ASSIGNMENT – 11

NAME: R.SURYANARAYANA

HT.NO: 2403A52038

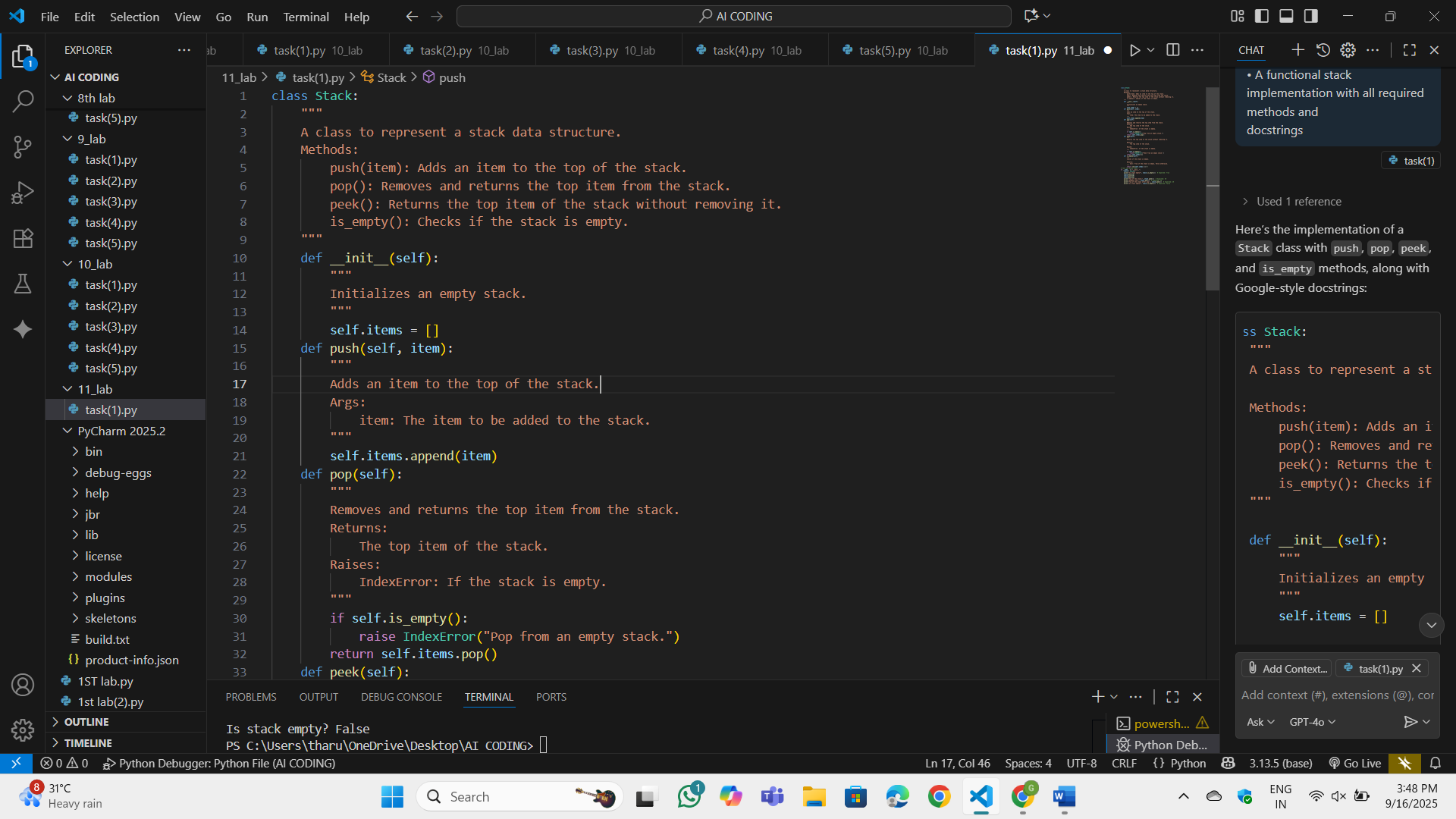
BATCH: 03

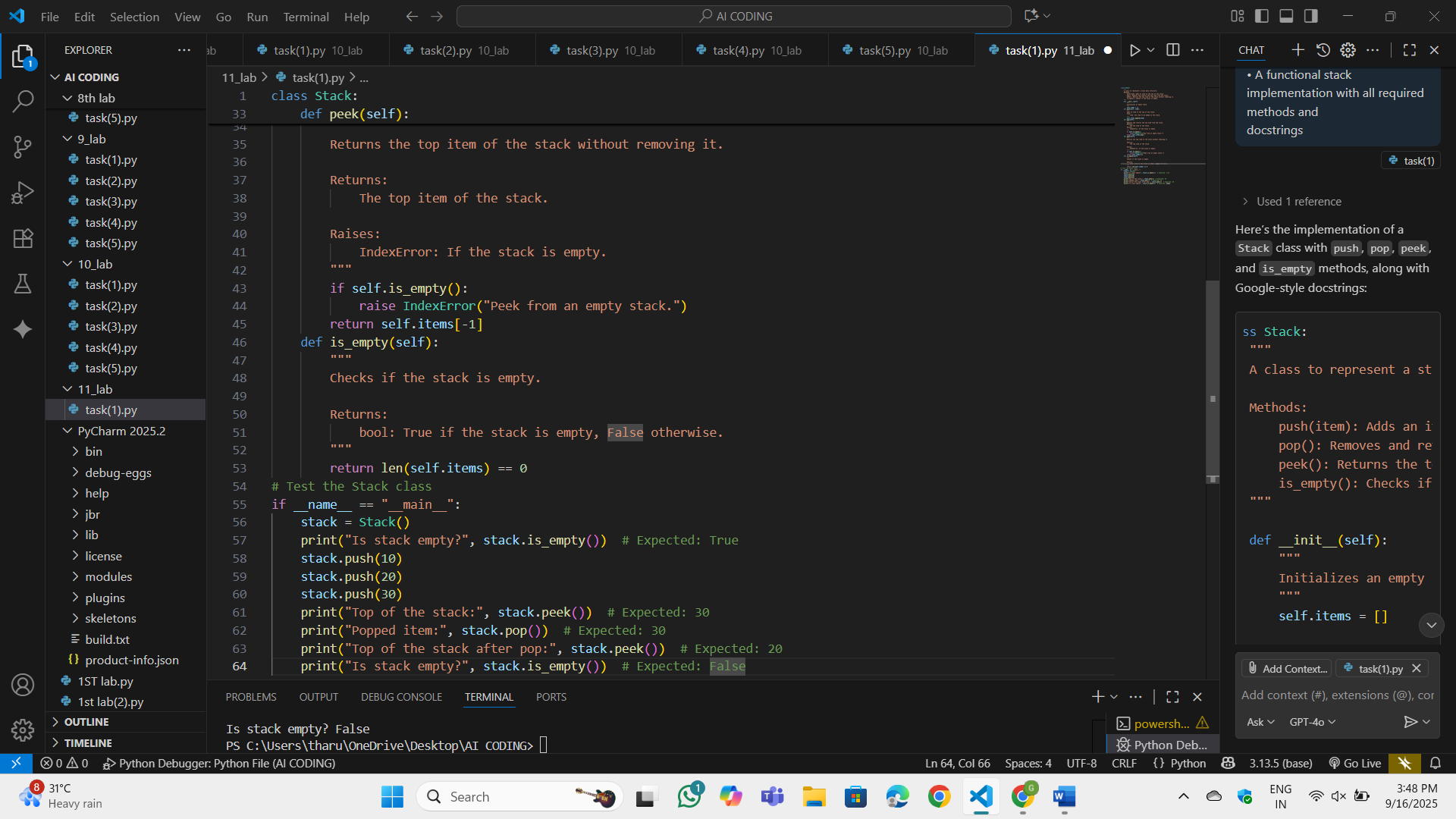
Task-1

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

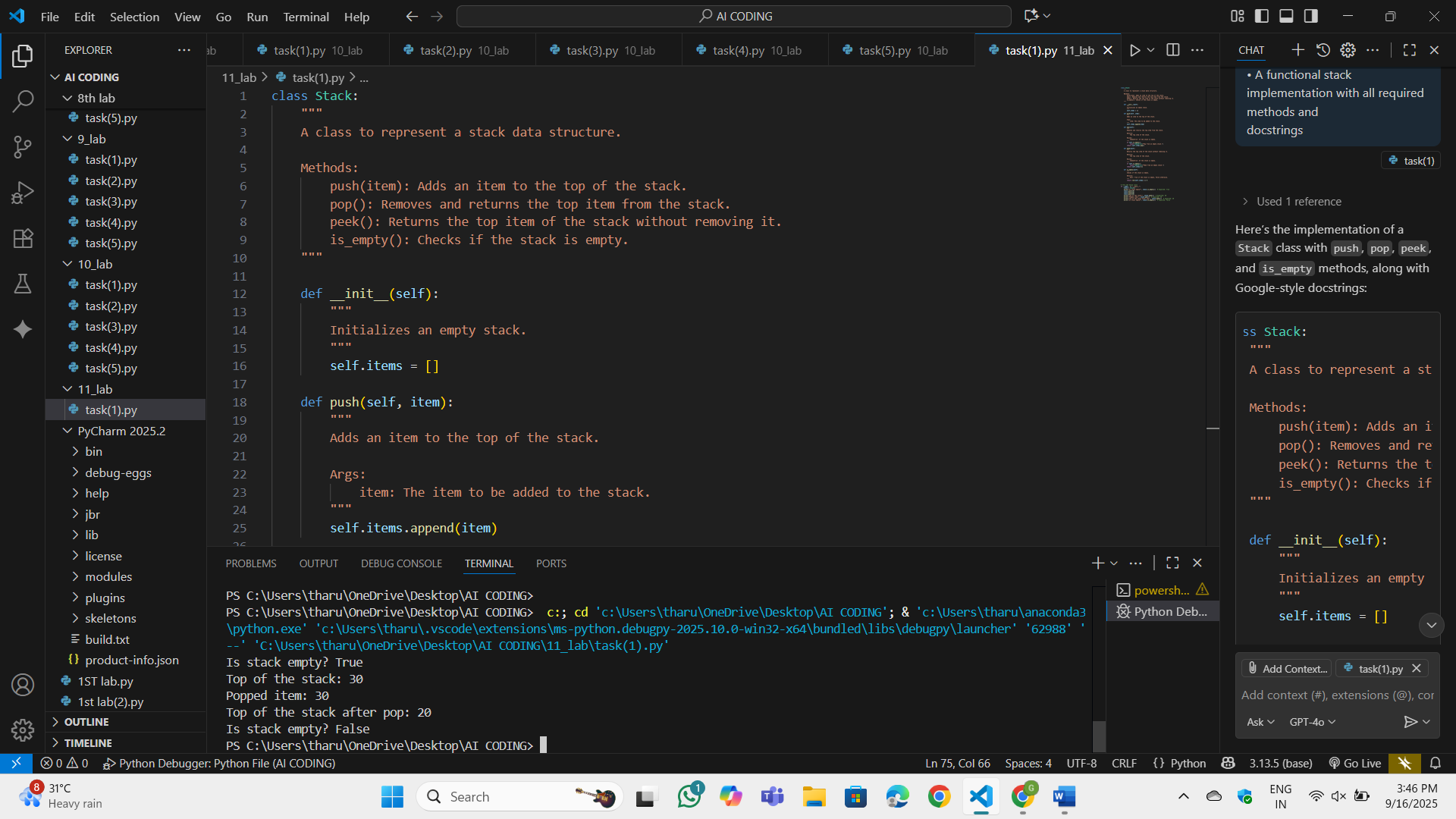
Prompt: generate a Stack class with push, pop, peek, and is\_empty  
methods.

Code:





OP:



Observation:

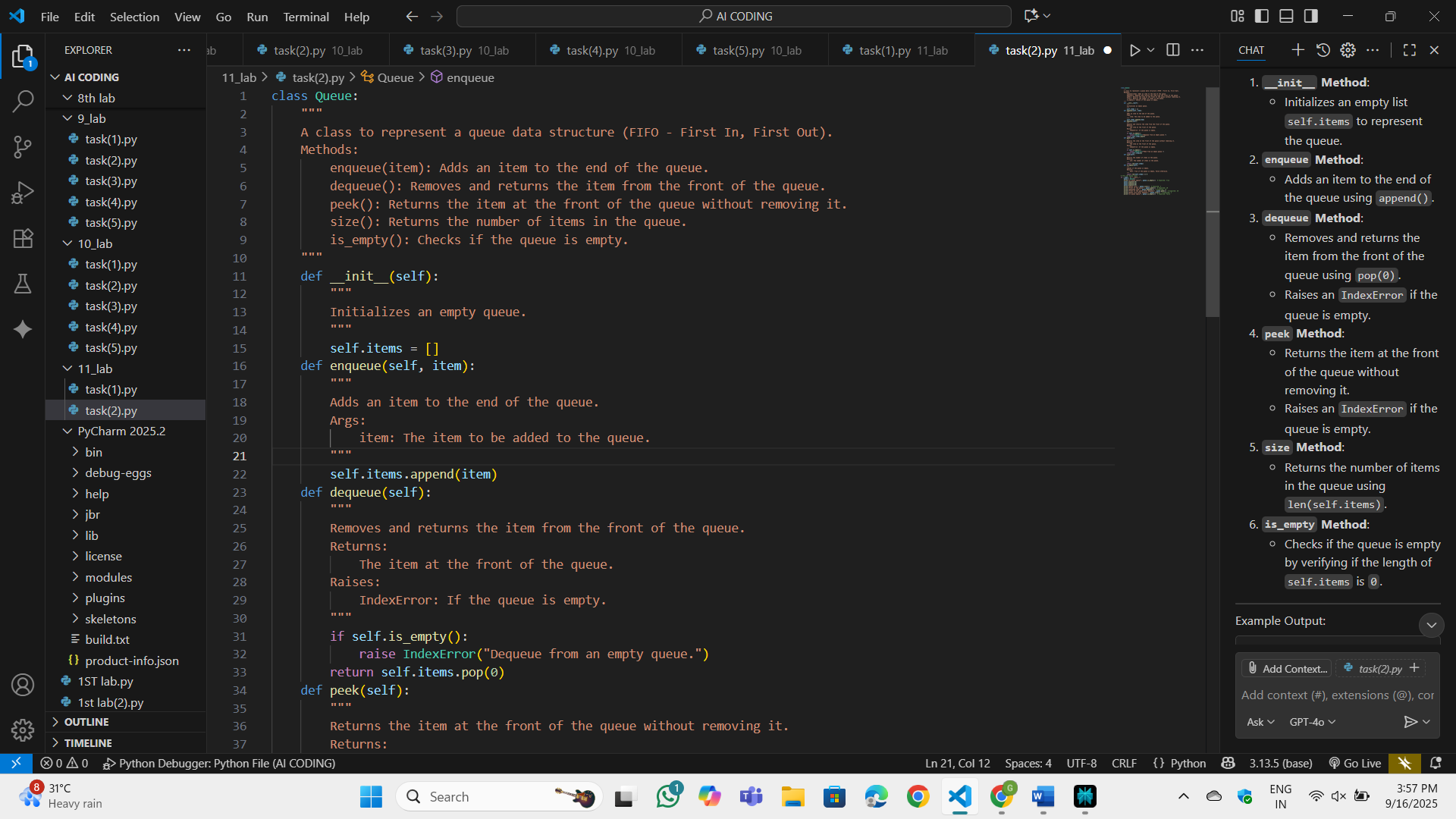
This program works like a collection where things are arranged one on top of another. At the beginning, the collection is empty. Whenever something new is added, it goes on the top, and if something needs to be removed, the latest one added comes out first. It also allows me to just see what is on top without removing it. There is even a way to check if the collection has nothing inside.

