# Understanding NumPy Library

## **Introduction**

NumPy (Numerical Python) is a core library for scientific computing in Python. It provides a high-performance multidimensional array object and tools for working with these arrays. It is the foundation for many other scientific computing libraries such as SciPy, Pandas, and Matplotlib.

## **1. Why NumPy over Python**

Using NumPy over plain Python offers several advantages:

* **Performance**: NumPy is faster due to its implementation in C.
* **Memory Efficiency**: NumPy arrays consume less memory than Python lists.
* **Convenience**: Provides extensive functionality for mathematical, logical, shape manipulation, sorting, selecting, I/O, and more.
* **Interoperability**: Compatible with other scientific libraries like SciPy, Pandas, and TensorFlow.

## **2. Different Functions in NumPy**

NumPy provides a wide array of functions:

* **Array Creation**: np.array(), np.zeros(), np.ones(), np.empty(), np.arange(), np.linspace()
* **Mathematical Operations**: np.add(), np.subtract(), np.multiply(), np.divide(), np.dot(), np.sqrt()
* **Statistical Operations**: np.mean(), np.median(), np.std(), np.var()
* **Random Number Generation**: np.random.rand(), np.random.randn(), np.random.randint()

## **3. Properties of ndarrays**

* **Shape**: The dimensions of the array.

Python Code

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

print(arr.shape) # (2, 3)

**Size**: The total number of elements in the array.

Python code

print(arr.size) # 6

* **Data Type**: The type of elements stored in the array.

Python code

print(arr.dtype) # dtype('int64')

## **4. NumPy Data Types**

NumPy supports a variety of data types:

* Integer types: np.int8, np.int16, np.int32, np.int64
* Unsigned integer types: np.uint8, np.uint16, np.uint32, np.uint64
* Floating-point types: np.float16, np.float32, np.float64
* Complex types: np.complex64, np.complex128
* Boolean type: np.bool\_
* Object type: np.object\_

## **5. Different Arithmetic Operations**

* **Element-wise Operations**:

Pythonc code

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

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

print(arr1 + arr2) # [5 7 9]

print(arr1 \* arr2) # [4 10 18]

* **Dot Product**:

Python code

print(np.dot(arr1, arr2)) # 32

* **Square root and Exponential** :

Python code

print(np.sqrt(arr1)) # [1. 1.41421356 1.73205081]

print(np.exp(arr1)) # [ 2.71828183 7.3890561 20.08553692]

## **6. Indexing and Slicing for 2D and 3D Arrays**

* **2D Array**:

Python code

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

print(arr[0, 1]) # 2

print(arr[:, 1]) # [2 5]

* **3D Array**:

Python code

arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

print(arr[0, 1, 1]) # 4

print(arr[:, 1, :]) # [[3 4] [7 8]]

## **7. Transposing Arrays**

Transpose changes the axes of the array.

Python code

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

print(arr.T) # [[1 4]

# [2 5]

# [3 6]]

## **8. Swapping Arrays**

Swap two axes of an array.

Python code

arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

print(np.swapaxes(arr, 0, 1)) # [[[1 2] [5 6]]

# [[3 4] [7 8]]]

## **9. Pseudo Random Arrays**

Generate arrays with random numbers.

Python code

random\_arr = np.random.rand(2, 3) # 2x3 array of random floats between 0 and 1

random\_int\_arr = np.random.randint(0, 10, (2, 3)) # 2x3 array of random integers between 0

## **10. Masking in Arrays**

Masking is useful for filtering elements.

Python code

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

mask = arr > 3

print(arr[mask]) # [4 5]

## **11. Different Operations on 2D Arrays**

* **Sum, Mean, Standard Deviation**:

Python code

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

print(np.sum(arr)) # 21

print(np.mean(arr)) # 3.5

print(np.std(arr)) # 1.707825127659933

* **Matrix Multiplication**:

Python code

arr1 = np.array([[1, 2], [3, 4]])

arr2 = np.array([[5, 6], [7, 8]])

print(np.matmul(arr1, arr2)) # [[19 22]

# [43 50]]

## **12. Universal Functions**

* **Arithmetic Functions**: np.add(), np.subtract(), np.multiply(), np.divide()
* **Trigonometric Functions**: np.sin(), np.cos(), np.tan()
* **Exponential and Logarithmic Functions**: np.exp(), np.log(), np.log10()
* **Statistical Functions**: np.mean(), np.median(), np.std(), np.var()
* **Comparison Functions**: np.greater(), np.less(), np.equal(), np.not\_equal()
* **Broadcasting**: Allows operations on arrays of different shapes.

Python code

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

scalar = 2

print(arr + scalar) # [3 4 5]

## **13. Playing with Shapes**

* **Reshape**:

Python code

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

reshaped\_arr = arr.reshape((3, 2))

* **Resize**:

Python code

arr.resize((3, 2))

* **Ravel and Flatten**:

Python code

print(arr.ravel()) # [1 2 3 4 5 6]

print(arr.flatten()) # [1 2 3 4 5 6]

* **Squeeze and Expand\_dims**:

Python code

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

print(arr.squeeze()) # [1 2 3]

print(np.expand\_dims(arr, axis=0).shape) # (1, 1, 1, 3)

## **14. Splitting and Joining Arrays**

* **Splitting**:

Python code

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

split\_arr = np.split(arr, 3) # [array([1, 2]), array([3, 4]), array([5, 6])]

* **Joining**:

Python code

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

arr2 = np.array([3, 4])

print(np.concatenate((arr1, arr2))) # [1 2 3 4]

## **15. Basic Information on Arrays**

* **Creation**:

Python code

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

* **Attributes**: shape, size, dtype
* **Methods**: reshape(), flatten(), ravel(), transpose()

Top of Form

Bottom of Form