AI ASSISTED CODING

LAB ASSIGNMENT: 11.2

NAME: V.Himesh Ram

H.NO: 2403A52045

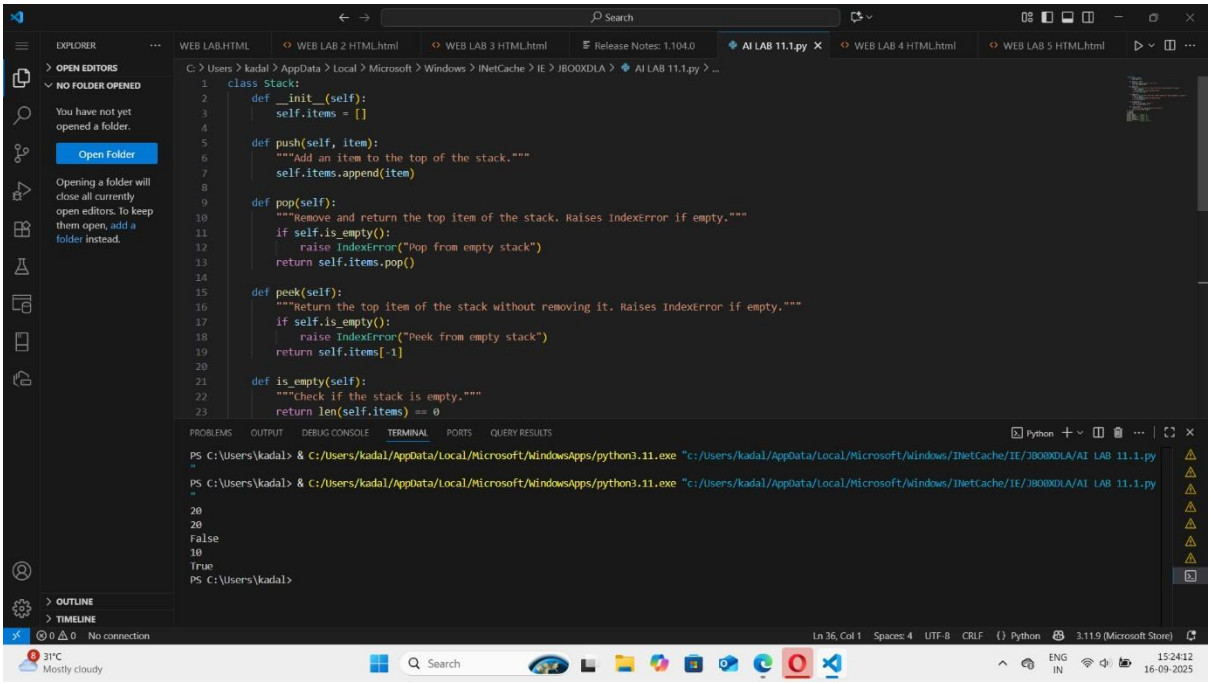
B.NO: 03 TASK:

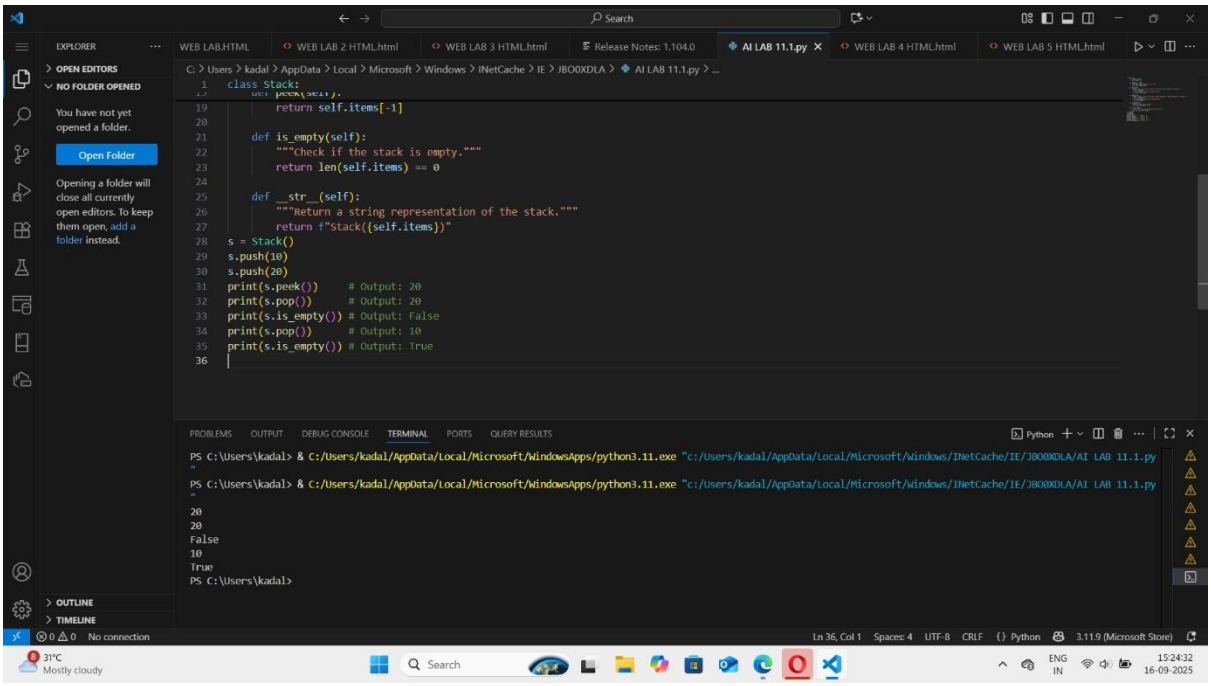
Stack Implementation
Task: Use AI to generate a Stack class with push, pop, peek, and is_empty methods.
Sample Input Code: class Stack: pass.

PROMPT:

Generate python code and stack Implementation
Task: Use AI to generate a Stack class with push, pop, peek, and is_empty methods.
Sample
Input Code: class
Stack: pass.

CODE & OUTPUT:





EXPLAINATION:

A **stack** is a linear data structure that follows the **LIFO** principle — **Last In, First Out**. Think of it like a stack of plates:

- You add (push) a plate to the top.
- You remove (pop) the top plate first.
- You can peek at the top plate without removing it.
- You can check if the stack is empty.

TASK 2:

Queue Implementation

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

Sample Input Code:

class Queue: pass.

PROMPT:

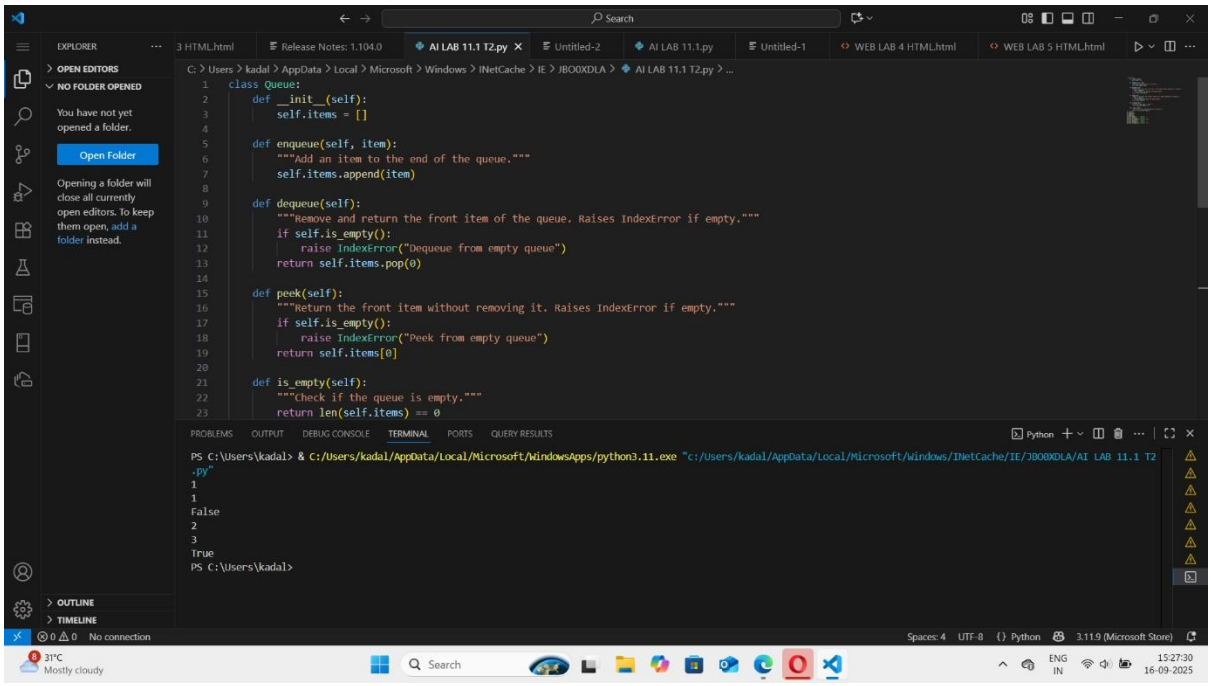
Generate python code and queue Implementation

