SR UNIVERSITY

School of Computer Science and Artificial Intelligence

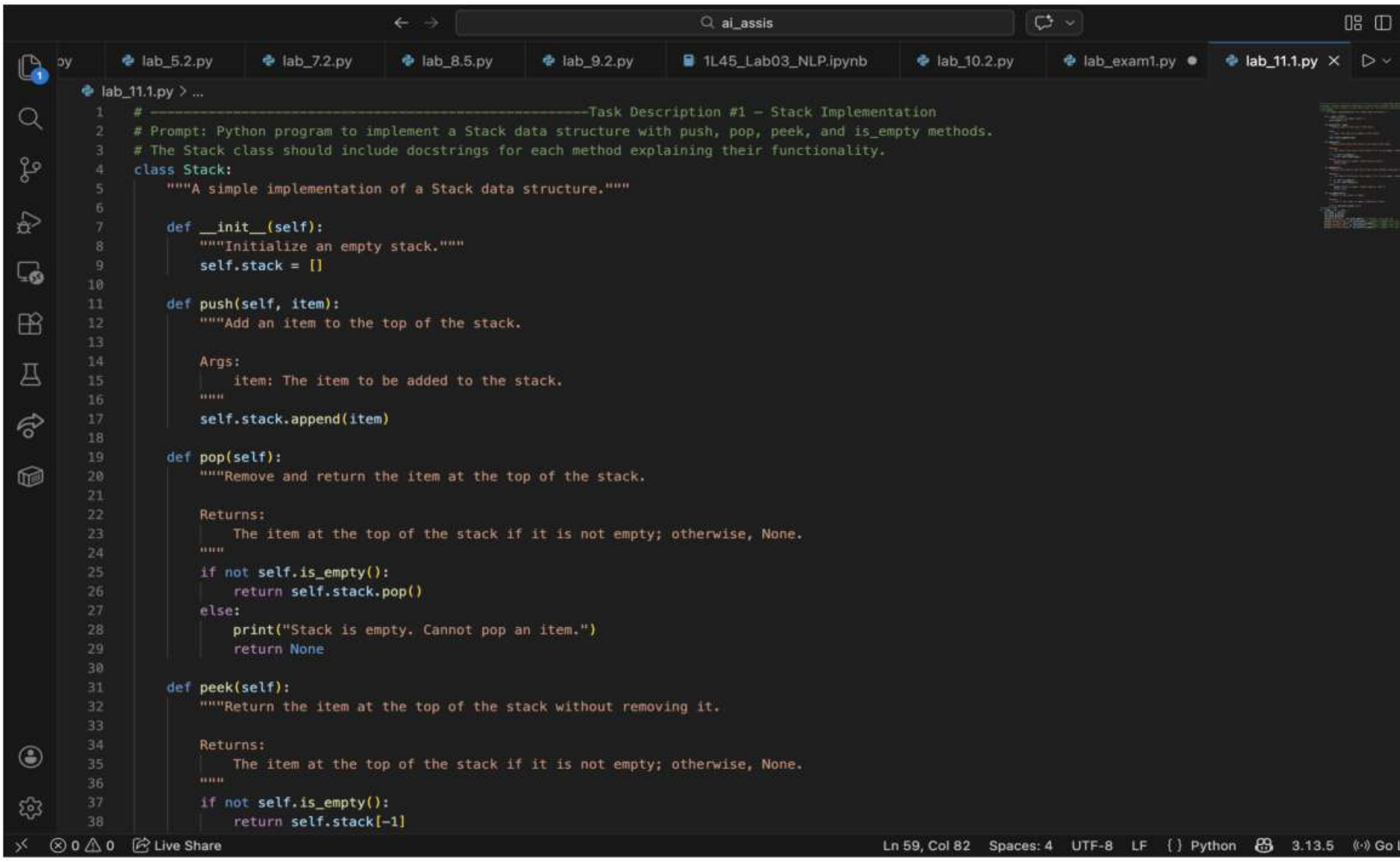
Lab Assignment # 11.1

Program	: B. Tech (CSE)
Specialization	: –
Course Title	: AI Assisted coding
Course Code	:
Semester	II
Academic Session	: 2025-2026
Name of Student	: Akash Reddy
Enrollment No.	: 2403A51L30
Batch No.	: 51
Date	:17-02-2026

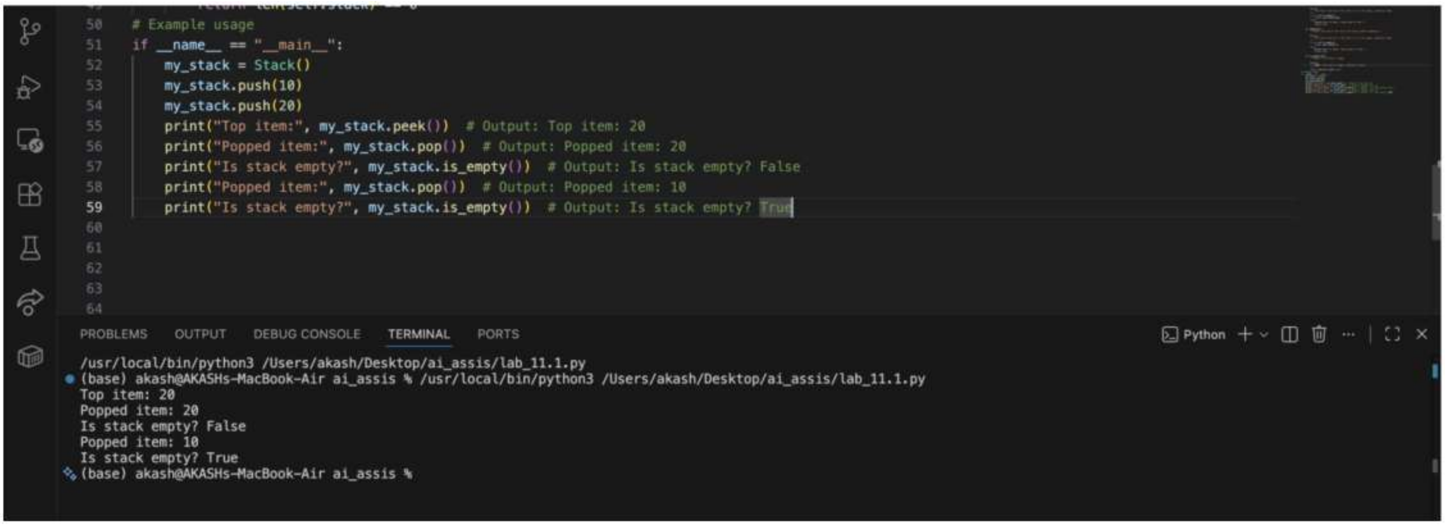
Task Description #1 – Stack Implementation

Prompt: Python program to implement a Stack data structure with push, pop, peek, and is_empty methods. The Stack class should include docstrings for each method explaining their functionality.

OUTPUT



```
# -------------------------------------------------------Task Description #1 – Stack Implementation
# Prompt: Python program to implement a Stack data structure with push, pop, peek, and is_empty methods.
# The Stack class should include docstrings for each method explaining their functionality.
class Stack:
    """A simple implementation of a Stack data structure."""

    def __init__(self):
        """Initialize an empty stack."""
        self.stack = []

    def push(self, item):
        """Add an item to the top of the stack.

        Args:
            item: The item to be added to the stack.
        """
        self.stack.append(item)

    def pop(self):
        """Remove and return the item at the top of the stack.

        Returns:
            The item at the top of the stack if it is not empty; otherwise, None.
        """
        if not self.is_empty():
            return self.stack.pop()
        else:
            print("Stack is empty. Cannot pop an item.")
            return None

    def peek(self):
        """Return the item at the top of the stack without removing it.

        Returns:
            The item at the top of the stack if it is not empty; otherwise, None.
        """
        if not self.is_empty():
            return self.stack[-1]
```



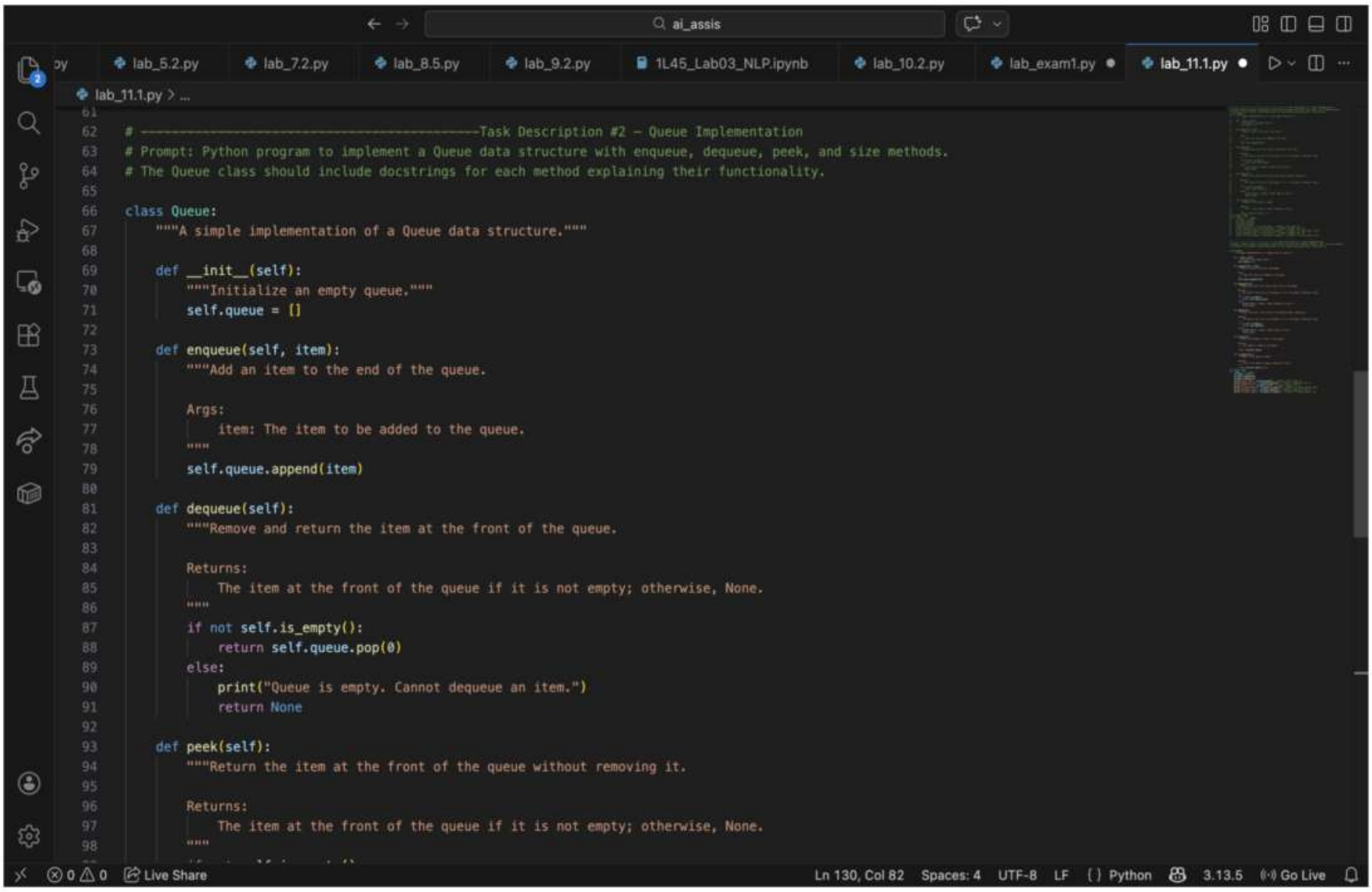
```python
# Example usage
if __name__ == "__main__":
    my_stack = Stack()
    my_stack.push(10)
    my_stack.push(20)
    print("Top item:", my_stack.peek())  # Output: Top item: 20
    print("Popped item:", my_stack.pop())  # Output: Popped item: 20
    print("Is stack empty?", my_stack.is_empty())  # Output: Is stack empty? False
    print("Popped item:", my_stack.pop())  # Output: Popped item: 10
    print("Is stack empty?", my_stack.is_empty())  # Output: Is stack empty? True
```

```
/usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
(base) akash@AKASHs-MacBook-Air ai_assis % /usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
Top item: 20
Popped item: 20
Is stack empty? False
Popped item: 10
Is stack empty? True
(base) akash@AKASHs-MacBook-Air ai_assis %
```

