# 📘 NumPy – Day 1 Notes

**Topic:** Introduction & Array Basics

## 🔹 1. What is NumPy?

* **NumPy = Numerical Python**
* A Python library used for **fast numerical and scientific computing**.
* Provides:  
  + Multidimensional arrays (ndarray)
  + Mathematical functions
  + Data manipulation tools

✅ **Why NumPy over Python Lists?**

* Faster (written in C, optimized)
* Less memory usage
* Vectorized operations (no loops needed)

## 🔹 2. Importing NumPy

**import numpy as np**

## 🔹 3. Creating Arrays

### From Python Lists

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

**arr2 = np.array([[1, 2], [3, 4]])** # 2D array

**arr3 = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])** # 3D array

**print(arr1)**  # [1 2 3 4]

**print(type(arr1))** # <class 'numpy.ndarray'>

### Special Arrays

**np.zeros((2, 3))** # 2x3 array filled with 0s

**np.ones((3, 3))** # 3x3 array filled with 1s

**np.eye(3)**  # Identity matrix (3x3)

**np.arange(1, 10, 2)**  # [1, 3, 5, 7, 9]

**np.linspace(0, 1, 5)** # 5 numbers between 0 and 1-Evenly spaced numbers

| **np.full((2,2), 7)** |  | #Array filled with 7 |
| --- | --- | --- |

## 🔹 4. Array Attributes

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

**[4, 5, 6]])**

**print(arr.ndim)** # Dimensions → 2

**print(arr.shape)** # Shape → (2, 3)

**print(arr.size)** # Total elements → 6

**print(arr.dtype)** # Data type → int64

## 🔹 5. Indexing & Slicing

## 🔹 1. Indexing in NumPy

Indexing means accessing elements from an array.

### 1D Array

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

**print(a[0])** # First element → 10

**print(a[-1])**  # Last element → 50

**print(a[2])**  # Third element → 30

### 2D Array (Rows & Columns)

**b = np.array([[1, 2, 3],**

**[4, 5, 6],**

**[7, 8, 9]])**

**print(b[0, 0])**  # Row 0, Col 0 → 1

**print(b[1, 2])** # Row 1, Col 2 → 6

**print(b[2, -1])**  # Last row, last column → 9

👉 Format: array[row\_index, column\_index]

### Higher Dimensions (3D Example)

**c = np.array([[[1, 2], [3, 4]],**

**[[5, 6], [7, 8]]])**

**print(c[0, 1, 1])**  # → 4 (block 0, row 1, col 1)

**print(c[1, 0, 0])** # → 5 (block 1, row 0, col 0)

## 🔹 2. Slicing in NumPy

Slicing means selecting **a range of elements**.  
 👉 Syntax: array[start:stop:step]

### 1D Slicing

**a = np.array([10, 20, 30, 40, 50, 60, 70])**

**print(a[1:5])**  # [20 30 40 50]

**print(a[:4])**  # [10 20 30 40] (from start to index 3)

**print(a[3:])**  # [40 50 60 70] (from index 3 to end)

**print(a[::2])**  # [10 30 50 70] (step = 2)

### 2D Slicing

**b = np.array([[1, 2, 3],**

**[4, 5, 6],**

**[7, 8, 9]])**

**print(b[0:2, 1:3])** # Rows 0-1, Cols 1-2 → [[2 3], [5 6]]

**print(b[:, 0])** # All rows, first column → [1 4 7]

**print(b[1, :])** # Entire row 1 → [4 5 6]

**print(b[:, ::2])**  # All rows, every 2nd column → [[1 3], [4 6], [7 9]]

## 🔹 3. Advanced Indexing

NumPy allows **fancy indexing** using lists/arrays of indices.

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

**print(a[[0, 2, 4]])** # [10 30 50] (pick multiple positions)

### 2D Fancy Indexing

**b = np.array([[1, 2, 3],**

**[4, 5, 6],**

**[7, 8, 9]])**

**rows = [0, 2]**

**cols = [1, 2]**

**print(b[rows, cols])** # [2 9] (elements at (0,1) and (2,2))

## 🔹 6. List vs NumPy Performance

**import time**

**lst = list(range(1000000))**

**arr = np.arange(1000000)**

**# Sum using list**

**start = time.time()**

**sum(lst)**

**print("List time:", time.time() - start)**

**# Sum using NumPy**

**start = time.time()**

**arr.sum()**

**print("NumPy time:", time.time() - start)**

✅ NumPy is much **faster** ⚡

## 📝 Mini Exercise (Classwork)

1. Create a 1D array of numbers 10–50.
2. Extract numbers from index 5 to 10.
3. Create a 3×3 array with numbers 1–9.
4. Slice the **second row** and **first column**.

# 📘 NumPy – Day 2 Notes

**Topic:** Array Operations, Reshaping & Broadcasting

## 🔹 1. Vectorized Operations

Instead of loops, NumPy allows element-wise operations directly.

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

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

**print(a + b)**  # [11 22 33 44]

**print(a - b)** # [ -9 -18 -27 -36]

**print(a \* b)** # [ 10 40 90 160]

**print(a / b)** # [0.1 0.1 0.1 0.1]

**print(a \*\* 2)**  # [ 1 4 9 16]

✅ Much faster than using for loops.

## 🔹 2. Aggregate Functions

Quick mathematical operations across arrays.

**arr = np.array([5, 10, 15, 20, 25])**

**print(arr.min())** # 5

**print(arr.max())**  # 25

**print(arr.sum())** # 75

**print(arr.mean())** # 15.0

**print(arr.std())**  # 7.07 (Standard Deviation)

👉 Works on 2D arrays too:

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

**print(mat.sum(axis=0))** # Column-wise sum → [5 7 9]

**print(mat.sum(axis=1))** # Row-wise sum → [6 15]

## 🔹 3. Reshaping Arrays

Change the structure without changing data.

**a = np.arange(12)** # [0 1 2 ... 11]

**print(a.reshape(3, 4))** # 3x4 matrix

**print(a.reshape(2, 6))** # 2x6 matrix

**print(a.reshape(4, 3))** # 4x3 matrix

**print(a.flatten())**  # Converts to 1D

**print(a.reshape(3, -1))**  # NumPy auto-calculates column size

## 🔹 4. Transpose

Flip rows & columns.

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

**print(mat.T)**

# [[1 4]

# [2 5]

# [3 6]]

## 🔹 5. Stacking & Splitting

### Stacking (Combine Arrays)

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

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

**print(np.hstack((a, b)))** # [1 2 3 4 5 6]

**print(np.vstack((a, b)))**

# [[1 2 3]

# [4 5 6]]

### Splitting (Divide Arrays)

**x = np.arange(1, 10)**

**print(np.split(x, 3))**  # 3 equal parts

# [array([1, 2, 3]), array([4, 5, 6]), array([7, 8, 9])]