**Assignment 11**

Htno : 2403a52423

**Task Description #1** – Stack Implementation

prompt **:** Use AI to generate a Stack class with push, pop, peek, and is\_empty  
methods.

**Sample Input Code:**  
class Stack:class Stack:

    """

    A simple stack implementation using Python list.

    Methods:

        push(item): Add an item to the top of the stack.

        pop(): Remove and return the top item of the stack.

        peek(): Return the top item without removing it.

        is\_empty(): Return True if the stack is empty, else False.

        display(): Print the current contents of the stack.

    """

    def \_\_init\_\_(self):

        self.\_data = []

    def push(self, item):

        """Push an item onto the stack."""

        self.\_data.append(item)

    def pop(self):

        """Pop the top item off the stack. Raises IndexError if empty."""

        if not self.\_data:

            raise IndexError("pop from empty stack")

        return self.\_data.pop()

    def peek(self):

        """Return the top item without removing it. Raises IndexError if empty."""

        if not self.\_data:

            raise IndexError("peek from empty stack")

        return self.\_data[-1]

    def is\_empty(self):

        """Check if the stack is empty."""

        return len(self.\_data) == 0

    def display(self):

        """Display the contents of the stack (top shown last)."""

        print("Stack contents:", self.\_data)

# ---------- TEST CASES ----------

if \_\_name\_\_ == "\_\_main\_\_":

    s = Stack()

    print("\n--- Stack Operations Demo ---")

    # Test 1: Stack should be empty initially

    assert s.is\_empty() == True

    s.display()

    # Test 2: Push items

    s.push(10)

    s.push(20)

    s.push(30)

    s.display()  # Expect [10, 20, 30]

    assert s.peek() == 30

    # Test 3: Pop one element

    popped = s.pop()

    print("Popped:", popped)

    s.display()  # Expect [10, 20]

    assert popped == 30

    # Test 4: Peek top element

    print("Top element:", s.peek())

    assert s.peek() == 20

    # Test 5: Pop remaining

    s.pop()

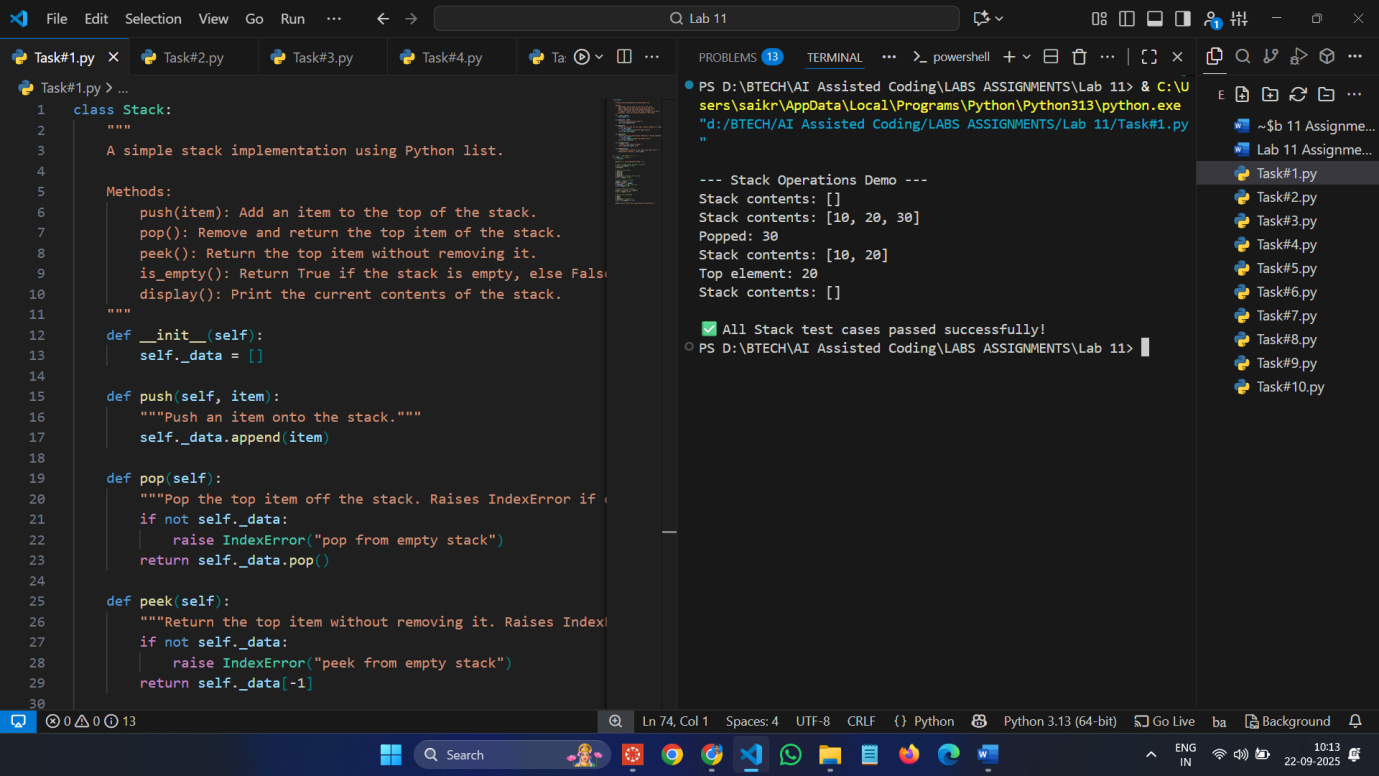
    s.pop()

    s.display()  # Expect []