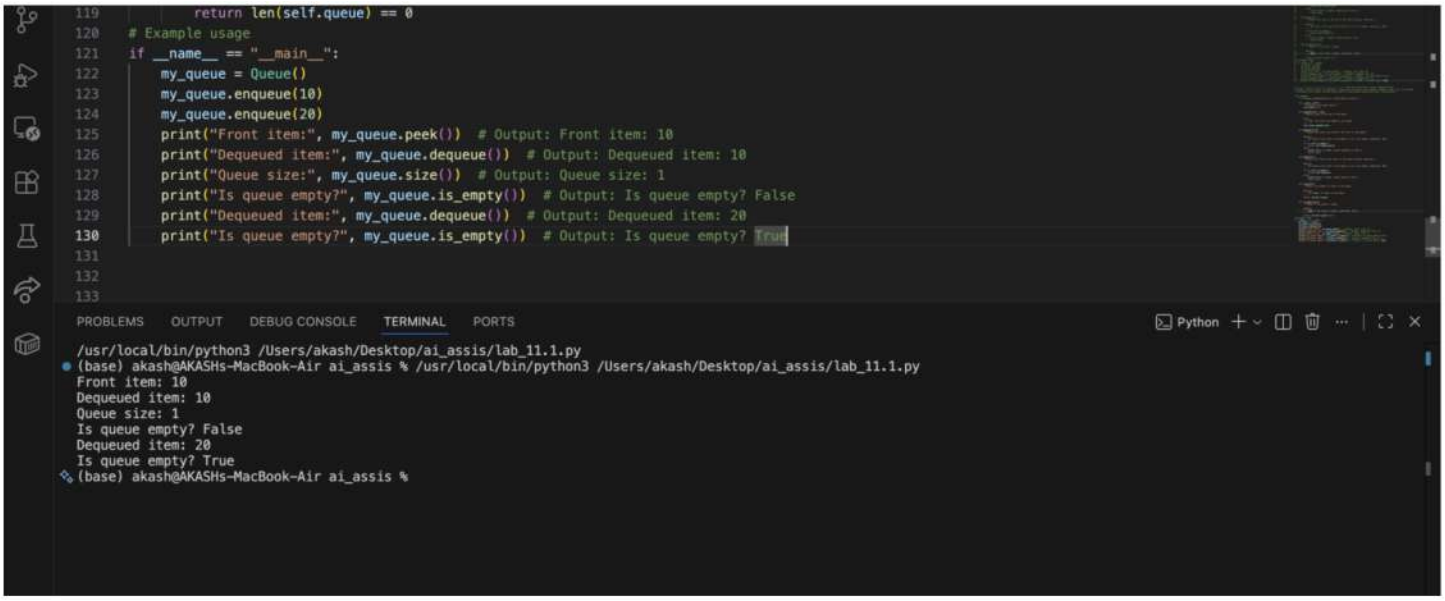
Task Description #2 – Queue Implementation

Prompt: Python program to implement a Queue data structure with enqueue, dequeue, peek, and size methods.The Queue class should include docstrings for each method explaining their functionality.

OUTPUT:



```python
# ---------------------------------------------Task Description #2 – Queue Implementation
# Prompt: Python program to implement a Queue data structure with enqueue, dequeue, peek, and size methods.
# The Queue class should include docstrings for each method explaining their functionality.

class Queue:
    """A simple implementation of a Queue data structure."""

    def __init__(self):
        """Initialize an empty queue."""
        self.queue = []

    def enqueue(self, item):
        """Add an item to the end of the queue.

        Args:
            item: The item to be added to the queue.
        """
        self.queue.append(item)

    def dequeue(self):
        """Remove and return the item at the front of the queue.

        Returns:
            The item at the front of the queue if it is not empty; otherwise, None.
        """
        if not self.is_empty():
            return self.queue.pop(0)
        else:
            print("Queue is empty. Cannot dequeue an item.")
            return None

    def peek(self):
        """Return the item at the front of the queue without removing it.

        Returns:
            The item at the front of the queue if it is not empty; otherwise, None.
        """
```



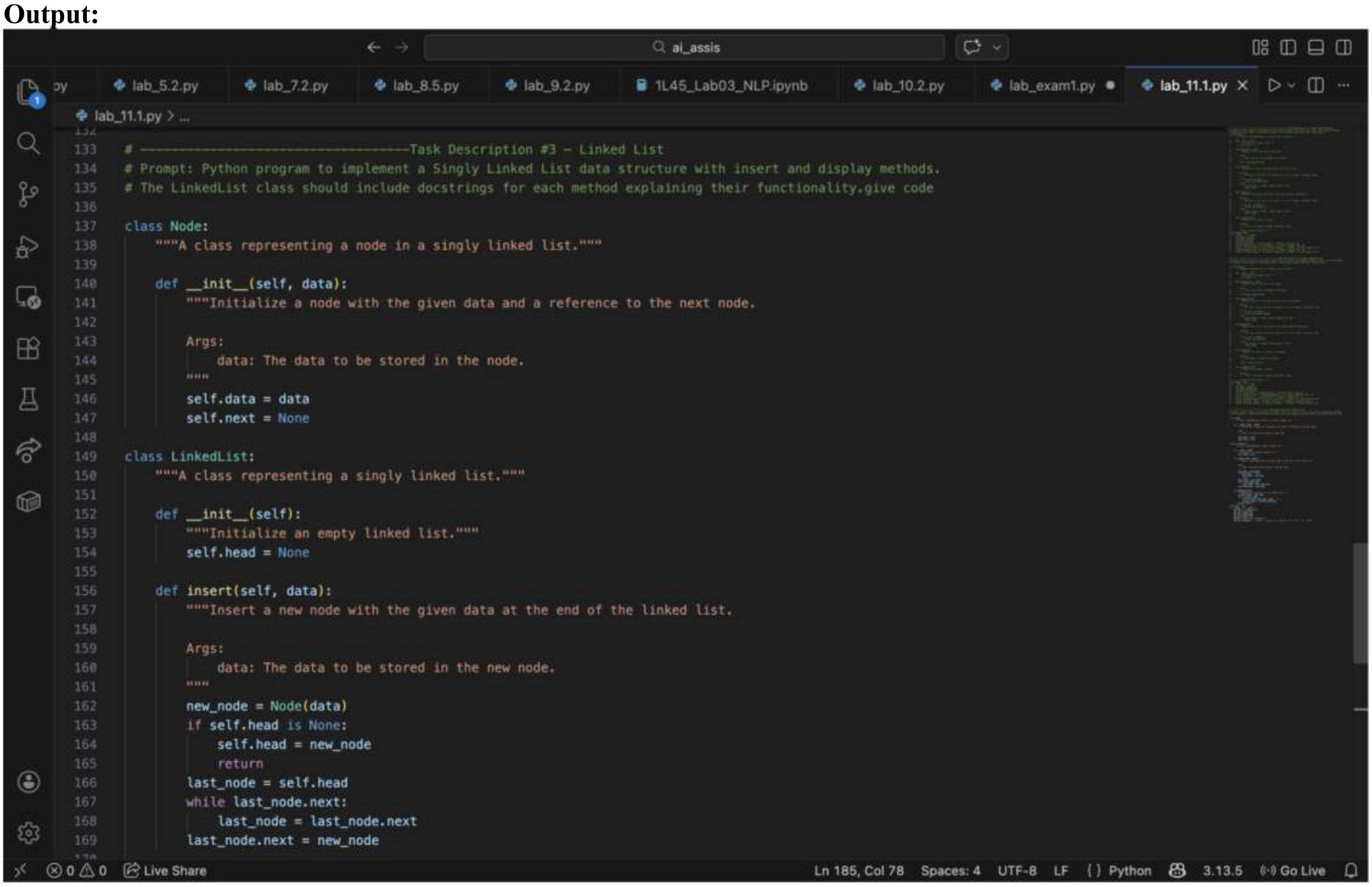
```
        return len(self.queue) == 0
# Example usage
if __name__ == "__main__":
    my_queue = Queue()
    my_queue.enqueue(10)
    my_queue.enqueue(20)
    print("Front item:", my_queue.peek())  # Output: Front item: 10
    print("Dequeued item:", my_queue.dequeue())  # Output: Dequeued item: 10
    print("Queue size:", my_queue.size())  # Output: Queue size: 1
    print("Is queue empty?", my_queue.is_empty())  # Output: Is queue empty? False
    print("Dequeued item:", my_queue.dequeue())  # Output: Dequeued item: 20
    print("Is queue empty?", my_queue.is_empty())  # Output: Is queue empty? True
```

```
/usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
(base) akash@AKASHs-MacBook-Air ai_assis % /usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
Front item: 10
Dequeued item: 10
Queue size: 1
Is queue empty? False
Dequeued item: 20
Is queue empty? True
(base) akash@AKASHs-MacBook-Air ai_assis %
```