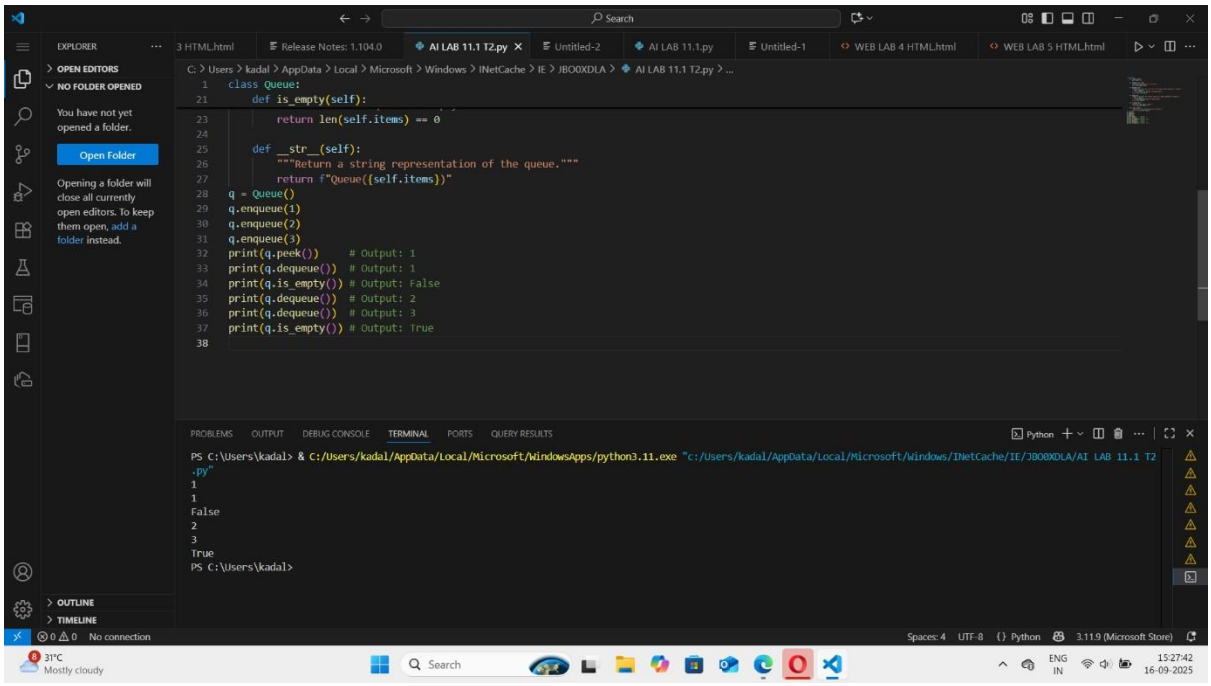
Task: Use AI to implement a Queue using Python lists.

Sample Input Code:

class Queue: pass.

CODE & OUTPUT:





EXPLAINATION:

🔍 Explanation

Method	Description	Time Complexity
`__init__`	Initializes an empty list to store queue elements	O①
`enqueue()`	Adds an item to the end of the list (rear of the queue)	O①
`dequeue()`	Removes and returns the first item (front of the queue)	O(n)
`peek()`	Returns the first item without removing it	O①
`is_empty()`	Checks if the queue is empty	O①

> ⚠️ Note: `dequeue()` uses `pop(0)`, which is O(n) because it shifts all remaining elements. For better performance, you can use `collections.deque`.

TASK 3:

Linked List

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

Sample Input Code:

class Node: pass.

PROMPT:

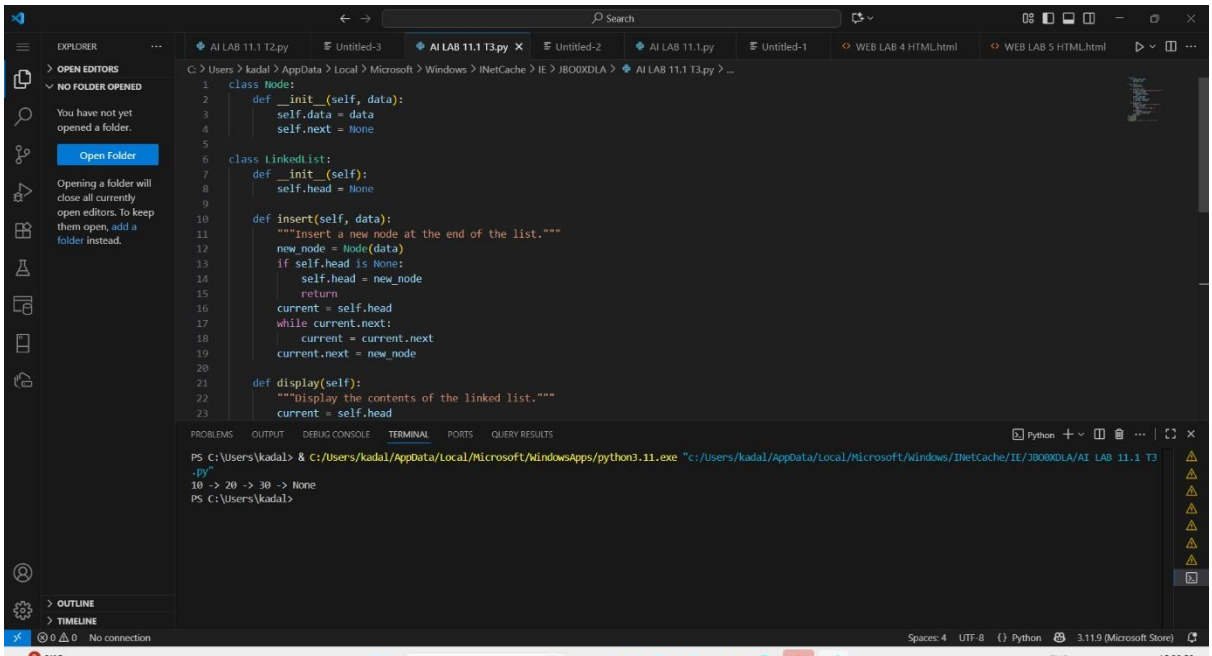
Generate python code and linked List

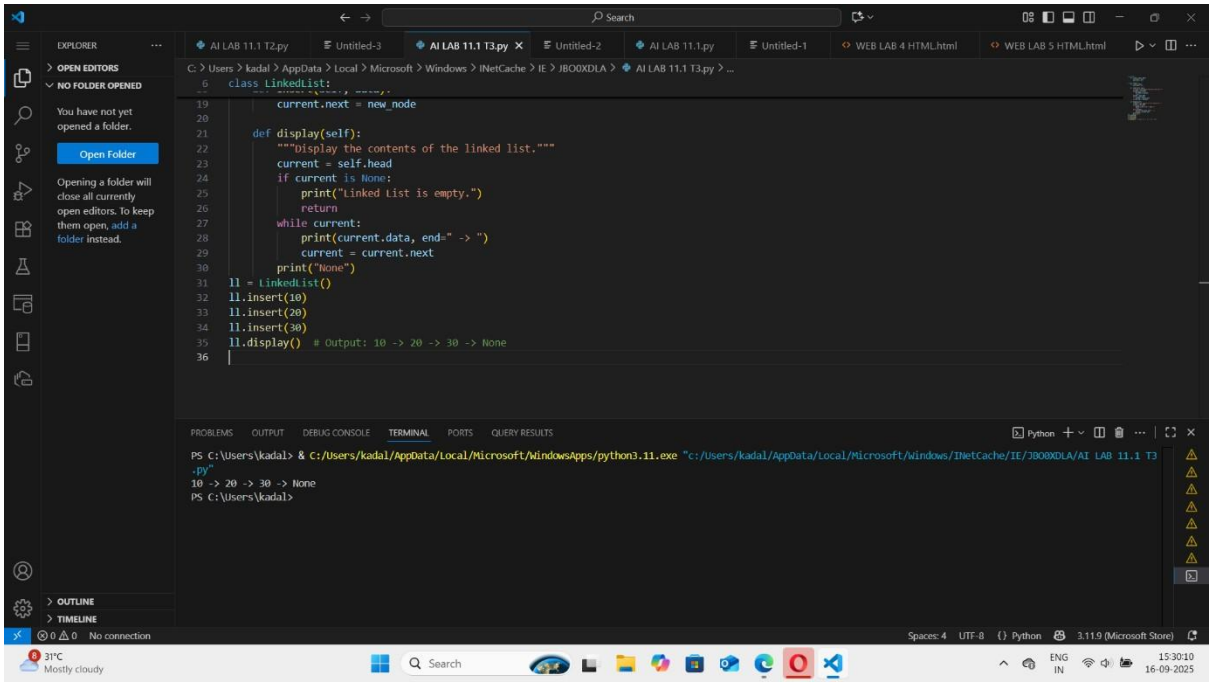
Task: Use AI to generate a Singly Linked List with insert and display methods.

Sample Input Code:

class Node: pass.

CODE & OUTPUT:





EXPLAINATION:

- Represents each element in the list.
- `data` : stores the value.
- `next` : points to the next node (or `None` if it's the last).

🧩 `LinkedList` Class

- Manages the chain of nodes.
- `head` : reference to the first node.

🔧 `insert(data)`

- Creates a new node.
- If the list is empty, sets it as the head.
- Otherwise, traverses to the end and links the new node.

TASK 4:

Binary Search Tree (BST)

Task: Use AI to create a BST with insert and inorder traversal methods.

Sample Input Code:

class BST: pass.

