PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Introduction to NumPy

UE25CS151A

Department of Computer Science and Engineering
Prof. Kundhavai K R, CSE Department

Introduction to NumPy

What is NumPy?

- NumPy = **Numerical Python**

- Core library for scientific & numerical computing

- Provides the **ndarray** → fast, efficient N-dimensional array

Introduction to NumPy

Why NumPy?

- **Fast:** Operations written in C → faster than Python lists
- **Memory Efficient:** Arrays use less memory
- **Convenient:** Many built-in mathematical & vectorized operations

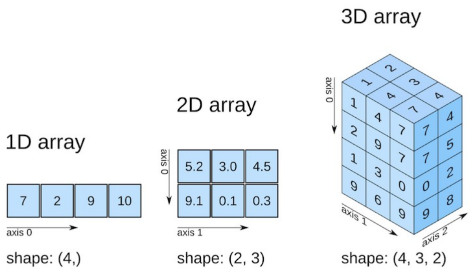
Installation & Import

```
pip install numpy
import numpy as np
```

NumPy Arrays: The Basics

NumPy arrays:

- Store values of **same data type**
- Can be:
  - **1D (Vector)**
  - **2D (Matrix)**
  - **N-Dimensional (Tensor)**



Array Creation and Manipulation

Create Arrays

np.zeros((3,3)) **Output:** [[0. 0. 0.] [0. 0. 0.] [0. 0. 0.]]

np.ones((2,4)) **Output:** [[1. 1. 1. 1.] [1. 1. 1. 1.]]

np.arange(10) **Output:** [0 1 2 3 4 5 6 7 8 9]

Array Creation and Manipulation

Basic Operations

```
a = np.array([1,2,3])
b = np.array([4,5,6])
a + b
```

Output:

```
 [5 7 9]
```

```
a * 2
```

Output:

```
[2 4 6]
```

Array Creation and Manipulation

Sorting

```
arr = np.array([4,1,6,2])

np.sort(arr)
```

Output:

```
 [1 2 4 6]
```

Concatenation

```
x = np.array([1,2])

y = np.array([3,4])
```

Array Creation and Manipulation

Reshaping

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

Output:

```
 [[1 2 3] [4 5 6]]
```

Array Attributes

Every array has:

- ndim → number of dimensions
  shape → size of each dimension
  size → total elements
  dtype → data type

Example:

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

Outputs:

- arr.ndim → 2
  arr.shape → (2,3)
  arr.size → 6
  arr.dtype → int64 / System dependent

Array Indexing

1. Indexing & Slicing (1D Arrays)

```
arr = np.arange(10)    # [0 1 2 3 4 5 6 7 8 9]
arr[3]       # 3
arr[2:5]      # [2 3 4]
```

2. Indexing (2D Arrays)

```
matrix = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]])

matrix[1, 2]   # 6   (row 2, column 3)
```

3. Boolean Indexing

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

        data[data > 25]    # [30 40 50]
```

Aggregation Functions

What Are Aggregation Functions?

Aggregation functions compute summary statistics from an array — such as max, min, average, and spread of values. They are essential for data analysis.

Common Aggregation Functions

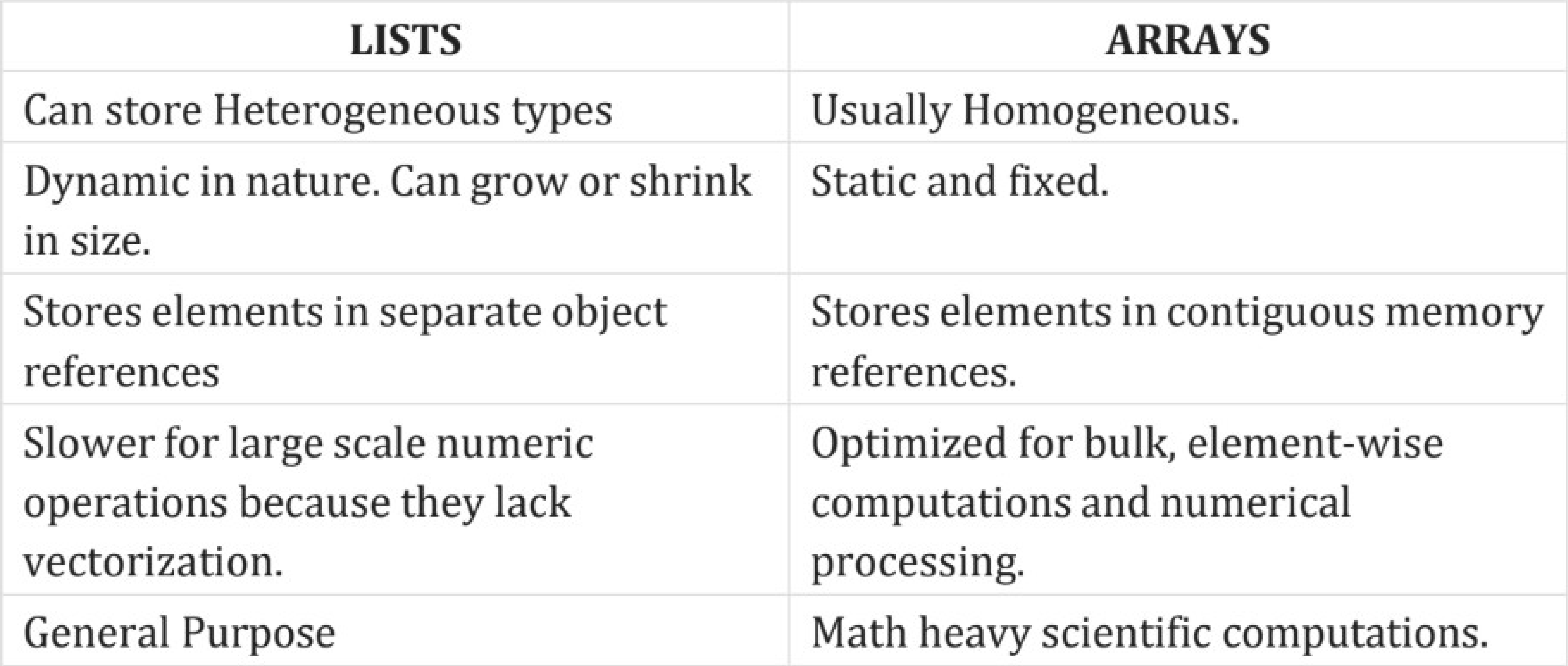
```
np.max(arr)     # Highest value

np.min(arr)     # Lowest value

np.mean(arr)    # Average (mean)

np.std(arr)     # Standard deviation (spread)
```

Arrays vs Lists



LISTS	ARRAYS
Can store Heterogeneous types	Usually Homogeneous.
Dynamic in nature. Can grow or shrink in size.	Static and fixed.
Stores elements in separate object references	Stores elements in contiguous memory references.
Slower for large scale numeric operations because they lack vectorization.	Optimized for bulk, element-wise computations and numerical processing.
General Purpose	Math heavy scientific computations.

THANK YOU

Department of Computer Science and Engineering

Prof. Kundhavai K R, CSE Department

Ack: Teaching Assistant:
Adithya Jeyaramsankar– PES2UG22CS029