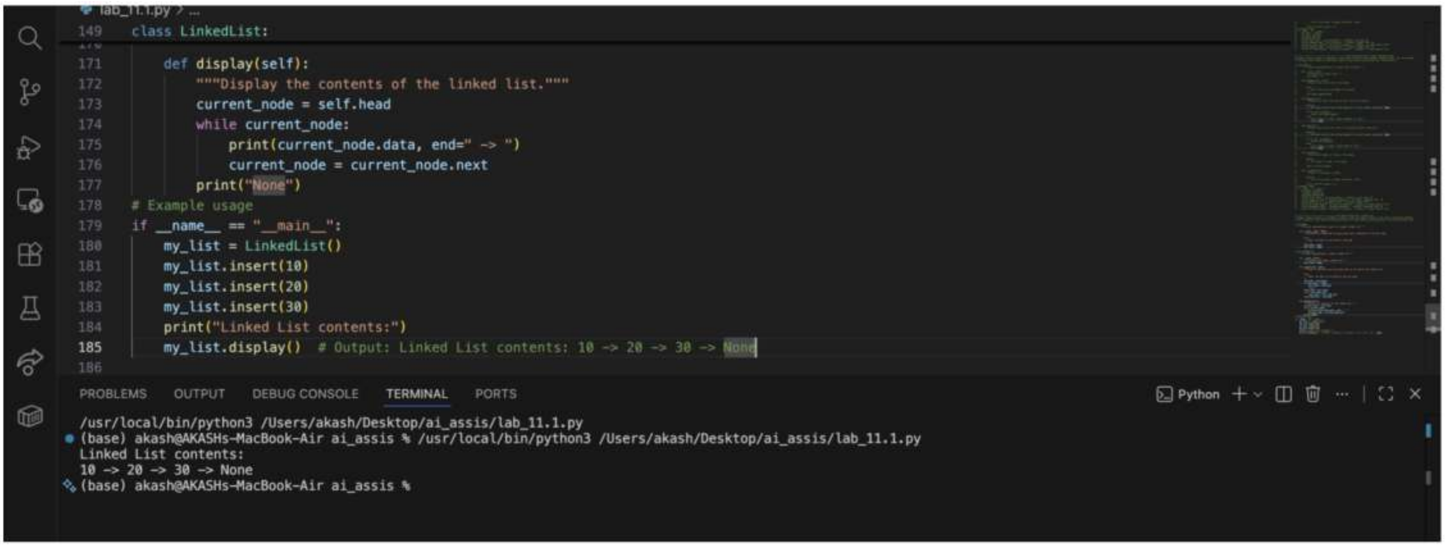
Task Description #3 – Linked List

Prompt: Python program to implement a Singly Linked List data structure with insert and display methods.The LinkedList class should include docstrings for each method explaining their functionality.give code

Output:



```
# -----------------------------------Task Description #3 – Linked List
# Prompt: Python program to implement a Singly Linked List data structure with insert and display methods.
# The LinkedList class should include docstrings for each method explaining their functionality.give code

class Node:
    """A class representing a node in a singly linked list."""

    def __init__(self, data):
        """Initialize a node with the given data and a reference to the next node.

        Args:
            data: The data to be stored in the node.
        """
        self.data = data
        self.next = None

class LinkedList:
    """A class representing a singly linked list."""

    def __init__(self):
        """Initialize an empty linked list."""
        self.head = None

    def insert(self, data):
        """Insert a new node with the given data at the end of the linked list.

        Args:
            data: The data to be stored in the new node.
        """
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return
        last_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node
```



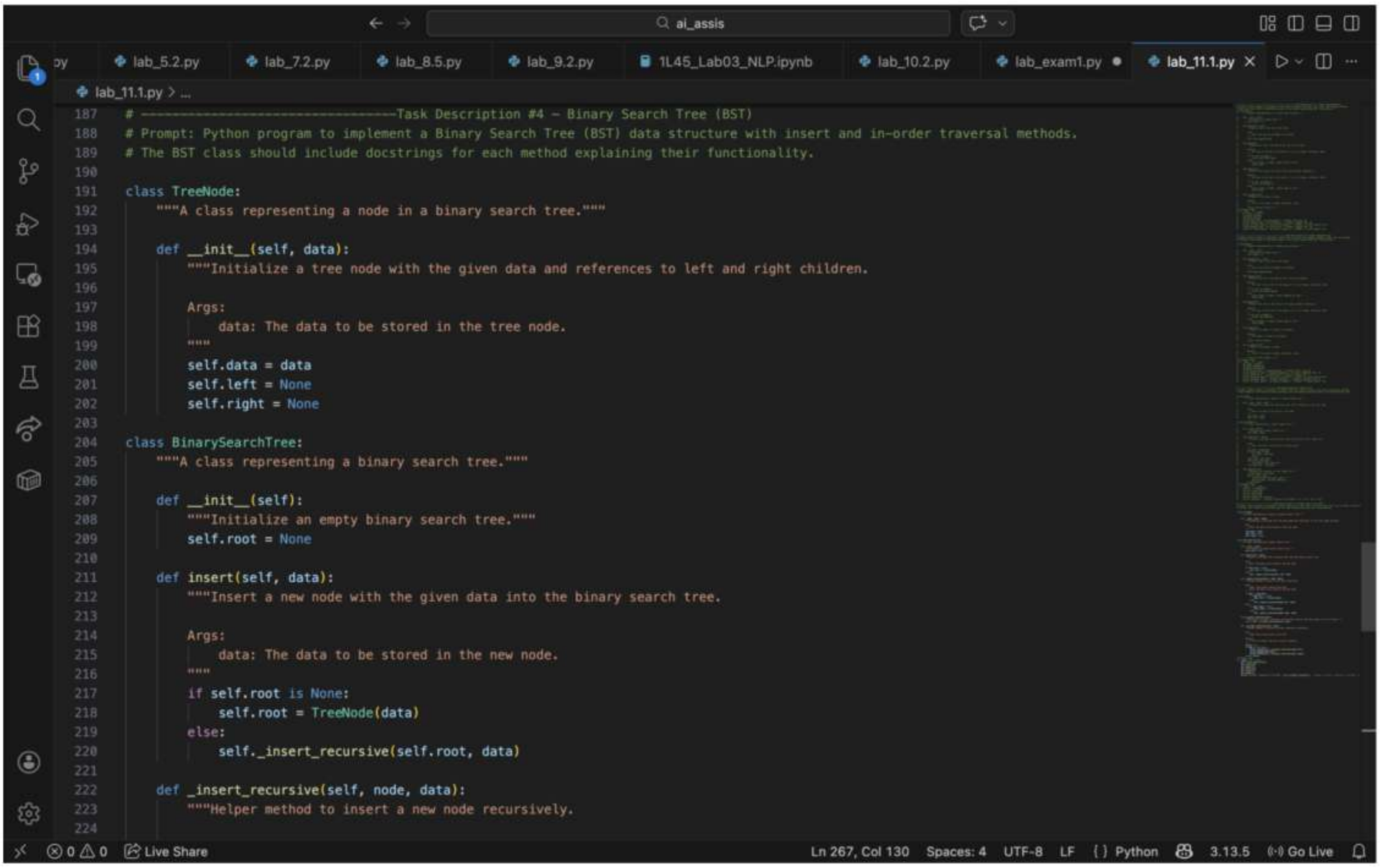
```python
class LinkedList:

    def display(self):
        """Display the contents of the linked list."""
        current_node = self.head
        while current_node:
            print(current_node.data, end=" -> ")
            current_node = current_node.next
        print("None")
# Example usage
if __name__ == "__main__":
    my_list = LinkedList()
    my_list.insert(10)
    my_list.insert(20)
    my_list.insert(30)
    print("Linked List contents:")
    my_list.display()  # Output: Linked List contents: 10 -> 20 -> 30 -> None

```

```
/usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
(base) akash@AKASHs-MacBook-Air ai_assis % /usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
Linked List contents:
10 -> 20 -> 30 -> None
(base) akash@AKASHs-MacBook-Air ai_assis %
```

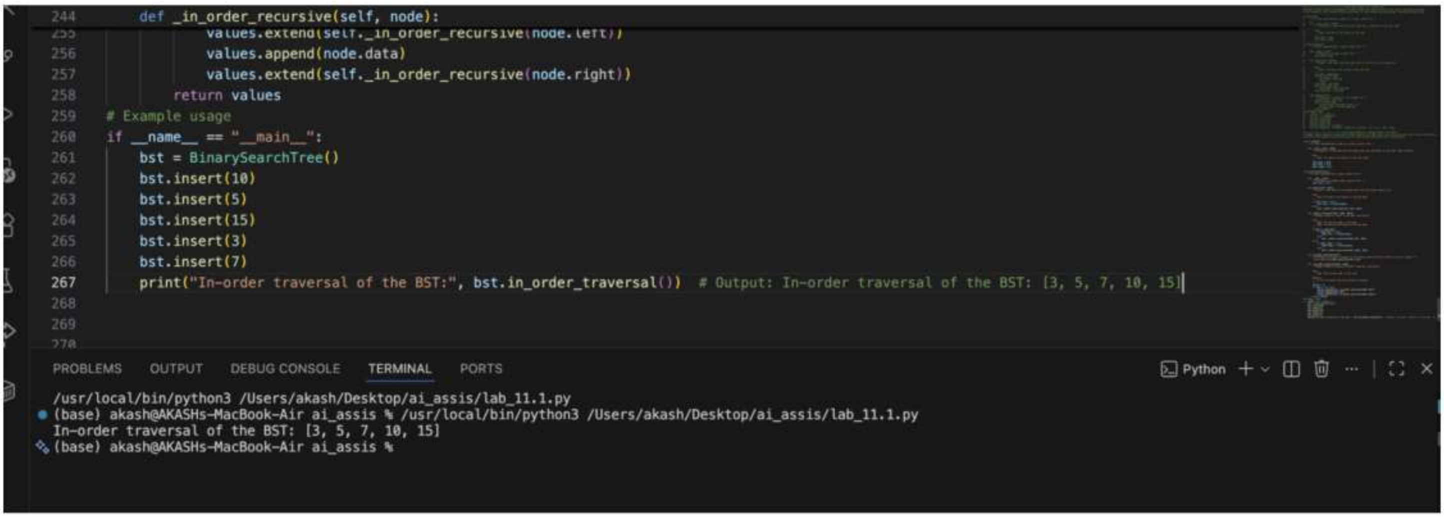
Task Description #4 – Binary Search Tree (BST)

Prompt: Python program to implement a Binary Search Tree (BST) data structure with insert and in-order traversal methods.The BST class should include docstrings for each method explaining their functionality.

