🎓 Course 2 — Data Structures with Python

📘 Course Overview

🧭 Course Description

This course provides a **comprehensive, hands-on foundation** in data structures using the Python programming language. Students will explore **how data is organized, accessed, and manipulated** efficiently through different structures, algorithms, and memory models. By the end of this course, learners will not only master the implementation and use of key data structures, but also develop the ability to **analyze algorithmic performance**, reason about **complexity**, and choose optimal data models for real-world applications.

Data Structures are the **backbone of efficient programming** — they define *how* information is stored and *how fast* it can be processed. This course bridges theory and practice, turning abstract concepts into concrete, code-level understanding.

🎯 Learning Outcomes

After completing this course, students will be able to:

- ✅ Explain the **concepts, properties, and use cases** of major data structures.
- ✅ Implement **arrays, linked lists, stacks, queues, trees, and graphs** in Python.
- ✅ Analyze **time and space complexity** using Big-O notation.
- ✅ Apply **recursion and backtracking** effectively in algorithm design.
- ✅ Compare and contrast **linear and non-linear** data structures.
- ✅ Design efficient **searching, sorting, and traversal algorithms**.
- ✅ Use Python's **built-in data structures (lists, dicts, sets, tuples)** optimally.

Course Topics Overview

Week	Main Topics	Core Focus
1	Introduction to Data Structures	Concept, classification, complexity
2	Arrays & Lists	Indexing, slicing, operations, memory
3	Stacks & Queues	LIFO/FIFO behavior, implementation
4	Linked Lists	Singly, doubly, circular linked lists
5	Trees	Binary tree, traversal, recursion
6	Binary Search Tree (BST), Heap	Searching, balancing, heap property
7	Hash Tables & Dictionaries	Hashing, collisions, performance
8	Graphs	Representation, BFS, DFS, shortest path
9	Recursion & Backtracking	Design pattern, call stack, optimization
10	Advanced Structures	AVL, priority queues, tries
11	Complexity & Optimization	O(n), O(log n), amortized analysis
12	Mini Project 1	Data structure selection & implementation
13	Real-world Applications	Search engine, cache, DB indexing
14	Mini Project 2	Graph traversal or pathfinding
15	Review & Testing Strategies	Debugging, profiling, benchmarking
16	Capstone Project	End-to-end algorithm & data model integration

🧠 Pedagogical Design

This course uses a **bottom-up learning approach**:

1. Begin with simple data structures.
2. Progressively build to complex, abstract ones.
3. Combine them through projects that require design reasoning and algorithmic thinking.

Each week includes:

- In-depth lecture explanation
- Illustrated diagrams (`images/*.png`)
- 3 in-class exercises (with full solutions)
- 3 take-home assignments
- Required readings
- Common mistakes & best practices
- A "Next Week Preview"

👨‍💻 Tools & Technologies

- Python 3.10+
- VSCode
- Libraries: `timeit`, `collections`, `heapq`, `networkx`, `graphviz` (for visualization)

📚 Reference Texts

Primary Books

- *Data Structures and Algorithms in Python* — Michael T. Goodrich et al.
- *Problem Solving with Algorithms and Data Structures Using Python* — Bradley N. Miller, David L. Ranum
- *Introduction to Algorithms* (CLRS) — Cormen et al.

Online Resources

- Python Docs: https://docs.python.org/3/tutorial/datastructures.html
- Visualgo: https://visualgo.net/en

Assessment & Projects

- **Mini Projects (Weeks 4, 8, 12)** Each focused on implementing a major data structure and demonstrating practical use.
- **Final Capstone (Weeks 13–16)** Design and implement a real-world system (e.g., pathfinding engine, cache simulator, or text search index).

Skills You'll Build

- Analytical and computational thinking
- Performance-driven design mindset
- Mastery of Python data modeling
- Algorithmic problem-solving with precision
- Debugging and complexity optimization