PROMPT:

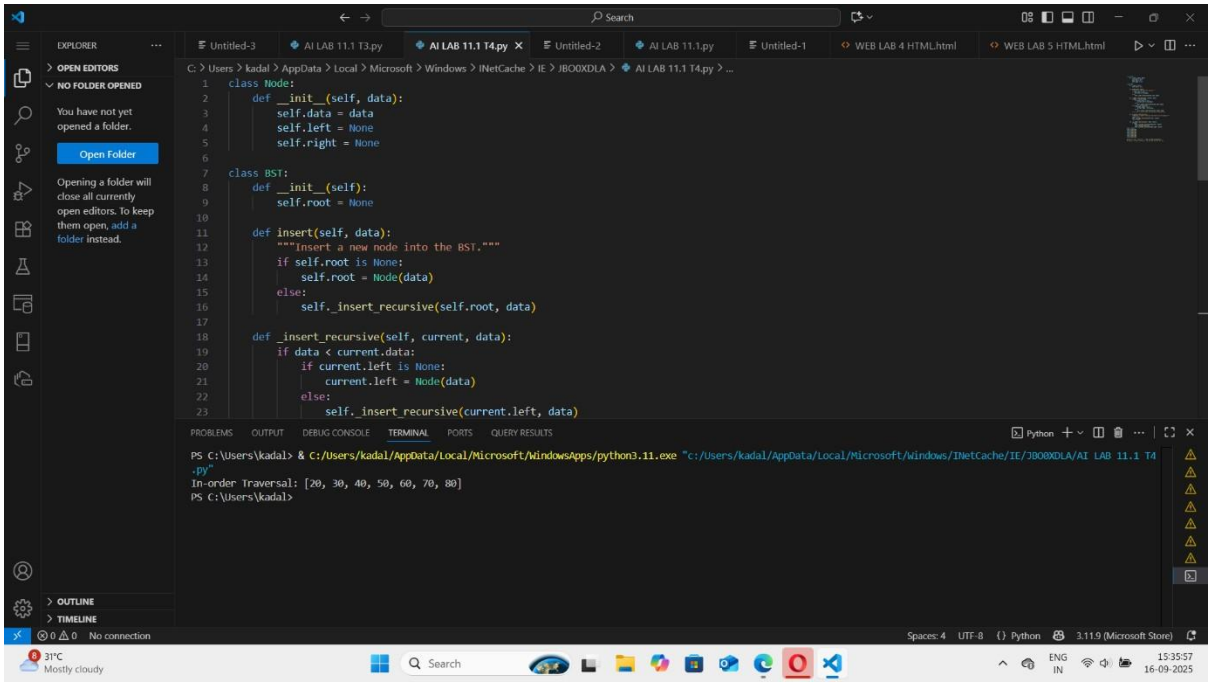
Generate python code and binary Search Tree (BST)

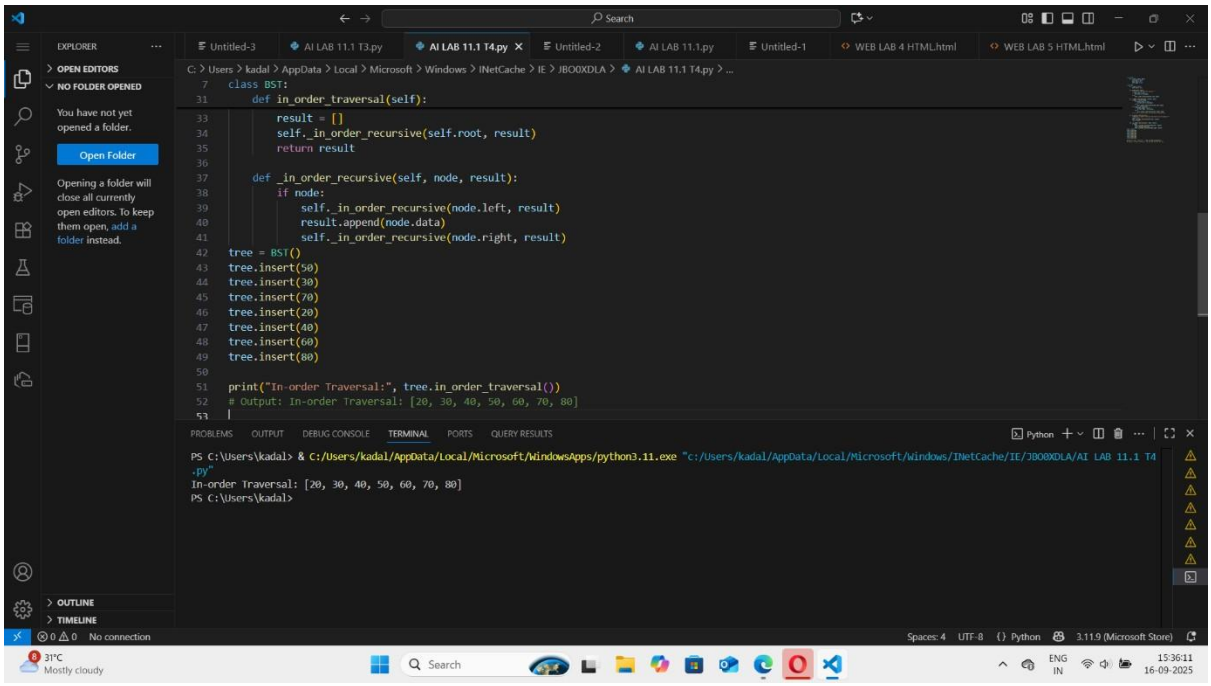
Task: Use AI to create a BST with insert and inorder traversal methods.

Sample Input Code:

class BST: pass.

CODE & OUTPUT:





EXPLAINATION:

- Adds a new value to the tree.
- If the tree is empty, it becomes the root.
- Otherwise, it uses `_insert_recursive()` to find the correct position:
  - If `data < current.data`: go left.
  - If `data > current.data`: go right.
  - If equal: skip (no duplicates).

👀 `in_order_traversal()`

- Returns a sorted list of values.
- Uses `_in_order_recursive()`:
  - Traverse left subtree.
  - Visit current node.
  - Traverse right subtree.

TASK 5:

Hash Table

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

Sample Input Code: class HashTable: pass.

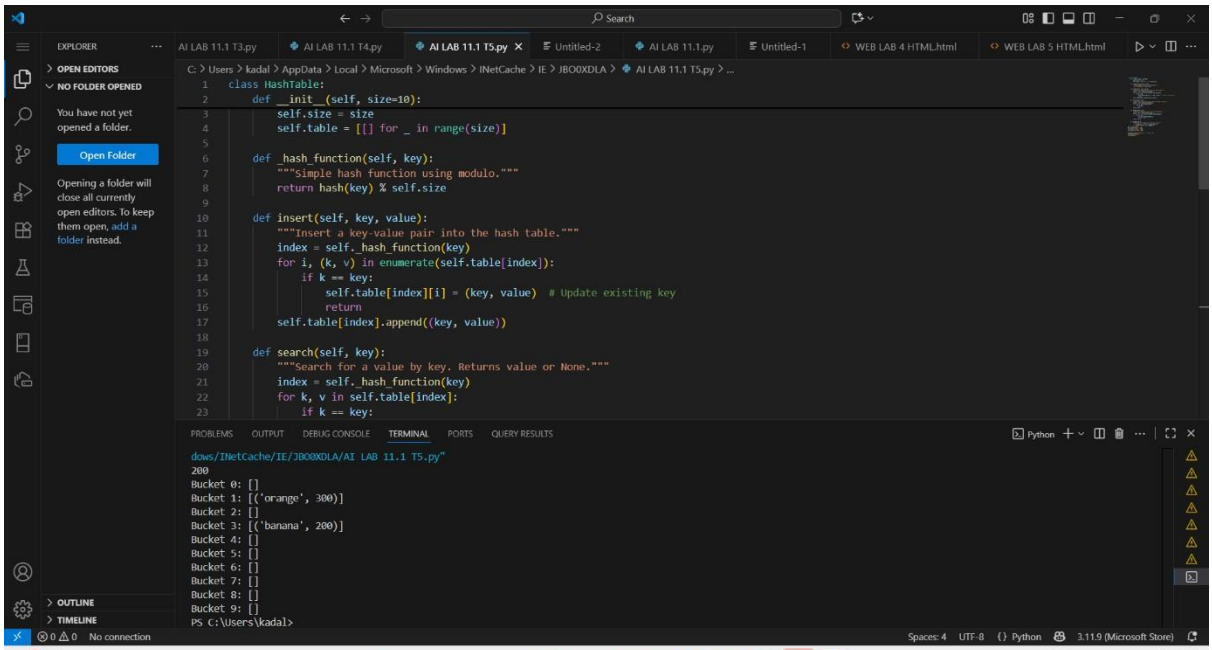
PROMPT:

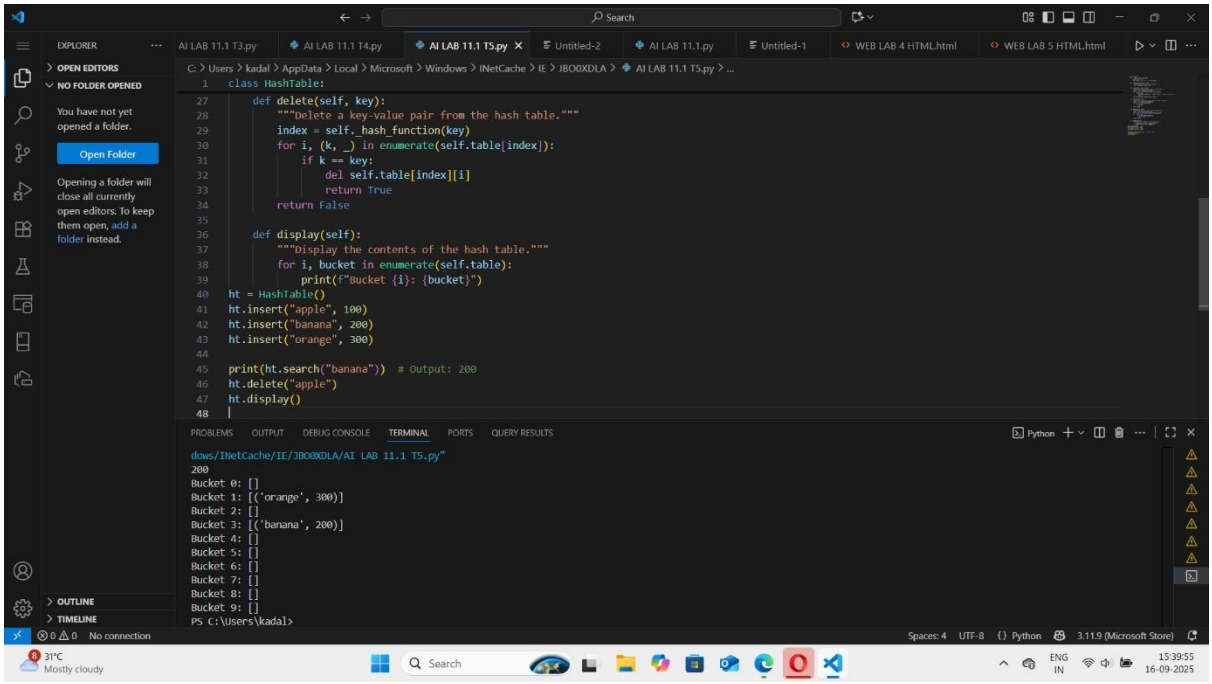
Generate python code and hash Table Task: Use AI to implement a hash table with

basic insert, search, and delete methods.

