SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type: Lab**	**Academic Year:**2025-2026
Course Coordinator Name	Venkataramana Veeramsetty		
Instructor(s) Name	Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)		
Course Code	24CS002PC215	**Course Title**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week6 - Monday	**Time(s)**	
Duration	2 Hours	**Applicable to Batches**	
AssignmentNumber:11.1(Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*Expected Time to complete*
1	**Lab 11 – Data Structures with AI: Implementing Fundamental Structures** **Lab Objectives** • Use AI to assist in designing and implementing fundamental data structures in Python. • Learn how to prompt AI for structure creation, optimization, and documentation. • Improve understanding of Lists, Stacks, Queues, Linked Lists, Trees, Graphs, and Hash Tables.	Week6 - Monday

- Enhance code quality with AI-generated comments and performance suggestions.

Task Description #1 – Stack Implementation

Task: Use AI to generate a Stack class with push, pop, peek, and is_empty methods.

Sample Input Code:

```
class Stack:
    pass
```

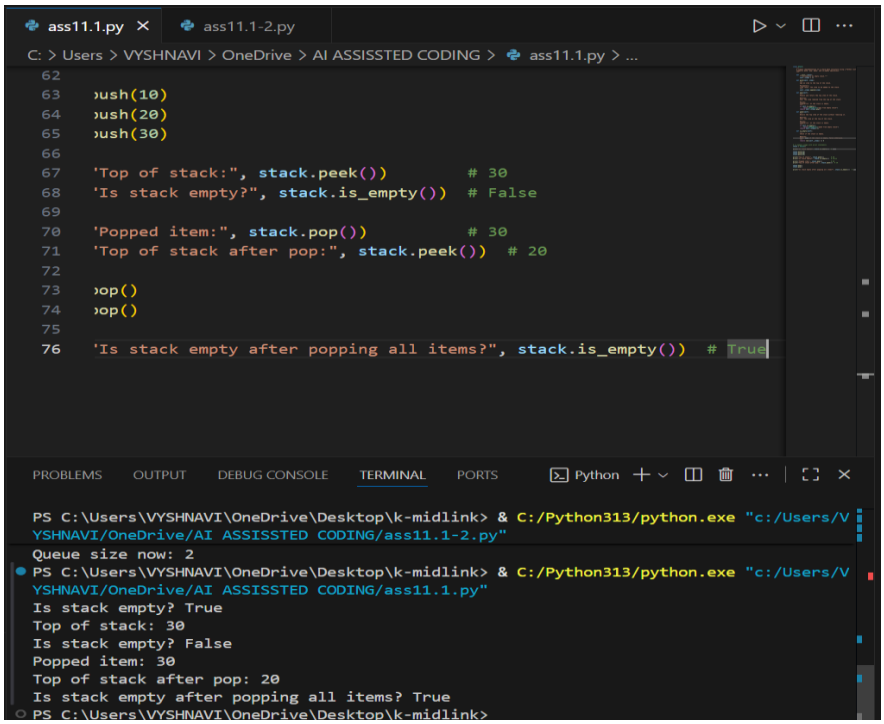
Expected Output:

- A functional stack implementation with all required methods and docstrings.

PROMPT:

"Generate a Stack class in Python with the following methods: push, pop, peek, and is_empty. Include comments explaining each method."

SCREENSHOT:



Task Description #2 – Queue Implementation

Task: Use AI to implement a Queue using Python lists.