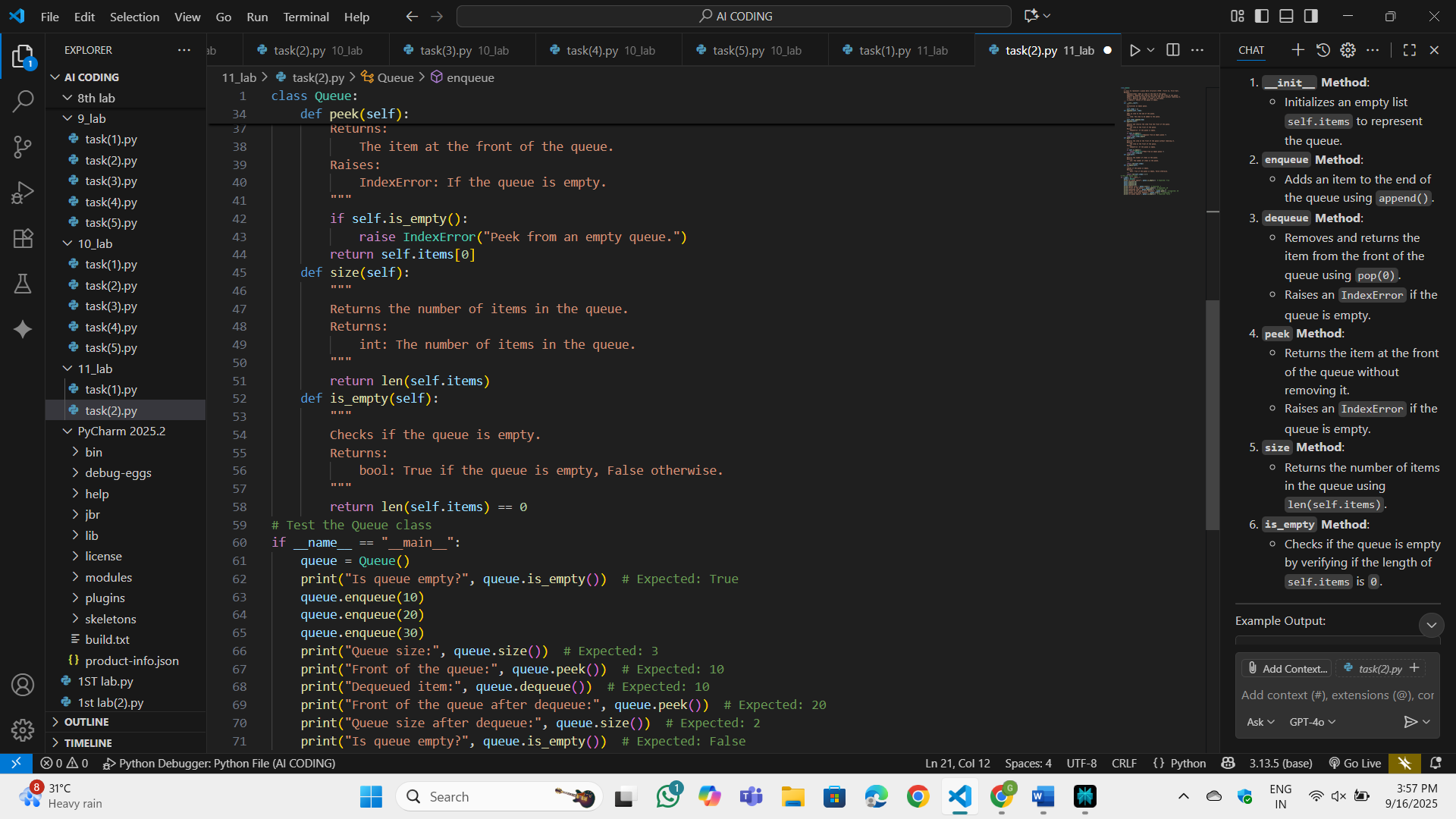
Task-2

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

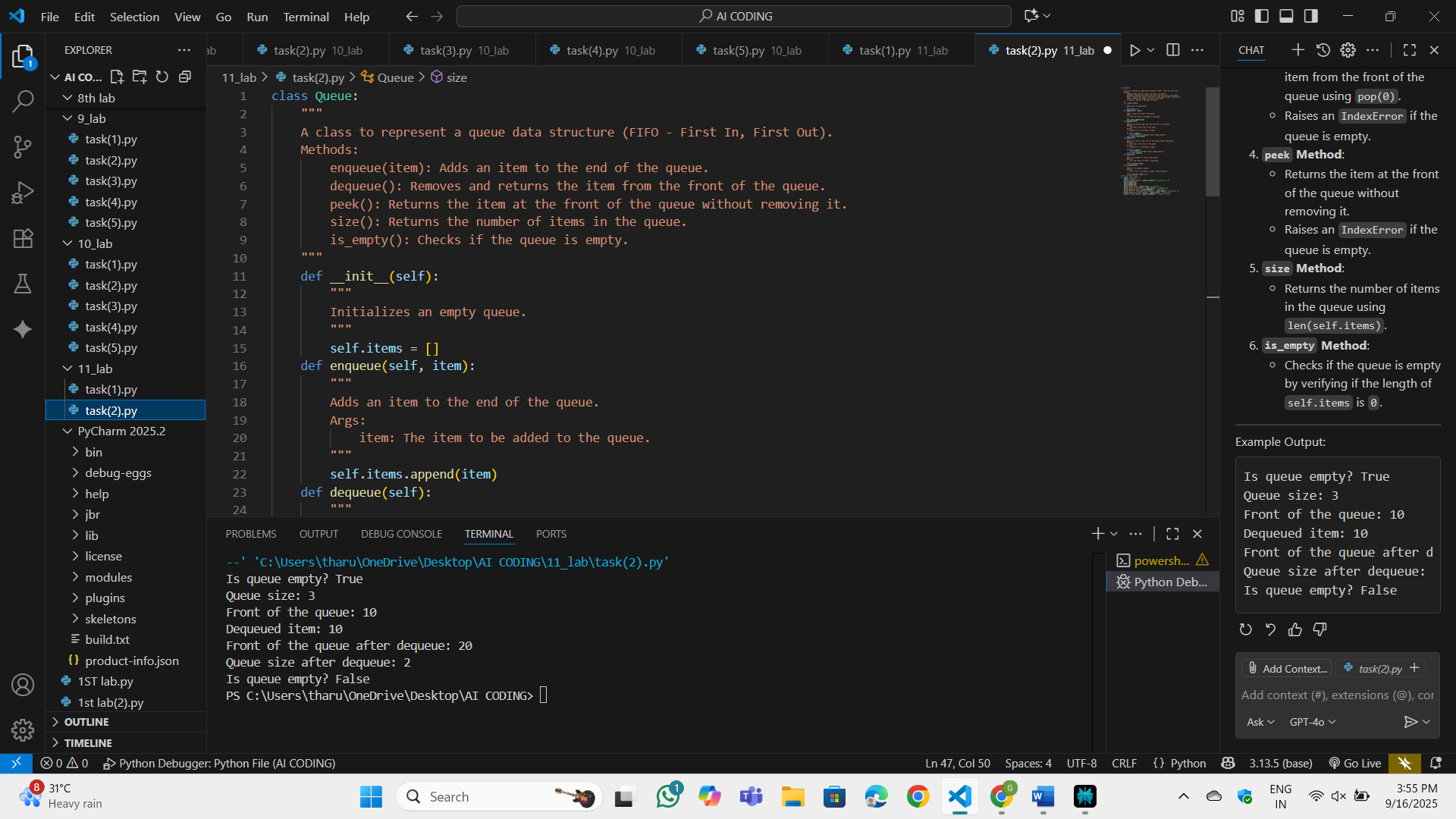
Prompt: implement a Queue using Python lists.

Code:





OP:



Observation:

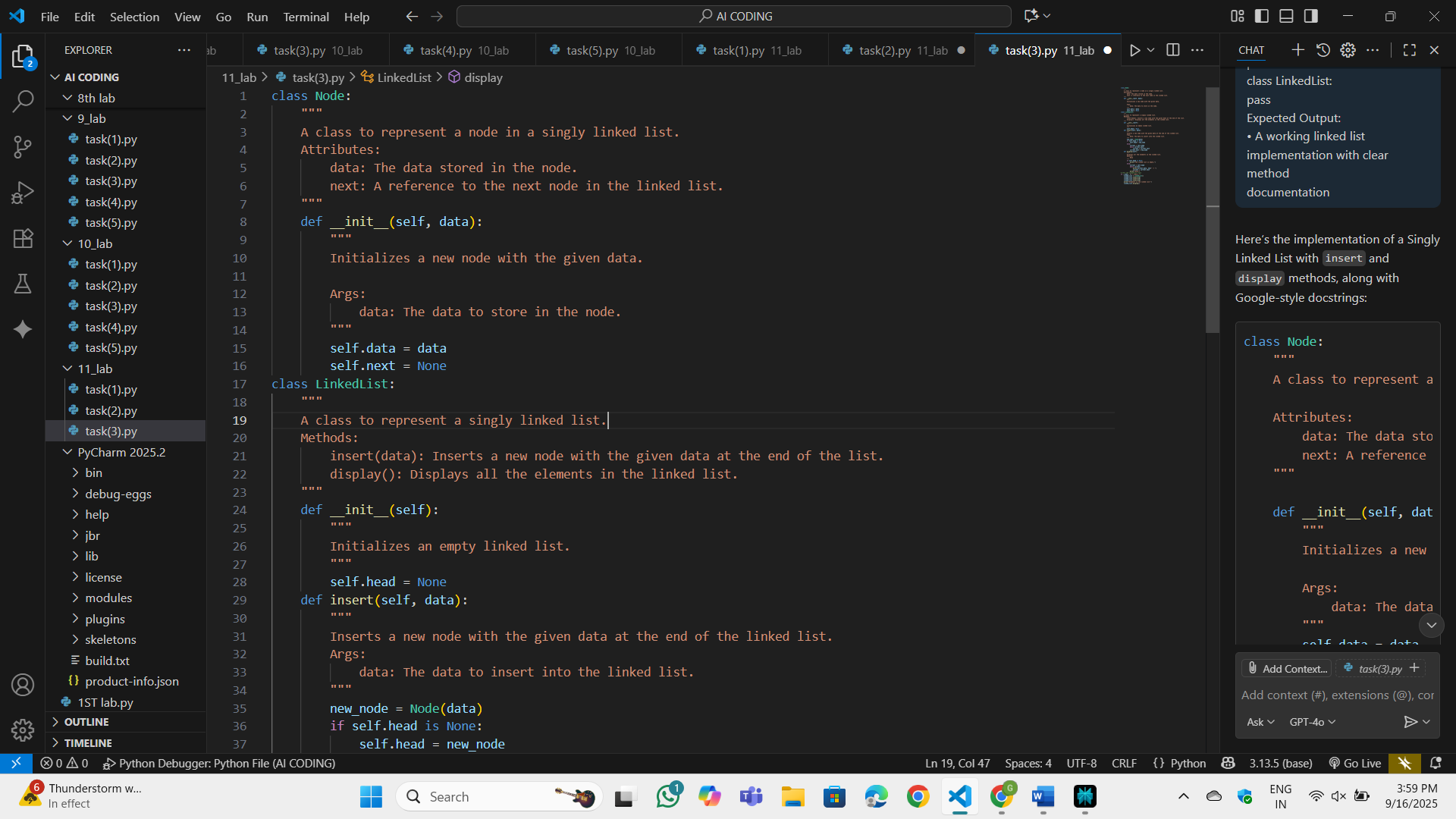
This program works like a line where people stand one after another. In the beginning, the line is empty. Whenever something new comes, it joins at the end of the line, and when something leaves, it is always the first one that came in. You can also just look at who is at the front without removing them. There’s a way to count how many are currently in the line, and also to check if the line is completely empty.

