 **NumPy**

**Syed Mansoor-ul Hassan Bukhari**

**NumPy:**

NumPy (Numerical Python) is a fundamental library for scientific computing in Python. It provides a powerful data structure—the n-dimensional array—that serves as the foundation for much of Python’s data science toolkit. Whether you’re a beginner or an experienced programmer, understanding NumPy is essential for working with data.

In this comprehensive tutorial, you’ll learn the core concepts of NumPy, how to create arrays, manipulate them, and apply these skills to real-world problems. Let’s dive in:

1. **Choosing NumPy: The Benefits**
   * **Speed**: NumPy uses C-based algorithms that execute in nanoseconds, making it faster than traditional Python loops.
   * **Fewer Loops**: NumPy reduces the need for explicit loops, leading to cleaner code.
   * **Clearer Code**: Without excessive loops, your code resembles the mathematical equations you’re working with.
2. **Installing NumPy**:
   * You can install NumPy using various methods:
     + With Anaconda: conda install numpy
     + With pip: pip install numpy
     + Using online editors like Repl.it or IPython, Notebooks, or JupyterLab.
3. **Hello NumPy: Curving Test Grades Tutorial**:
   * Create NumPy arrays using different methods.
   * Perform basic calculations and manipulations on arrays.
4. **Getting Into Shape: Array Shapes and Axes**:
   * Understand array shapes (dimensions) and axes (dimensions along which operations occur).
5. **Broadcasting**:
   * Learn how NumPy broadcasts arrays to perform element-wise operations.
6. **Data Science Operations**:
   * Filtering, ordering, and aggregating data.
   * Indexing, masking, and filtering arrays.
   * Transposing, sorting, and concatenating arrays.
7. **Optimizing Storage: Data Types**:
   * Explore numerical types (int, bool, float, complex).
   * Understand string types and structured arrays.
8. **Creating NumPy Arrays**:
   * Import NumPy: After installation, use import numpy as np.
   * Create arrays:
     + From lists: np.array([1, 2, 3])
     + Zeros/ones: np.zeros((3, 4)), np.ones((2, 2))
     + Random: np.random.rand(2, 3)
     + Range: np.arange(0, 10, 2)
9. **Array Properties**:
   * Shape: Dimensions (e.g., (2, 3) for a 2x3 array).
   * Data type: Homogeneous elements (int, float, etc.).
   * Size: Total number of elements.
10. **Manipulating Arrays**:
    * Indexing: Access elements (e.g., arr[0, 1]).
    * Slicing: Extract subarrays (e.g., arr[:, 1:3]).
    * Reshaping: Change dimensions (e.g., arr.reshape(3, 2)).
    * Broadcasting: Perform element-wise operations.
11. **Common Operations**:
    * Mathematical: +, -, \*, /, np.dot().
    * Aggregation: sum(), mean(), max(), min().
    * Sorting: np.sort(), np.argsort().
12. **Data Types**:
    * Integers: int8, int16, int32, int64.
    * Floating-point: float16, float32, float64.
    * Complex: complex64, complex128.

Let’s dive into the essential NumPy functions and their usage. I’ll provide code examples for each function

**Creating Arrays**:

np.array(): Create an array from a list or tuple.

import numpy as np

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

1. **Array Properties**:
   * arr.shape: Get the shape (dimensions) of the array.
   * arr.ndim: Get the number of dimensions.
   * arr.dtype: Get the data type of elements in the array.
2. **Mathematical Operations**:
   * Element-wise operations:

result = arr + 10

* + Matrix multiplication:

matrix\_product = np.dot(arr1, arr2)

1. **Indexing and Slicing**:
   * Access elements:

element = arr[2]

* + Slice arrays:

subarray = arr[1:4]

1. **Reshaping Arrays**:
   * Change array shape:

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

1. **Joining and Splitting Arrays**:
   * Join arrays:

joined\_arr = np.concatenate((arr1, arr2))

* + Split arrays:

split\_arr = np.array\_split(arr, 3)

1. **Searching and Filtering**:
   * Find indices where a condition is met:

indices = np.where(arr > 3)

* + Filter elements based on a condition:

filtered\_arr = arr[arr > 2]

1. **Sorting Arrays**:
   * Sort an array:

sorted\_arr = np.sort(arr)

1. **Matrix Operations**:
   * Create matrices:

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

* + Transpose a matrix:

transposed\_matrix = matrix.T

1. **Random Number Generation**:
   * Generate random numbers:

random\_numbers = np.random.rand(5)

Let’s delve into some advanced concepts of NumPy. These topics will deepen your understanding and empower you to work more effectively with this powerful library:

1. **Broadcasting**:
   * Broadcasting allows NumPy to perform element-wise operations on arrays with different shapes. It enables efficient computation without explicitly replicating data.
   * Example:

import numpy as np

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

b = np.array([[10], [20], [30]])

result = a + b # Broadcasting adds 10, 20, 30 to each element of 'a'

1. **Advanced Indexing**:
   * Beyond basic slicing, NumPy supports advanced indexing techniques like boolean indexing, integer arrays, and fancy indexing.
   * Example:

arr = np.array([10, 20, 30, 40, 50])

mask = arr > 20

filtered\_values = arr[mask] # Select elements where the condition is True

1. **Linear Algebra**:
   * NumPy provides functions for matrix operations, eigenvalues, eigenvectors, and solving linear equations.
   * Example:

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

eigenvalues, eigenvectors = np.linalg.eig(A)

1. **Stacking and Splitting**:
   * Combine or split arrays along specified axes.
   * Example:

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

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

stacked = np.vstack((arr1, arr2)) # Stack vertically

1. **Memory Views and Strides**:
   * Understand how NumPy handles memory layout and strides for efficient data access.
   * Example:

arr = np.arange(12).reshape(3, 4)

view = arr[::2, ::2] # Create a memory view with custom strides

1. **Universal Functions (ufuncs)**:
   * NumPy’s ufuncs provide element-wise operations (e.g., sin, cos, exp) that are optimized for performance.
   * Example:

angles = np.linspace(0, 2 \* np.pi, 100)

sine\_values = np.sin(angles)