NUMPY

NumPy is a fundamental library for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.

Importing NumPy:

NumPy is a powerful library in Python for numerical operations. To use its functionalities, you need to import it. The conventional alias for NumPy is "np".

Syntax: import numpy as np.

Creating NumPy Arrays:

NumPy arrays are versatile data structures used to store elements of the same type. They can be created from lists or tuples and using built-in functions.

Syntax: np.array(object) (Or) np.array(object,dtype)

Using Built-in Functions:

Here.,Shape=Dimension(row,column)

Zero Array:

Syntax: np.zeros ((shape))

Ones Array:

Syntax: np.ones ((shape))

Full Array:

Syntax: np.full ((shape),value)

Array Attributes:

NumPy arrays have several attributes that provide information about their properties.

Shape:

Syntax: array.shape

Size:

Syntax: array.size

Datatype:

Syntax: array.dtype

Dimension:

Syntax: array.ndim

Indexing and Slicing:

Indexing and slicing allow you to access and manipulate array elements efficiently.

Indexing:

1D Array Indexing:  
Syntax: array [index]

2D Array Indexing:

Syntax: array [row\_index,column\_index]

Slicing:

Syntax: array [start:stop:step]

Mathematical Operations:

Perform various mathematical operations on NumPy arrays for efficient computations.

Element-wise Operations:

Element-wise operations are performed independently on each element of the array. These include addition, subtraction, multiplication, and division.

Addition:

Syntax: np.add(array1, array2)

Subtraction:

Syntax: np.subtract(array1, array2)

Multiplication:

Syntax: np.multiply(array1, array2)

Division:

Syntax: np.divide(array1, array2)

Mathematical Functions:

NumPy provides various mathematical functions that operate element-wise.

Square Root:

Syntax: np.sqrt(array)

Exponential:

Syntax: np.exp(array)

Logarithm:

Syntax: np.log(array)

Aggregation Operations:

NumPy provides functions to perform operations across all elements of the array, such as sum, mean, and product.

Sum:

Syntax: np.sum(array)

Mean:

Syntax: np.mean(array)

Product:

Syntax: np.prod(array)

Aggregate Functions on Multi-dimensional Arrays:

When working with multi-dimensional arrays, aggregation functions can be applied along specific axis.

Sum along Axis:

Syntax: np.sum(array, axis=0)

Mean along Axis:

Syntax: np.mean(array, axis=1)

Linear Algebra Operations:

NumPy includes functions for linear algebra operations such as dot products, matrix multiplications, and solving linear equations.

Dot Product:

Syntax: np.dot(array1, array2)

Matrix Multiplication:

Syntax: np.matmul(array1, array2)

Solving Linear Equations:

Syntax: np.linalg.solve(A, B)

Statistical Functions:

NumPy also provides functions for various statistical computations.

Standard Deviation:

Syntax: np.std(array)

Variance:

Syntax: np.var(array)

Reshaping Array:

Reshaping arrays in NumPy is a common operation that allows you to change the shape of an existing array without changing its data.

Reshape(): Used to give a new shape to an array without changing its data.

Syntax: array.reshape(Dimensions)

Ravel(): Returns a flattened array.

Syntax: array.ravel()

Flatten(): Similar to ravel() but returns a copy rather than a view.

Syntax: array.flatten()

Transpose(): Reverses or permutes the axis of an array.

Syntax: array.transpose()

Resize(): changes the shape and size of an array in-place

Syntax: array.resize()

Reshape(-1): Using -1 in the reshape() function allows NumPy to infer the appropriate dimension size.

Syntax: array.reshape(rows,-1)

Loading and Saving Data:

Loading and saving data in NumPy is straightforward and can be accomplished using several functions designed for different formats and purposes.

Save and load:

These functions are used to save and load binary files in .npy format, which is NumPy's own binary format.

Saving Data:

Syntax: np.save('my\_array.npy', arr)

(OR)

Savez(): These functions are used to save multiple arrays into a single .npz file, which is a zipped archive of .npy files.

Syntax: np.savez('my\_array.npy', arr1,arr2)

Loading Data:

Syntax: loaded\_arr = np.load('my\_array.npy')

print(loaded\_arr) (OR)

loaded\_arr = np.load('my\_array.npz')

Compressed Save:

Save the arrays to a compressed file named

as 'arrays\_compressed.npz'.

Syntax:

np.savez\_compressed('arrays\_compressed.npz',arr1,arr2)

Savetxt() and loadtxt(): Used to save and load data in text format.

Savetxt():

Syntax: np.savetxt('my\_array.txt', arr, delimiter=',')

Loadtxt():

Syntax: np.loadtxt('my\_array.txt', delimiter=',')

Genfromtxt():This function is used to load data from a text file with more options for handling missing values and different data types.

Syntax: np.genfromtxt('data.txt', delimiter=',')

Saving and Loading Data to a Binary File:

Used for reading and writing binary data to and from files.

Syntax:

To Save: arr.tofile('my\_array.bin')

To Load: np.fromfile('my\_array.bin', dtype=np.int64)

Handling CSV Files with pandas:

For more complex CSV operations, you can use the pandas library and then convert the data to a NumPy array.

Syntax:

import pandas as pd

# Load the CSV file into a pandas DataFrame

df = pd.read\_csv('data.csv')

# Convert the DataFrame to a NumPy array

arr = df.values

print(arr)