# Lab 11 – Data Structures with AI: Implementing Fundamental Structures

**Assignment number: 11.4**

**Enrollment number:2503A51L36**

**Name: Suhana Rehan**

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

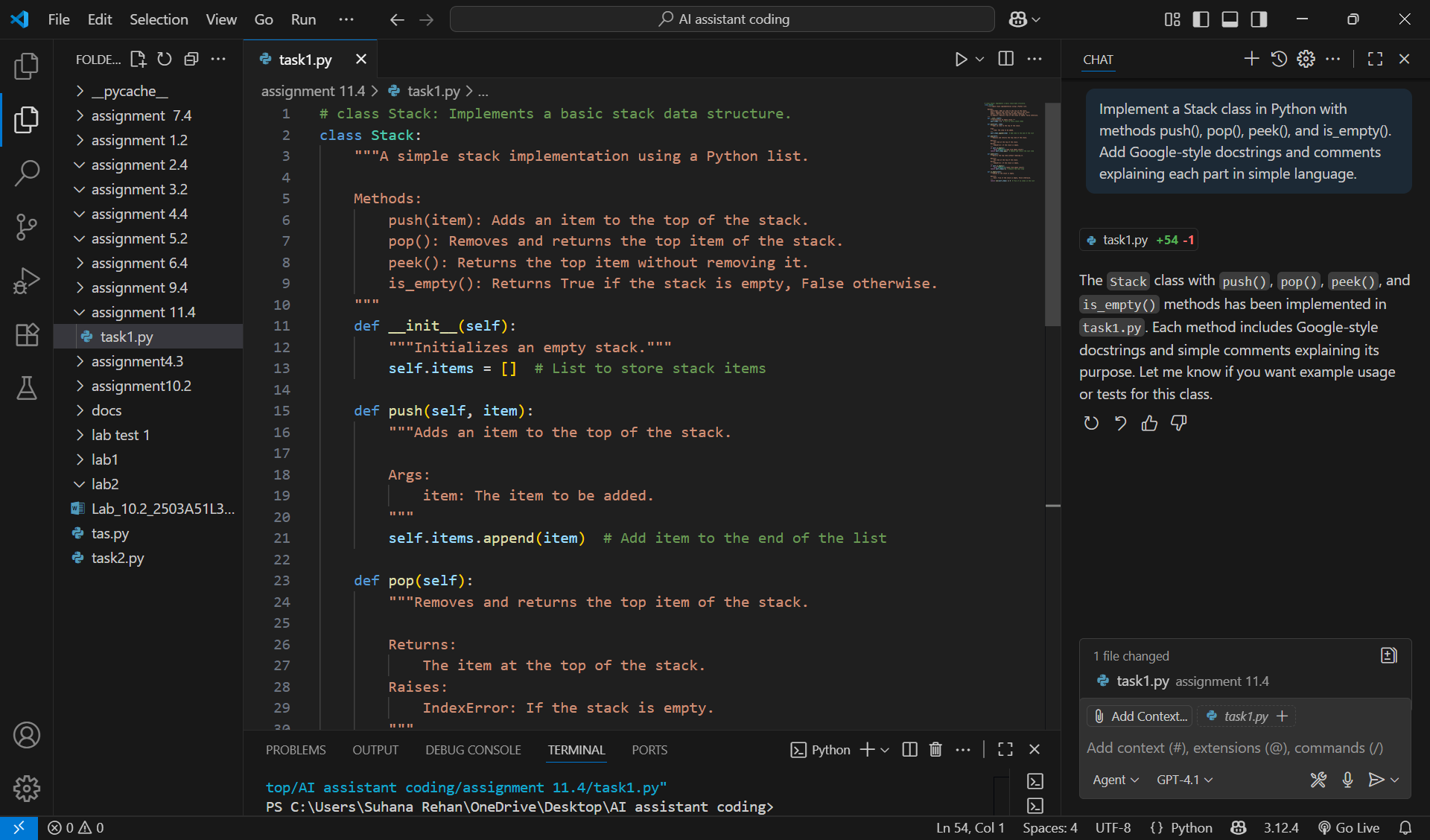
**Task 1: Implementing a Stack (LIFO)**

* **Task**: Use AI to help implement a **Stack** class in Python with the following operations: push(), pop(), peek(), and is\_empty().
* **Instructions**:
  + Ask AI to generate code skeleton with docstrings.
  + Test stack operations using sample data.
  + Request AI to suggest optimizations or alternative implementations (e.g., using collections.deque).