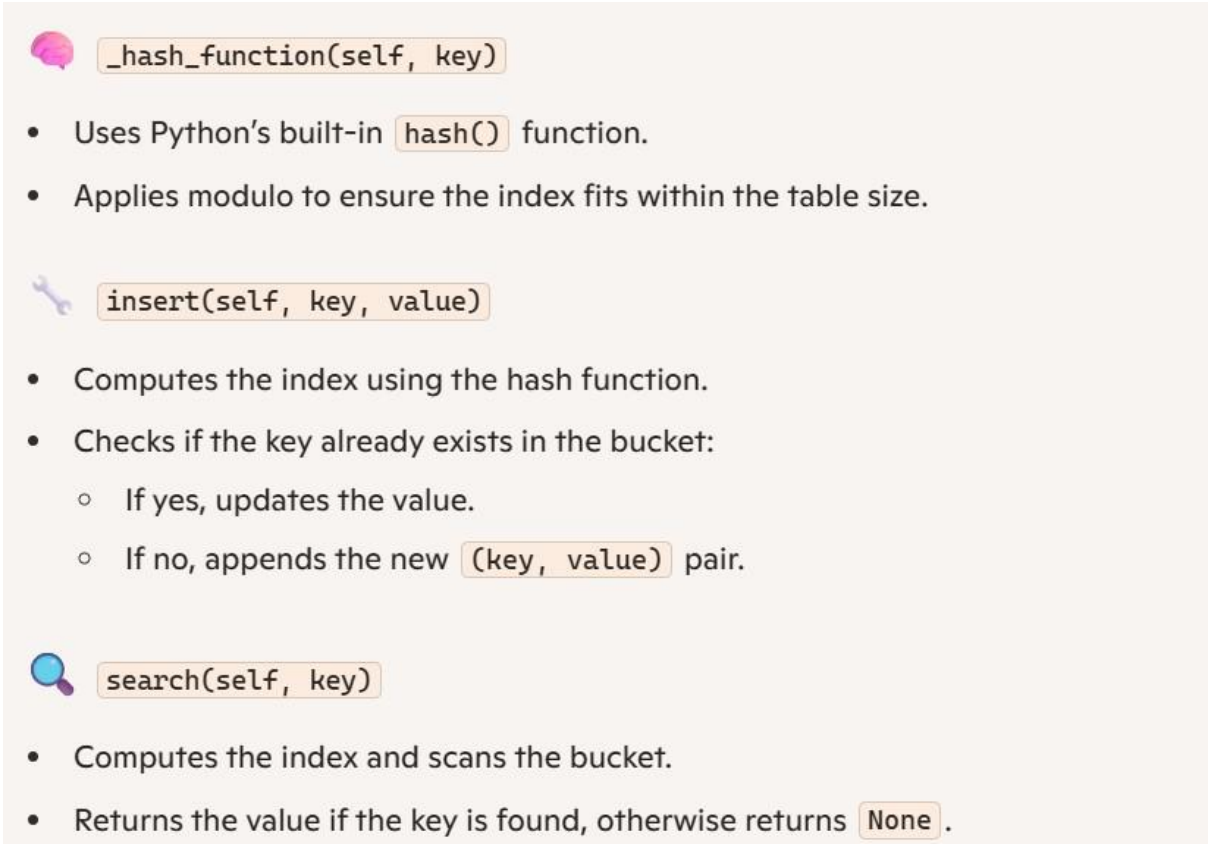
Sample Input Code: class HashTable: pass.

CODE & OUTPUT:





EXPLAINATION:



🧠 `_hash_function(self, key)`

- Uses Python's built-in `hash()` function.
- Applies modulo to ensure the index fits within the table size.

🔧 `insert(self, key, value)`

- Computes the index using the hash function.
- Checks if the key already exists in the bucket:
  - If yes, updates the value.
  - If no, appends the new `(key, value)` pair.

🔍 `search(self, key)`

- Computes the index and scans the bucket.
- Returns the value if the key is found, otherwise returns `None`.

TASK 6:

Graph Representation

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

Sample Input Code: class

Graph:

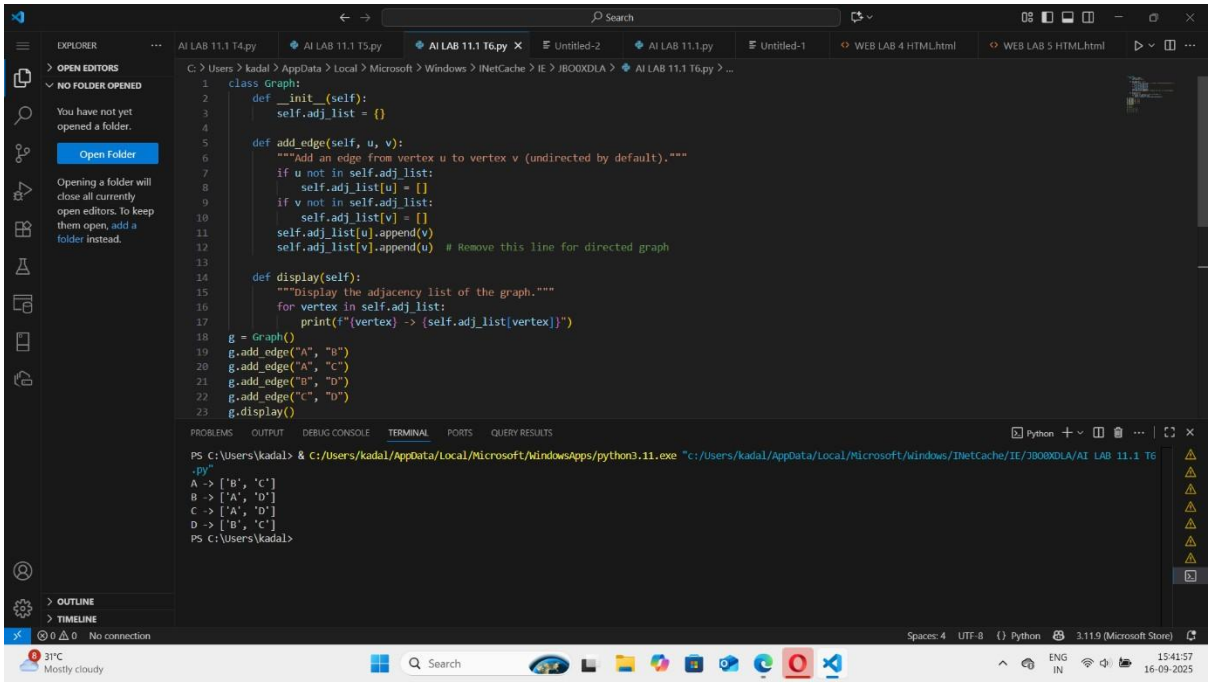
pass.

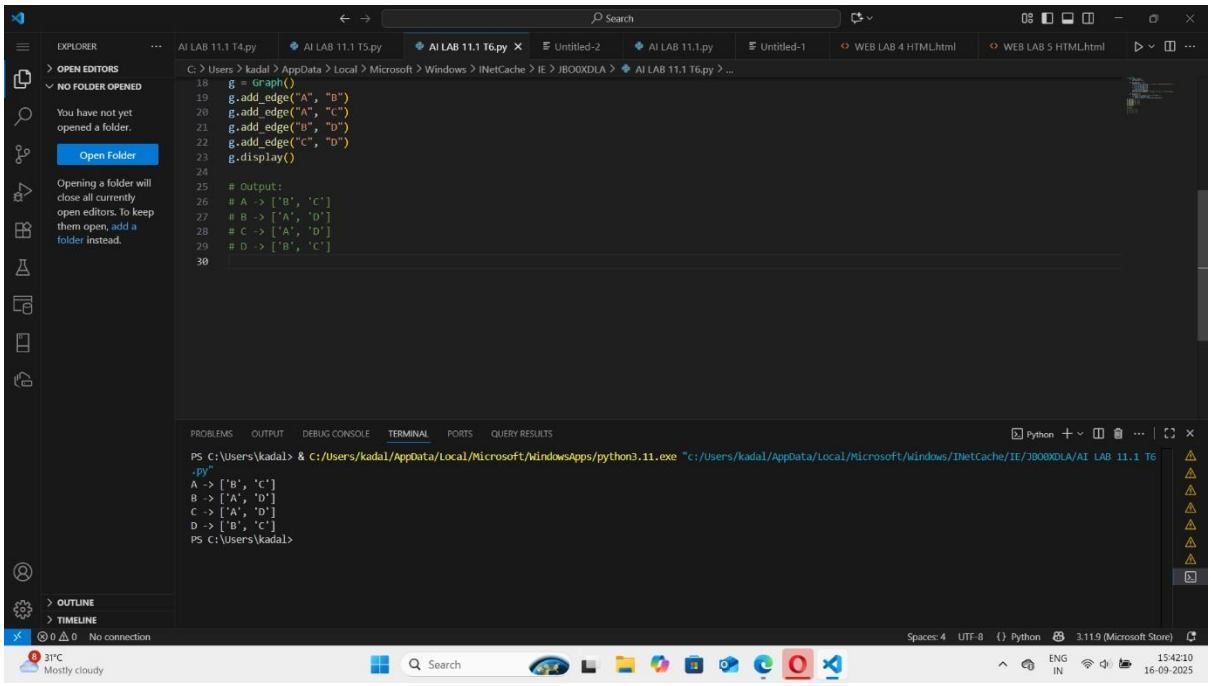
PROMPT:

Generate python code and graph Representation

Task: Use AI to implement a graph using an adjacency list.
Sample Input Code:
class Graph: pass.