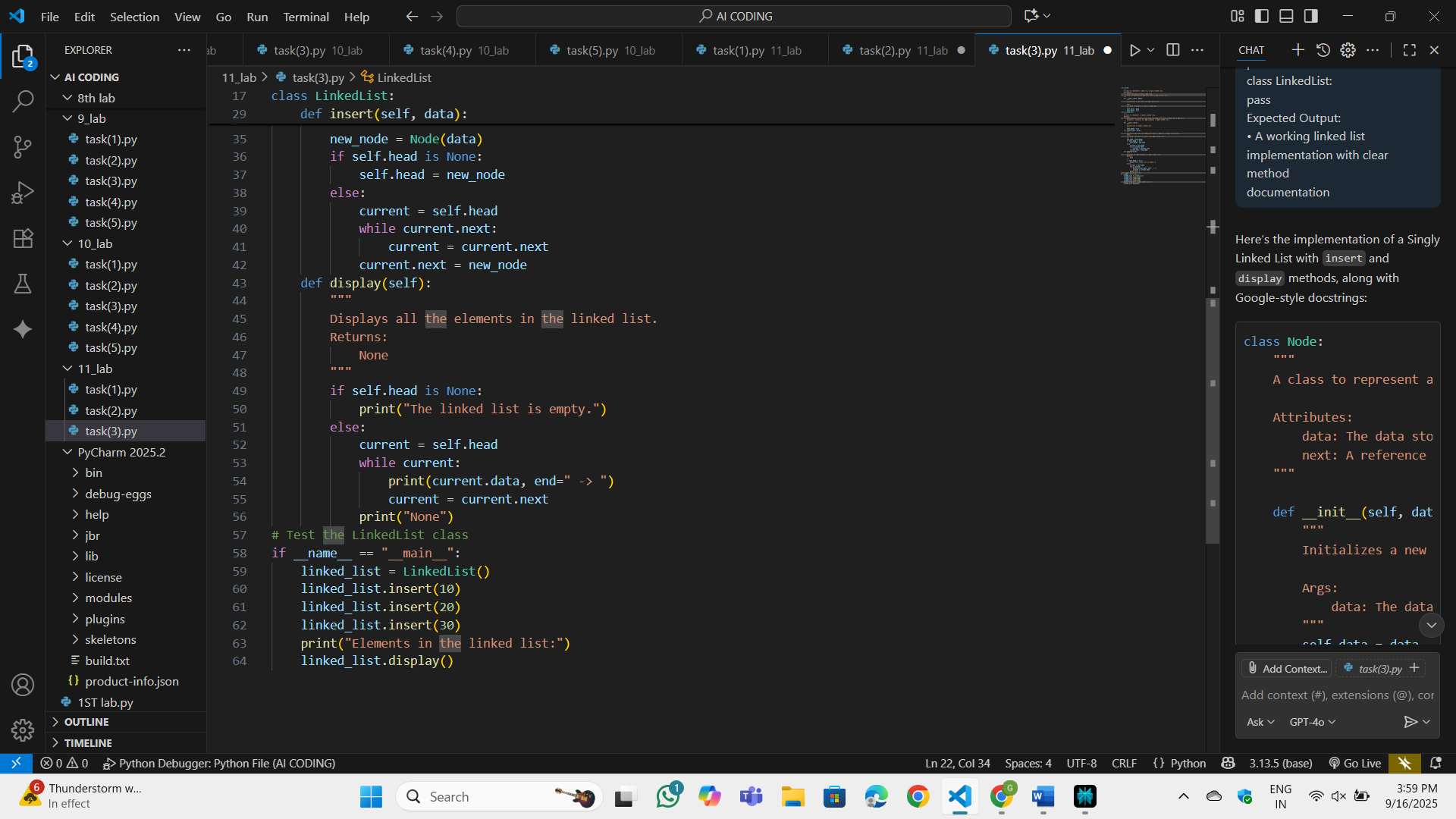
Task-3:

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

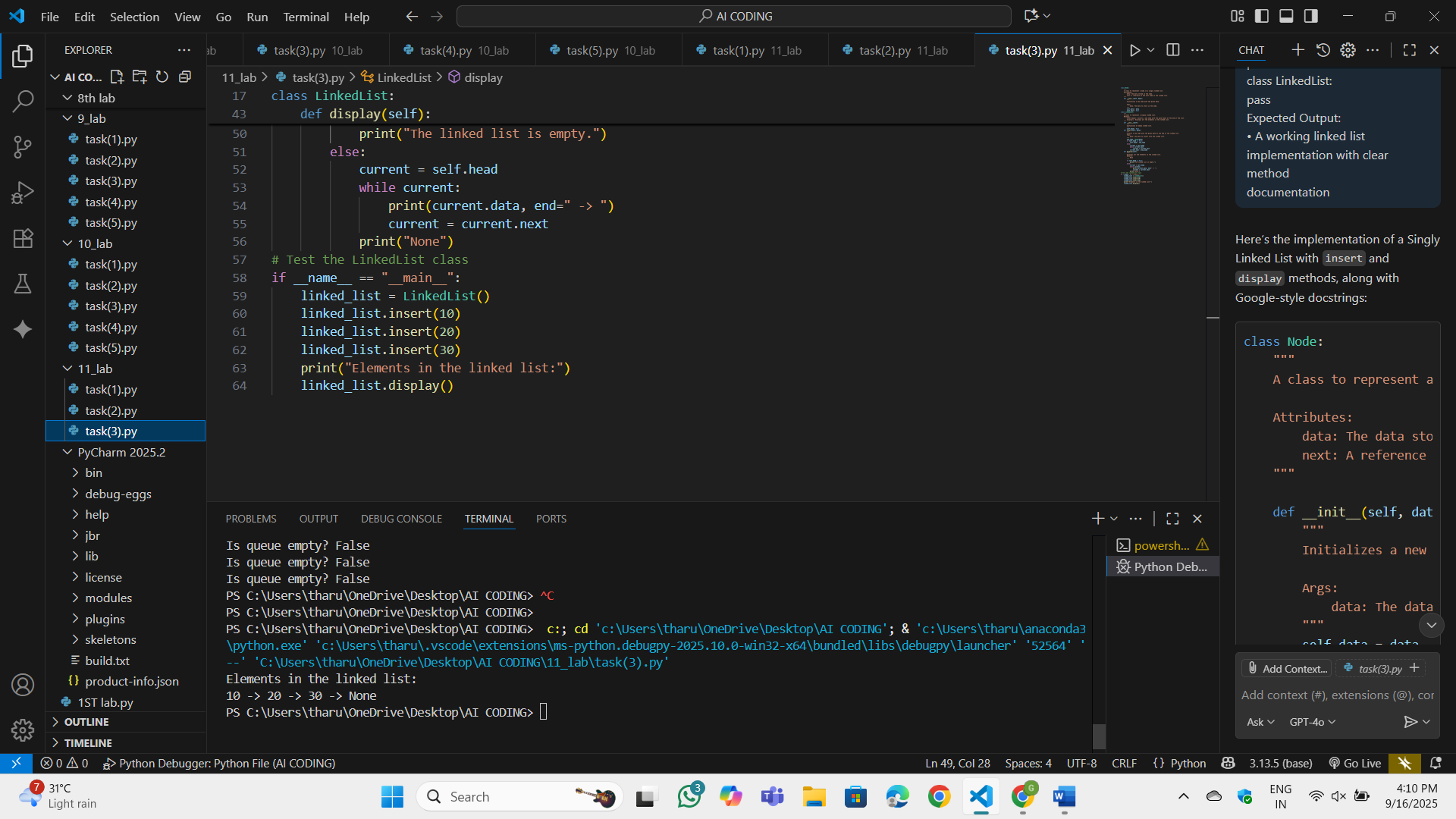
Prompt: generate a Singly Linked List with insert and display methods. A working linked list implementation with clear method  
documentation

Code:





OP:



Observation:

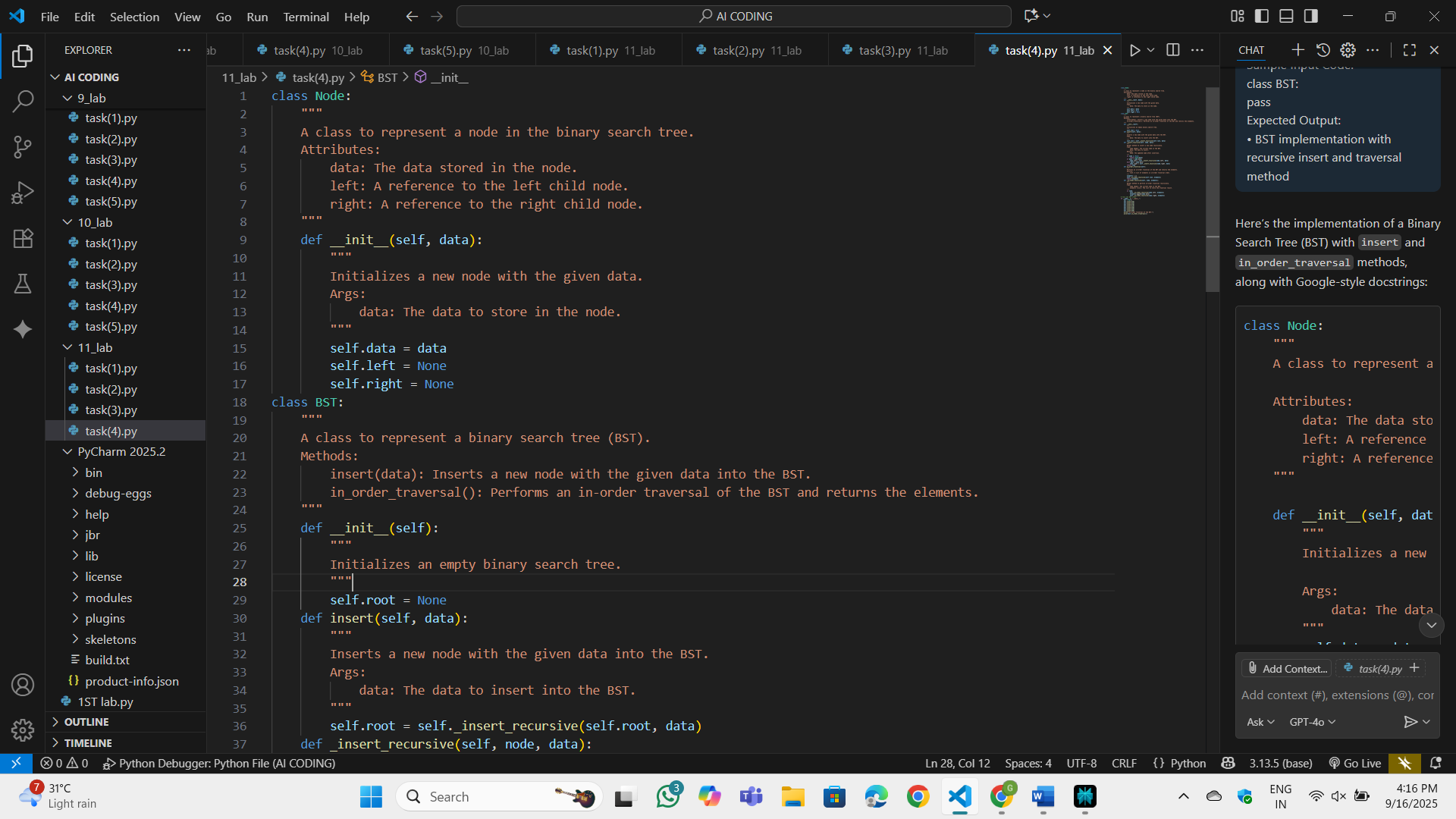
This program works like a line where people stand one after another. In the beginning, the line is empty. Whenever something new comes, it joins at the end of the line, and when something leaves, it is always the first one that came in. You can also just look at who is at the front without removing them. There’s a way to count how many are currently in the line, and also to check if the line is completely empty.