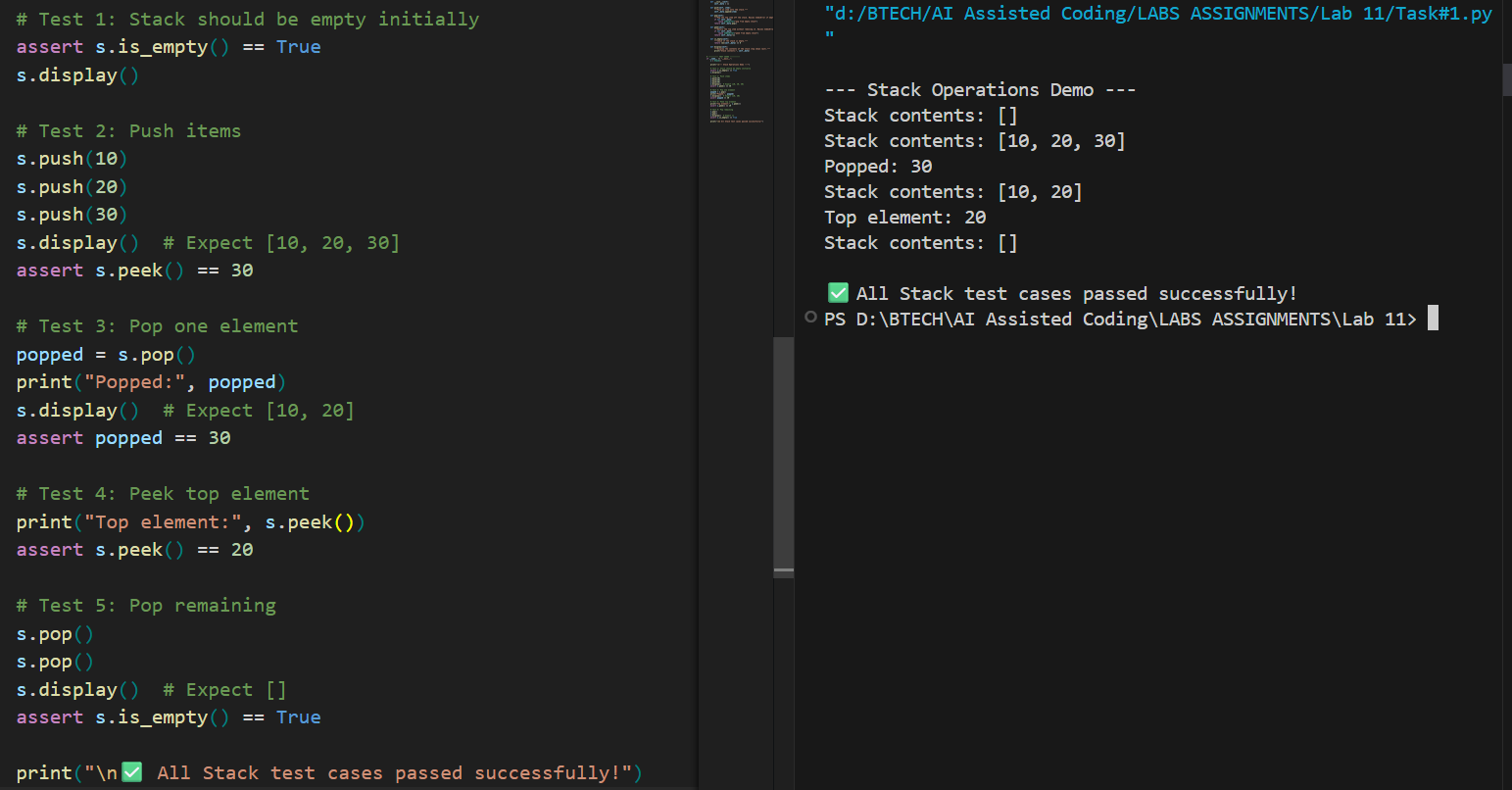
    assert s.is\_empty() == True

    print("\n✅ All Stack test cases passed successfully!")

