MA5741: Object-Oriented Programming (Python Edition)

Structure:

- **Lecture (concept focus):** The main conceptual topics that will be covered in the lecture for that week. These are typically theoretical explanations and discussions.
- **Lab/studio (hands-on):** Practical activities, coding exercises, or projects to be completed during a lab or studio session. This is where students apply the concepts learned in the lecture.
- **Independent work (≈4 h):** The estimated time (around 4 hours) or specific tasks that students are expected to complete on their own outside of class. This often includes readings, additional exercises, or project work.

Detailed Contents:

Week 1:

- **Lecture:** Python basics, virtual environments (for managing dependencies), and type hints (for better code readability and error checking).
- **Lab:** Setting up the development environment and working on a "small coding kata" (a short, focused coding challenge).
- **Independent Work:** Reading Chapters 1-4 of Ramalho's book (likely "Fluent Python").

Week 2:

- **Lecture:** Deeper dive into Python's object model, namespaces, and special methods like `__init__` (constructor) and `__str__` (string representation).
- **Lab:** Designing and implementing `Point` and `Vector` classes.
- **Independent Work:** Completing exercises 1-3.

Week 3:

- **Lecture:** Object-oriented principles like encapsulation, properties, and the distinction between composition and inheritance.

- **Lab:** Refactoring code from Week 2 and writing unit tests using `pytest`.
- **Independent Work:** Reading sections 1-2 of Freeman's material (possibly "Head First Design Patterns" or similar).

Week 4:

- **Lecture:** SOLID principles (a set of design principles for object-oriented programming) and UML class diagrams (visual representations of class structures).
- **Lab:** Drawing UML diagrams from an existing code repository and starting the Test-Driven Development (TDD) cycle.
- **Independent Work:** Writing test cases.

Week 5:

- **Lecture:** Introduction to algorithm analysis (how to evaluate algorithm efficiency) and specific search algorithms: linear and binary search.
- **Lab:** Implementing and benchmarking (measuring performance) these search algorithms.
- **Independent Work:** Reading sections 1.1-1.2 of Dasgupta's book (likely an algorithms textbook).

Week 6:

- **Lecture:** Sorting algorithms: insertion sort and selection sort.
- **Lab:** Creating time-plots with `matplotlib` to visualize the performance of these sorts.
- **Independent Work:** Reading Kleinberg's Chapter 2 notes (another algorithms text).

Week 7:

- **Lecture:** More advanced sorting algorithms: merge sort, quicksort, and the idea behind Timsort (Python's built-in hybrid sort).
- **Lab:** Refactoring a list sorter using the "strategy pattern" (a design pattern).
- **Independent Work:** Writing a blog reflection.

Week 8:

- **Lecture:** Data structures: stacks, queues, and deques (double-ended queues).
- **Lab:** Implementing a generic stack/queue class with type hints.
- **Independent Work:** Preparing for Quiz 1.

Week 9:

- **Lecture:** Tree data structures: Binary Search Trees (BST), AVL trees (self-balancing BSTs), heaps, and priority queues.
- **Lab:** A mini-project involving a heap-based job scheduler.
- **Independent Work:** Extra AVL tree drills.

Week 10:

- **Lecture:** Graph data structures: Breadth-First Search (BFS), Depth-First Search (DFS), and topological sort.
- **Lab:** Visualizing DFS/BFS with `networkx` (a Python library for graph manipulation).
- **Independent Work:** Working on past-year problems.

Week 11:

- **Lecture:** Shortest path algorithms (Dijkstra's algorithm) and Minimum Spanning Tree (MST) algorithms (Prim's and Kruskal's).
- **Lab:** Path-finding on a transport graph.
- **Independent Work:** Preparing for Quiz 2.

Week 12:

- **Lecture:** Divide-and-conquer strategy; memoization.
- **Lab:** Implementing Karatsuba multiplication and a memoized coin-change problem solution.
- **Independent Work:** Dynamic Programming (DP) worksheet.

Week 13:

- **Lecture:** Dynamic Programming (Longest Increasing Subsequence - LIS, knapsack problem).
- **Lab:** Profiling and optimization with `cProfile`. Starting a "Capstone coding" project.

Week 14:

- **Lecture:** Concurrency (using `asyncio` and `multiprocessing`); packaging and Continuous Integration (CI).
- **Lab:** Peer code-review and capstone project demonstrations.
- **Independent Work:** Project report.

In summary, this is a comprehensive course outline covering fundamental Python programming, object-oriented design, data structures, algorithms, and some advanced topics like concurrency and software engineering practices.

Core Text

- Goodrich, Tamassia & Goldwasser, Data Structures and Algorithms in Python

References

- Jon Kleinberg & Éva Tardos, Algorithm Design
- Sanjoy Dasgupta, Christos Papadimitriou & Umesh Vazirani, Algorithms
- Luciano Ramalho, Fluent Python
- Steve Freeman & Nat Pryce, Growing Object-Oriented Software, Guided by Tests

Evaluation

Item	Weight
Weekly lab notebooks (auto-graded)	25 %
Three in-class quizzes (Weeks 4, 9, 13)	15 %
Mid-semester exam (Week 8)	20 %
Capstone project (team of 2–3; repo + report + demo)	30 %
Class engagement & peer-review quality	10 %