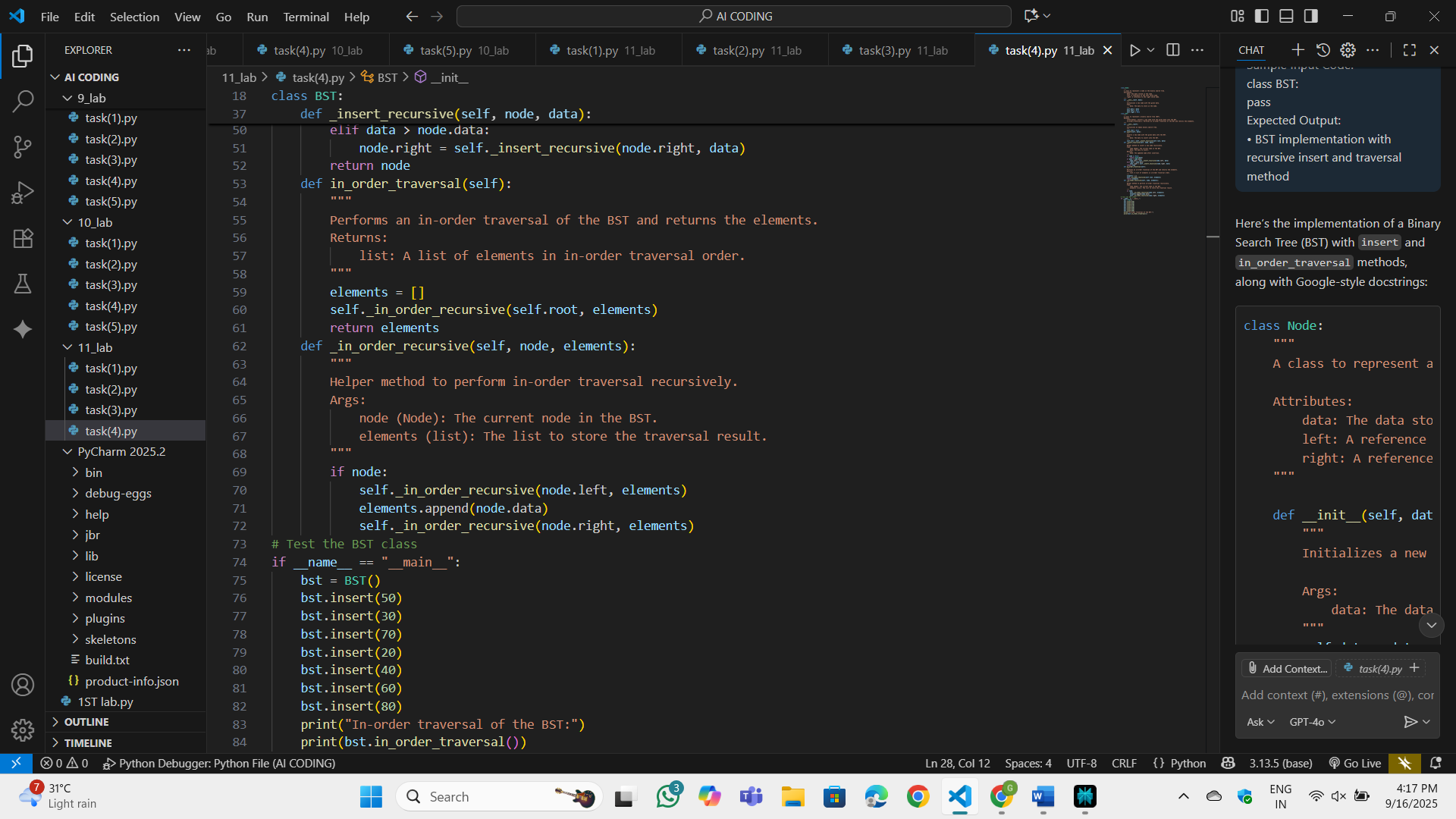
Task-4:

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 method

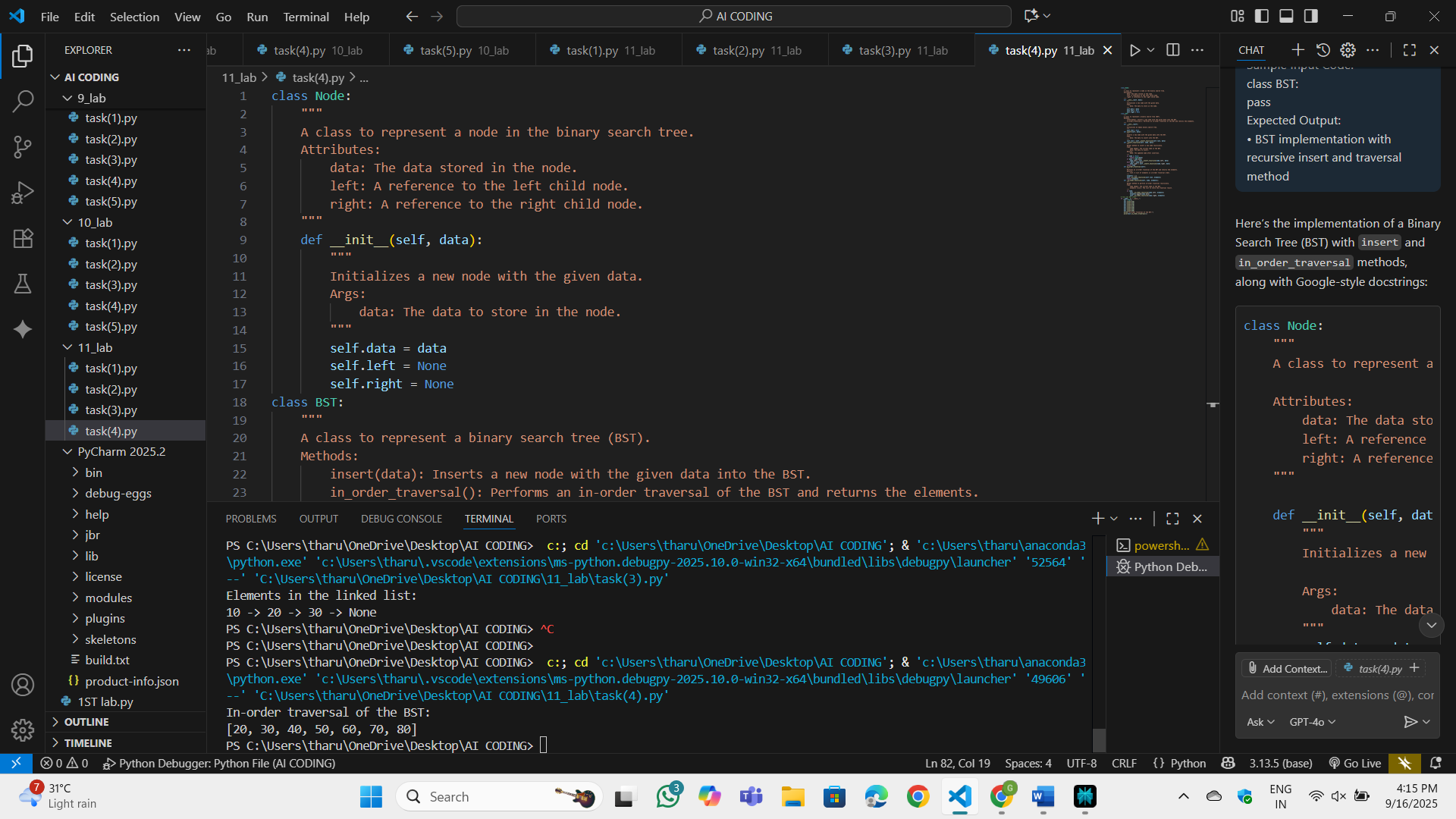
Prompt: create a BST with insert and in-order traversal methods.