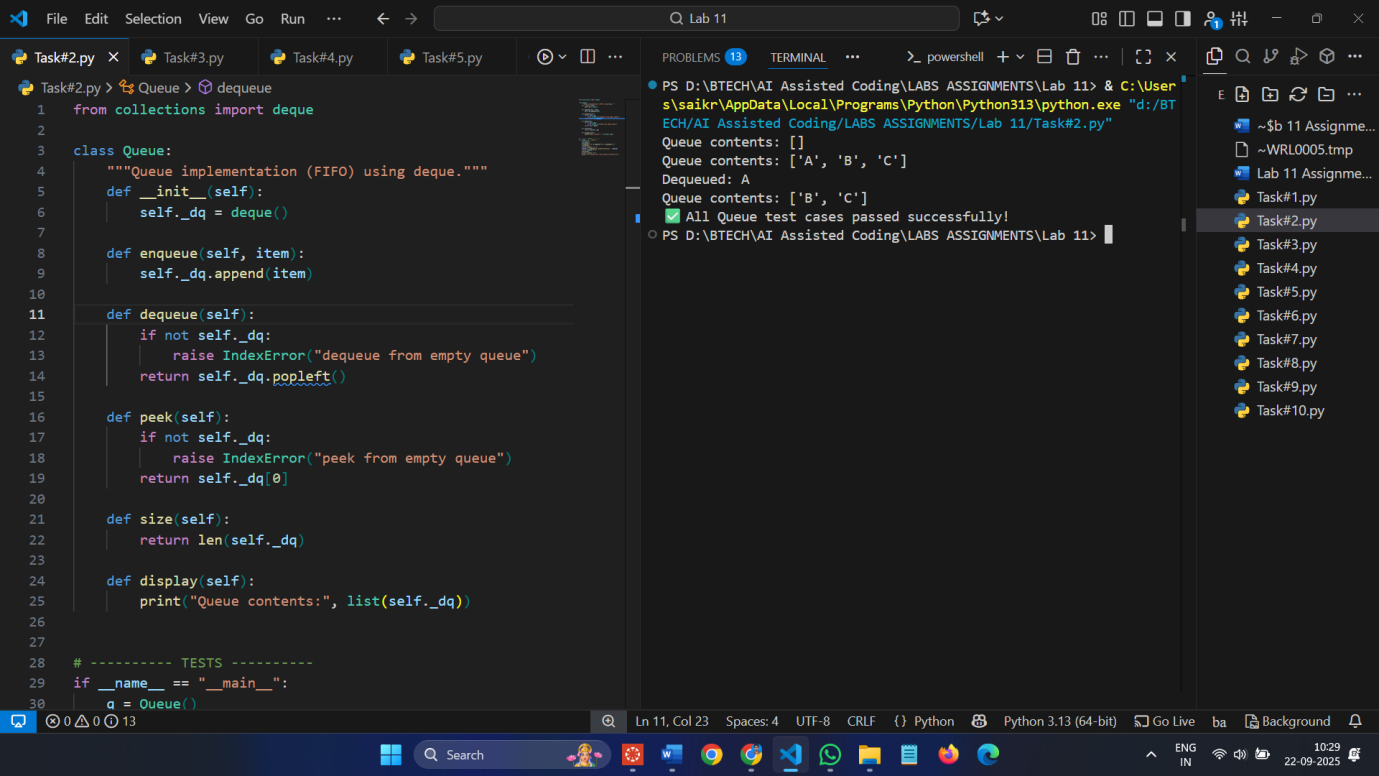
pass  
**Expected Output:**



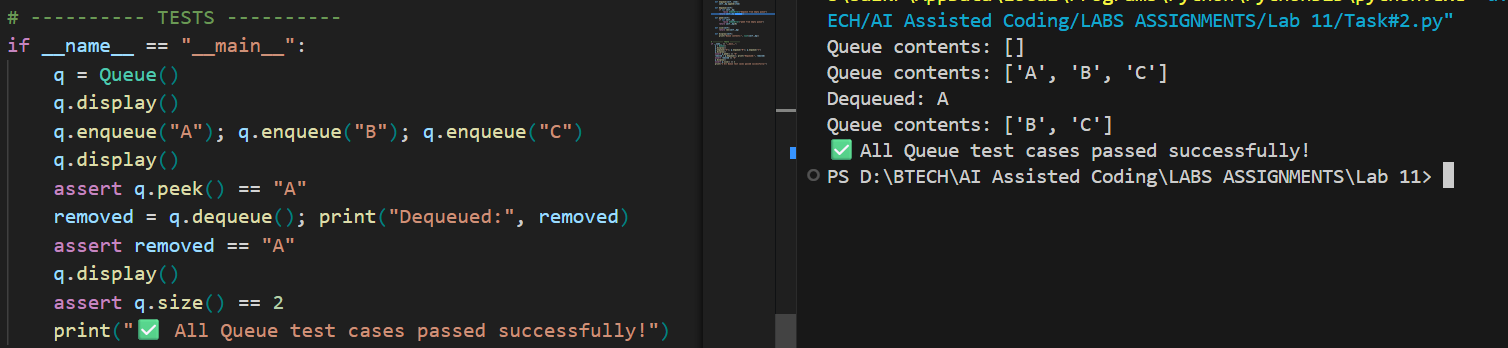
**3 assert test cases :**

****

**Task Description #2** – Queue Implementation  
**Task:** Use AI to implement a Queue using Python lists.  
Prompt **:**  
class Queue:  
pass  
**Expected Output:**

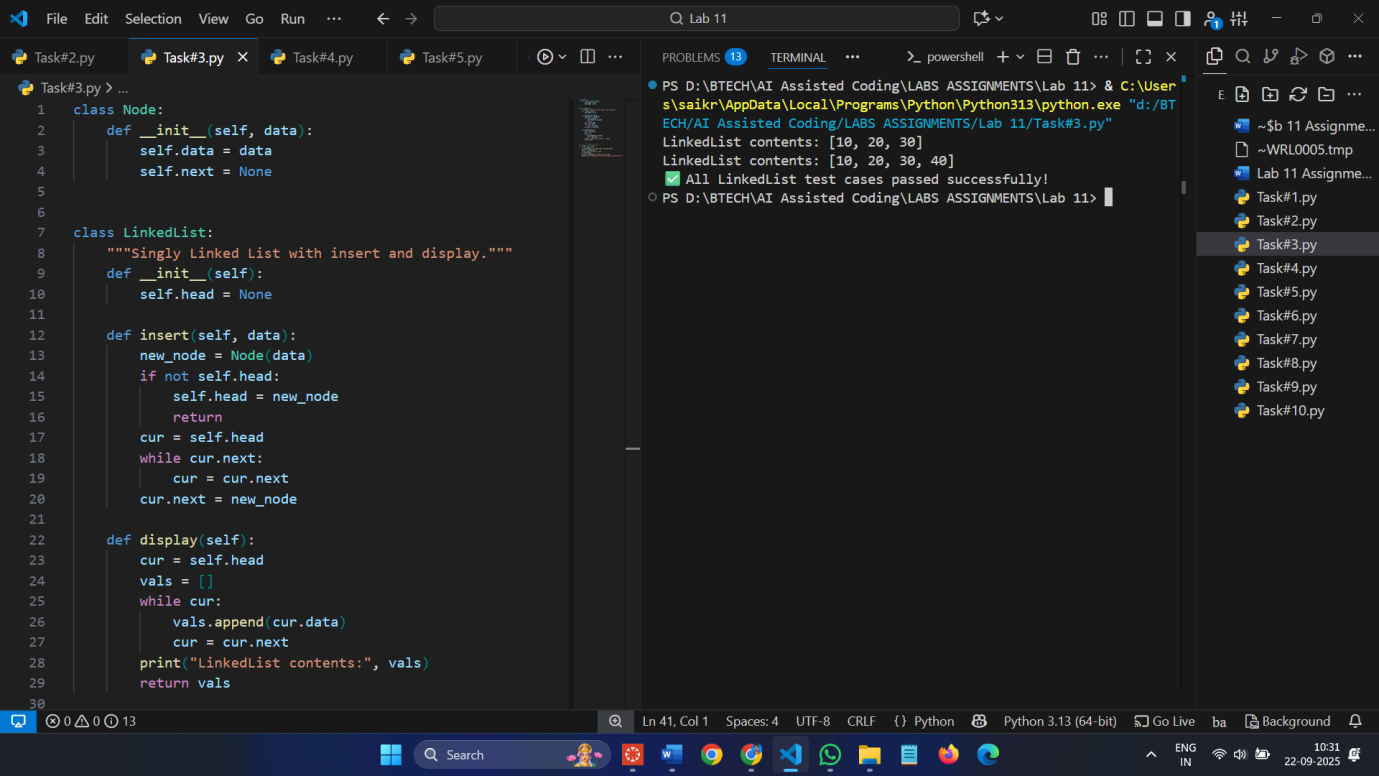


**3 assert test cases :**

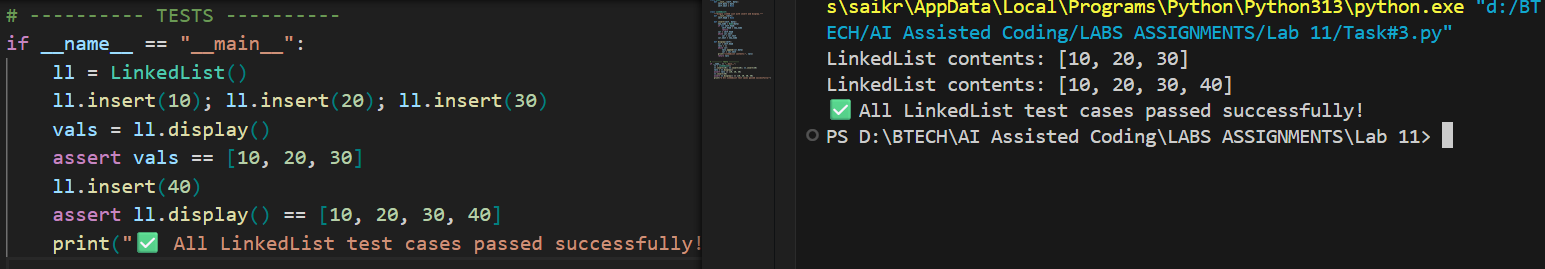


**Task Description #3** – Linked List  
prompt: Use AI to generate a Singly Linked List with insert and display methods.  
**Sample Input Code:**  
class Node:  
pass  
class LinkedList:  
pass

**Expected Output:**

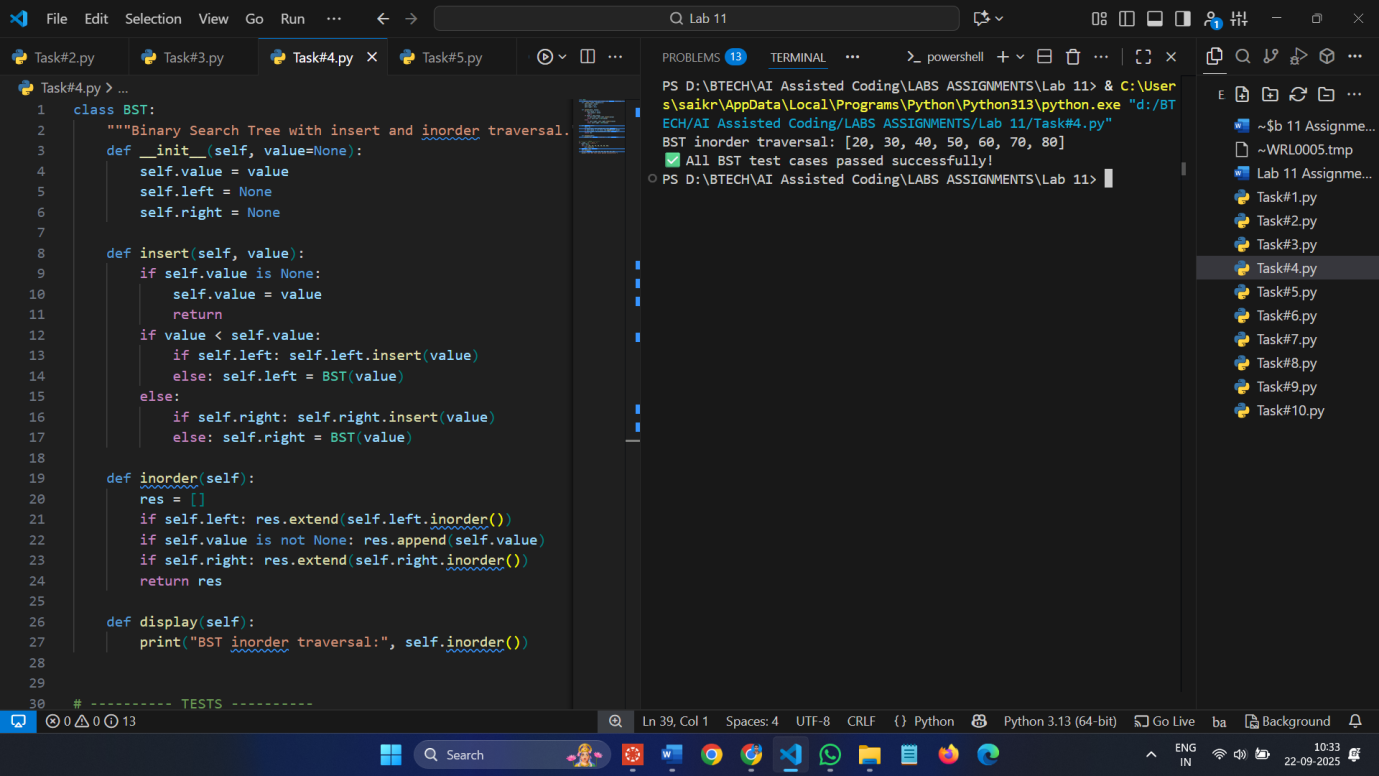


**3 assert test cases :**

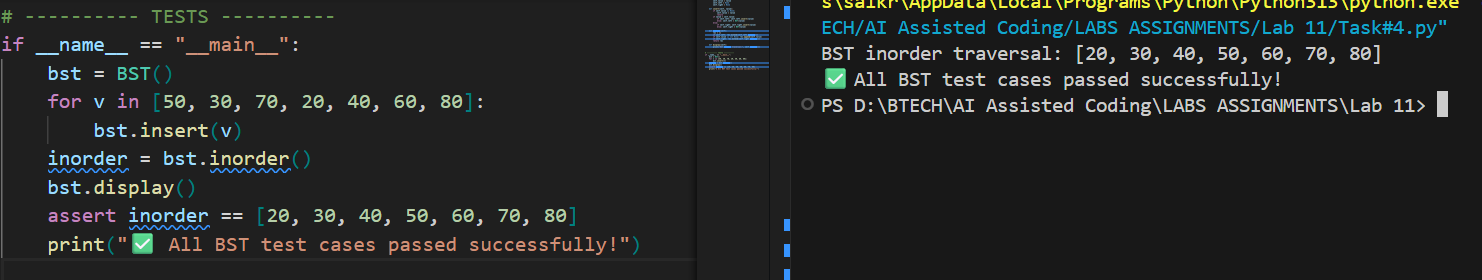


**Task Description #4** – Binary Search Tree (BST)  
prompt **:** Use AI to create a BST with insert and in-order traversal methods.  
**Sample Input Code:**  
class BST:  
pass