Sample Input Code:

```
class Queue:
    pass
```

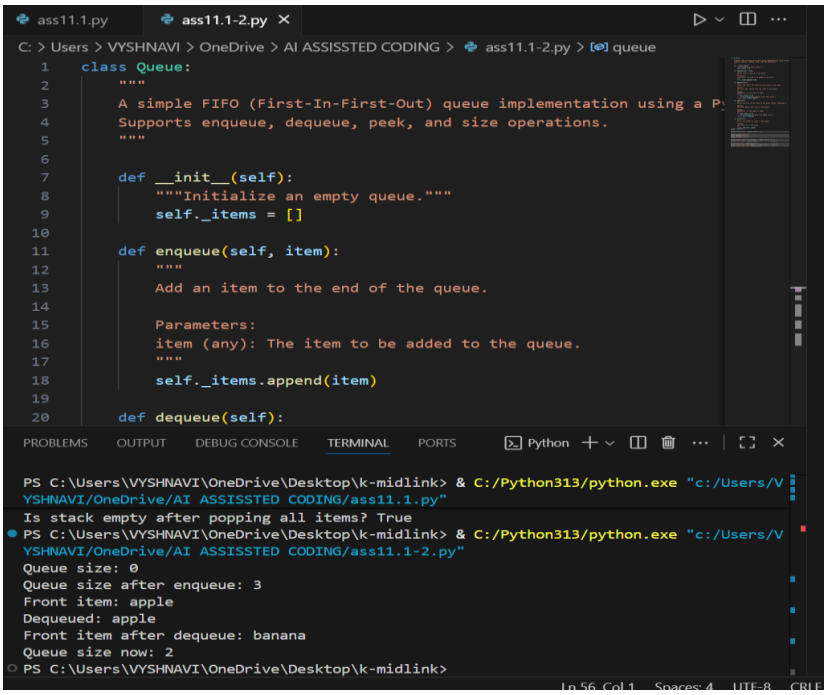
Expected Output:

- FIFO-based queue class with enqueue, dequeue, peek, and size methods.

  PROMPT:

  "Write a Python class called Queue that uses a list to implement."

  SCREENSHOT:



Task Description #3 – Linked List

Task: Use AI to generate a Singly Linked List with insert and display methods.

Sample Input Code:

```
class Node:
    pass

class LinkedList:
    pass
```

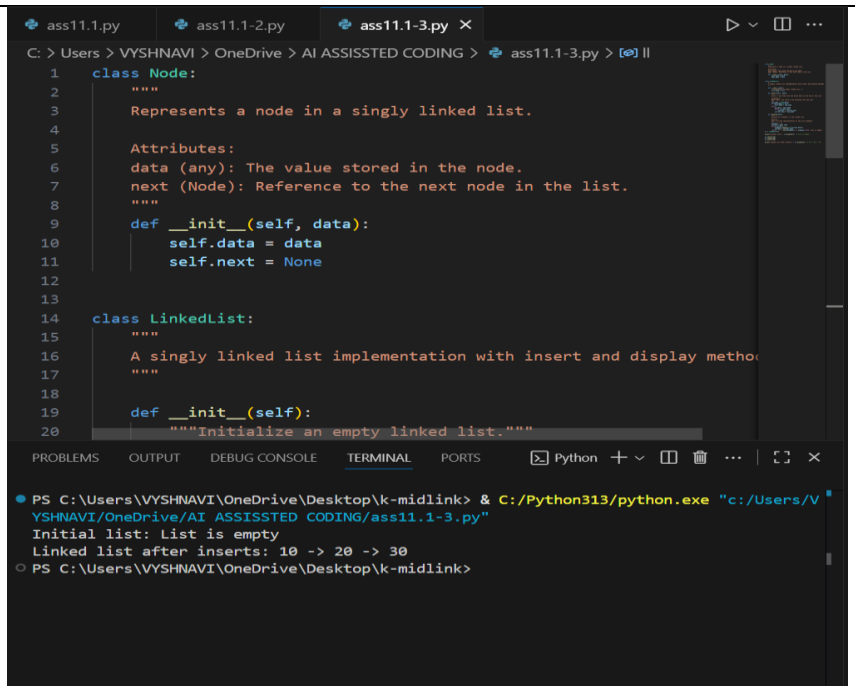
Expected Output:

- A working linked list implementation with clear method documentation.

PROMPT:

"Write a Python class for a Singly Linked List with two methods: insert to add a node at the end, and display to print all node values. Include comments explaining each part of the code."

SCREENSHOT:



Task Description #4 – Binary Search Tree (BST)

Task: Use AI to create a BST with insert and in-order traversal methods.