Code:





OP:



Observation:

This program is about creating and organizing a tree-like structure where each piece of data is stored in special boxes called nodes. Every node has a value, along with links that can connect to smaller nodes on the left and larger nodes on the right. When a new value is added, it is placed in the proper position by comparing it with existing values until it finds its correct spot. There is also a way to go through the tree in order, which means visiting the left side first, then the main value, and finally the right side, so all the values come out sorted.

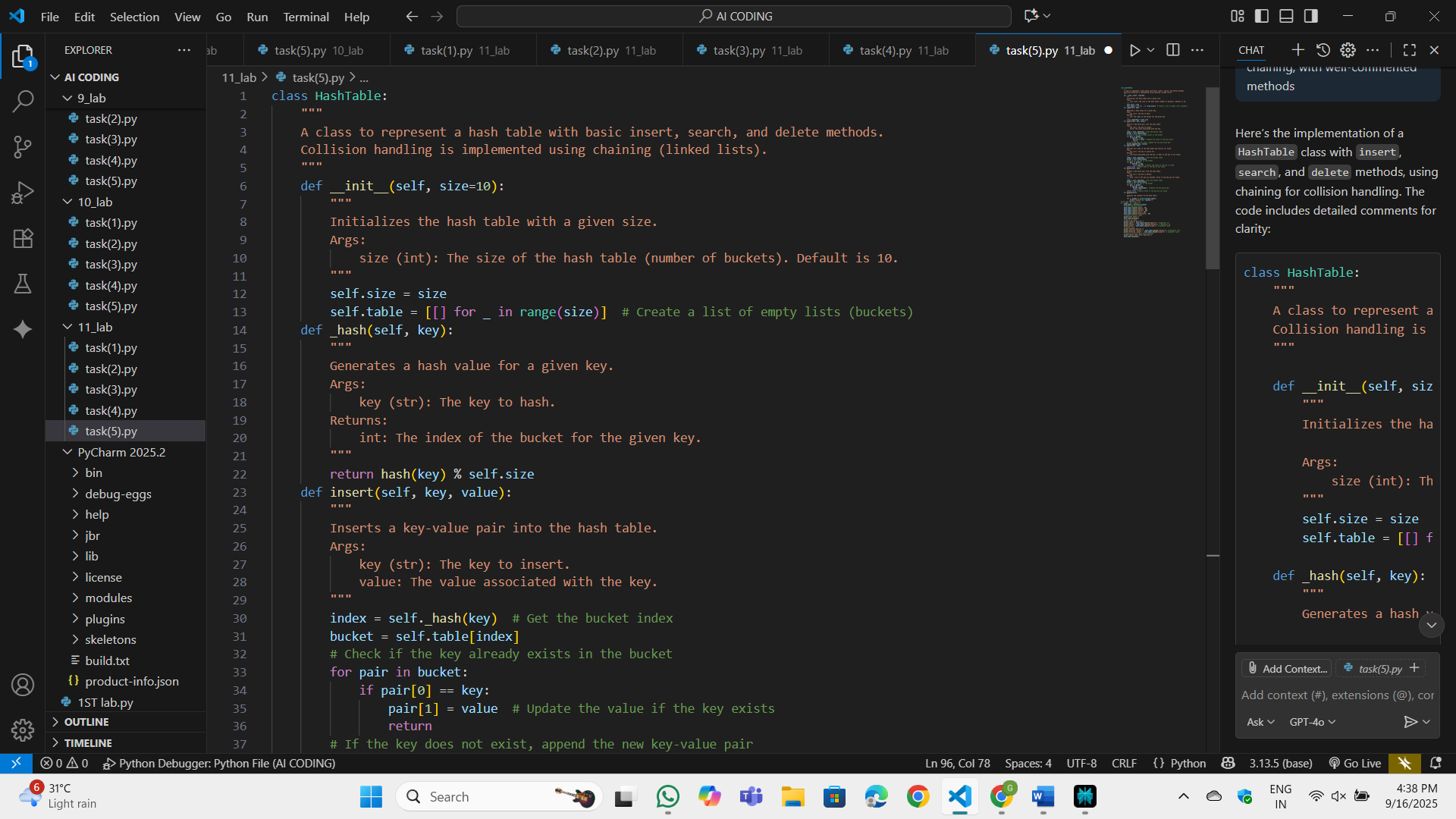
Task-5:

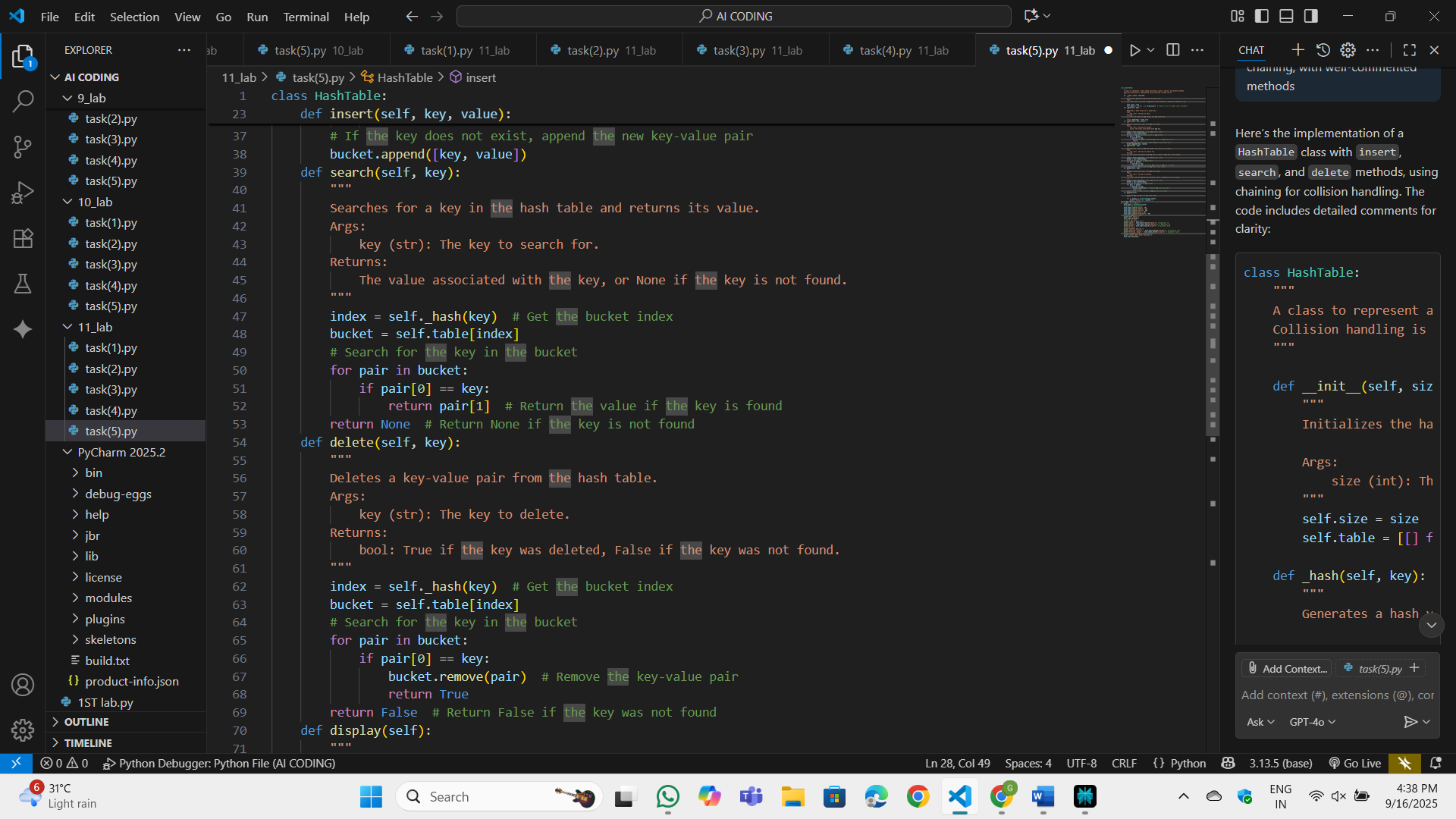
Task: Use AI to implement a hash table with basic insert, search, and delete

