🧩 Course 2 — Data Structures with Python

Week 1: Introduction to Data Structures

🎯 Learning Objectives

By the end of this week, students will be able to: -

✅ Define what a data structure is and explain why it is essential in programming.

✅ Differentiate between **linear** and **non-linear** data structures.

✅ Understand and calculate **time and space complexity** using Big-O notation.

✅ Recognize trade-offs between different data representations.

✅ Write simple Python programs that demonstrate basic data manipulation.

📘 1. Topic Overview: What are Data Structures?

A **data structure** is a *systematic way of organizing and storing data* so it can be accessed and modified efficiently.
Every program relies on data — numbers, text, records — and how you *structure* that data affects performance, scalability, and clarity.

Real-world analogy:

Think of a data structure as a **toolbox**.
Each compartment (array, stack, queue, etc.) serves a purpose.
Choosing the right one determines how fast and easy you can retrieve your tools (data).

🧠 2. Why Data Structures Matter

Efficient data structures: - Reduce memory usage - Improve runtime performance - Make code scalable - Simplify debugging and maintenance

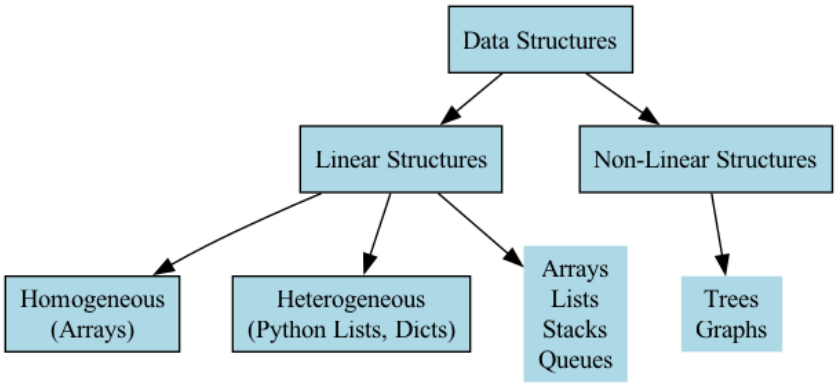
Example:

```
# Searching for a name in a list
names = ["Alice", "Bob", "Charlie", "Diana"]
if "Charlie" in names:
    print("Found!")
```

But what if there are **10 million names**? You'd need more efficient structures (e.g., *hash tables*, *binary search trees*) to search faster.

⚙️ 3. Classification of Data Structures

Type	Description	Examples
Linear	Elements arranged sequentially	Arrays, Lists, Stacks, Queues
Non-linear	Hierarchical relationships	Trees, Graphs
Homogeneous	Same data type elements	Arrays
Heterogeneous	Different data types	Python Lists, Dictionaries



Data Structure Classification Diagram

🧩 4. Abstract Data Types (ADT)

An **Abstract Data Type (ADT)** defines *what* operations can be performed, not *how* they are implemented.

Example:

- **Stack ADT** operations: `push`, `pop`, `peek`
- Implementation could use **list**, **deque**, or even **linked list**

ADT gives conceptual clarity — the behavior matters more than the mechanics.

---

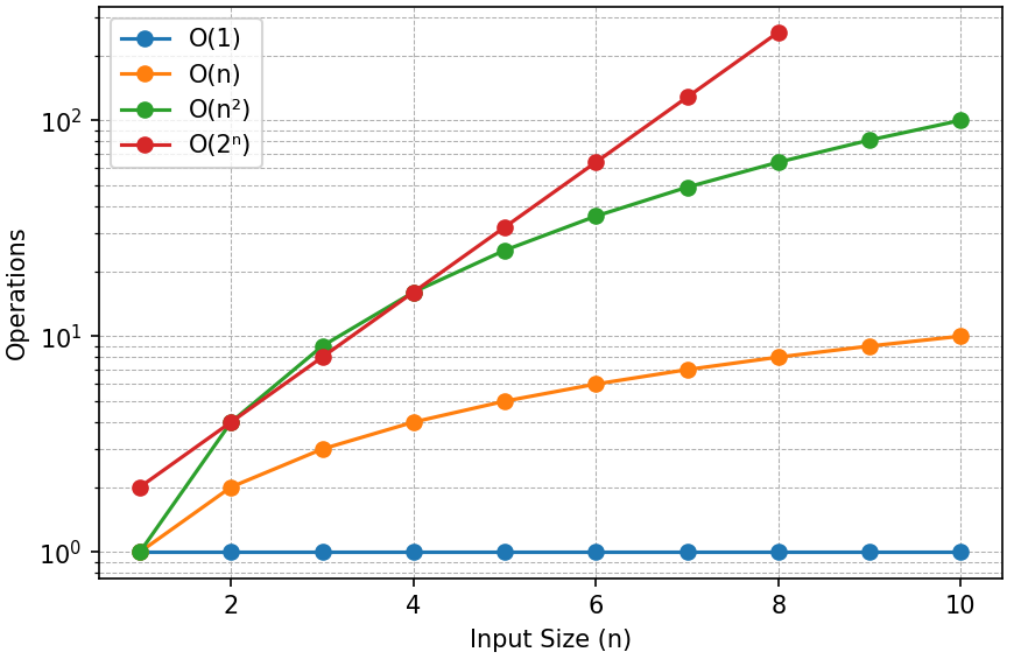
5. Introduction to Complexity Analysis

Performance is often measured by **how fast (time)** and **how much memory (space)** an algorithm uses.