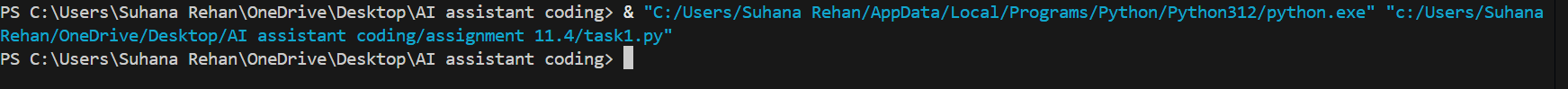
**Prompt:**

* Implement a Stack class in Python with methods push(), pop(), peek(), and is\_empty(). Add Google-style docstrings and comments explaining each part in simple language.
* Test stack operations using sample data.
* Suggest optimizations or alternative implementations (e.g., using collections.deque).

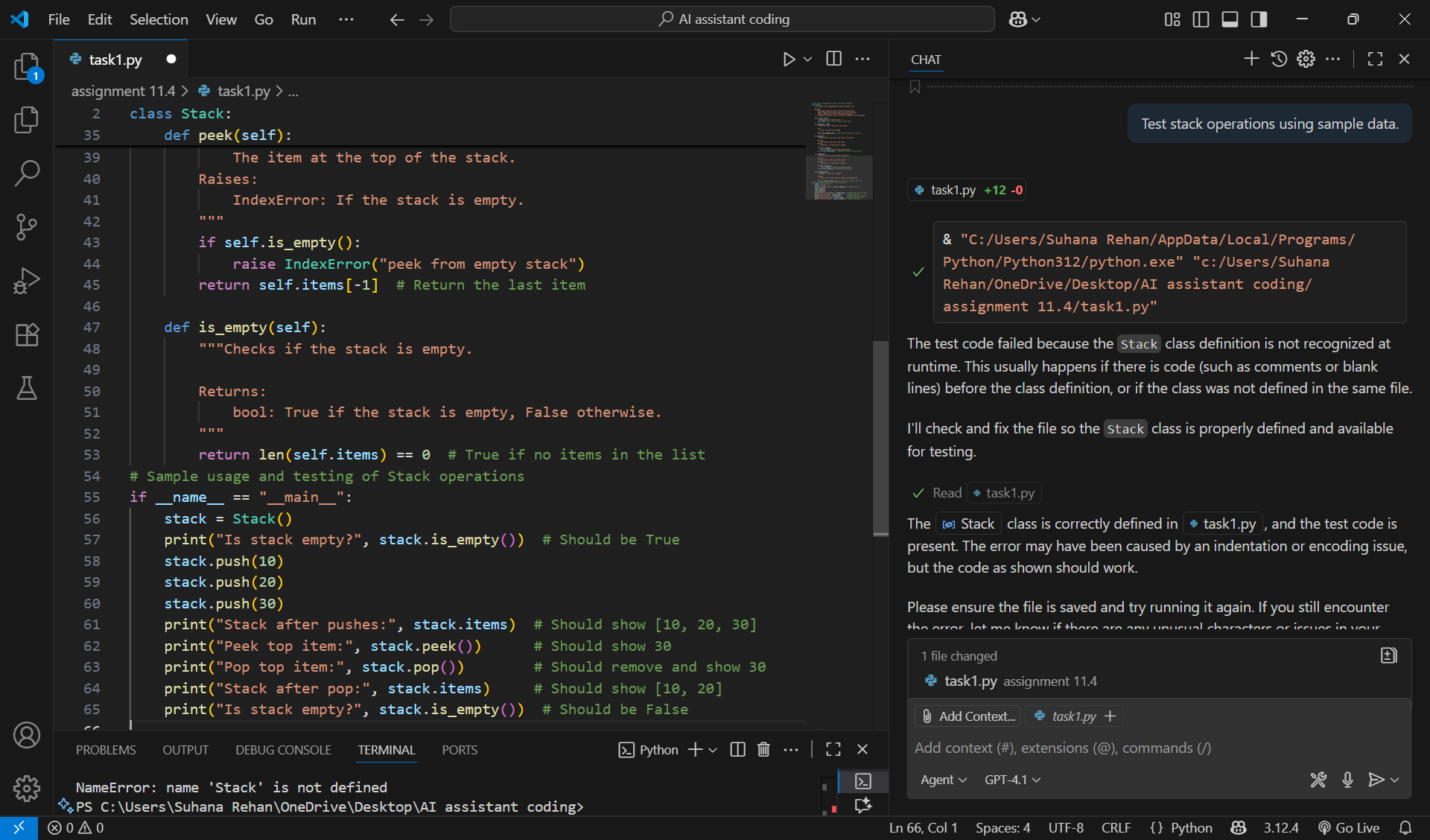
**Code:**

****

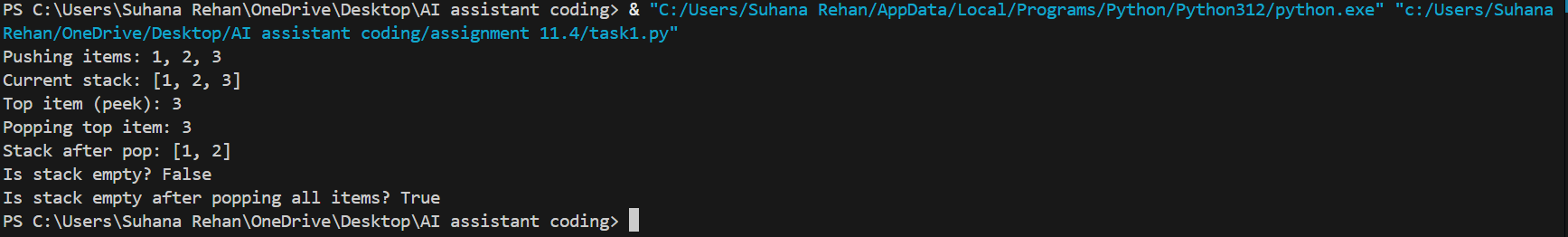
**Output:**

****

**Code:**

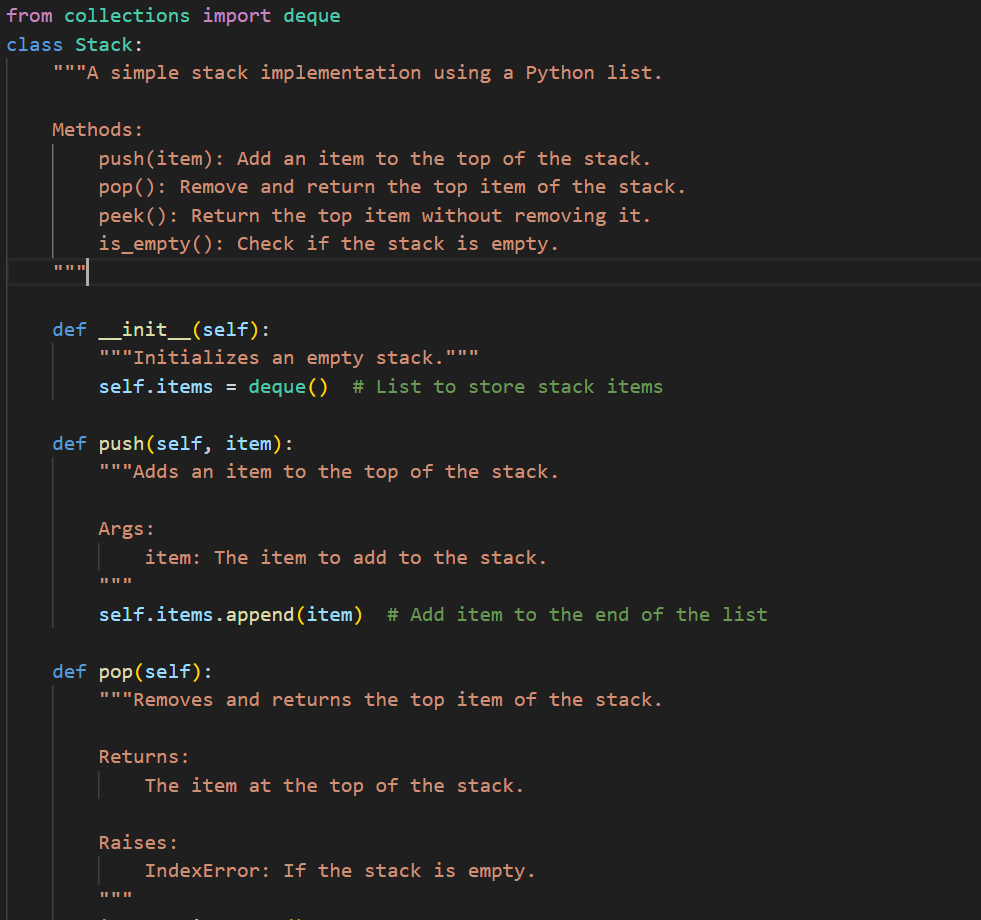
****

**Output:**

****

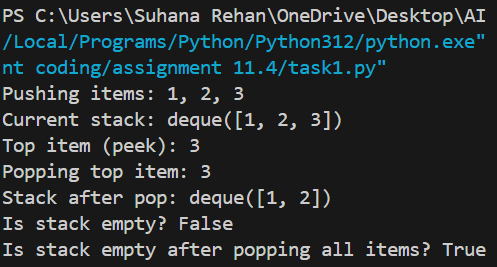