**Expected Output:**



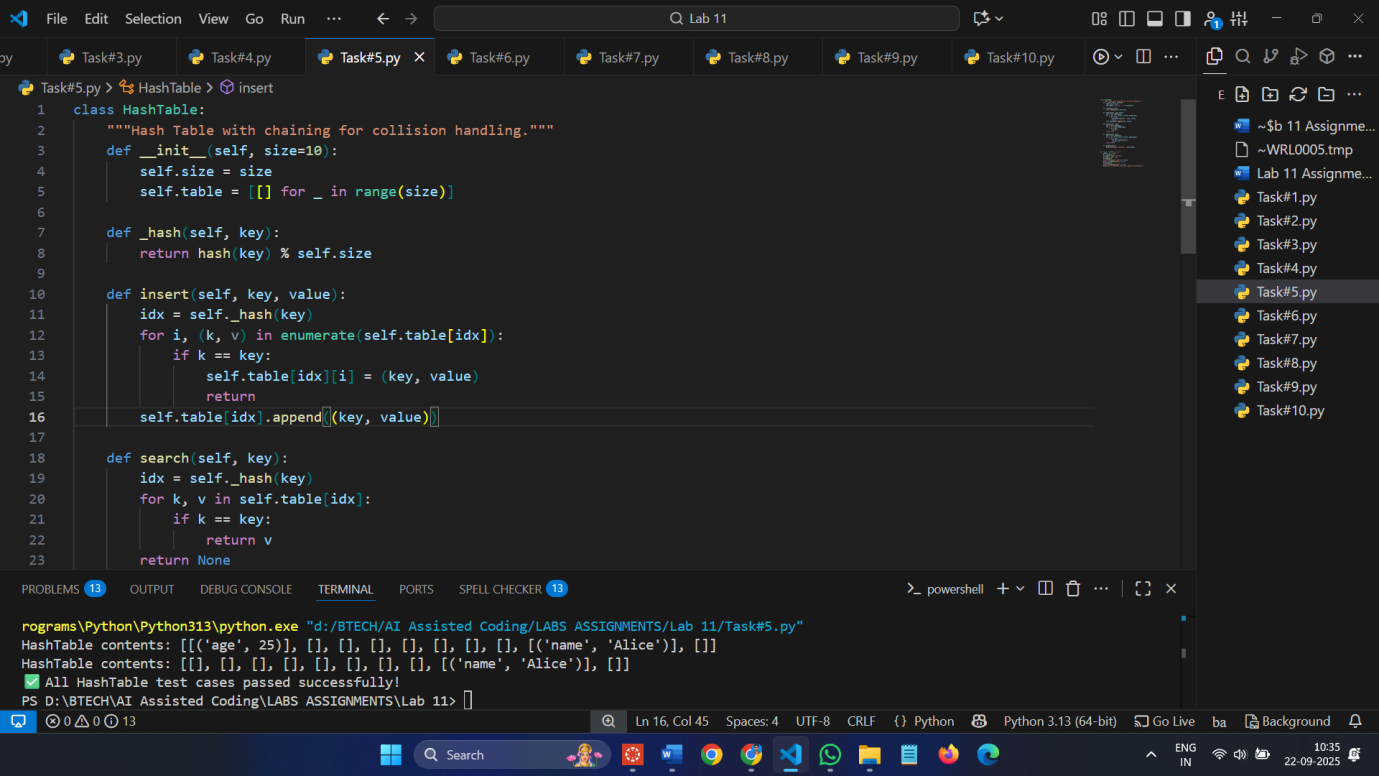
**3 assert test cases :**



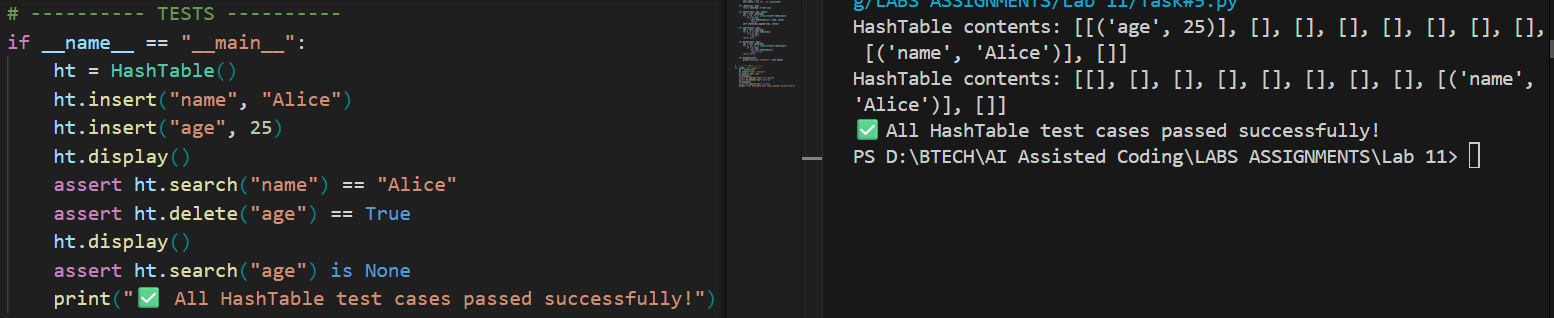
**Task Description #5** – Hash Table  
prompt **:** Use AI to implement a hash table with basic insert, search, and delete

