✅ Day 2: NumPy Arrays – Creation, Data Types & Basic Properties

📌 1. What is an ndarray?

A **NumPy array** is an **n-dimensional** array object called **`ndarray`**.

Key features:

- All elements in an array are of the **same type** (homogeneous)
- Arrays can have any number of **dimensions**: 1D, 2D, 3D, etc.
- Arrays are **contiguous in memory**, enabling fast access and operations

🧠 Why Arrays over Lists in Data Engineering?

Feature	Python List	NumPy ndarray
Type	Heterogeneous	Homogeneous
Speed	Slower	Much faster
Memory	More	Less
Vectorized Ops	No	Yes
Use Case	Small datasets	Large datasets, real-time logs, telemetry

In real-time data pipelines, where **speed, memory efficiency, and structure** matter, `ndarray` is the go-to structure.

🛠️ 2. Creating NumPy Arrays

✅ 1. From Python Lists / Tuples

```
import numpy as np

# 1D Array
arr1d = np.array([1, 2, 3, 4])
print(arr1d)

# 2D Array
arr2d = np.array([[1, 2], [3, 4]])
print(arr2d)
```

✅ 2. Using Built-in NumPy Functions

Function	Description
`np.zeros()`	Creates array filled with 0s
`np.ones()`	Creates array filled with 1s
`np.full()`	Creates array filled with a specific value
`np.arange()`	Creates evenly spaced values within a range
`np.linspace()`	Creates evenly spaced numbers over a specified interval
`np.eye()`	Creates identity matrix
`np.random.rand()`	Creates array with random values (0 to 1)
`np.random.randint()`	Creates random integers within a range

```
zeros = np.zeros((3, 3))
ones = np.ones((2, 2))
filled = np.full((2, 3), 99)
arng = np.arange(0, 10, 2)
lin = np.linspace(0, 1, 5)
identity = np.eye(3)
rand_uniform = np.random.rand(2, 2)
rand_int = np.random.randint(1, 100, size=(3, 3))
```

🎯 Real-Time Use Case:

Let’s say your IoT device is expected to send 5000 values every second from each sensor.

```
# Simulating 5000 random values between 10°C and 100°C
sensor_data = np.random.uniform(10, 100, size=5000)

# Flagging risky readings
risk = sensor_data > 90
print("Risky entries:", np.sum(risk))
```

Efficient memory usage and fast operations make NumPy arrays perfect for such pipelines.

🔍 3. Understanding Array Attributes

Attribute	Description	Example
`ndim`	Number of dimensions	`arr.ndim`
`shape`	Tuple of dimensions	`arr.shape`
`size`	Total number of elements	`arr.size`
`dtype`	Data type of elements	`arr.dtype`
`itemsize`	Size in bytes of one element	`arr.itemsize`
`nbytes`	Total bytes consumed	`arr.nbytes`

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

print("Dimensions:", arr.ndim)
print("Shape:", arr.shape)
print("Size:", arr.size)
print("Data type:", arr.dtype)
print("Item size (bytes):", arr.itemsize)
print("Total bytes:", arr.nbytes)
```

🔢 4. NumPy Data Types (`dtype`)

NumPy provides support for many data types:

Data Type	Code	Description
`int32`, `int64`	`'i'`	Integer
`float32`, `float64`	`'f'`	Floating-point
`bool_`	`'b'`	Boolean
`complex64`	`'c'`	Complex numbers
`object_`	`'O'`	Generic Python object

Example:

```
arr_int = np.array([1, 2, 3], dtype=np.int32)
arr_float = np.array([1.5, 2.3], dtype=np.float64)
```

🏗️ 5. Changing Data Types

Use `.astype()` to convert:

```
arr = np.array([1.5, 2.7, 3.9])
arr_int = arr.astype(int)
```

🧪 6. Practical Example: Converting Sensor Values to Int for Storage

```
float_readings = np.random.uniform(20, 30, size=10)
int_readings = float_readings.astype(np.int32)
```

This can reduce **storage cost** or **database size** in ADF pipelines or ADLS Gen2.

📚 7. Summary of Day 2

Topic	Summary
`ndarray`	n-dimensional fast array object
Creation	Lists, zeros, ones, arange, random, etc.
dtype	Data type of all elements
Attributes	ndim, shape, size, dtype, itemsize, nbytes
Real-time Use	Efficient storage, fast filtering, numeric transformation

---

🧠 Interview Questions – Day 2

1. What is the difference between Python list and NumPy ndarray?
2. What are various ways to create a NumPy array?
3. What is the difference between `arange()` and `linspace()`?
4. How do you change the data type of an ndarray?
5. What are the key attributes of a NumPy array and why are they important?

✅ Day 2 Tasks for Practice

🧪 Coding Practice:

1. Create arrays using different creation methods (zeros, ones, arange, linspace, full, etc.)
2. Simulate random sensor data and filter high-risk entries

3. Practice using `dtype`, `.astype()`, and array attributes