OUTPUT:



```python
# ---------------------------------Task Description #4 - Binary Search Tree (BST)
# Prompt: Python program to implement a Binary Search Tree (BST) data structure with insert and in-order traversal methods.
# The BST class should include docstrings for each method explaining their functionality.

class TreeNode:
    """A class representing a node in a binary search tree."""

    def __init__(self, data):
        """Initialize a tree node with the given data and references to left and right children.

        Args:
            data: The data to be stored in the tree node.
        """
        self.data = data
        self.left = None
        self.right = None

class BinarySearchTree:
    """A class representing a binary search tree."""

    def __init__(self):
        """Initialize an empty binary search tree."""
        self.root = None

    def insert(self, data):
        """Insert a new node with the given data into the binary search tree.

        Args:
            data: The data to be stored in the new node.
        """
        if self.root is None:
            self.root = TreeNode(data)
        else:
            self._insert_recursive(self.root, data)

    def _insert_recursive(self, node, data):
        """Helper method to insert a new node recursively.

```



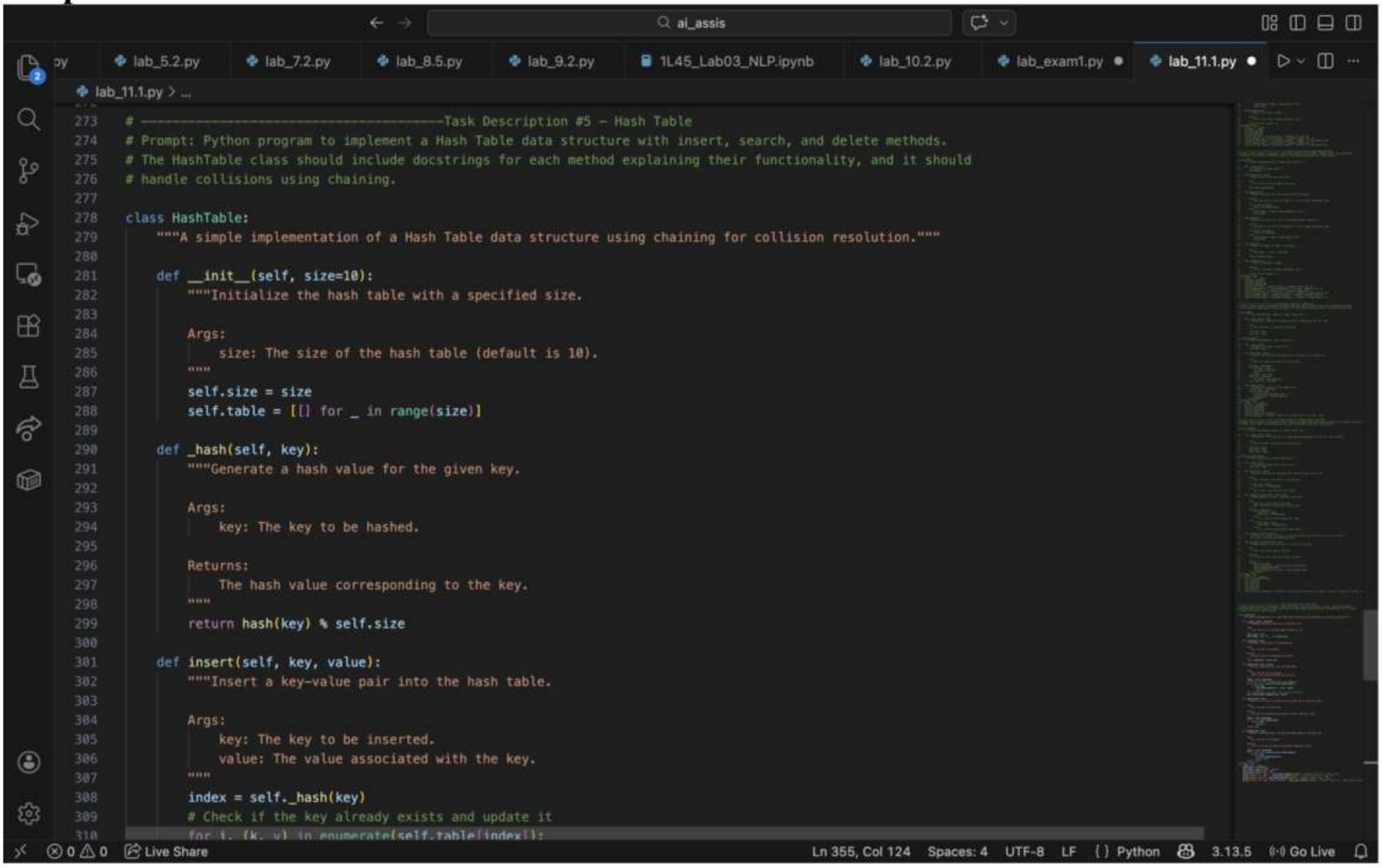
```
    def _in_order_recursive(self, node):
            values.extend(self._in_order_recursive(node.left))
            values.append(node.data)
            values.extend(self._in_order_recursive(node.right))
        return values
# Example usage
if __name__ == "__main__":
    bst = BinarySearchTree()
    bst.insert(10)
    bst.insert(5)
    bst.insert(15)
    bst.insert(3)
    bst.insert(7)
    print("In-order traversal of the BST:", bst.in_order_traversal())  # Output: In-order traversal of the BST: [3, 5, 7, 10, 15]
```

```
/usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
(base) akash@AKASHs-MacBook-Air ai_assis % /usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
In-order traversal of the BST: [3, 5, 7, 10, 15]
(base) akash@AKASHs-MacBook-Air ai_assis %
```

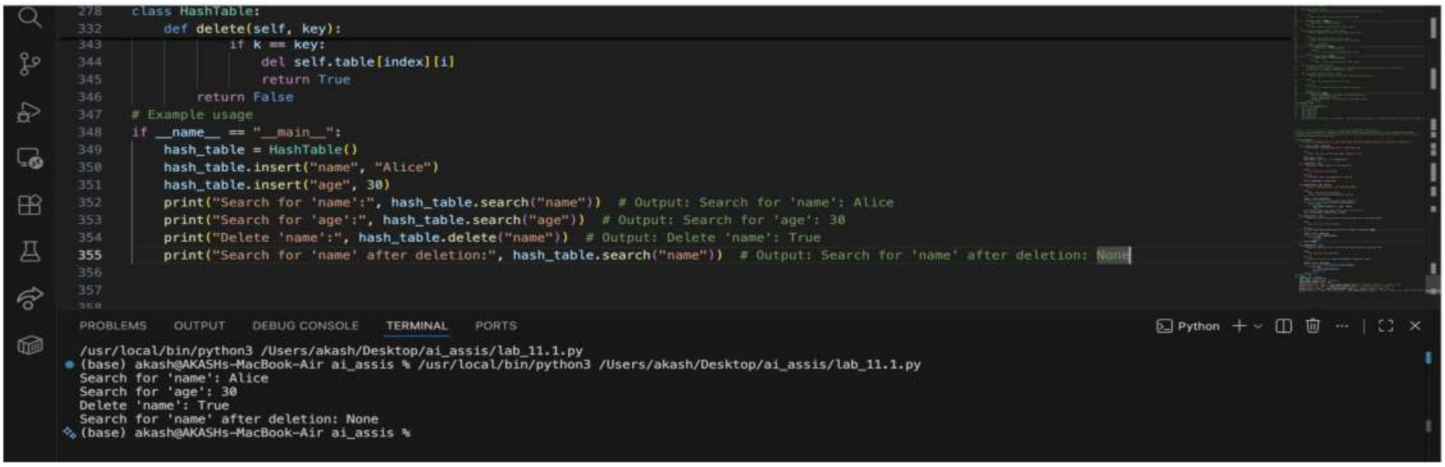
Task Description #5 – Hash Table

Prompt: Python program to implement a Hash Table data structure with insert, search, and delete methods.The HashTable class should include docstrings for each method explaining their functionality, and it should handle collisions using chaining.

Output:



```
# ----------------------------------------Task Description #5 - Hash Table
# Prompt: Python program to implement a Hash Table data structure with insert, search, and delete methods.
# The HashTable class should include docstrings for each method explaining their functionality, and it should
# handle collisions using chaining.

class HashTable:
    """A simple implementation of a Hash Table data structure using chaining for collision resolution."""

    def __init__(self, size=10):
        """Initialize the hash table with a specified size.

        Args:
            size: The size of the hash table (default is 10).
        """
        self.size = size
        self.table = [[] for _ in range(size)]

    def _hash(self, key):
        """Generate a hash value for the given key.

        Args:
            key: The key to be hashed.

        Returns:
            The hash value corresponding to the key.
        """
        return hash(key) % self.size

    def insert(self, key, value):
        """Insert a key-value pair into the hash table.

        Args:
            key: The key to be inserted.
            value: The value associated with the key.
        """
        index = self._hash(key)
        # Check if the key already exists and update it
        for i, (k, v) in enumerate(self.table[index]):
```