methods.  
**Sample Input Code:**  
class HashTable:  
pass

**Expected Output:**

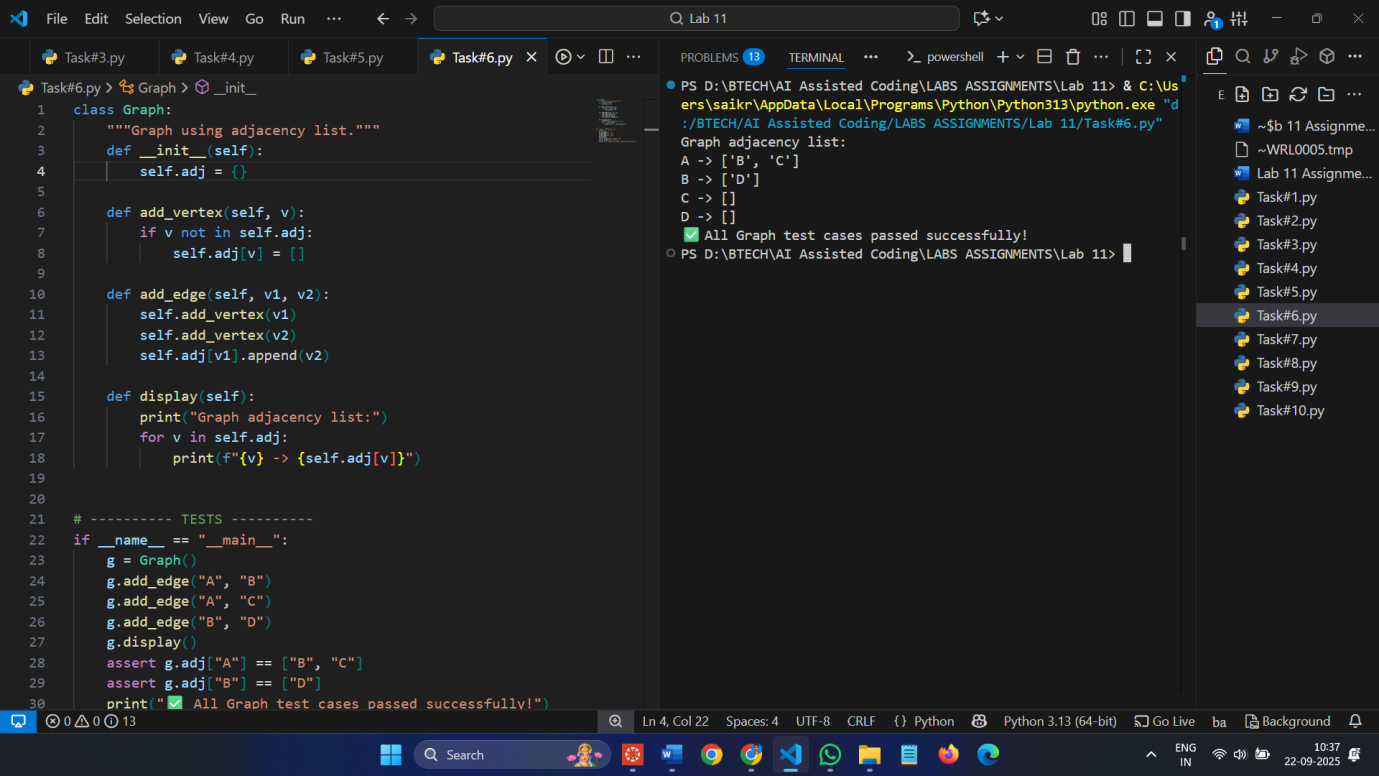


**3 assert test cases :**

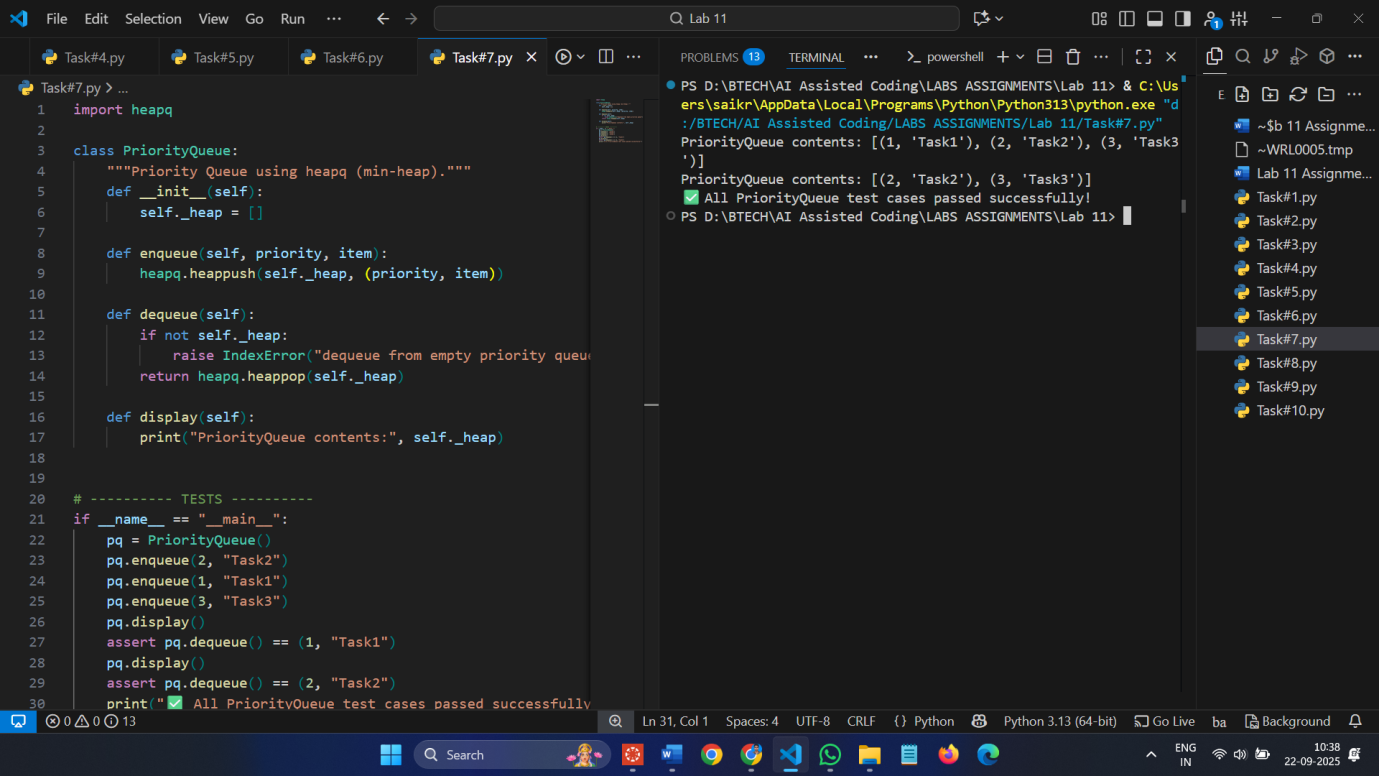


**Task Description #6** – Graph Representation  
prompt**:** Use AI to implement a graph using an adjacency list.  
**Sample Input Code:**  
class Graph:  
pass

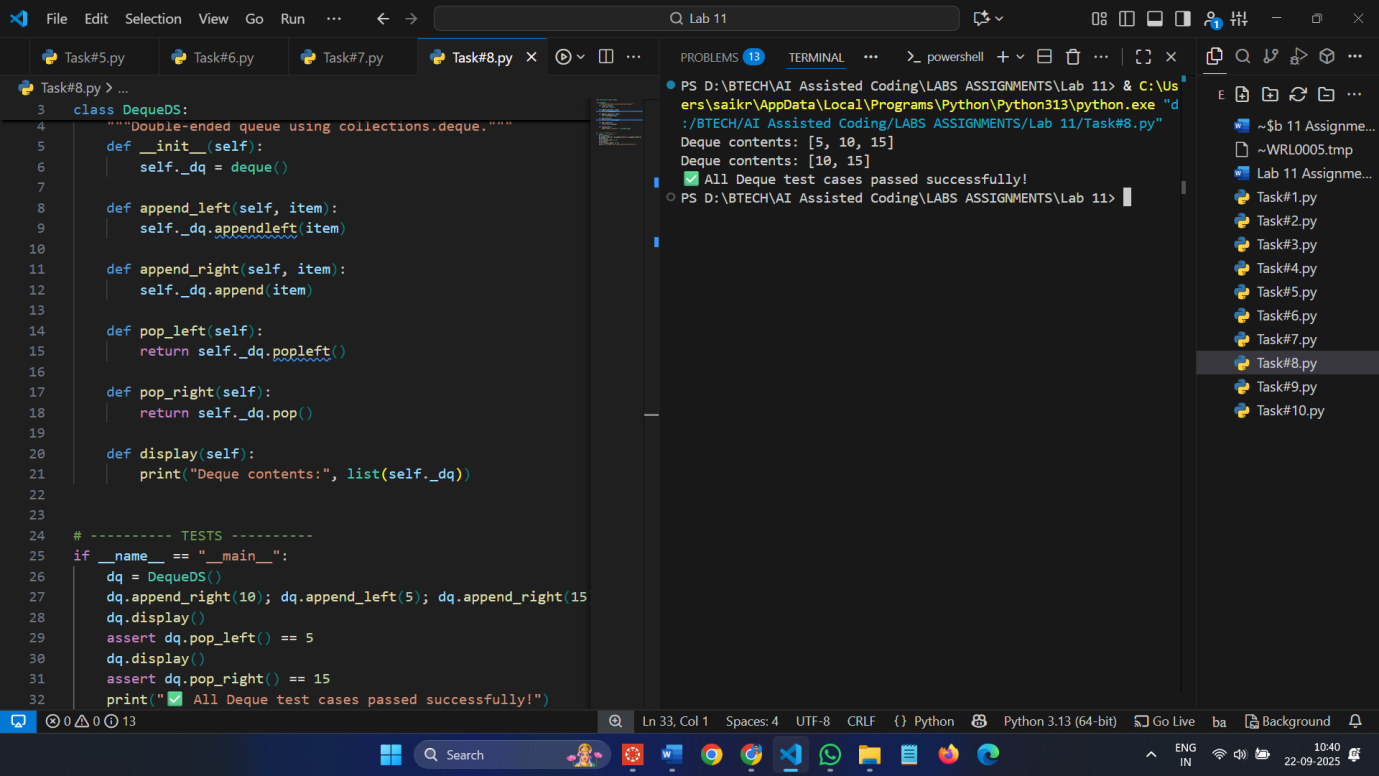
**Expected Output / 3 assert test cases :**



**Task Description #7** – Priority Queue  
prompt**:** Use AI to implement a priority queue using Python’s heapq module.  
**Sample Input Code:**  
class PriorityQueue:  
pass  
**Expected Output / 3 assert test cases :**

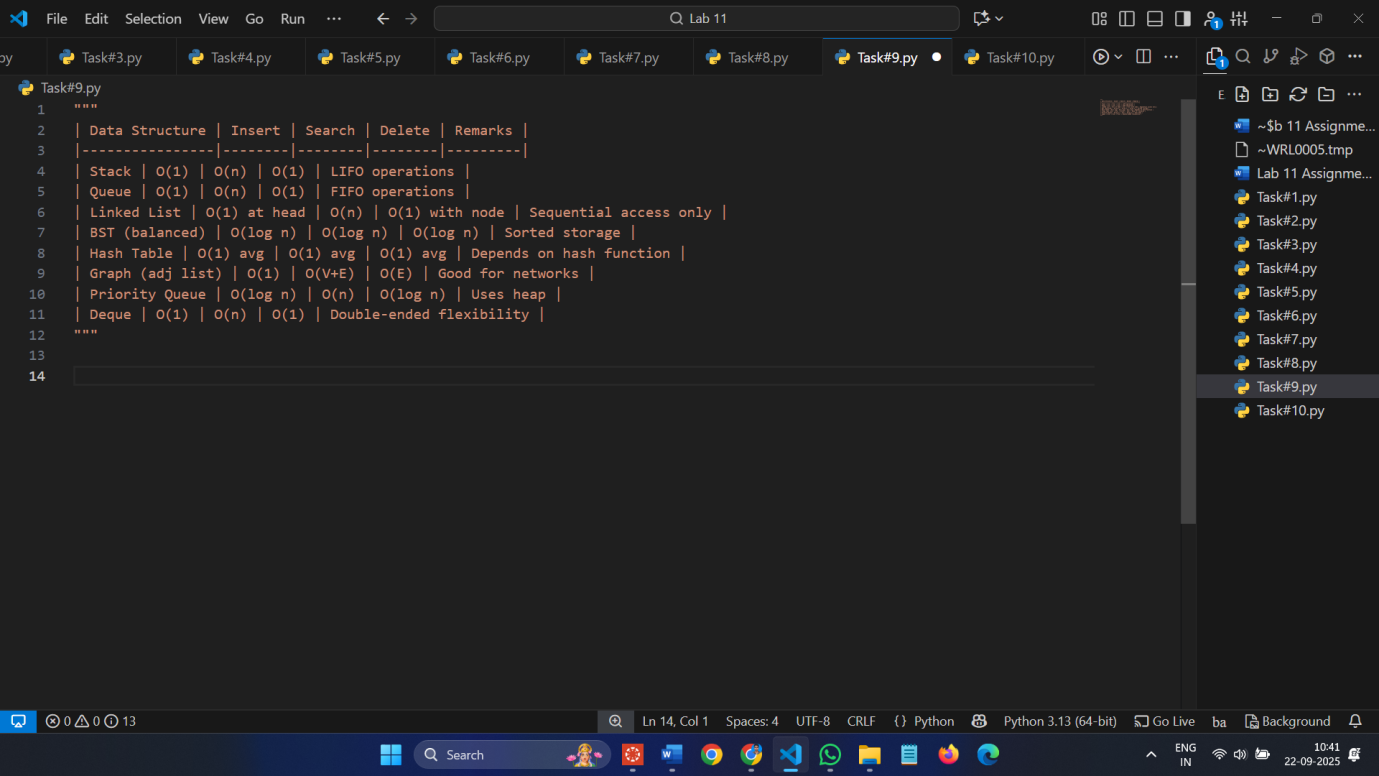


**Task Description #8** – Deque  
**Task:** Use AI to implement a double-ended queue using collections.deque.  
**Sample Input Code:**  
class DequeDS:  
pass  
**Expected Output / 3 assert test cases :**



