NumPy Notes: A Beginner's Guide.

What is NumPy?

- NumPy stands for Numerical Python.
- It is a powerful library in Python used for working with arrays (lists of numbers) and performing mathematical operations efficiently.

2. Getting Started with NumPy

1. Importing NumPy

To use NumPy in your Python programs, you first need to import it:

```
import numpy as np
```

Here, np is a common alias for NumPy, making it easier to reference in your code.

2. Creating Your First Array

Arrays are central to NumPy. They are similar to lists in Python but come with additional functionalities.

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

Output:

```
[1 2 3 4]
```

This creates a simple one-dimensional array with four elements.

3. Doing Math with Arrays

NumPy allows you to perform mathematical operations on entire arrays quickly and easily.

Example: Adding Two Arrays

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

print(arr1 + arr2)
```

Output:

```
[5 7 9]
```

Each element in `arr1` is added to the corresponding element in `arr2`.
We added two arrays element-wise, so `1+4=5`, `2+5=7`, and `3+6=9`.

Working with Array Shapes

1. Reshaping Arrays

You can change the shape (dimensions) of an array. For example, you can turn a one-dimensional array into a two-dimensional array.

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

Output:

```
[[1 2]
 [3 4]]
```

The array is now reshaped into a 2x2 matrix.

2. Flattening Arrays

Flattening converts a multi-dimensional array into a one-dimensional array.

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

Output:

```
[1 2 3 4]
```

The 2x2 matrix was flattened back into a single row.

Indexing and Slicing: Accessing Array Elements

1. Basic Indexing

Indexing is used to access individual elements in an array.

Example:

```
arr = np.array([10, 20, 30])
print(arr[1])
```

Output:

```
20
```

Here, `arr[1]` gives you the element at index 1 (which is `20`).

2. Slicing

Slicing is used to access a range of elements in an array.

Example:

```
arr = np.array([10, 20, 30, 40])
print(arr[1:3])
```

Output:

```
[20 30]
```

We sliced the array to get elements from index 1 to 2 (index 3 is not included).

Broadcasting: Easy Math Operations

Broadcasting allows you to perform operations between arrays of different shapes or between an array and a single value.

Example: Adding a Scalar to an Array

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

Output:

```
[6 7 8]
```

Here, 5 is added to each element of the array.

Advanced Array Operations: Matrix Math

NumPy makes it easy to perform operations on matrices (2D arrays), such as matrix multiplication.

Example:

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

Output:

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

Matrix multiplication results in a new matrix where each element is the sum of the products of corresponding elements.

Why Use NumPy?

- **Efficiency**: NumPy operations are faster and require less memory than traditional Python lists.
- **Convenience**: It provides a wide range of mathematical functions and easy array manipulation.
- **Versatility**: It's widely used in data science, machine learning, and scientific computing.

Key Points to Remember

- **Arrays** are like super-powered lists.
- You can perform **math operations** on entire arrays with a single line of code.
- Use **indexing and slicing** to access or modify specific elements.
- **Reshape** arrays to change their dimensions.
- NumPy supports **advanced math operations**, making it a powerful tool for numerical computing.

Next Steps

- Practice creating and manipulating arrays using NumPy.
- Explore more advanced features like statistical operations and data handling with NumPy.

- Combine NumPy with other libraries like pandas and Matplotlib for data analysis and visualization.