Sample Input Code:

class BST:

pass

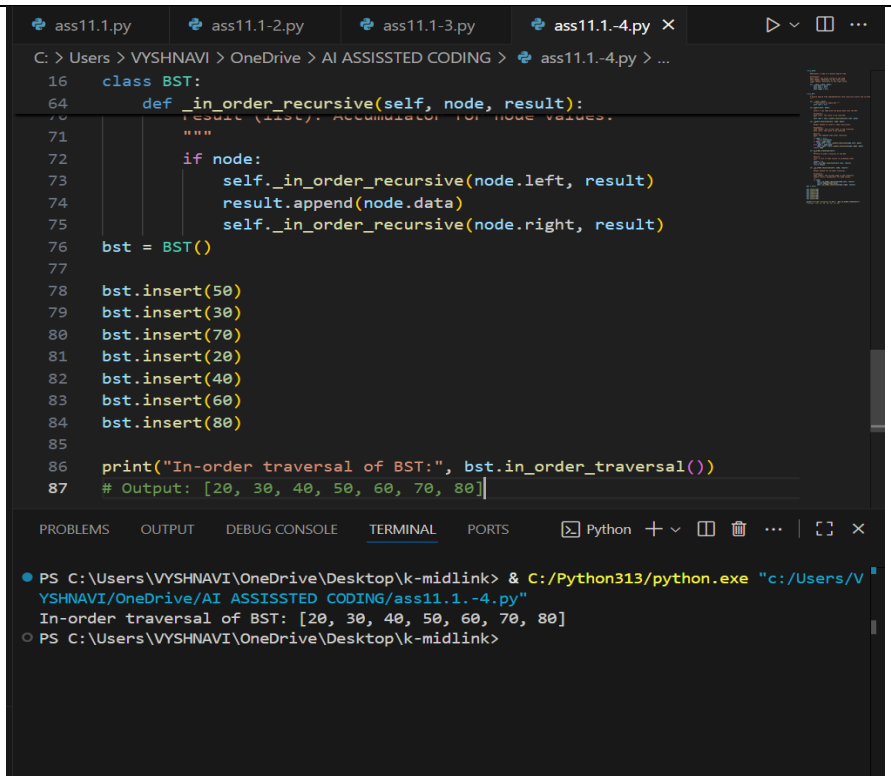
Expected Output:

- BST implementation with recursive insert and traversal methods.

PROMPT:

"Write a Python class for a Binary Search Tree with two methods: insert to add nodes, and in_order_traversal to print the tree in sorted order. Include comments explaining each method."

SCREENSHOT:



Task Description #5 – Hash Table

Task: Use AI to implement a hash table with basic insert, search, and delete methods.

Sample Input Code:

class HashTable:

pass

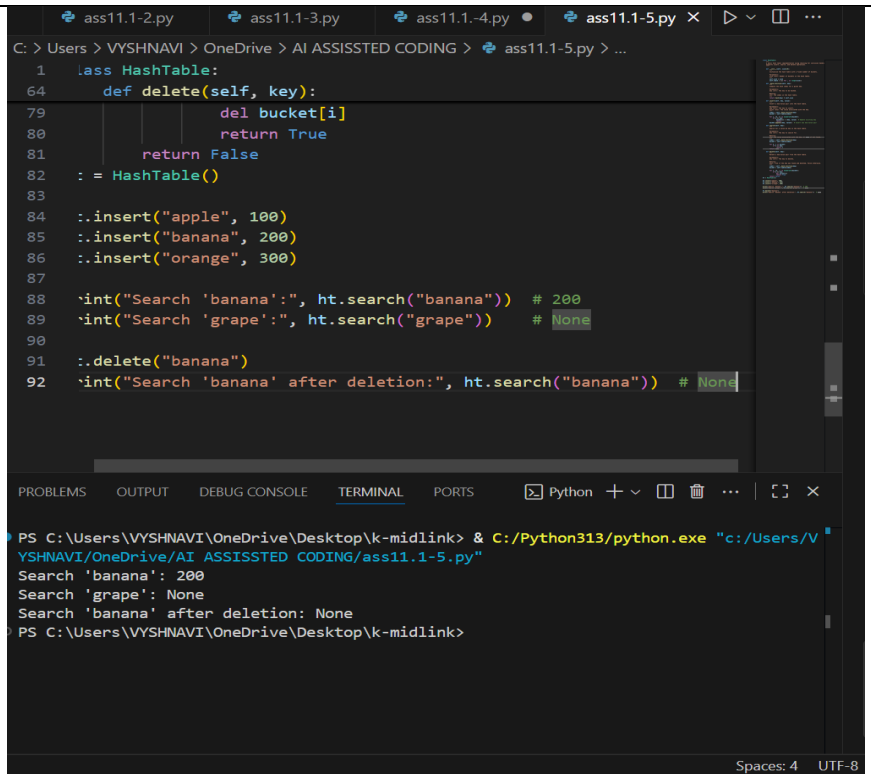
Expected Output:

- Collision handling using chaining, with well-commented methods.