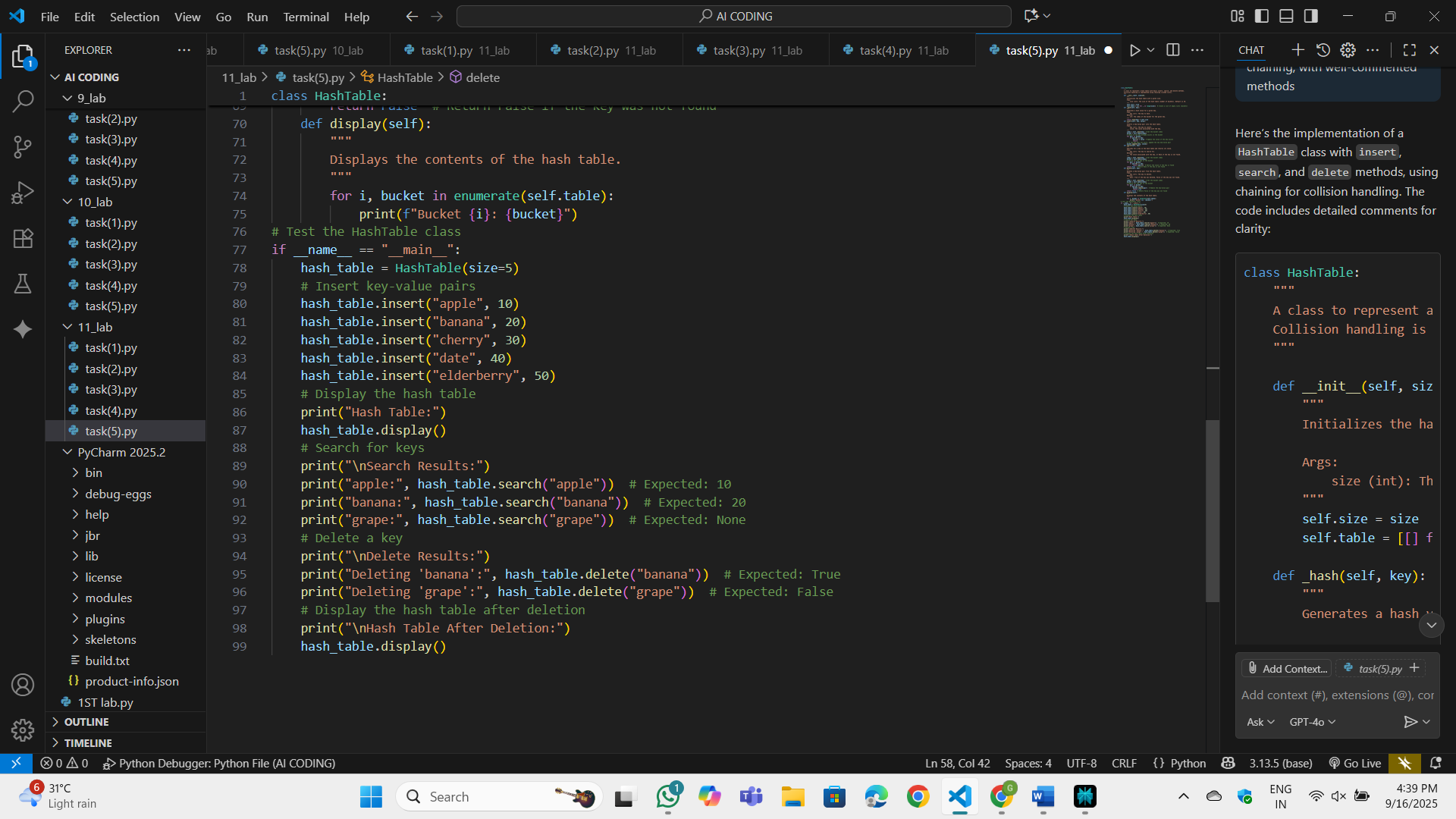
methods.  
Sample Input Code:  
class HashTable:  
pass  
Expected Output:  
• Collision handling using chaining, with well-commented methods

Prompt: implement a hash table with basic insert, search, and delete. Collision handling using chaining, with well-commented methods

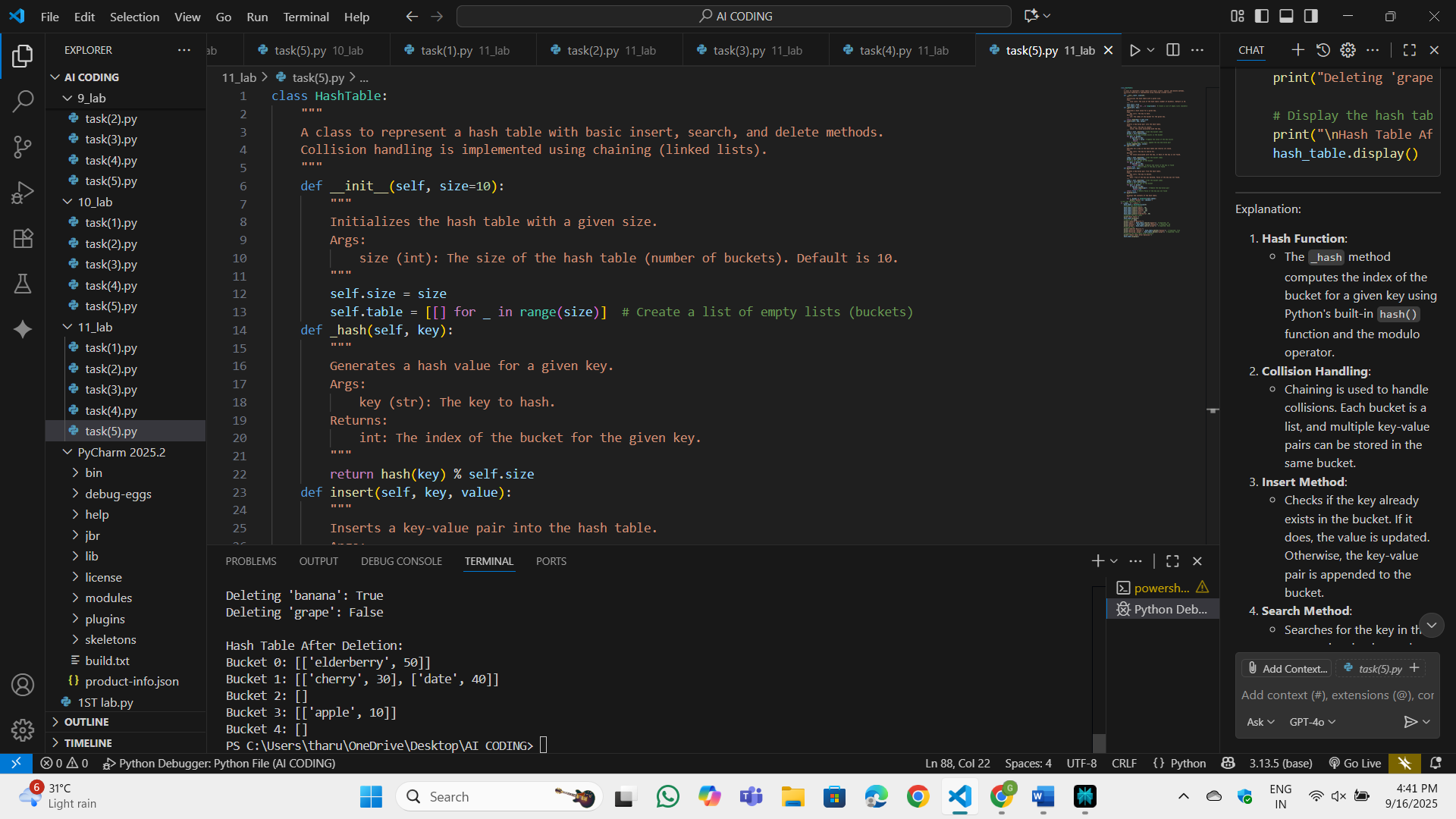
Code:







OP:



Observation:

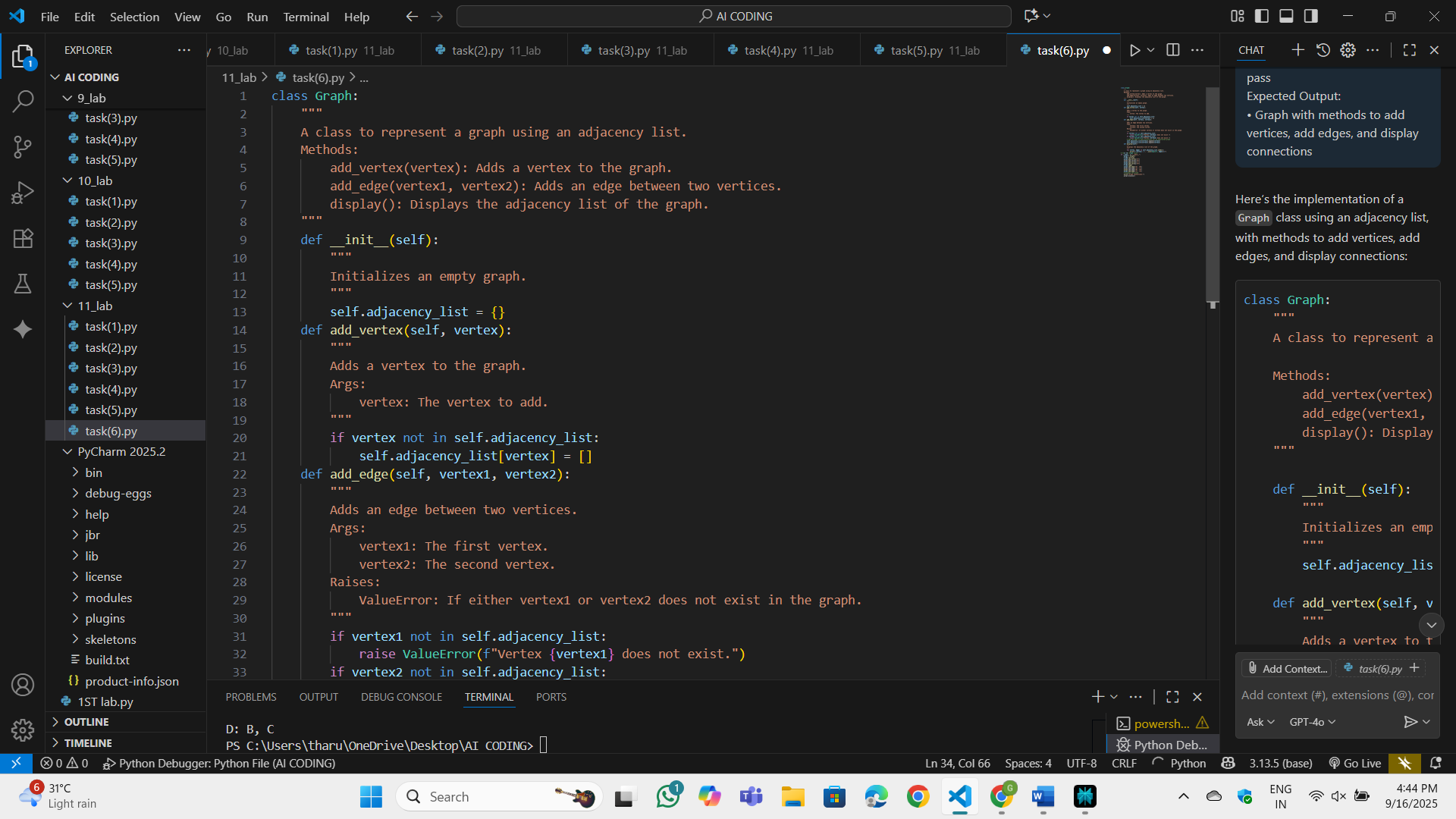
This program is about storing data in a special table where each piece of information is placed in a specific spot calculated from its key. Sometimes, more than one key can end up in the same spot, and in that case, they are simply kept together in a small list at that position. When adding something new, if the key already exists, its value gets updated; if not, the new pair is added. To look up information, it searches the correct spot and returns the value if the key is found, or nothing if it isn’t. You can also remove a key from the table, and there’s a way to display everything stored inside.