CODE & OUTPUT:





EXPLAINATION:

- Initializes an empty list called `heap`.
- This list will store tuples of `(priority, item)`.

🔧 `insert(priority, item)`

- Uses `heapq.heappush()` to add a tuple to the heap.
- The heap maintains order based on the **priority** (lowest number = highest priority).

🗑️ `remove()`

- Uses `heapq.heappop()` to remove and return the item with the **lowest priority value**.
- Raises an error if the queue is empty.

👀 `peek()`

- Returns the item with the highest priority without removing it.

TASK 7:

Priority Queue

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

Sample Input Code: class PriorityQueue: pass.

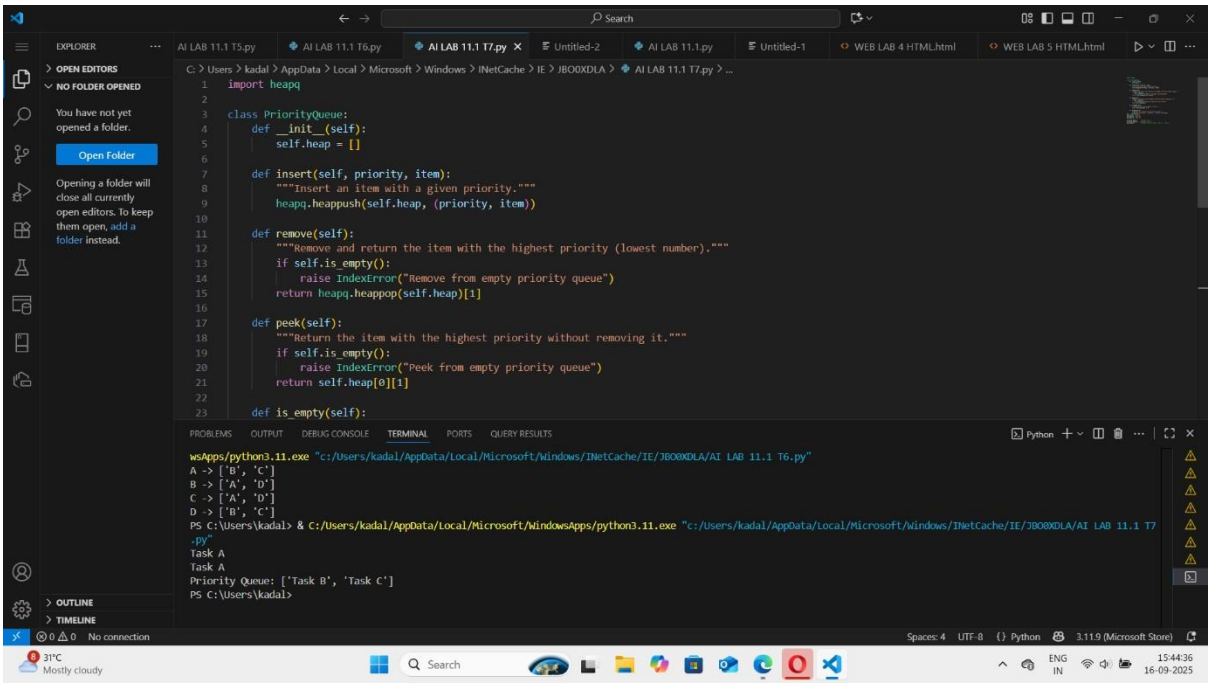
PROMPT:

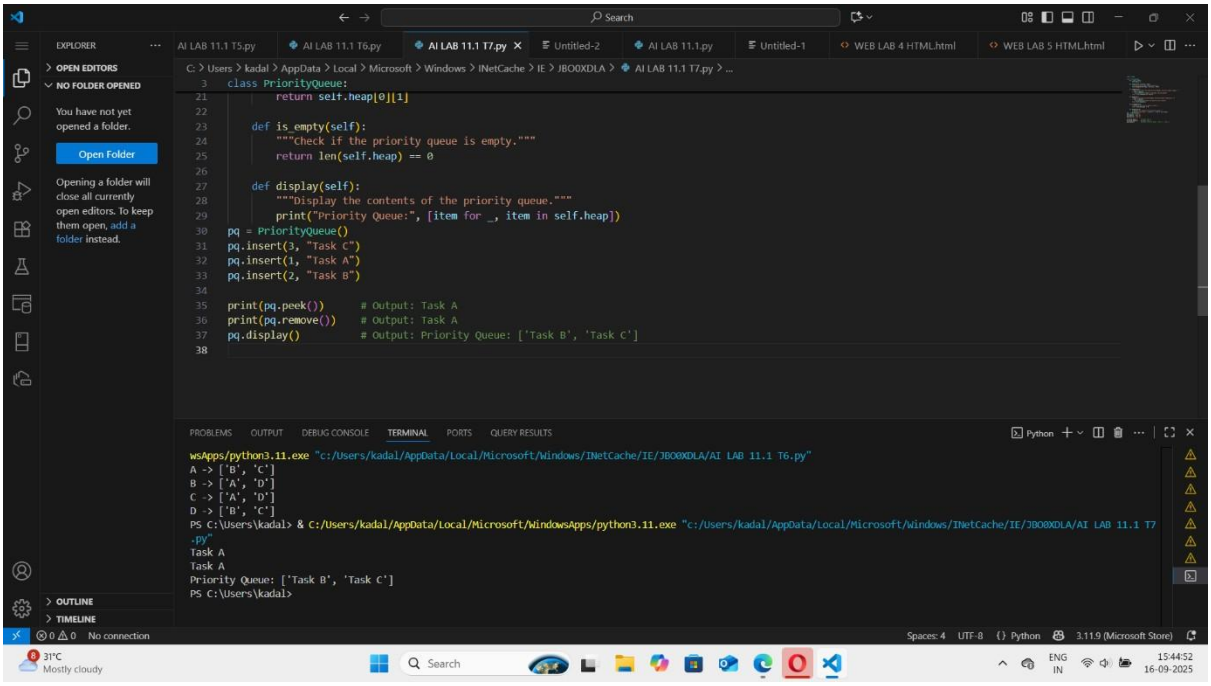
Generate a python code and priority Queue

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

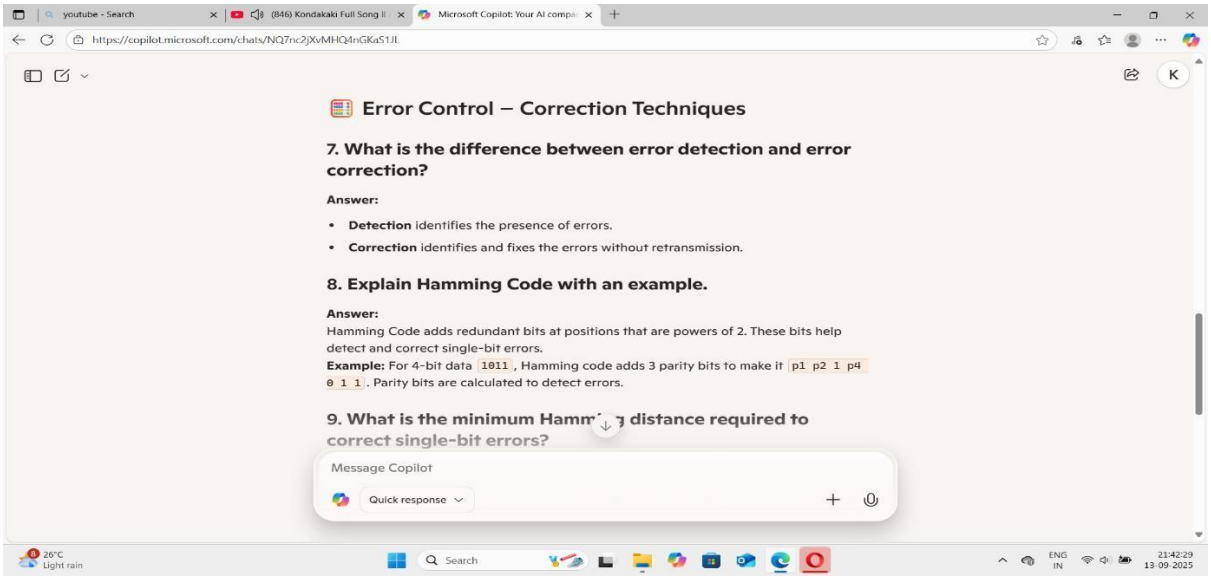
Sample Input Code: class PriorityQueue: pass.

CODE & OUTPUT:





EXPLAINATION:



TASK 8:

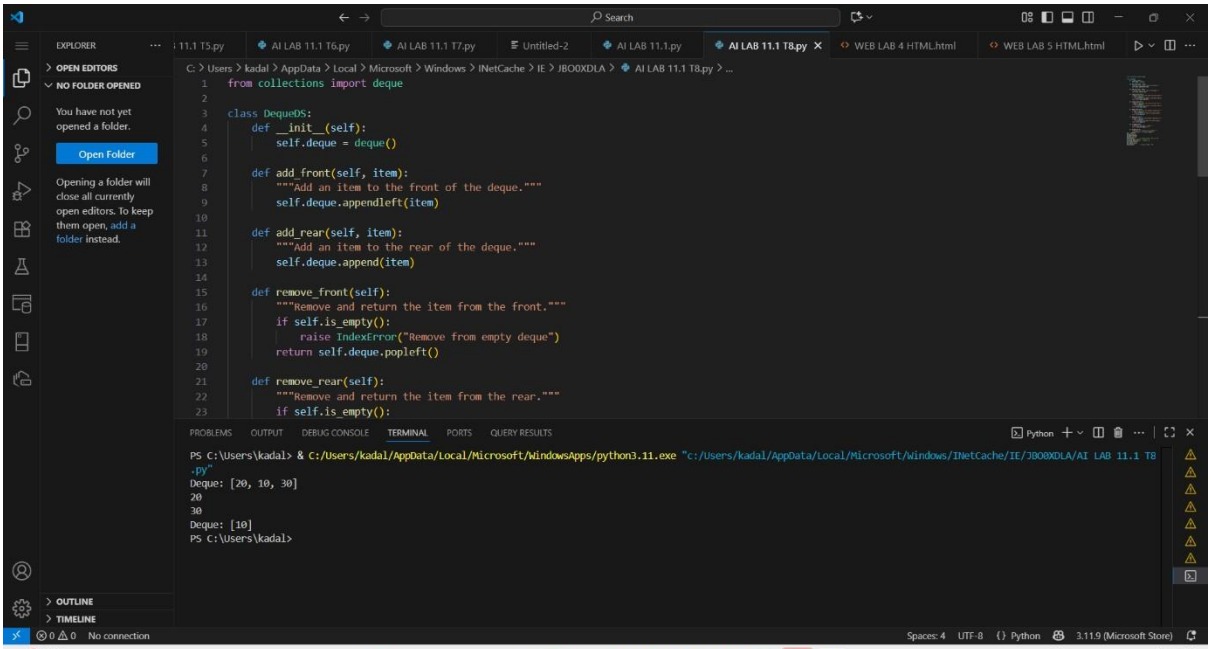
Deque

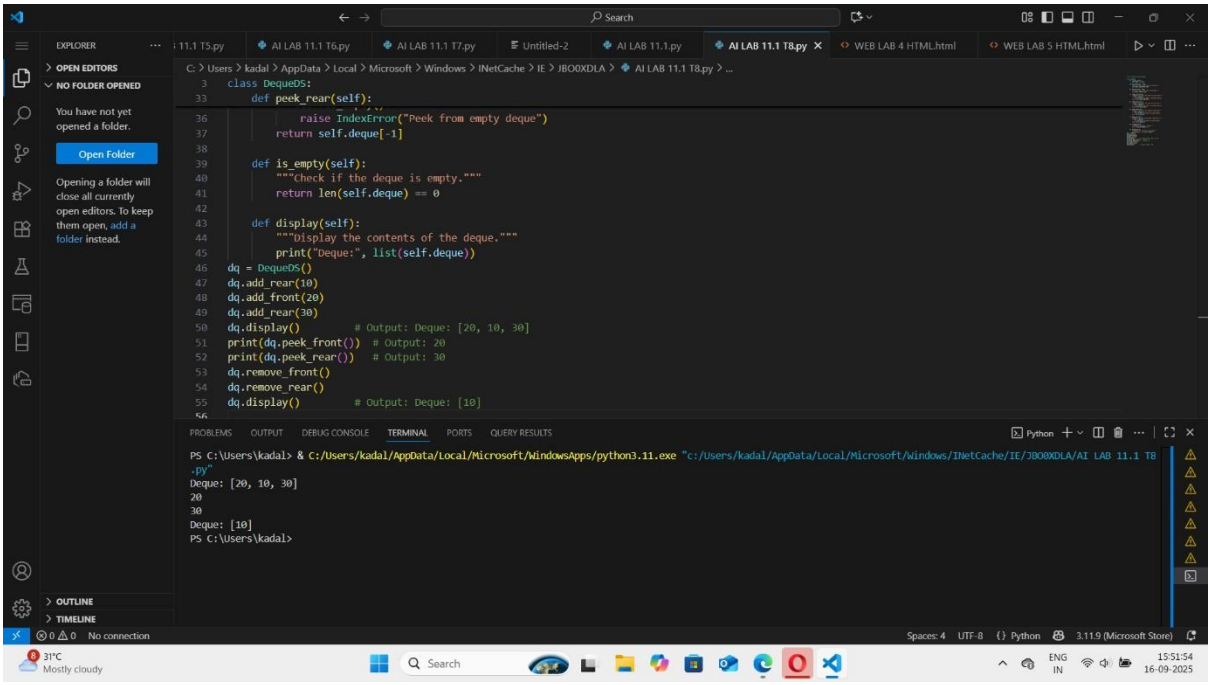
Task: Use AI to implement a double-ended queue using collections.deque.
Sample Input Code:
class DequeDS: pass.

PROMPT:

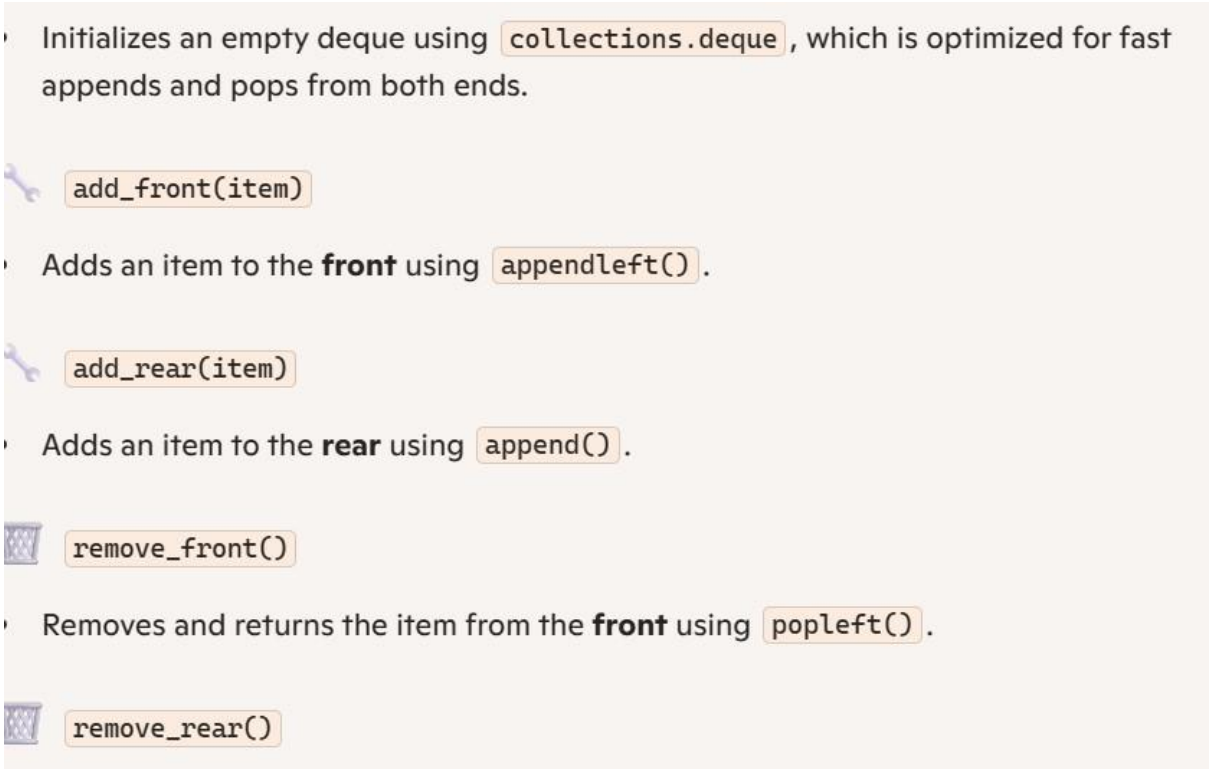
Generate python code and deque Task:
Use AI to implement a double-ended queue using collections.deque.
Sample Input Code:
class DequeDS: pass.

CODE &OUTPUT:





EXPLAINATION:



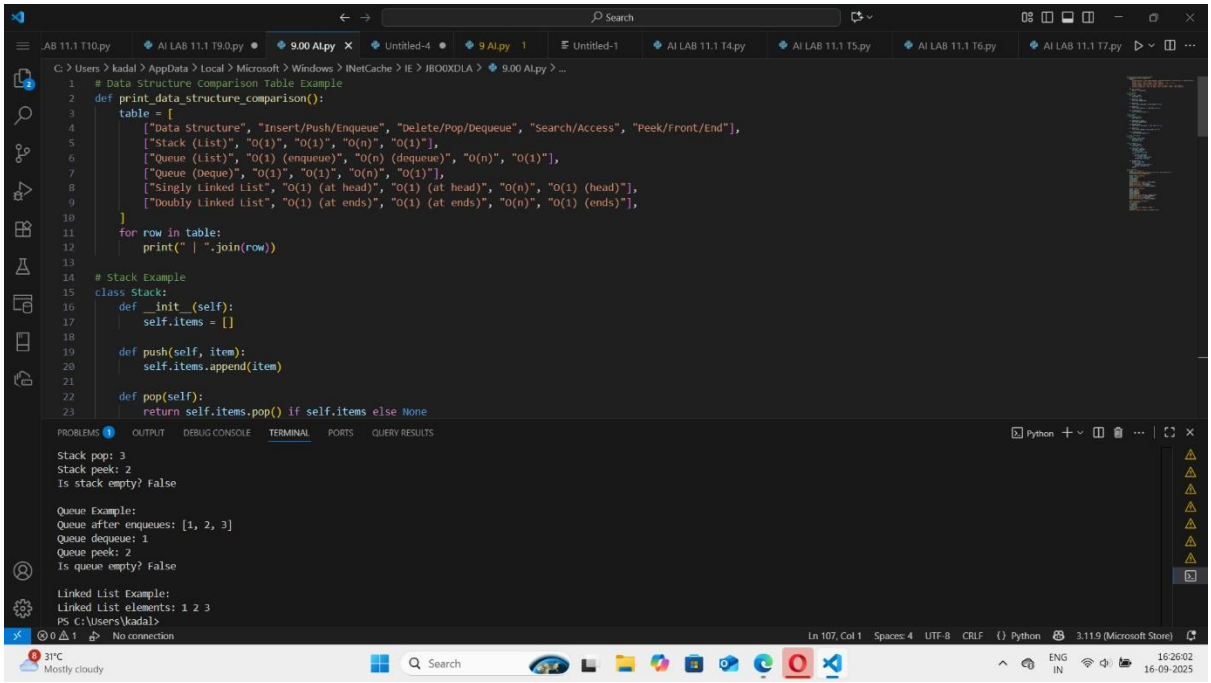
TASK 9:

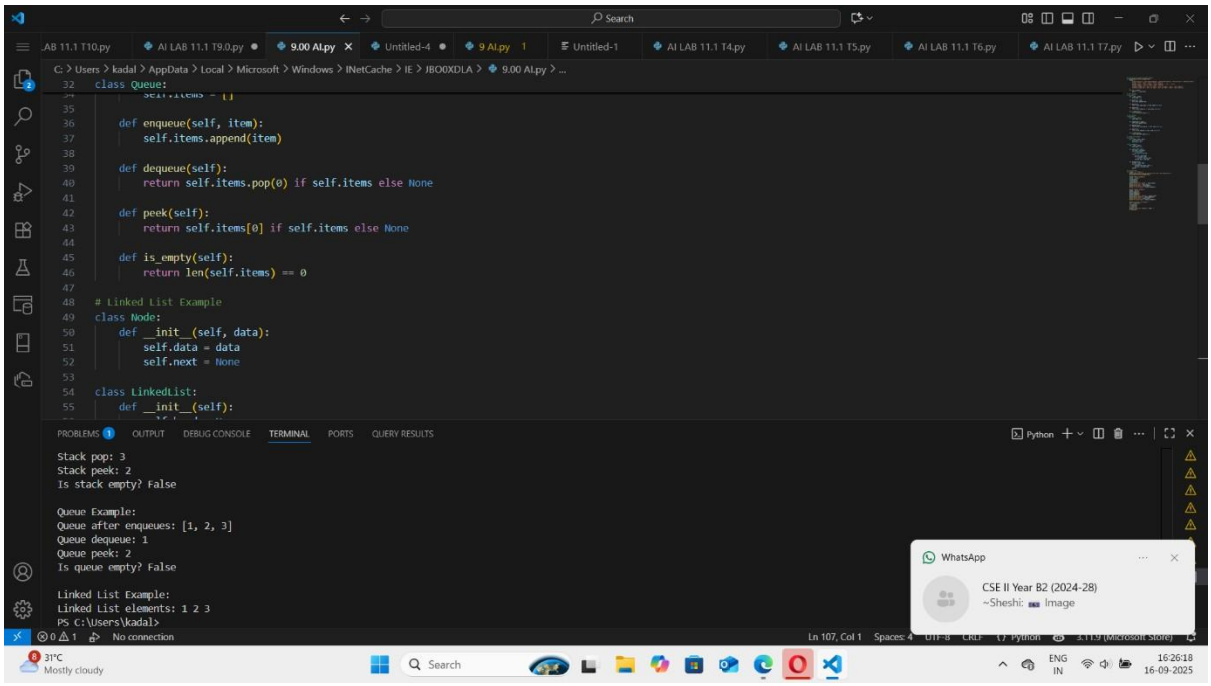
AI-Generated Data Structure Comparisons Task:
Use AI to generate a comparison table of different data structures (stack, queue, linked list, etc.) including time complexities.
Sample Input Code:
No code, prompt AI for a data structure comparison table.

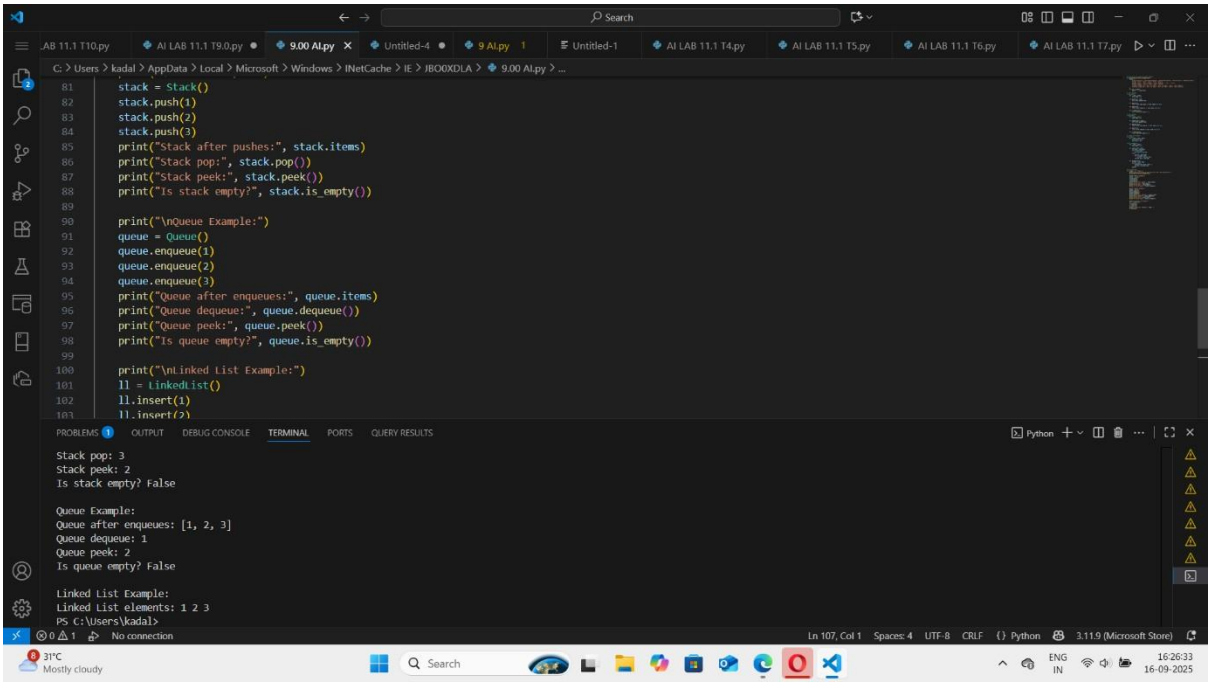
PROMPT:

Generate python code and 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.

CODE & OUTPUT:


Search
.AB 11.1 T10.py
AI LAB 11.1 T9.0.py
9.00 AI.py
Untitled-4
9 AI.py 1
Untitled-1
AI LAB 11.1 T4.py
AI LAB 11.1 T5.py
AI LAB 11.1 T6.py
AI LAB 11.1 T7.py
C: > Users > kadal > AppData > Local > Microsoft > Windows > INetCache > IE > JBO0XDLA > 9.00 AI.py > ...
# Data Structure Comparison Table Example
def print_data_structure_comparison():
table = [
["Data Structure", "Insert/Push/Enqueue", "Delete/Pop/Dequeue", "Search/Access", "Peek/Front/End"],
["Stack (List)", "O(1)", "O(1)", "O(n)", "O(1)"],
["Queue (List)", "O(1) (enqueue)", "O(n) (dequeue)", "O(n)", "O(1)"],
["Queue (Deque)", "O(1)", "O(1)", "O(n)", "O(1)"],
["Singly Linked List", "O(1) (at head)", "O(1) (at head)", "O(n)", "O(1) (head)"],
["Doubly Linked List", "O(1) (at ends)", "O(1) (at ends)", "O(n)", "O(1) (ends)"],
]
for row in table:
print(" | ".join(row))
# Stack Example
class Stack:
def __init__(self):
self.items = []
def push(self, item):
self.items.append(item)
def pop(self):
return self.items.pop() if self.items else None
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
QUERY RESULTS
Python
Stack pop: 3
Stack peek: 2
Is stack empty? False
Queue Example:
Queue after enqueues: [1, 2, 3]
Queue dequeue: 1
Queue peek: 2
Is queue empty? False
Linked List Example:
Linked List elements: 1 2 3
PS C:\Users\kadal>
No connection
Ln 107, Col 1
Spaces: 4
UTF-8
CRLF
Python
3.11.9 (Microsoft Store)
31°C
Mostly cloudy
Search
ENG
IN
16:26:02
16-09-2025


Search
.AB 11.1 T10.py
AI LAB 11.1 T9.0.py
9.00 AI.py
Untitled-4
9 AI.py 1
Untitled-1
AI LAB 11.1 T4.py
AI LAB 11.1 T5.py
AI LAB 11.1 T6.py
AI LAB 11.1 T7.py
C: > Users > kadal > AppData > Local > Microsoft > Windows > INetCache > IE > JBO0XDLA > 9.00 AI.py > ...
class Queue:
def enqueue(self, item):
self.items.append(item)
def dequeue(self):
return self.items.pop(0) if self.items else None
def peek(self):
return self.items[0] if self.items else None
def is_empty(self):
return len(self.items) == 0
# Linked List Example
class Node:
def __init__(self, data):
self.data = data
self.next = None
class LinkedList:
def __init__(self):
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
QUERY RESULTS
Python
Stack pop: 3
Stack peek: 2
Is stack empty? False
Queue Example:
Queue after enqueues: [1, 2, 3]
Queue dequeue: 1
Queue peek: 2
Is queue empty? False
Linked List Example:
Linked List elements: 1 2 3
PS C:\Users\kadal>
WhatsApp
CSE II Year B2 (2024-28)
~Sheshi: Image
No connection
Ln 107, Col 1
Spaces: 4
UTF-8
CRLF
Python
3.11.9 (Microsoft Store)
31°C
Mostly cloudy
Search
ENG
IN
16:26:18
16-09-2025



EXPLAINATION:

`def print_data_structure_comparison():`

- Defines a function that prints a formatted comparison table.