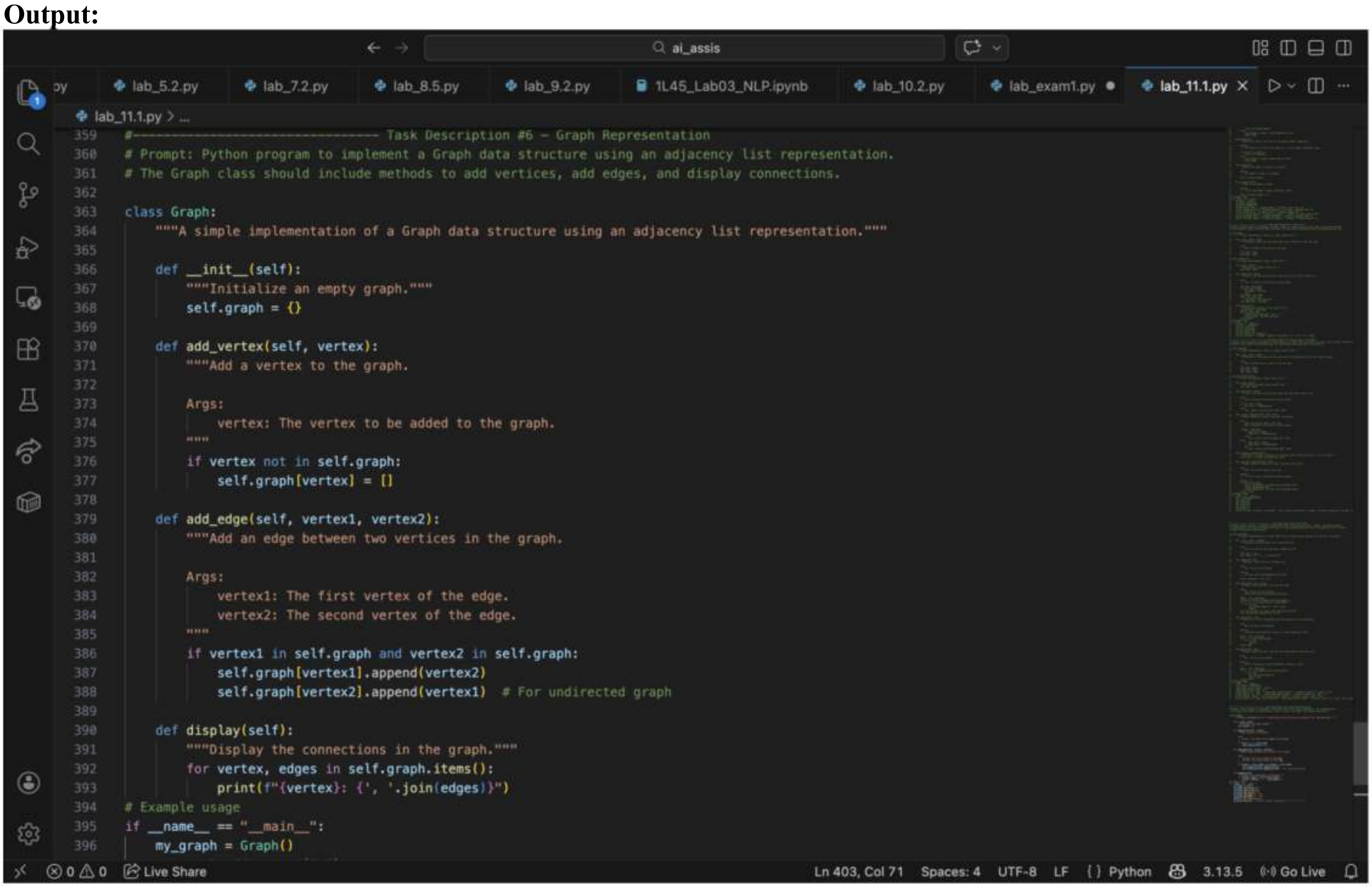
```
class HashTable:
    def delete(self, key):
            if k == key:
                del self.table[index][i]
                return True
        return False
# Example usage
if __name__ == "__main__":
    hash_table = HashTable()
    hash_table.insert("name", "Alice")
    hash_table.insert("age", 30)
    print("Search for 'name':", hash_table.search("name"))  # Output: Search for 'name': Alice
    print("Search for 'age':", hash_table.search("age"))  # Output: Search for 'age': 30
    print("Delete 'name':", hash_table.delete("name"))  # Output: Delete 'name': True
    print("Search for 'name' after deletion:", hash_table.search("name"))  # Output: Search for 'name' after deletion: None
```

```
/usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
(base) akash@AKASHs-MacBook-Air ai_assis % /usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
Search for 'name': Alice
Search for 'age': 30
Delete 'name': True
Search for 'name' after deletion: None
(base) akash@AKASHs-MacBook-Air ai_assis %
```

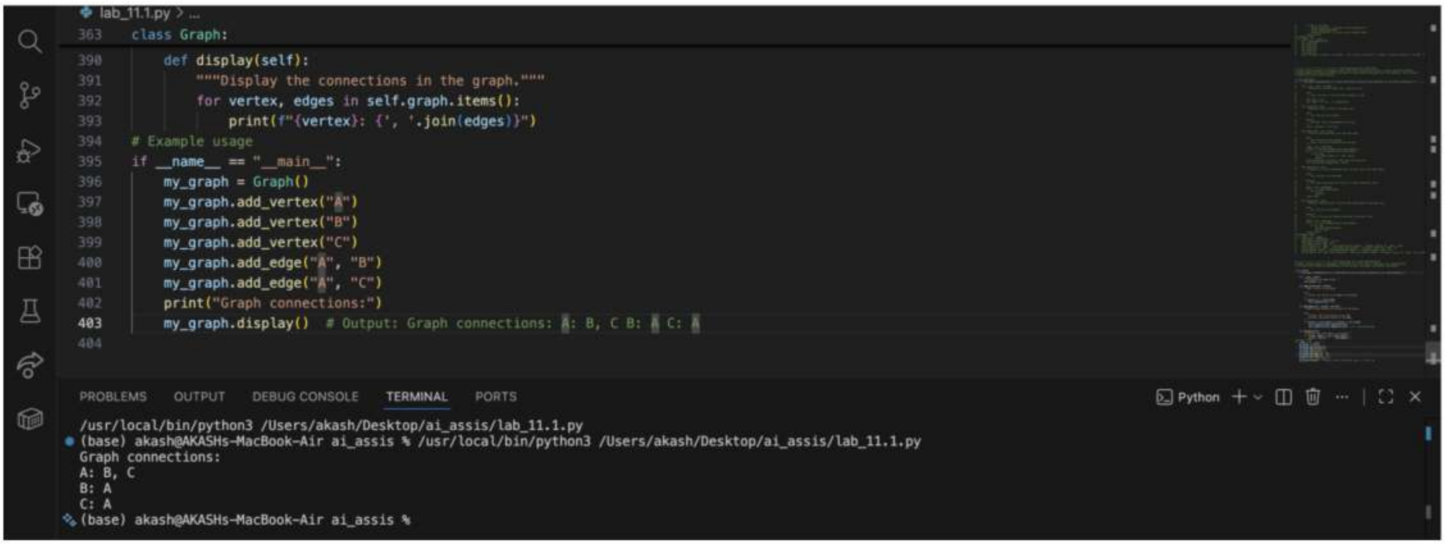
Task Description #6 – Graph Representation

Prompt: Python program to implement a Graph data structure using an adjacency list representation.The Graph class should include methods to add vertices, add edges, and display connections.

Output:



```
#---------------------------------- Task Description #6 – Graph Representation
# Prompt: Python program to implement a Graph data structure using an adjacency list representation.
# The Graph class should include methods to add vertices, add edges, and display connections.

class Graph:
    """A simple implementation of a Graph data structure using an adjacency list representation."""

    def __init__(self):
        """Initialize an empty graph."""
        self.graph = {}

    def add_vertex(self, vertex):
        """Add a vertex to the graph.

        Args:
            vertex: The vertex to be added to the graph.
        """
        if vertex not in self.graph:
            self.graph[vertex] = []

    def add_edge(self, vertex1, vertex2):
        """Add an edge between two vertices in the graph.

        Args:
            vertex1: The first vertex of the edge.
            vertex2: The second vertex of the edge.
        """
        if vertex1 in self.graph and vertex2 in self.graph:
            self.graph[vertex1].append(vertex2)
            self.graph[vertex2].append(vertex1)  # For undirected graph

    def display(self):
        """Display the connections in the graph."""
        for vertex, edges in self.graph.items():
            print(f"{vertex}: {', '.join(edges)}")
# Example usage
if __name__ == "__main__":
    my_graph = Graph()
```



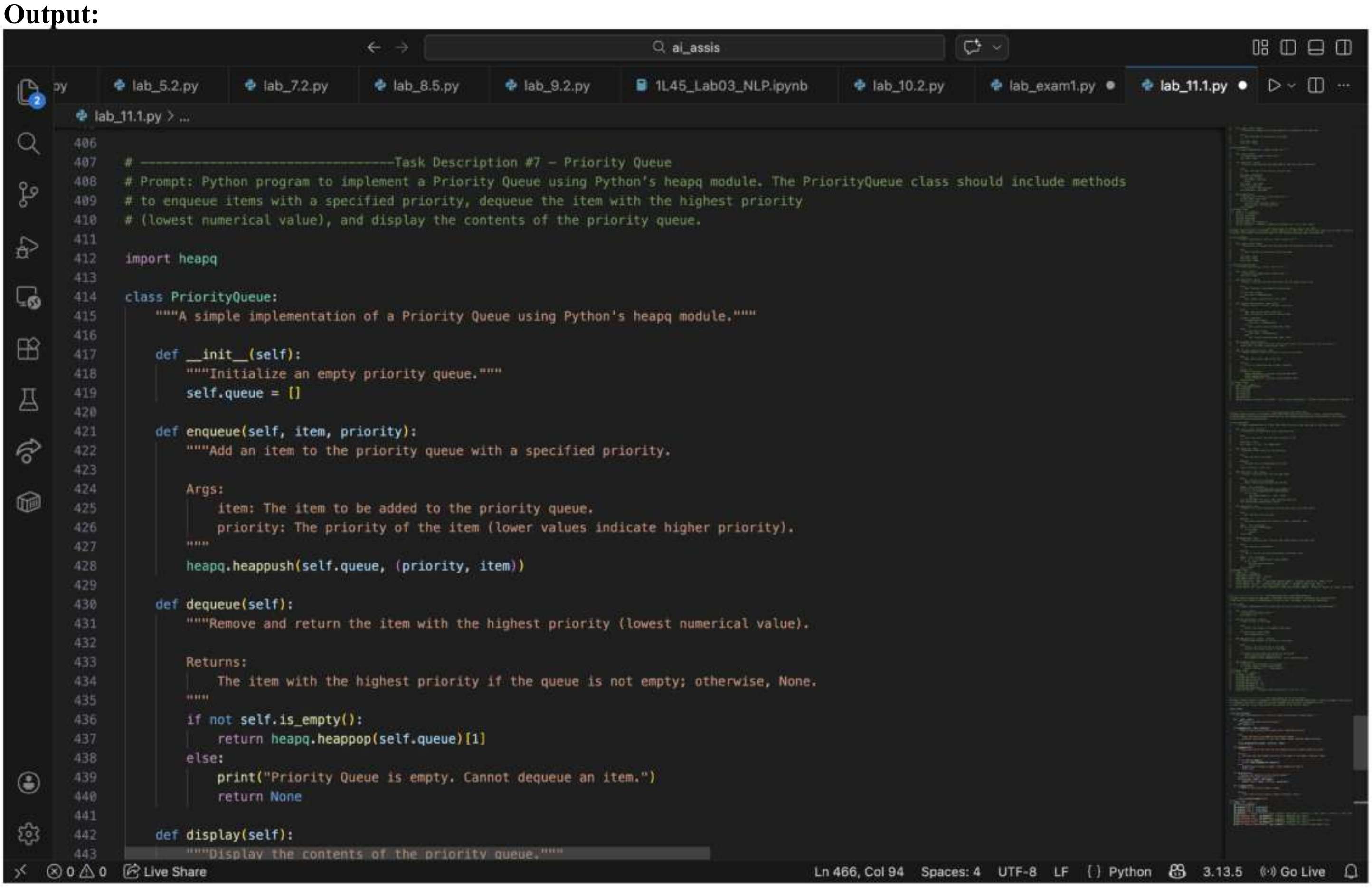
```python
class Graph:
    def display(self):
        """Display the connections in the graph."""
        for vertex, edges in self.graph.items():
            print(f"{vertex}: {', '.join(edges)}")
# Example usage
if __name__ == "__main__":
    my_graph = Graph()
    my_graph.add_vertex("A")
    my_graph.add_vertex("B")
    my_graph.add_vertex("C")
    my_graph.add_edge("A", "B")
    my_graph.add_edge("A", "C")
    print("Graph connections:")
    my_graph.display()  # Output: Graph connections: A: B, C B: A C: A
```

```
/usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
(base) akash@AKASHs-MacBook-Air ai_assis % /usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
Graph connections:
A: B, C
B: A
C: A
(base) akash@AKASHs-MacBook-Air ai_assis %
```