Big-O Notation Examples:

Operation	Example Code	Time Complexity
Access by index	`arr[i]`	O(1)
Search unsorted list	`x in arr`	O(n)
Append	`arr.append(x)`	O(1) amortized
Sort	`arr.sort()`	O(n log n)

Visualization:



Time Complexity Graph

👨‍💻 6. Python and Built-in Data Structures

Python gives several high-level, flexible structures:

- **List** — ordered, mutable sequence
- **Tuple** — ordered, immutable sequence
- **Dict** — key-value mapping
- **Set** — unordered collection of unique elements

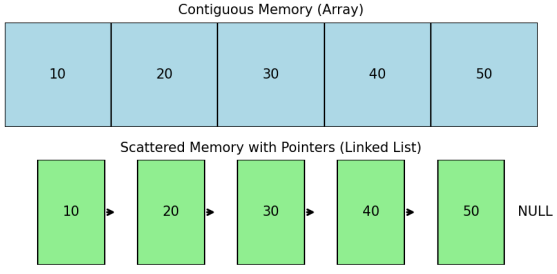
Example:

```
student = {
    "name": "Alice",
    "age": 20,
    "courses": ["Math", "CS"]
}
print(student["courses"][0])
```

🧠 7. Memory Model and Data Representation

All data structures rely on **memory layout**:

- Arrays store elements **contiguously**
- Linked lists store elements **scattered**, connected by pointers (references)



Memory Layout Comparison

Understanding memory helps explain *why* some operations are faster.

🧩 In-Class Exercises

Exercise 1 — Identify Data Structures

List 5 real-world systems (e.g., social media, e-commerce) and identify **one data structure** each might use internally.

Exercise 2 — Trace Big-O

For each operation below, determine its Big-O:

1. Access first element of list
2. Search for an item in unsorted list
3. Insert element at the start of list

Exercise 3 — Python ADT Simulation

Simulate a Stack ADT using a Python list.

```
stack = []
stack.append("A")     # push
stack.append("B")
stack.pop()           # pop
print(stack)
```

✅ In-Class Exercise Solutions

Exercise 1 Example Answers:

System	Data Structure
Instagram Feed	Queue
Browser History	Stack
Address Book	Hash Table
GPS Routing	Graph
E-commerce Cart	List

Exercise 2 Answers:

1. O(1)
2. O(n)
3. O(n)

Exercise 3 Output:

```
['A']
```

🏠 Take-Home Assignments

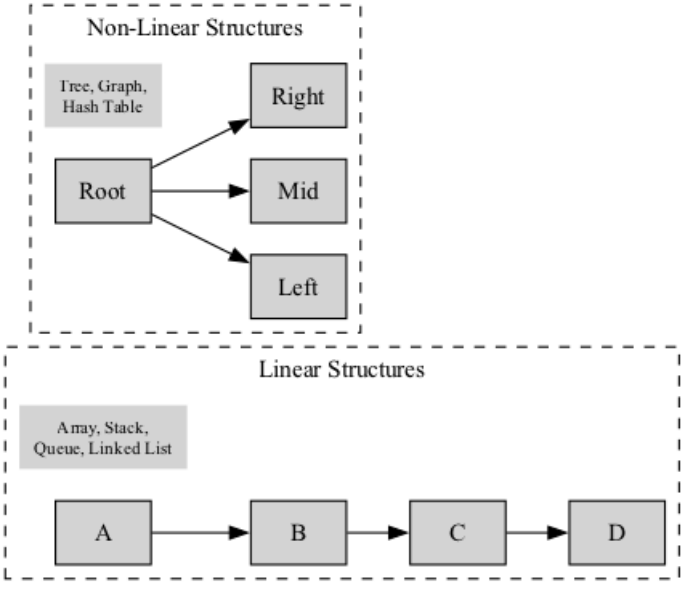
Assignment 1 — Complexity Comparison

Write two Python scripts:

1. Linear search in a list
2. Dictionary lookup Measure their performance using `timeit`.

Assignment 2 — Diagram Drawing

Draw (or code-generate later) a diagram showing:



- Linear vs Non-linear data structures

Assignment 3 — Build Custom List ADT

Implement a simple `CustomList` class supporting:

- `insert(value)`
- `delete(index)`
- `display()`

❌ Common Mistakes & 💡 Best Practices

Common Mistakes

- ❌ Confusing *data types* (int, str) with *data structures* (list, dict)
- ❌ Ignoring time complexity when choosing a structure
- ❌ Misunderstanding mutability (list vs tuple)
- ❌ Using wrong data structure for wrong problem (e.g., using list instead of dict)

Best Practices

- ✅ Always analyze the problem before picking a structure
- ✅ Prefer Python's built-in structures for simplicity and optimization
- ✅ Visualize data flow with diagrams before implementation
- ✅ Comment your code to clarify complexity assumptions

🔭 Next Week Preview

Next week we will explore:

1. **Arrays & Lists** in depth
2. **Indexing, slicing, and iteration**
3. **Internal memory allocation**
4. **Practical performance testing**