`table = [...]`

- A list of lists, where each inner list represents a row in the table.
- The first row is the header: column titles like "Insert", "Delete", etc.
- Each subsequent row compares a specific data structure.

`" | ".join(row)`

- Joins each element in the row with `" | "` to mimic a table format.
- This makes the output readable and aligned like a markdown-style table.

TASK 10:

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: o Stack o Queue o Priority Queue o Linked List

o Binary Search Tree (BST)
o Graph o Hash Table o
Deque

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

PROMPT:

Generate python code and 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.

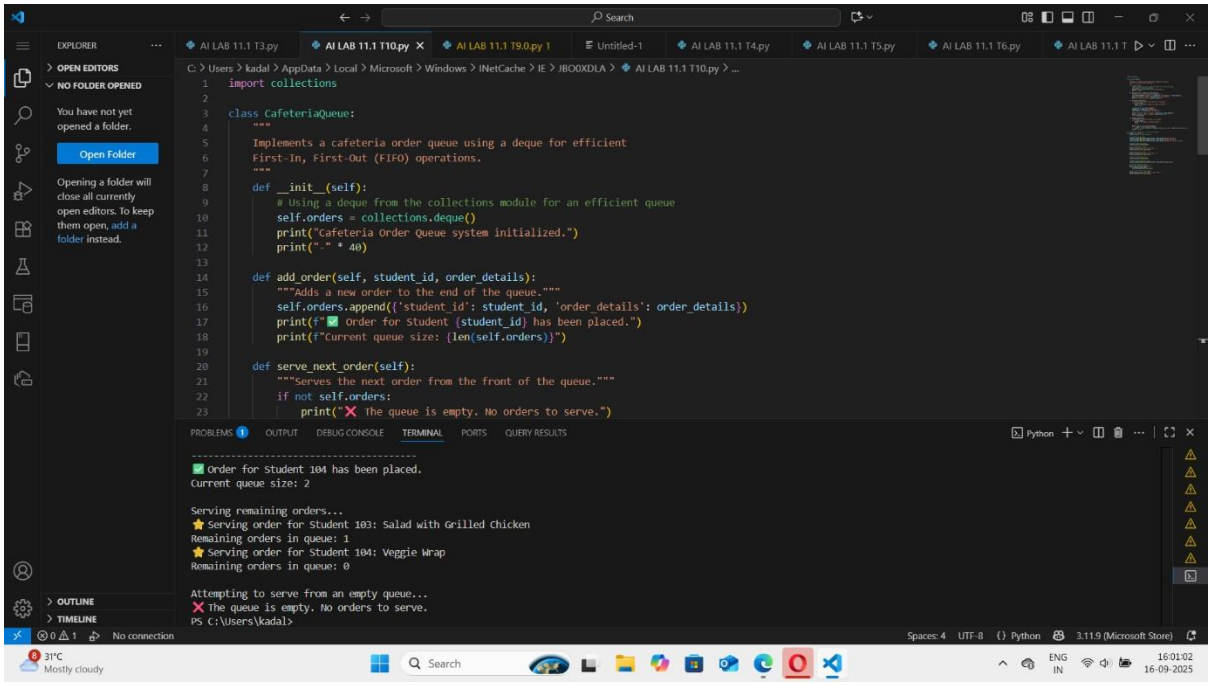
2. Event Registration System – Manage participants in events with quick search and removal.
3. Library Book Borrowing – Keep track of available books and their due dates.
4. Bus Scheduling System – Maintain bus routes and stop connections.
5. Cafeteria Order Queue – Serve students in the order they arrive.

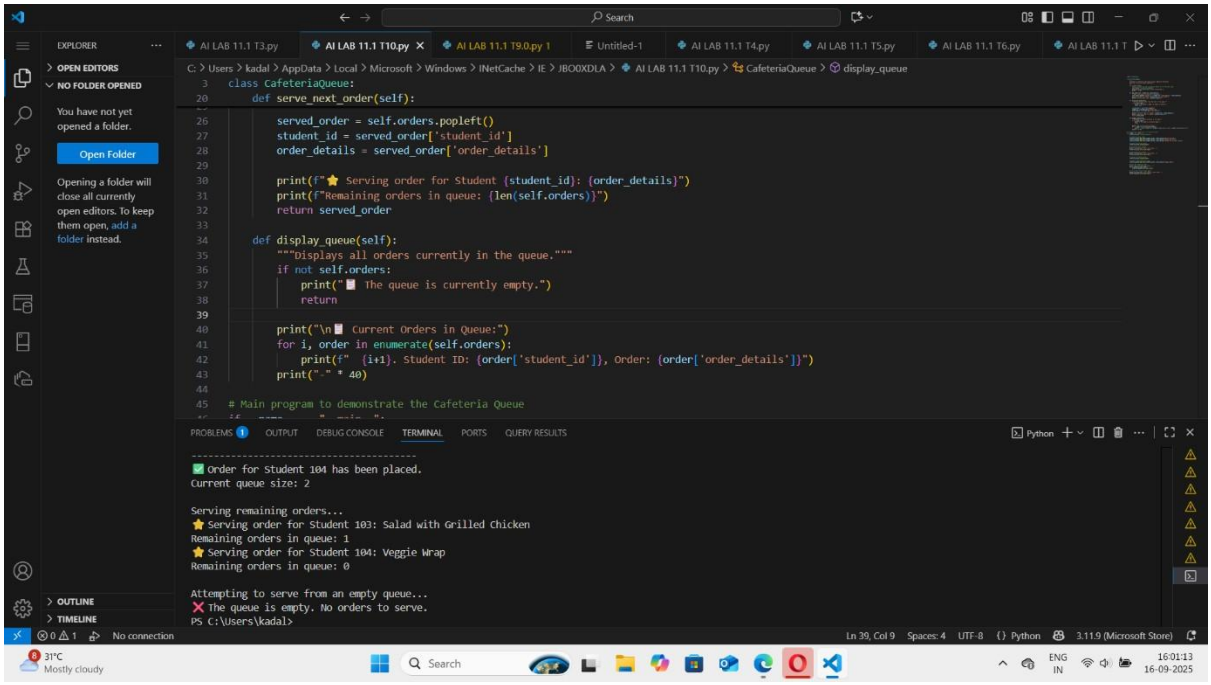
Student Task:

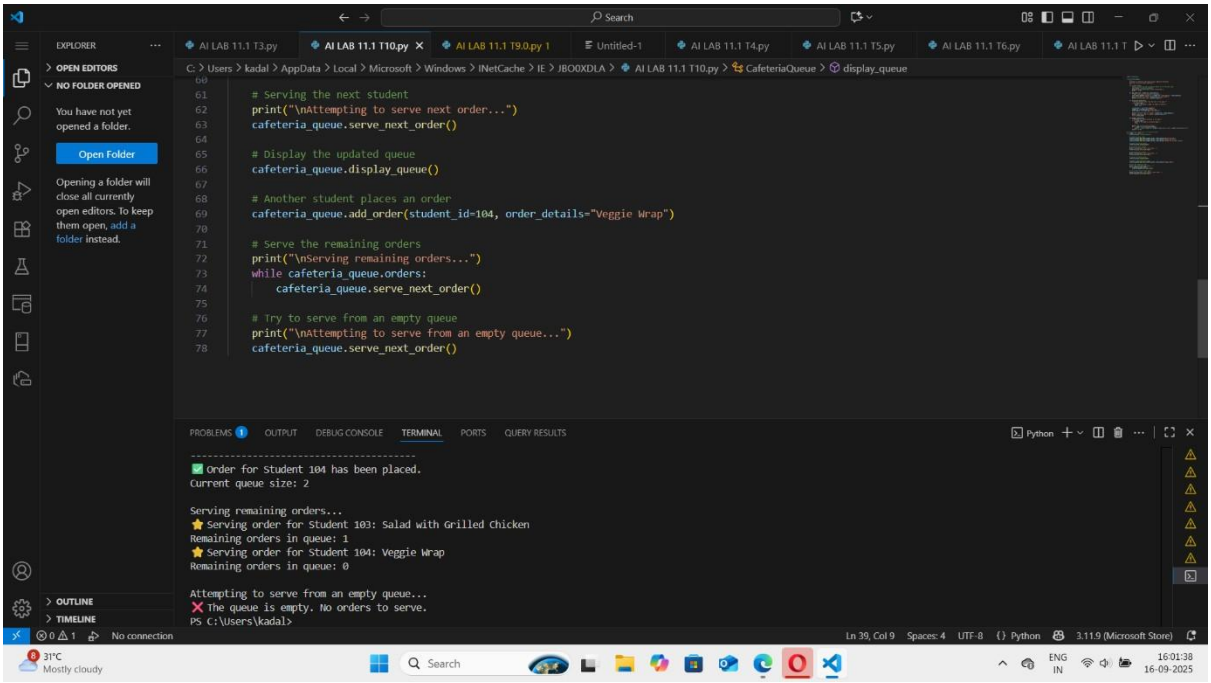
• For each feature, select the most appropriate data structure from the list below:
o Stack o Queue o Priority Queue o Linked List
o Binary Search Tree (BST)
o Graph o
Hash Table o
Deque
• Justify your choice in 2–3 sentences per feature.

- Implement one selected feature as a working Python program with AI- assisted code generation.

CODE & OUTPUT:







EXPLAINATION:

- Initializes an empty list `queue` to store student names.

🍽️ `place_order(student_name)`

- Adds a student to the end of the queue using `append()` — O(1) time.

📃 `serve_order()`

- Removes and returns the first student using `pop(0)` — O(n) time due to shifting.
- Raises an error if the queue is empty.

👀 `peek_next()`

- Returns the first student without removing them — O(1) time.

✅ `is_empty()`

- Checks if the queue is empty — O(1) time.