PROMPT:

"Write a Python class for a hash table using lists. Include methods for insert, search, and delete. Add comments explaining each method."

SCREENSHOT:



Task Description #6 – Graph Representation

Task: Use AI to implement a graph using an adjacency list.

Sample Input Code:

class Graph:

pass

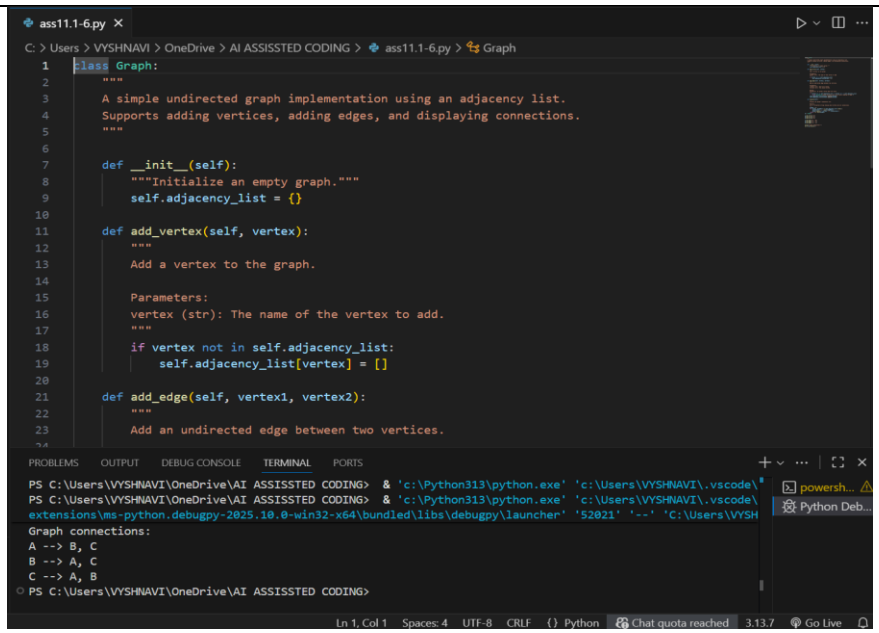
Expected Output:

- Graph with methods to add vertices, add edges, and display connections.

PROMPT:

"Write a Python class to represent a graph using an adjacency list. Include methods to add vertices, add edges, and display the graph. Add comments explaining each method."

SCREENSHOT:



Task Description #7 – Priority Queue

Task: Use AI to implement a priority queue using Python's heapq module.

Sample Input Code:

class PriorityQueue:

pass

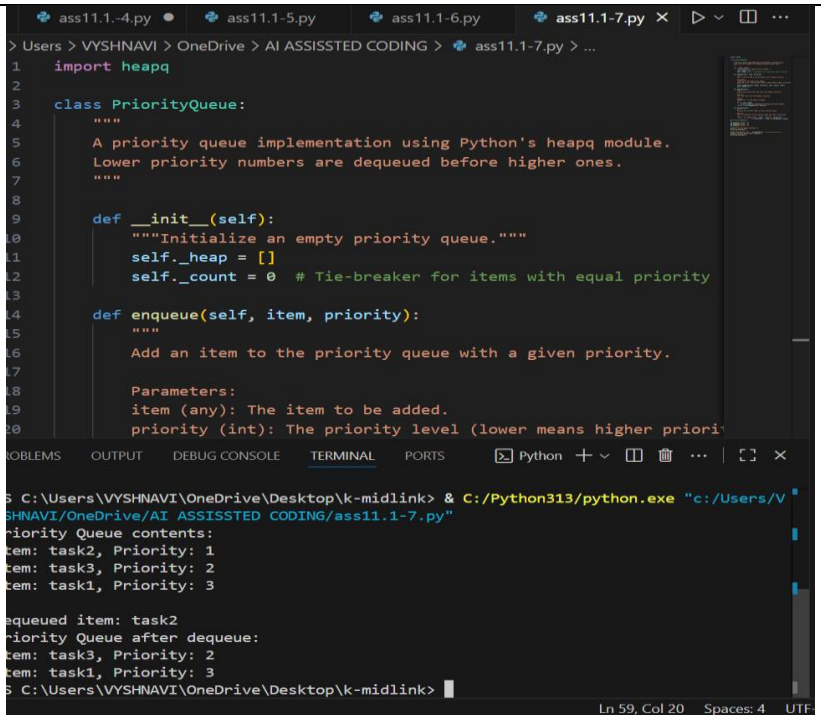
Expected Output:

- Implementation with enqueue (priority), dequeue (highest priority), and display methods.

  PROMPT:

  "Write a Python class for a priority queue using the heapq module. Include methods to insert items with priority, remove the highest priority item, and peek at the highest priority item. Add comments explaining each method."

  SCREENSHOT:



Task Description #8 – Deque

Task: Use AI to implement a double-ended queue using collections.deque.

Sample Input Code:

class DequeDS:

pass

Expected Output:

- Insert and remove from both ends with docstrings.

PROMPT:

"Write a Python class that wraps collections.deque to implement a double-ended queue. Include methods to add and remove elements from both ends, and display the current state. Add comments explaining each method."

SCREENSHOT:

C: > Users > VYSHNAVI > OneDrive > AI ASSISSTED CODING > ass11.1-8.py > ...

```
class DequeDS:
    def insert_front(self, item):

        Insert an item at the front of the deque.

        Parameters:
        item (any): The item to insert.
        """
        self._deque.appendleft(item)

    def insert_rear(self, item):
        """
        Insert an item at the rear of the deque.

        Parameters:
        item (any): The item to insert.
        """
        self._deque.append(item)

    def remove_front(self):
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python

```
PS C:\Users\VYSHNAVI\OneDrive\Desktop\k-midlink> & C:/Python313/python.exe "c:/Users/V
YSHNAVI/OneDrive/AI ASSISSTED CODING/ass11.1-8.py"
Deque contents: 5 <-> 10 <-> 20 <-> 30
Removed from front: 5
Removed from rear: 30
Deque contents: 10 <-> 20
PS C:\Users\VYSHNAVI\OneDrive\Desktop\k-midlink>
```

Task Description #9 – AI-Generated Data Structure Comparisons

Task: Use AI to generate a comparison table of different data structures (stack, queue, linked list, etc.) including time complexities.

Sample Input Code:

No code, prompt AI for a data structure comparison table

Expected Output:

- A markdown table with structure names, operations, and complexities.

Structure	Access	Search	Insert	Delete	Notes
Array	O①	O(n)	O(n)	O(n)	Fast access, fixed size
Dyn. Array	O①*	O(n)	O①*	O(n)	Resizable, amortized ops
Singly List	O(n)	O(n)	O①	O①	Insert/delete at head
Doubly List	O(n)	O(n)	O①	O①	Bidirectional links
Stack (Arr)	O(n)	O(n)	O①	O①	LIFO, fixed size
Stack (List)	O(n)	O(n)	O①	O①	LIFO, dynamic
Queue (Arr)	O(n)	O(n)	O①	O①	FIFO, may need shifting
Queue (List)	O(n)	O(n)	O①	O①	FIFO, dynamic
Circ. Queue	O(n)	O(n)	O①	O①	Wrap-around buffer
Deque	O(n)	O(n)	O①	O①	Insert/delete both

Task Description #10 Real-Time Application Challenge – Choose the Right Data Structure

Scenario:

Your college wants to develop a Campus Resource Management System that handles:

1. Student Attendance Tracking – Daily log of students entering/exiting the campus.
2. Event Registration System – Manage participants in events with quick search and removal.
3. Library Book Borrowing – Keep track of available books and their due dates.
4. Bus Scheduling System – Maintain bus routes and stop connections.
5. Cafeteria Order Queue – Serve students in the order they arrive.

Student Task:

- For each feature, select the most appropriate data structure from the list below:
    - Stack
    - Queue
    - Priority Queue
    - Linked List
    - Binary Search Tree (BST)
    - Graph
    - Hash Table
    - Deque

- Justify your choice in 2–3 sentences per feature.
- Implement one selected feature as a working Python program with AI-assisted code generation.

Expected Output:

- A table mapping feature → chosen data structure → justification.
- A functional Python program implementing the chosen feature with comments and docstrings.

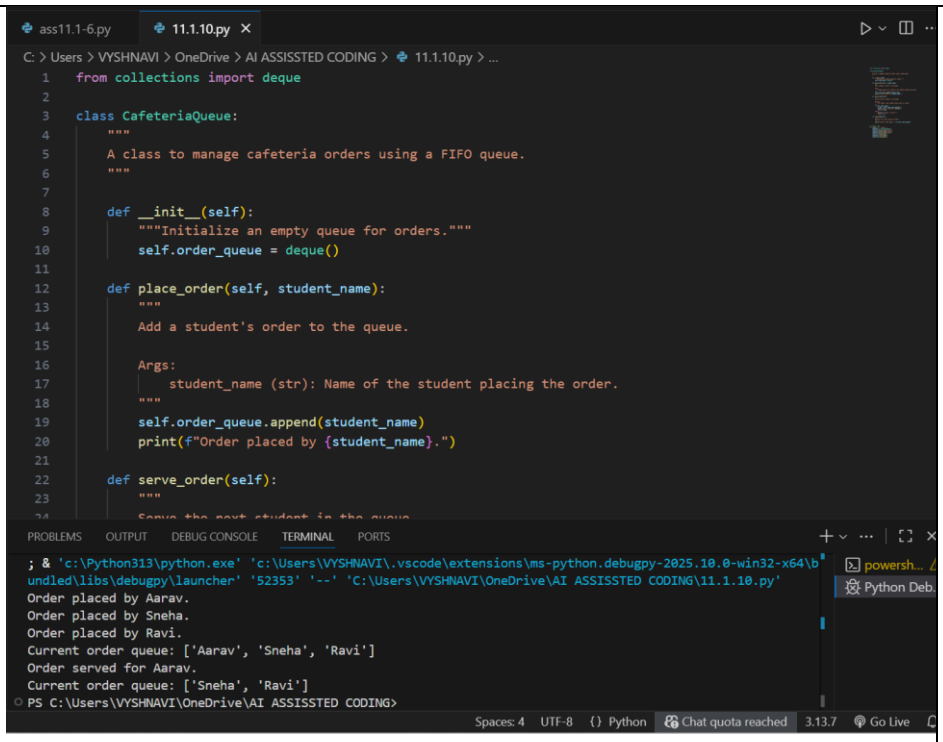
PROMPT:

Choose the best data structure for each campus system feature (attendance, events, library, buses, cafeteria) from a given list. Justify each choice in 2–3 sentences. Then, implement one feature in Python with clear comments and docstrings.

SCREENSHOT:

Feature	Chosen Data Structure	Justification
1. Student Attendance Tracking	**Stack**	A stack is ideal for tracking entry/exit logs in reverse chronological order. It allows quick access to the most recent activity, which is useful for audits or tracing last movements.
2. Event Registration System	**Hash Table**	A hash table enables constant-time search, insertion, and deletion of participants using unique IDs. This ensures fast registration lookup and efficient removal.
3. Library Book Borrowing	**Priority Queue**	A priority queue can manage books based on due dates, allowing the system to prioritize overdue returns or upcoming deadlines.
4. Bus Scheduling System	**Graph**	A graph is perfect for modeling bus routes and stop connections, where nodes represent stops and edges represent routes. It supports pathfinding and route optimization.
5. Cafeteria Order Queue	**Queue**	A queue ensures students are served in the order they arrive (FIFO), maintaining fairness and simplicity in processing orders.

CODE:



✅ Deliverables (For All Tasks)

1. AI-generated prompts for code and test case generation.
2. At least 3 assert test cases for each task.
3. AI-generated initial code and execution screenshots.
4. Analysis of whether code passes all tests.
5. Improved final version with inline comments and explanation.
6. Compiled report (Word/PDF) with prompts, test cases, assertions, code, and output.