**Code:**

****



**Output:**

****

**Observation:**

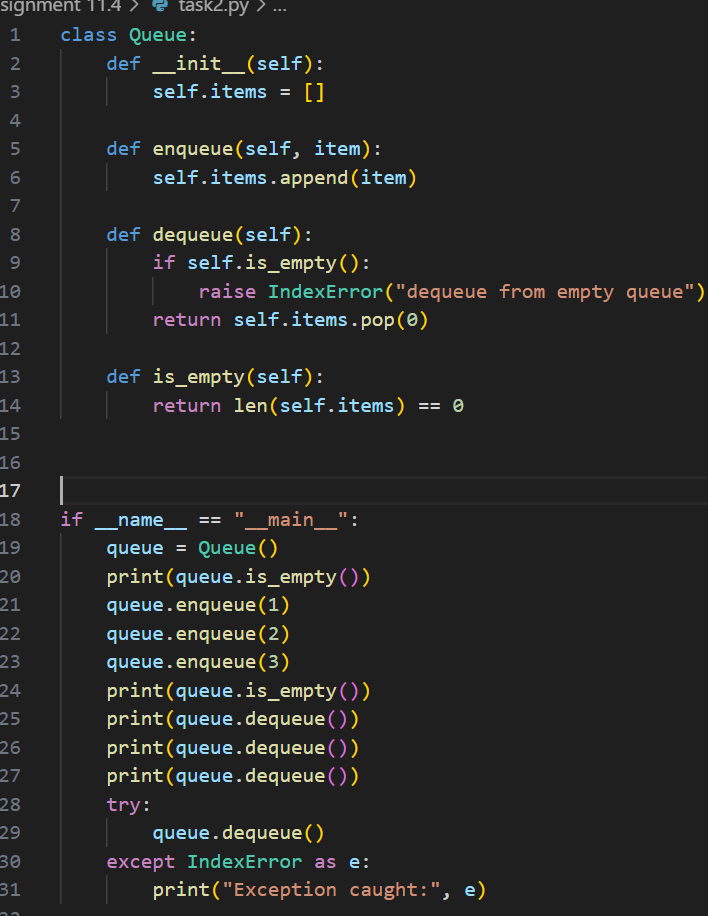
I learned how a stack works with the LIFO method using Python’s list.  
I made sure to handle errors when popping or peeking from an empty stack.  
AI helped me with the structure, but I test the sample data by my own

**Task 2: Queue Implementation with Performance Review**

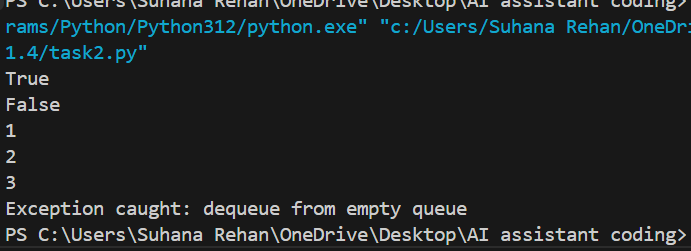
* **Task**: Implement a **Queue** with enqueue(), dequeue(), and is\_empty() methods.
* **Instructions**:
  + First, implement using Python lists.
  + Then, ask AI to review performance and suggest a more efficient implementation (using collections.deque).

**Prompt:** Review performance and suggest a more efficient implementation (using collections.deque).

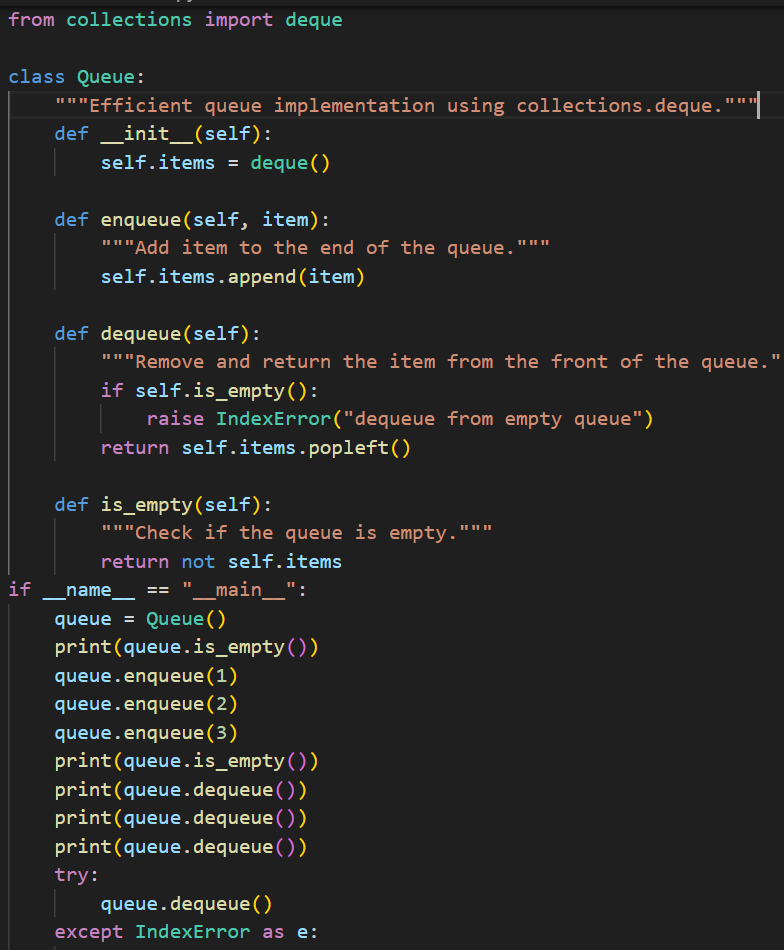
**Code without AI:**

****

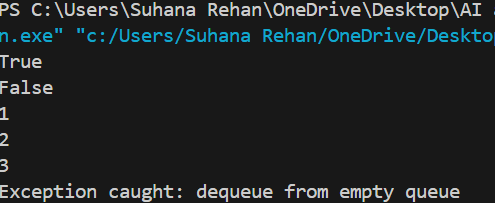
**Output without AI**:



**Code with AI:**



**Output with AI**:



**Observation:**

Here the manual took more time to execute and ai generate code took less time to execute

Ai generated code looks more put-together than manually written one

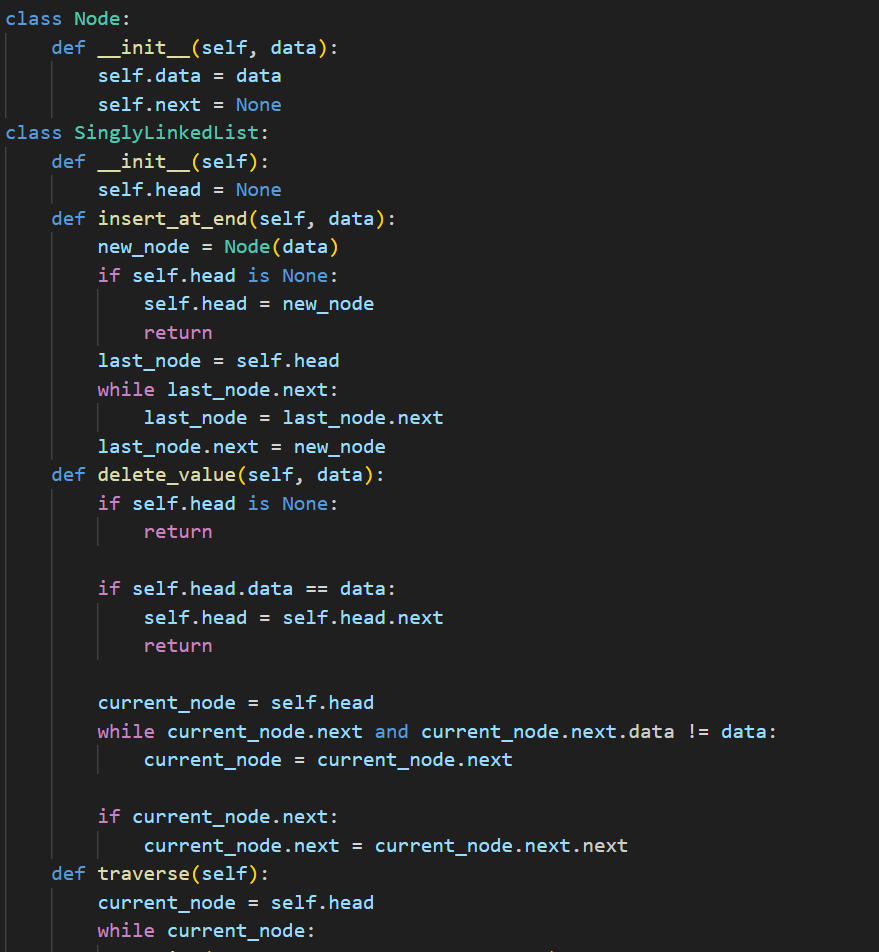
**Task 3: Singly Linked List with Traversal**

* **Task**: Implement a **Singly Linked List** with operations: insert\_at\_end(), delete\_value(), and traverse().
* **Instructions**:
  + Start with a simple class-based implementation (Node, LinkedList).
  + Use AI to generate inline comments explaining pointer updates (which are non-trivial).
  + Ask AI to suggest test cases to validate all operations.

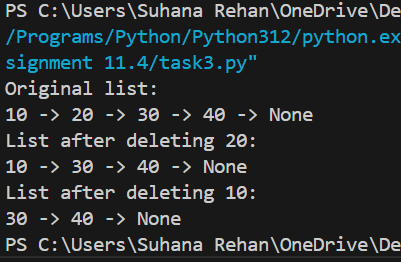
**Prompt:**

Generate inline comments explaining pointer updates and Suggest test cases to validate all operations

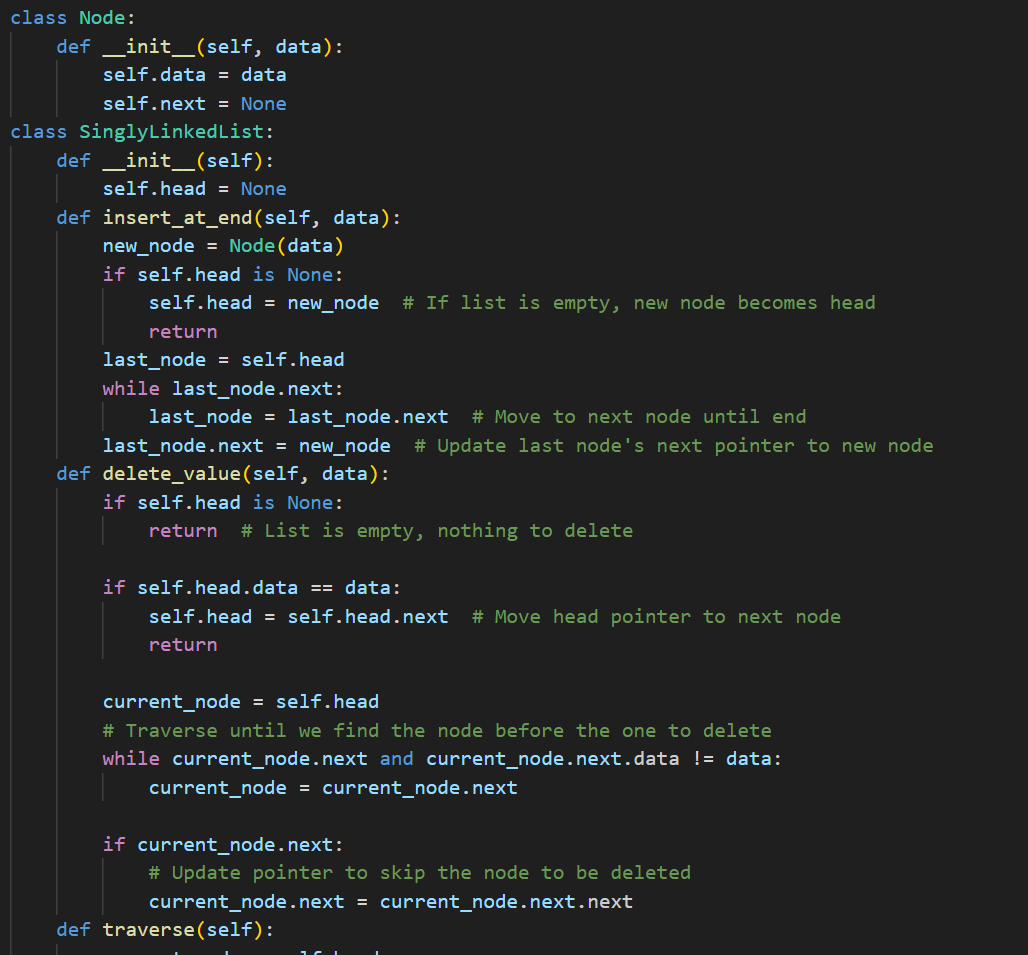
**Code:**



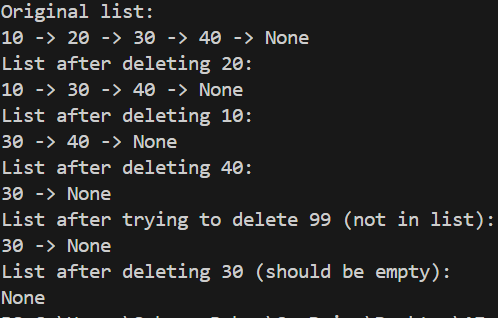
**Output**:

****

**Code after AI:**

****

**Output after AI:**

****

**Observation:**

AI generated clear comments explaining the logic of insertions and deletions and Suggested test cases to validate all operations

**Task 4: Binary Search Tree (BST)**

* **Task**: Implement a **Binary Search Tree** with methods for insert(), search(), and inorder\_traversal().
* **Instructions**:
  + AI with a partially written Node and BST class
  + Ask AI to complete missing methods and add docstrings.
  + Test with a list of integers and compare outputs of search() for present vs absent elements.

**Prompt:**

**Code:**

**Output**:

**Observation:**

* + A BST class with clean implementation, meaningful docstrings, and correct traversal output.

**Task 5: Graph Representation and BFS/DFS Traversal**

* **Task**: Implement a **Graph** using an adjacency list, with traversal methods BFS() and DFS().
* **Instructions**:
  + Start with an adjacency list dictionary.
  + Ask AI to generate BFS and DFS implementations with inline comments.
  + Compare recursive vs iterative DFS if suggested by AI.

**Prompt:**

**Code:**

**Output**:

**Observation:**

**`** A graph implementation with BFS and DFS traversal methods, with AI-generated comments explaining traversal steps.