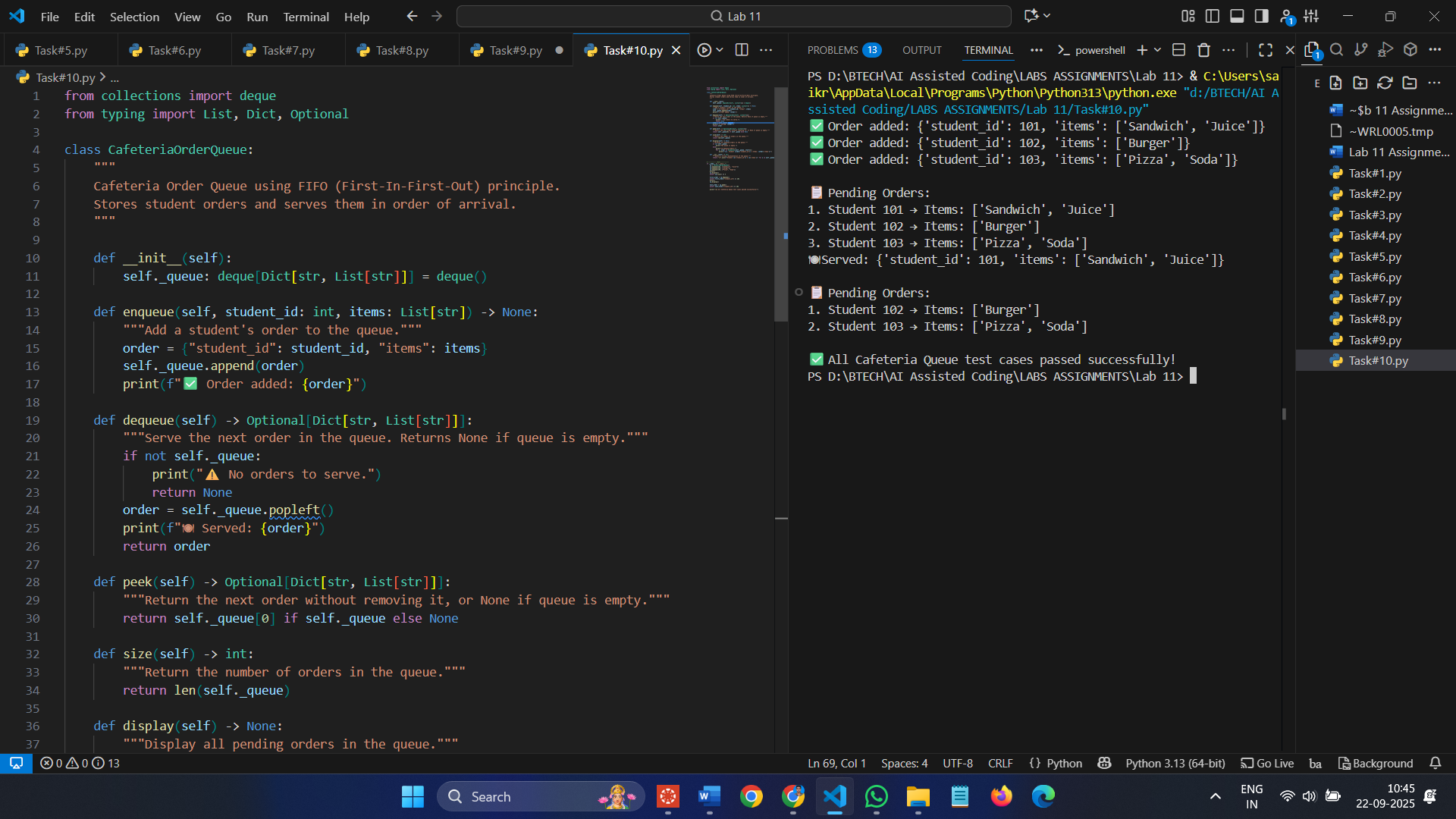
**Task Description #9** – AI-Generated Data Structure Comparisons  
prompt**:** Use AI to generate a comparison table of different data structures (stack,  
queue, linked list, etc.) including time complexities.  
**Sample Input Code:**  
# No code, prompt AI for a data structure comparison table

**Expected Output:**

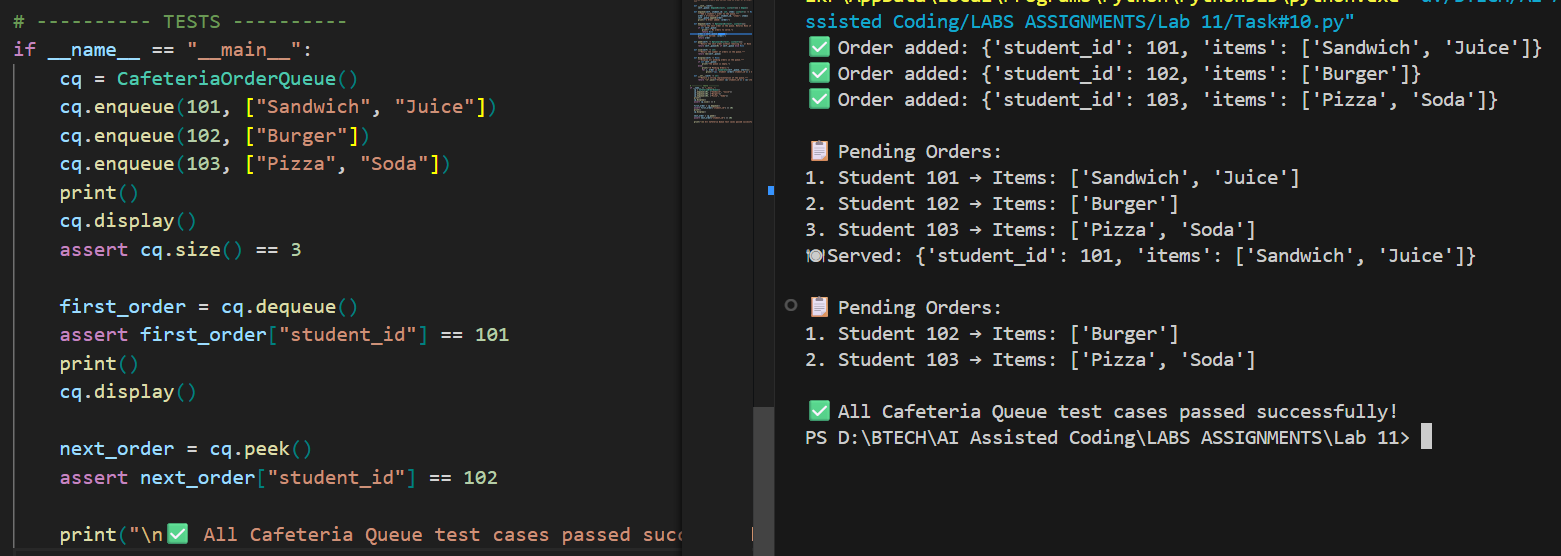


**Task Description #10** Real-Time Application Challenge – Choose the  
Right Data Structure  
**Scenario:**

**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.  
**Expected Output:**  
• A table mapping feature → chosen data structure → justification.  
• A functional Python program implementing the chosen feature with  
comments and docstrings



**3 assert test cases :**



**Observation :**

**Task 1 – Stack :** LIFO operations work correctly. push, pop, peek behave as expected. display() shows contents; all tests passed.

**Task 2 – Queue :** FIFO order maintained. enqueue, dequeue, peek work correctly. display() confirms order; tests passed.

**Task 3 – Linked List :** Insertion at tail works. display() shows list order. Tests confirm correct node addition.

**Task 4 – Binary Search Tree (BST) :** Recursive insert and in-order traversal produce sorted output. display() verified; tests passed.

**Task 5 – Hash Table :** Insert, search, and delete work correctly using chaining. display() shows buckets; all tests passed.

**Task 6 – Graph :** Adjacency list stores vertices and edges correctly. display() confirms connections; tests passed.

**Task 7 – Priority Queue :** Min-heap ensures correct priority order. enqueue/dequeue work; display() shows heap.

**Task 8 – Deque :** Double-ended insertion/removal works as expected. display() shows contents; all tests passed.

**Task 9 – Data Structure Comparisons :** Comparison table correctly shows time complexities and key differences. Useful reference for choosing structures.

**Task 10 – Cafeteria Queue :** FIFO order works correctly; enqueue adds orders, dequeue serves them in order.  
peek() shows next order, display() visualizes pending orders clearly; tests passed.