**NumPy Explained: A Deep Dive into Key Concepts**

NumPy is a powerful Python library for numerical computing. It provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these structures efficiently. Below, we explore key NumPy topics:

**1. Array Creation**

NumPy provides various methods to create arrays:

* **From Python Lists**: np.array([1, 2, 3])
* **Using arange**: np.arange(0, 10, 2) # [0, 2, 4, 6, 8]
* **Using linspace**: np.linspace(0, 1, 5) # [0. 0.25 0.5 0.75 1.]
* **Using zeros, ones, full**:
* np.zeros((2,2)) # [[0. 0.]
* # [0. 0.]]
* **Random Arrays**: np.random.rand(3, 3) # Random values in [0, 1]

**2. Basic Operations**

NumPy allows element-wise operations:

* **Arithmetic Operations**
* a = np.array([1, 2, 3])
* b = np.array([4, 5, 6])
* print(a + b) # [5 7 9]
* print(a \* b) # [4 10 18]
* print(a \*\* 2) # [1 4 9]
* **Aggregation Functions**:
* arr = np.array([1, 2, 3, 4])
* print(arr.sum()) # 10
* print(arr.mean()) # 2.5
* print(arr.min()) # 1
* print(arr.max()) # 4

**3. Universal Functions (ufuncs)**

Universal functions (ufuncs) operate element-wise on arrays and are optimized for performance.

Examples:

arr = np.array([0, np.pi / 2, np.pi])

print(np.sin(arr)) # [0. 1. 0.]

print(np.exp(arr)) # Exponential function

print(np.log(arr + 1)) # Logarithm (avoiding log(0) error)

**4. Indexing, Slicing, and Iterating**

NumPy follows zero-based indexing like Python.

* **Indexing**:
* arr = np.array([10, 20, 30, 40])
* print(arr[2]) # 30
* **Slicing**:
* arr = np.array([0, 1, 2, 3, 4, 5])
* print(arr[1:4]) # [1 2 3]
* print(arr[:3]) # [0 1 2]
* print(arr[-2:]) # [4 5]
* **Multidimensional Indexing**:
* arr = np.array([[1, 2, 3], [4, 5, 6]])
* print(arr[1, 2]) # 6
* print(arr[:, 1]) # [2 5] (All rows, column index 1)
* **Iterating**:
* for row in arr:
* print(row)

**5. Reshaping Arrays**

NumPy allows changing the shape of an array without modifying data.

* **Reshaping**:
* arr = np.arange(6).reshape(2, 3)
* print(arr)
* # [[0 1 2]
* # [3 4 5]]
* **Flattening**:
* print(arr.ravel()) # [0 1 2 3 4 5]

**6. Images as Arrays**

Images can be represented as NumPy arrays where each pixel is a numerical value.

* **Grayscale Image**: 2D array
* **Color Image (RGB)**: 3D array (height × width × 3)

import matplotlib.pyplot as plt

import numpy as np

img = np.random.rand(100, 100) # Random grayscale image

plt.imshow(img, cmap="gray")

plt.show()

**7. Image Manipulation**

NumPy provides tools to manipulate images:

* **Cropping**:
* cropped = img[10:50, 20:60]
* **Flipping**:
* flipped = np.flip(img, axis=0) # Vertical flip
* **Color Channel Manipulation (for RGB images)**:
* img[:, :, 0] = 0 # Remove red channel

**8. Views and NumPy Arrays**

* **View (Shallow Copy)**:
* a = np.array([1, 2, 3])
* b = a.view()
* b[0] = 99
* print(a) # [99 2 3] (Changes reflect in `a` as well)
* **Copy (Deep Copy)**:
* b = a.copy()
* b[0] = 42
* print(a) # [99 2 3] (Original remains unchanged)

**9. Deep Copies of Arrays**

* np.copy() ensures that modifying one array does not affect the other.

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

b = np.copy(a)

b[0] = 100

print(a) # [1 2 3] (No change)

**10. Index Masks**

NumPy allows boolean masking to filter elements.

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

mask = arr > 15

print(arr[mask]) # [20 30 40]

**11. Array Broadcasting**

Broadcasting allows operations on arrays of different shapes.

Example:

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

arr2 = np.array([[10], [20], [30]])

result = arr1 + arr2

print(result)

# [[11 12 13]

# [21 22 23]

# [31 32 33]]

This automatically expands arr1 to match arr2.