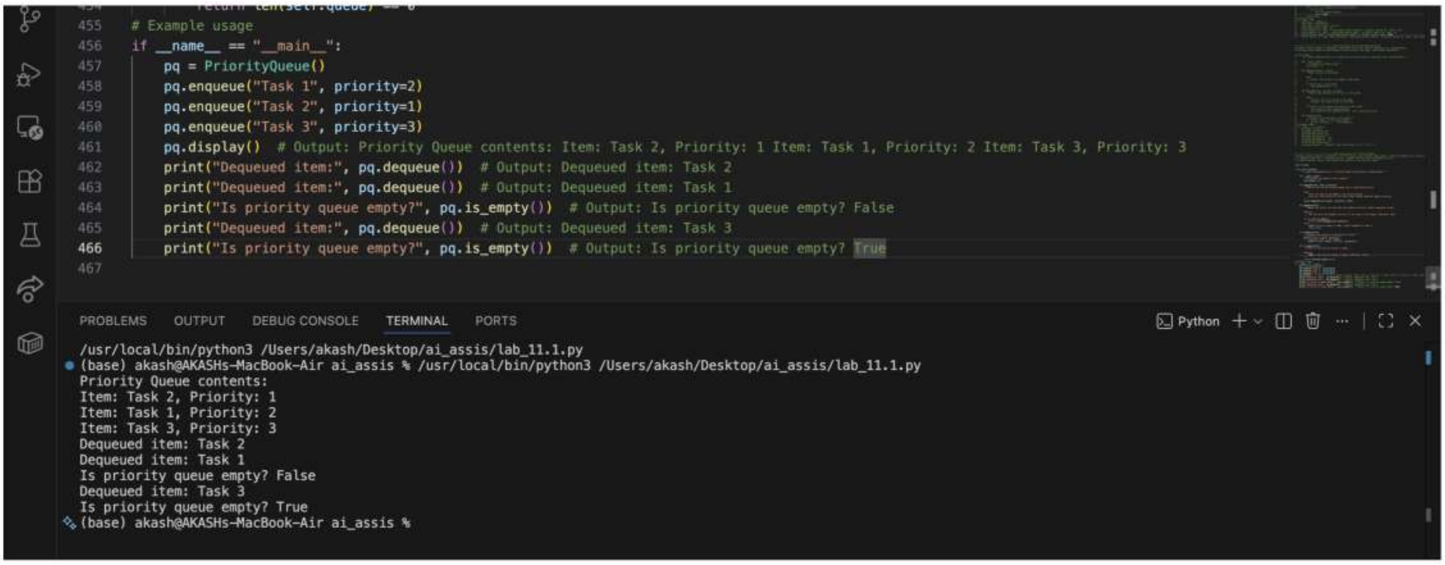
Task Description #7 – Priority Queue

Prompt: Python program to implement a Priority Queue using Python’s heapq module. The PriorityQueue class should include methods to enqueue items with a specified priority, dequeue the item with the highest priority

Output:



```python
# ----------------------------------Task Description #7 – Priority Queue
# Prompt: Python program to implement a Priority Queue using Python's heapq module. The PriorityQueue class should include methods
# to enqueue items with a specified priority, dequeue the item with the highest priority
# (lowest numerical value), and display the contents of the priority queue.

import heapq

class PriorityQueue:
    """A simple implementation of a Priority Queue using Python's heapq module."""

    def __init__(self):
        """Initialize an empty priority queue."""
        self.queue = []

    def enqueue(self, item, priority):
        """Add an item to the priority queue with a specified priority.

        Args:
            item: The item to be added to the priority queue.
            priority: The priority of the item (lower values indicate higher priority).
        """
        heapq.heappush(self.queue, (priority, item))

    def dequeue(self):
        """Remove and return the item with the highest priority (lowest numerical value).

        Returns:
            The item with the highest priority if the queue is not empty; otherwise, None.
        """
        if not self.is_empty():
            return heapq.heappop(self.queue)[1]
        else:
            print("Priority Queue is empty. Cannot dequeue an item.")
            return None

    def display(self):
        """Display the contents of the priority queue."""
```



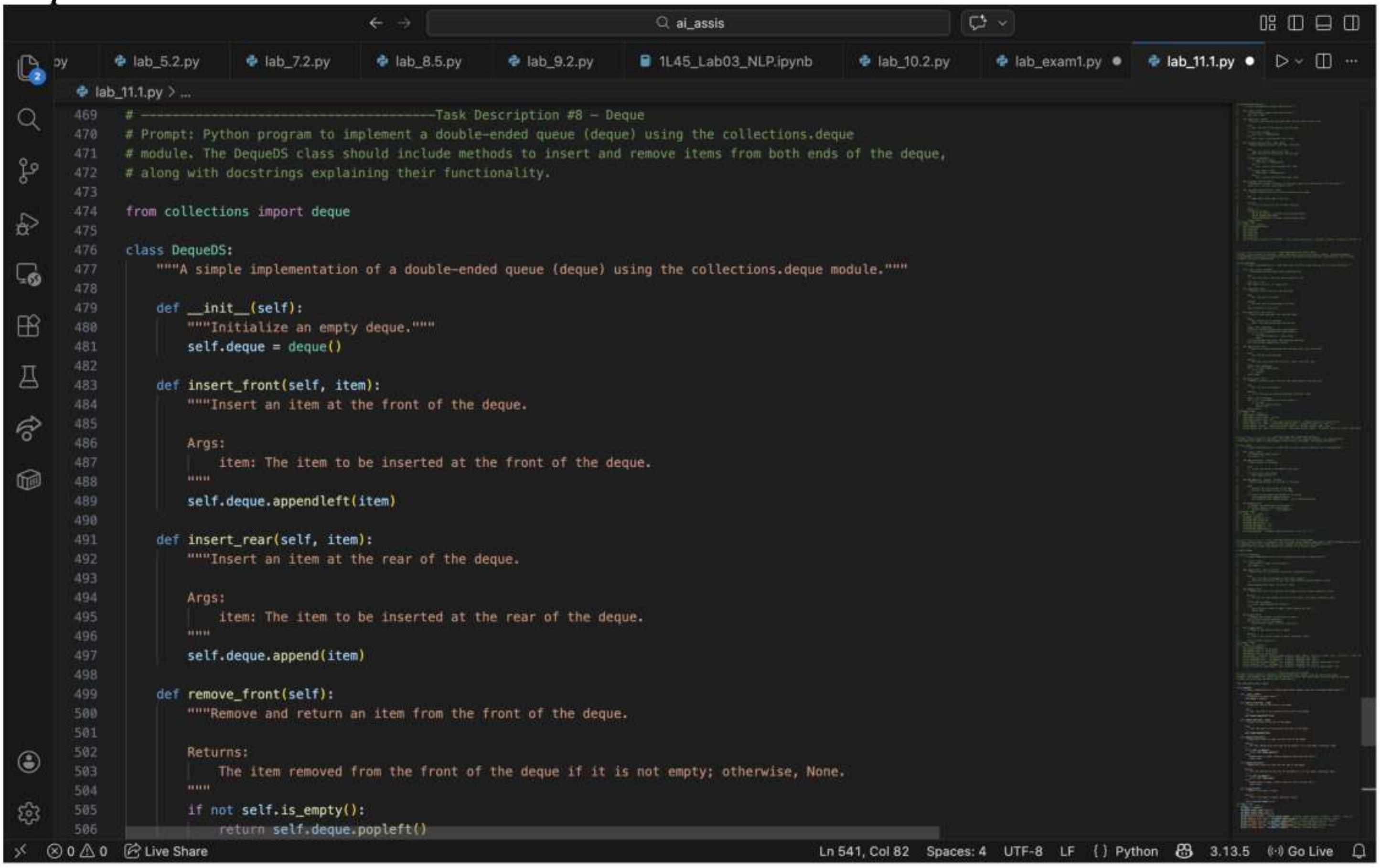
```python
        return len(self.queue) == 0
# Example usage
if __name__ == "__main__":
    pq = PriorityQueue()
    pq.enqueue("Task 1", priority=2)
    pq.enqueue("Task 2", priority=1)
    pq.enqueue("Task 3", priority=3)
    pq.display()  # Output: Priority Queue contents: Item: Task 2, Priority: 1 Item: Task 1, Priority: 2 Item: Task 3, Priority: 3
    print("Dequeued item:", pq.dequeue())  # Output: Dequeued item: Task 2
    print("Dequeued item:", pq.dequeue())  # Output: Dequeued item: Task 1
    print("Is priority queue empty?", pq.is_empty())  # Output: Is priority queue empty? False
    print("Dequeued item:", pq.dequeue())  # Output: Dequeued item: Task 3
    print("Is priority queue empty?", pq.is_empty())  # Output: Is priority queue empty? True
```

```
/usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
(base) akash@AKASHs-MacBook-Air ai_assis % /usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
Priority Queue contents:
Item: Task 2, Priority: 1
Item: Task 1, Priority: 2
Item: Task 3, Priority: 3
Dequeued item: Task 2
Dequeued item: Task 1
Is priority queue empty? False
Dequeued item: Task 3
Is priority queue empty? True
(base) akash@AKASHs-MacBook-Air ai_assis %
```

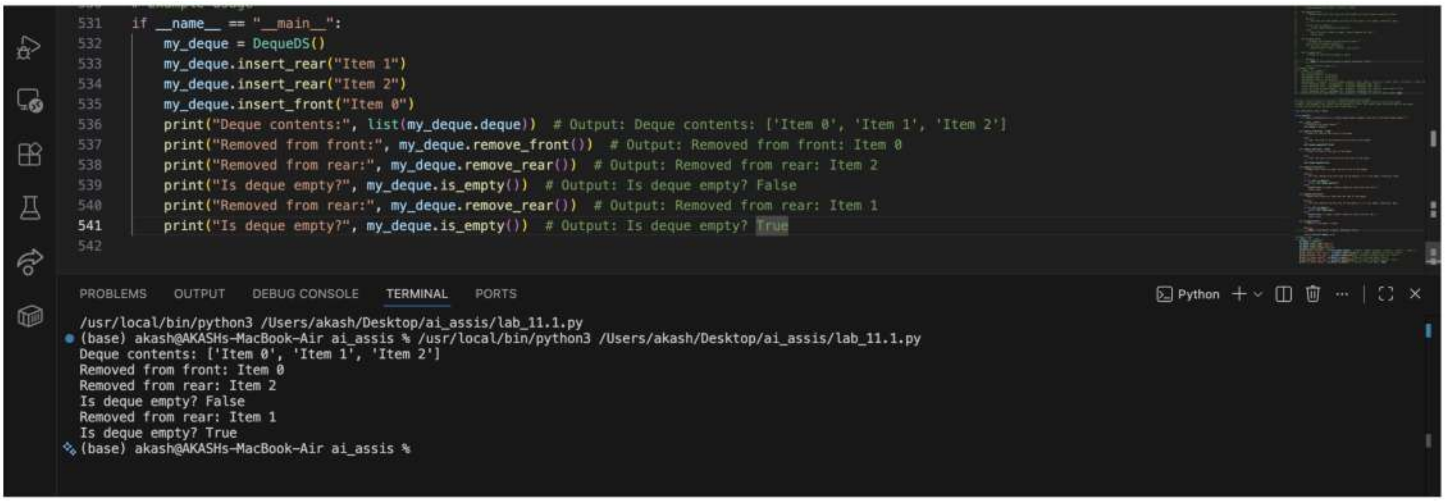
Task Description #8 – Deque

Prompt: Python program to implement a double-ended queue (deque) using the collections.deque module. The DequeDS class should include methods to insert and remove items from both ends of the deque, along with docstrings explaining their functionality.

