Introduction to NumPy

Page 1 – Why NumPy Matters

1 | What is NumPy?

NumPy (Numerical Python) is the **fundamental numerical-computing library for Python**. It introduces a fast, memory-efficient *n-dimensional array object* (`ndarray`) plus a rich collection of mathematical functions that operate on these arrays.

2 | Why Use NumPy Instead of Pure Python?

Pure-Python Lists	NumPy Arrays
Store elements as separate Python objects → **slow & memory-heavy**	Store homogeneous, C-contiguous data → **up to 50× faster**
Loops are explicit, written in Python	Most operations are **vectorised** in C
Limited math utilities	1,000+ ready-made numerical functions

> **Bottom line:** If you plan to crunch numbers in Python—data science, machine learning, image processing, signal analysis—NumPy is usually your first import.

3 | Installing NumPy

```
# Using pip (PyPI)
pip install numpy

# Or via conda
conda install numpy
```

NumPy is also bundled with popular scientific distributions like **Anaconda** and **Miniconda**.

4 | Creating Your First Array

```
import numpy as np

a = np.array([1, 2, 3])       # 1-D array
b = np.zeros((2, 3))          # 2x3 array of 0s
c = np.random.randn(3, 3)     # 3x3 array of Gaussian random numbers
```

Every NumPy program starts with `import numpy as np`—a community convention.

Page 2 – Core Concepts & Everyday Operations

5 | Key Array Attributes

```
>>> x = np.arange(12).reshape(3,4)
>>> x.ndim        # Number of dimensions
2
>>> x.shape       # Tuple of axis lengths
(3, 4)
>>> x.dtype       # Data type of elements
dtype('int64')
>>> x.nbytes      # Memory usage in bytes
96
```

- **ndim** – dimensionality (1-D, 2-D, …)
- **shape** – size along each axis
- **dtype** – fixed-width type (`int32`, `float64`, `bool`, custom)

6 | Slicing & Broadcasting

```
y = x[:, 1:3]     # View: all rows, cols 1-2
y[0, 0] = 999     # Modifies *x* too (views share data)

# Broadcasting: combine shapes (3,4) and (1,4)
row_sums = x + np.array([10, 20, 30, 40])
```

- **Views, not copies**: most slices point to the same memory—fast but beware of side effects.
- **Broadcasting** automatically stretches compatible shapes, eliminating explicit loops.

7 | Vectorised Maths

```
# Element-wise operations
z = np.sin(x) + np.log1p(x)

# Reduction operations
total = x.sum()          # sum of all elements
col_means = x.mean(axis=0)
```

Compute millions of operations per second in underlying C/Fortran—no Python loops, just concise code.

8 | Interoperability & the SciPy Ecosystem

- **Pandas** uses NumPy arrays under the hood for DataFrames.
- **Matplotlib**, **SciPy**, **scikit-learn**, **TensorFlow/PyTorch** all accept or output `ndarray`s.
- `ndarray` can **share memory** with C/C++, Fortran, and GPU libraries via the buffer protocol.

9 | Where to Go Next

1. **Indexing tricks**: boolean masks, fancy indexing.
2. **Linear algebra**: `np.linalg.svd`, `np.matmul`, eigen-decomposition.
3. **Random sampling**: `numpy.random` (new API in NumPy 1.22+).
4. **Performance tuning**: `numba`, `cython`, or move to the GPU with **CuPy**.

✨ Summary

NumPy turns Python into a **powerful, production-grade numerical platform**. Master its arrays, and the rest of the PyData stack falls into place. Happy vectorising!