy as np
  arr = np.array([1, 2, 3, 4, 5])
  x = arr.view()
  x[0] = 31
  print(arr)
  print(x)
  ```
- The original array SHOULD be affected by the changes made to the view.
  ```