Output:



```python
# ----------------------------------------Task Description #8 – Deque
# Prompt: Python program to implement a double-ended queue (deque) using the collections.deque
# module. The DequeDS class should include methods to insert and remove items from both ends of the deque,
# along with docstrings explaining their functionality.

from collections import deque

class DequeDS:
    """A simple implementation of a double-ended queue (deque) using the collections.deque module."""

    def __init__(self):
        """Initialize an empty deque."""
        self.deque = deque()

    def insert_front(self, item):
        """Insert an item at the front of the deque.

        Args:
            item: The item to be inserted at the front of the deque.
        """
        self.deque.appendleft(item)

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

        Args:
            item: The item to be inserted at the rear of the deque.
        """
        self.deque.append(item)

    def remove_front(self):
        """Remove and return an item from the front of the deque.

        Returns:
            The item removed from the front of the deque if it is not empty; otherwise, None.
        """
        if not self.is_empty():
            return self.deque.popleft()
```



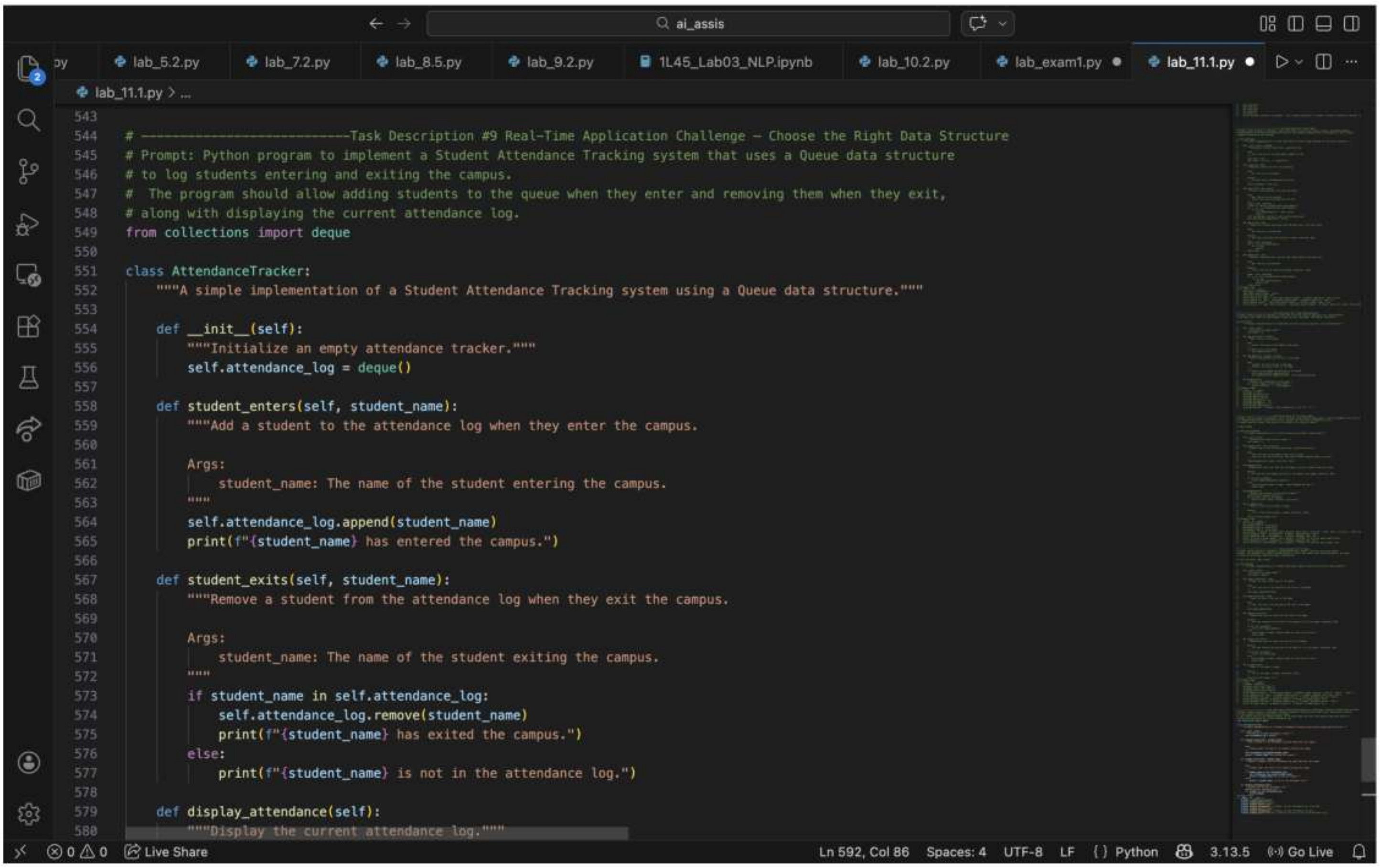
```python
if __name__ == "__main__":
    my_deque = DequeDS()
    my_deque.insert_rear("Item 1")
    my_deque.insert_rear("Item 2")
    my_deque.insert_front("Item 0")
    print("Deque contents:", list(my_deque.deque))  # Output: Deque contents: ['Item 0', 'Item 1', 'Item 2']
    print("Removed from front:", my_deque.remove_front())  # Output: Removed from front: Item 0
    print("Removed from rear:", my_deque.remove_rear())  # Output: Removed from rear: Item 2
    print("Is deque empty?", my_deque.is_empty())  # Output: Is deque empty? False
    print("Removed from rear:", my_deque.remove_rear())  # Output: Removed from rear: Item 1
    print("Is deque empty?", my_deque.is_empty())  # Output: Is deque empty? True
```

```
/usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
(base) akash@AKASHs-MacBook-Air ai_assis % /usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
Deque contents: ['Item 0', 'Item 1', 'Item 2']
Removed from front: Item 0
Removed from rear: Item 2
Is deque empty? False
Removed from rear: Item 1
Is deque empty? True
(base) akash@AKASHs-MacBook-Air ai_assis %
```

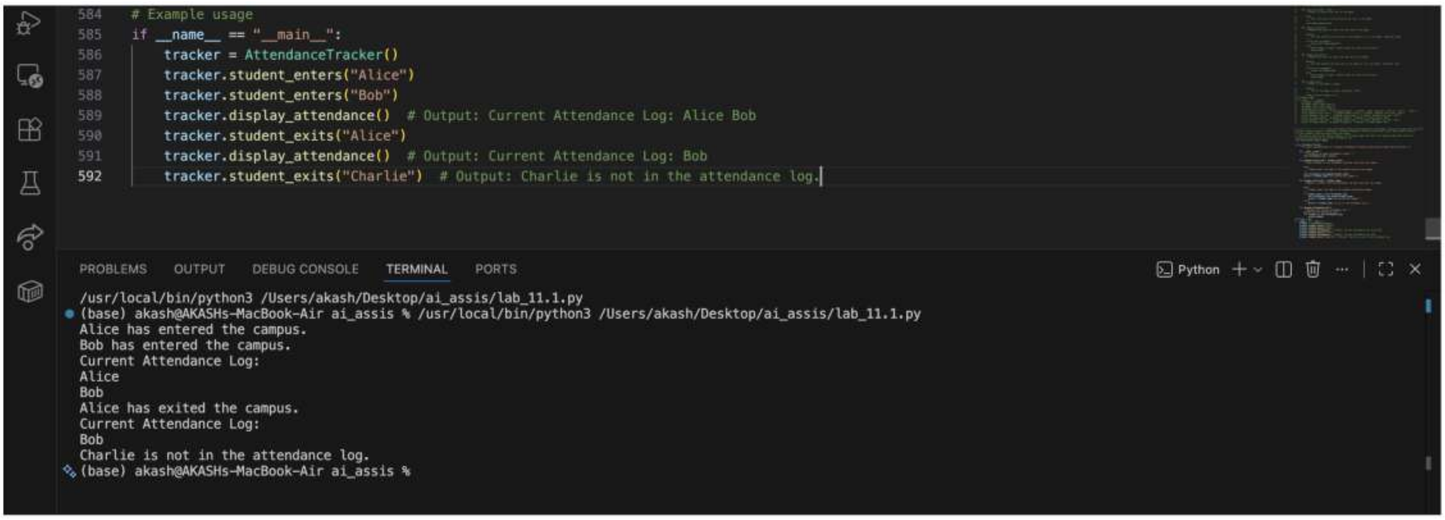
Task Description #9 Real-Time Application Challenge – Choose the Right Data Structure

Prompt: Python program to implement a Student Attendance Tracking system that uses a Queue data structure to log students entering and exiting the campus.The program should allow adding students to the queue when they enter and removing them when they exit, along with displaying the current attendance log.from collections import deque

Output:



```python

# ----------------------------Task Description #9 Real-Time Application Challenge – Choose the Right Data Structure
# Prompt: Python program to implement a Student Attendance Tracking system that uses a Queue data structure
# to log students entering and exiting the campus.
#  The program should allow adding students to the queue when they enter and removing them when they exit,
# along with displaying the current attendance log.
from collections import deque

class AttendanceTracker:
    """A simple implementation of a Student Attendance Tracking system using a Queue data structure."""

    def __init__(self):
        """Initialize an empty attendance tracker."""
        self.attendance_log = deque()

    def student_enters(self, student_name):
        """Add a student to the attendance log when they enter the campus.

        Args:
            student_name: The name of the student entering the campus.
        """
        self.attendance_log.append(student_name)
        print(f"{student_name} has entered the campus.")

    def student_exits(self, student_name):
        """Remove a student from the attendance log when they exit the campus.

        Args:
            student_name: The name of the student exiting the campus.
        """
        if student_name in self.attendance_log:
            self.attendance_log.remove(student_name)
            print(f"{student_name} has exited the campus.")
        else:
            print(f"{student_name} is not in the attendance log.")

    def display_attendance(self):
        """Display the current attendance log."""
```