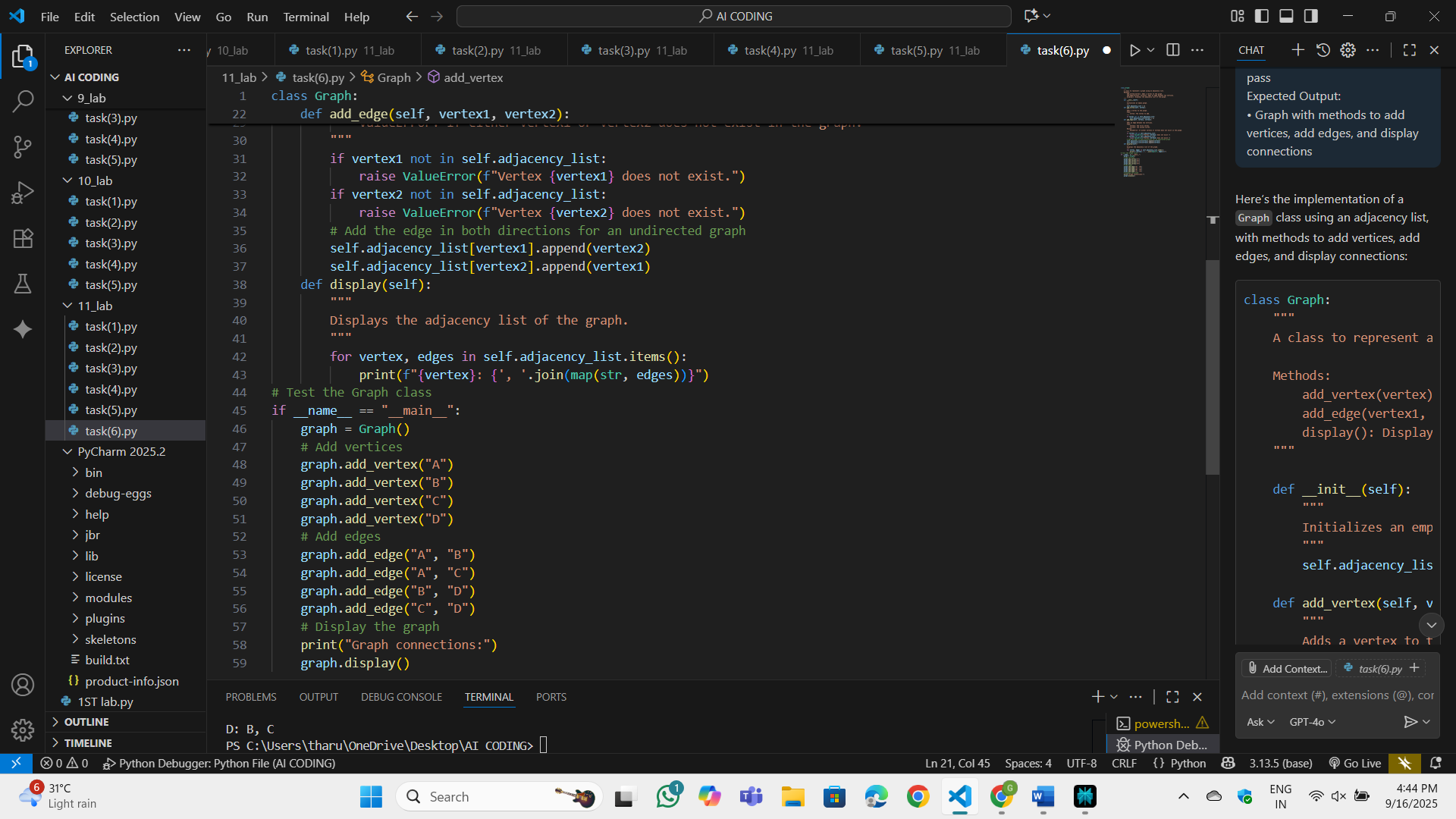
Task 6:

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

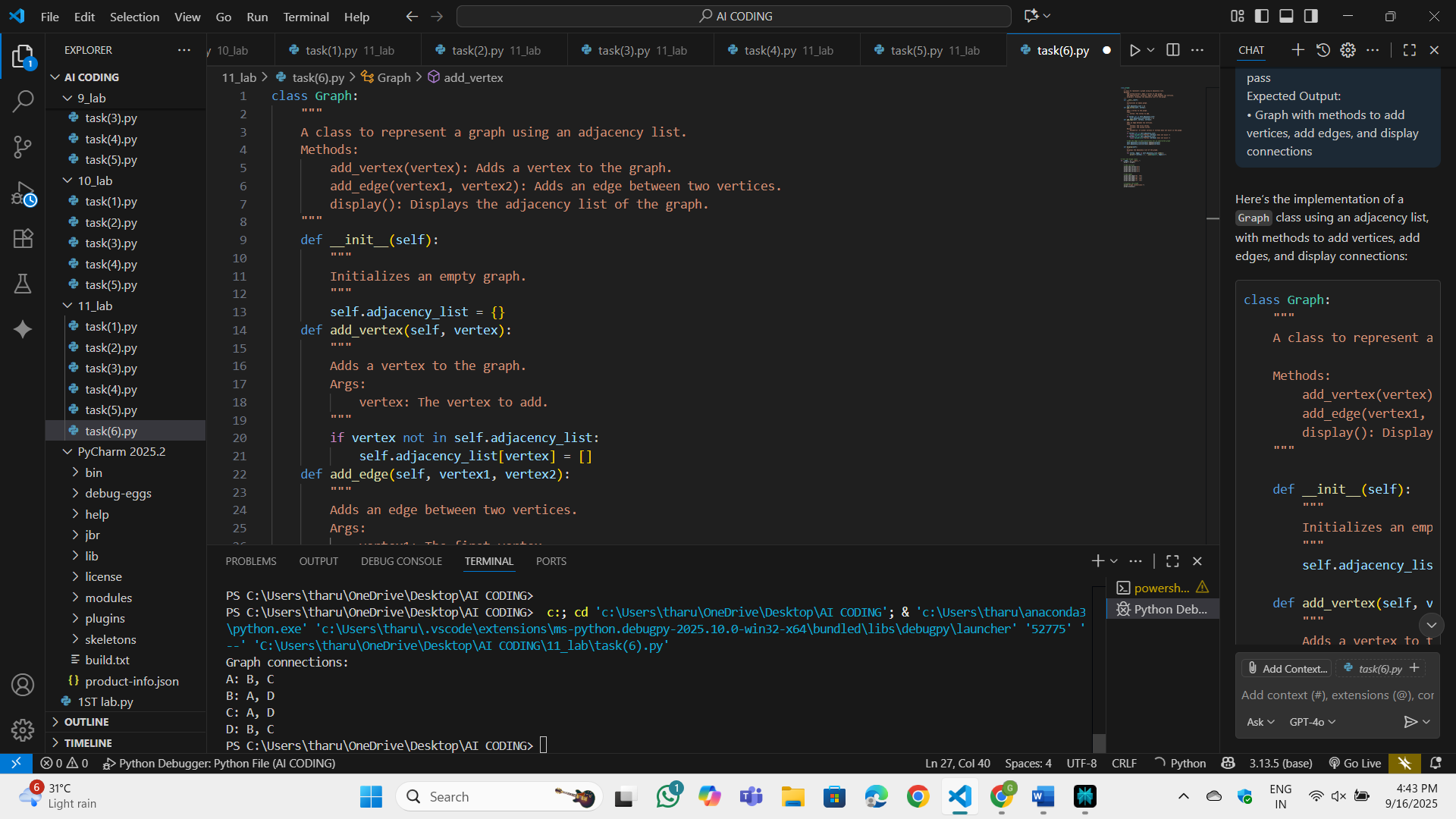
Prompt: implement a graph using an adjacency list.

Code:





OP:



Observation:

This setup treats a network like a map where each point keeps a small list of its neighboring points it directly connects to. New points can be added by creating an empty spot for their connections, and links between two points are recorded on both sides so each knows about the other. If a link is requested between points that don’t exist, it’s considered a mistake and the process is stopped with an error message. There’s also a simple way to go through every point and show which other points it’s connected to, making the whole map easy to read.

Task 7:

ask: 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:

Code:

OP:

Observation:

Task 8:

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:

Code:

OP:

Observation:

Task 9:

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

Prompt:

Code:

OP:

Observation:

Task 10

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.  
Prompt:

Code:

OP:

Observation: