NumPy (Numerical Python) is a fundamental Python library for data science (DS) and machine learning. It provides fast, powerful, and flexible tools for handling large datasets efficiently.

Array:

- Fixed Size – The size of an array is usually defined at the time of creation.
- Same Data Type – All elements in an array must be of the same type.
- Indexed Access – Elements are accessed using an index, starting from 0 (zero-based index in most programming languages).
- Efficient Retrieval – Arrays allow fast access to elements using their index.

Why Use NumPy in Data Science?

1. **Performance**: NumPy arrays are much faster than Python lists because they use fixed memory allocation and optimized C-based operations.
2. **Convenience**: It provides built-in mathematical functions for complex operations.
3. **Efficiency**: NumPy uses less memory than lists and enables vectorized operations (operations on entire arrays instead of element-by-element loops).

1. Installing and Importing NumPy

pip install numpy

Then, import it in your Python script:

import numpy as np

2. Creating NumPy Arrays

A NumPy array (ndarray) is similar to a list but more efficient.

From a Python list

```
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

Multi-dimensional Array

```
matrix = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix)
```

Array with all zeros

```
zeros = np.zeros((3, 3))  # 3x3 matrix of zeros
print(zeros)
```

Array with all ones

```
ones = np.ones((2, 4))  # 2x4 matrix of ones
print(ones)
```

Random Numbers

```
random_arr = np.random.rand(3, 3)  # 3x3 matrix with random values
print(random_arr)
```

3. Array Attributes

check various properties of a NumPy array:

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


print(arr.shape)   # (2, 3) -> 2 rows, 3 columns
print(arr.size)    # 6 -> total number of elements
```

```
print(arr.dtype)   # int64 -> data type of elements
print(arr.ndim)    # 2 -> number of dimensions
```

4. Indexing and Slicing

NumPy indexing is similar to Python lists.

Access Elements

```
arr = np.array([10, 20, 30, 40, 50])
print(arr[0])   # 10
print(arr[1])   # 20
print(arr[-1])  # 50
```

Access Multi-Dimensional Elements

```
matrix = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix[0, 0])  # 1 (first row, first column)
print(matrix[1, 2])  # 6 (second row, third column)
```

Slicing Arrays

```
arr = np.array([10, 20, 30, 40, 50])
print(arr[1:4])   # [20 30 40] (index 1 to 3)
print(arr[:3])    # [10 20 30] (first 3 elements)
print(arr[::2])   # [10 30 50] (every second element)
```

5. Mathematical Operations

NumPy makes mathematical operations simple.

Basic Math

```
arr = np.array([1, 2, 3, 4])
```

```
print(arr + 2)  # [3 4 5 6]
print(arr * 3)  # [3 6 9 12]
print(arr / 2)  # [0.5 1.  1.5 2. ]
```

Element-wise Operations

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
print(arr1 + arr2)  # [5 7 9]
print(arr1 * arr2)  # [4 10 18]
```

Matrix Multiplication

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
print(np.dot(A, B))

# [[19 22]
#  [43 50]]
```

Aggregate Functions

```
arr = np.array([1, 2, 3, 4, 5])
print(np.sum(arr))    # 15 (sum of all elements)
print(np.mean(arr))   # 3.0 (average)
print(np.max(arr))    # 5 (maximum value)
print(np.min(arr))    # 1 (minimum value)
print(np.std(arr))    # 1.41 (standard deviation)
```

Reshaping and Transposing

Reshape an Array

```
arr = np.array([1, 2, 3, 4, 5, 6])
reshaped_arr = arr.reshape((2, 3))  # Reshape into 2 rows, 3 columns
print(reshaped_arr)
```