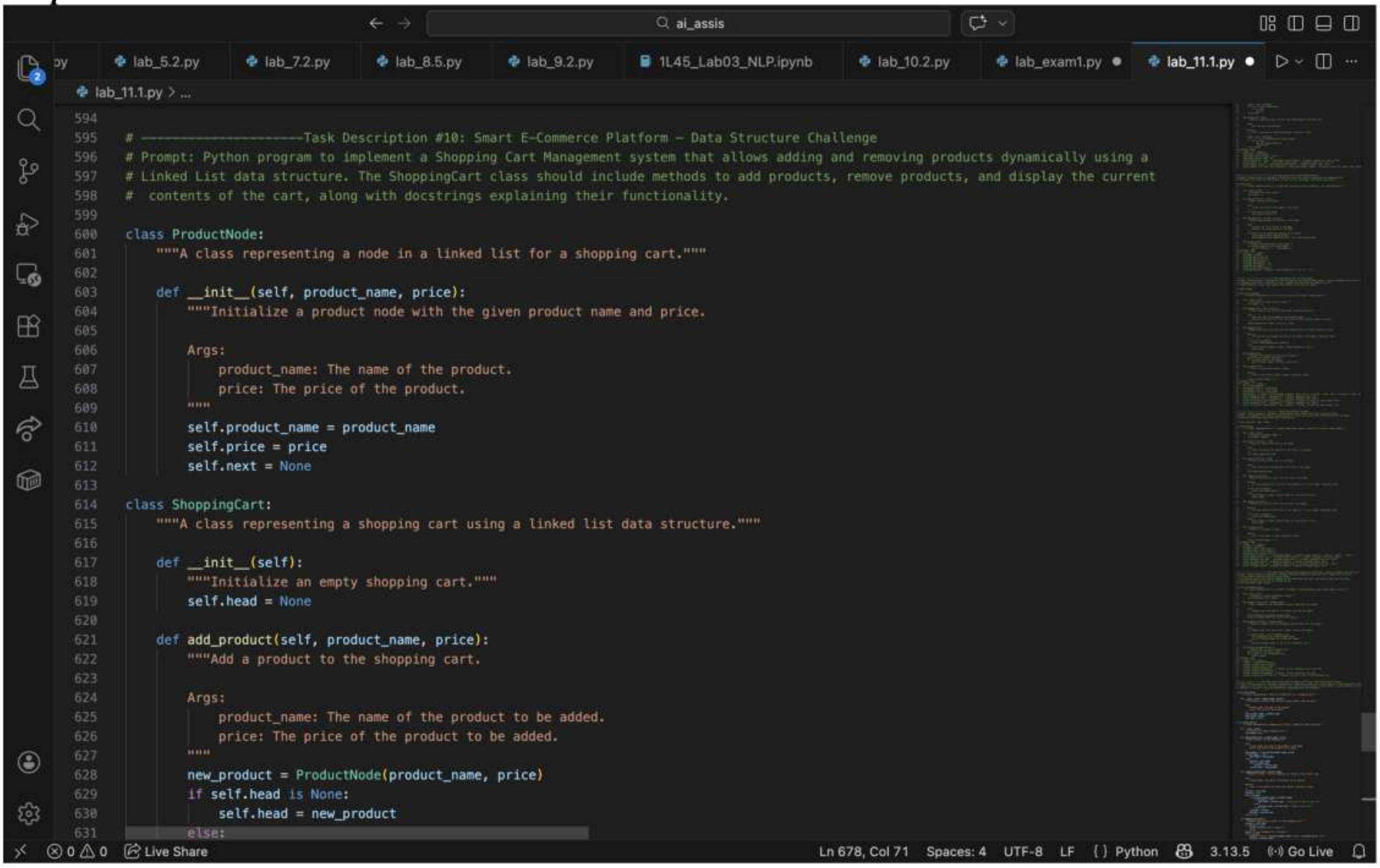
```python
# Example usage
if __name__ == "__main__":
    tracker = AttendanceTracker()
    tracker.student_enters("Alice")
    tracker.student_enters("Bob")
    tracker.display_attendance()  # Output: Current Attendance Log: Alice Bob
    tracker.student_exits("Alice")
    tracker.display_attendance()  # Output: Current Attendance Log: Bob
    tracker.student_exits("Charlie")  # Output: Charlie is not in the attendance log.
```

```
/usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
(base) akash@AKASHs-MacBook-Air ai_assis % /usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
Alice has entered the campus.
Bob has entered the campus.
Current Attendance Log:
Alice
Bob
Alice has exited the campus.
Current Attendance Log:
Bob
Charlie is not in the attendance log.
(base) akash@AKASHs-MacBook-Air ai_assis %
```

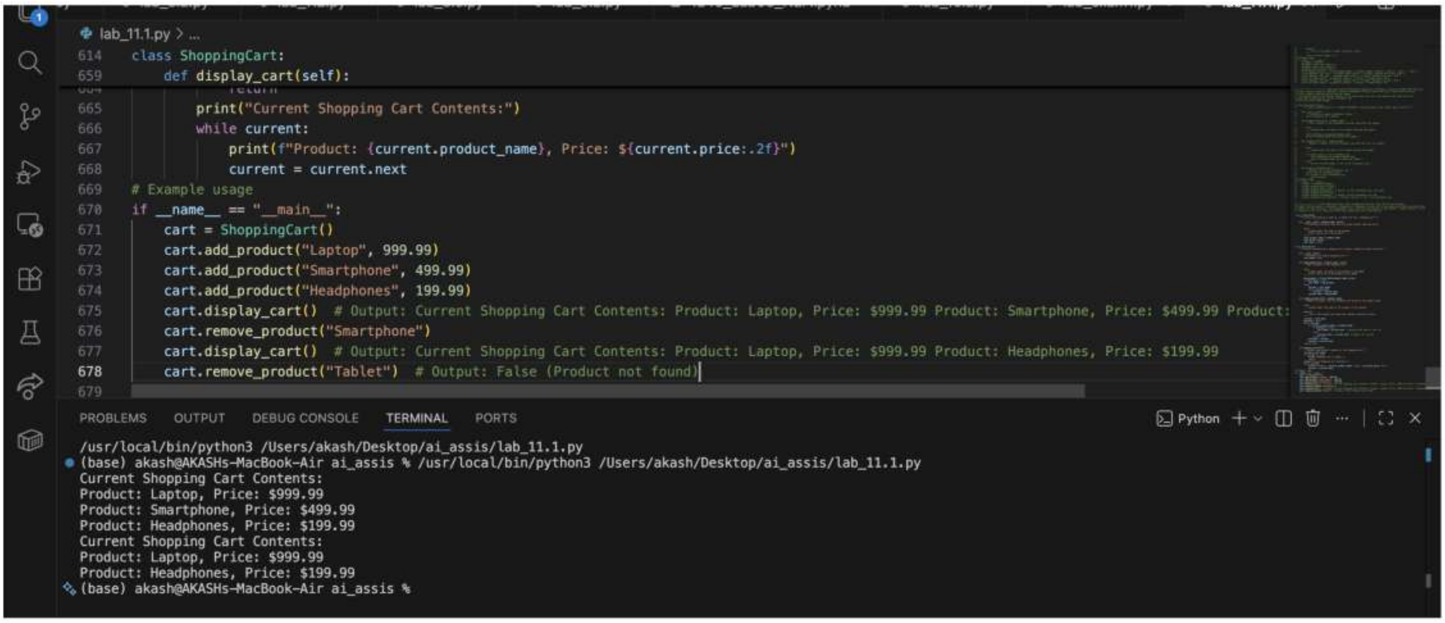
Task Description #10: Smart E-Commerce Platform – Data Structure Challenge

Prompt: Python program to implement a Shopping Cart Management system that allows adding and removing products dynamically using a Linked List data structure. The ShoppingCart class should include methods to add products, remove products, and display the current contents of the cart, along with docstrings explaining their functionality.

Output:



```python

# ---------------------Task Description #10: Smart E-Commerce Platform – Data Structure Challenge
# Prompt: Python program to implement a Shopping Cart Management system that allows adding and removing products dynamically using a
# Linked List data structure. The ShoppingCart class should include methods to add products, remove products, and display the current
#  contents of the cart, along with docstrings explaining their functionality.

class ProductNode:
    """A class representing a node in a linked list for a shopping cart."""

    def __init__(self, product_name, price):
        """Initialize a product node with the given product name and price.

        Args:
            product_name: The name of the product.
            price: The price of the product.
        """
        self.product_name = product_name
        self.price = price
        self.next = None

class ShoppingCart:
    """A class representing a shopping cart using a linked list data structure."""

    def __init__(self):
        """Initialize an empty shopping cart."""
        self.head = None

    def add_product(self, product_name, price):
        """Add a product to the shopping cart.

        Args:
            product_name: The name of the product to be added.
            price: The price of the product to be added.
        """
        new_product = ProductNode(product_name, price)
        if self.head is None:
            self.head = new_product
        else:
```



```python
class ShoppingCart:
    def display_cart(self):
                return
            print("Current Shopping Cart Contents:")
            while current:
                print(f"Product: {current.product_name}, Price: ${current.price:.2f}")
                current = current.next
# Example usage
if __name__ == "__main__":
    cart = ShoppingCart()
    cart.add_product("Laptop", 999.99)
    cart.add_product("Smartphone", 499.99)
    cart.add_product("Headphones", 199.99)
    cart.display_cart()  # Output: Current Shopping Cart Contents: Product: Laptop, Price: $999.99 Product: Smartphone, Price: $499.99 Product:
    cart.remove_product("Smartphone")
    cart.display_cart()  # Output: Current Shopping Cart Contents: Product: Laptop, Price: $999.99 Product: Headphones, Price: $199.99
    cart.remove_product("Tablet")  # Output: False (Product not found)
```

```
/usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
(base) akash@AKASHs-MacBook-Air ai_assis % /usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
Current Shopping Cart Contents:
Product: Laptop, Price: $999.99
Product: Smartphone, Price: $499.99
Product: Headphones, Price: $199.99
Current Shopping Cart Contents:
Product: Laptop, Price: $999.99
Product: Headphones, Price: $199.99
(base) akash@AKASHs-MacBook-Air ai_assis %
```