**Assignment-11.5**

**Name: Jannu Athyunnatha**

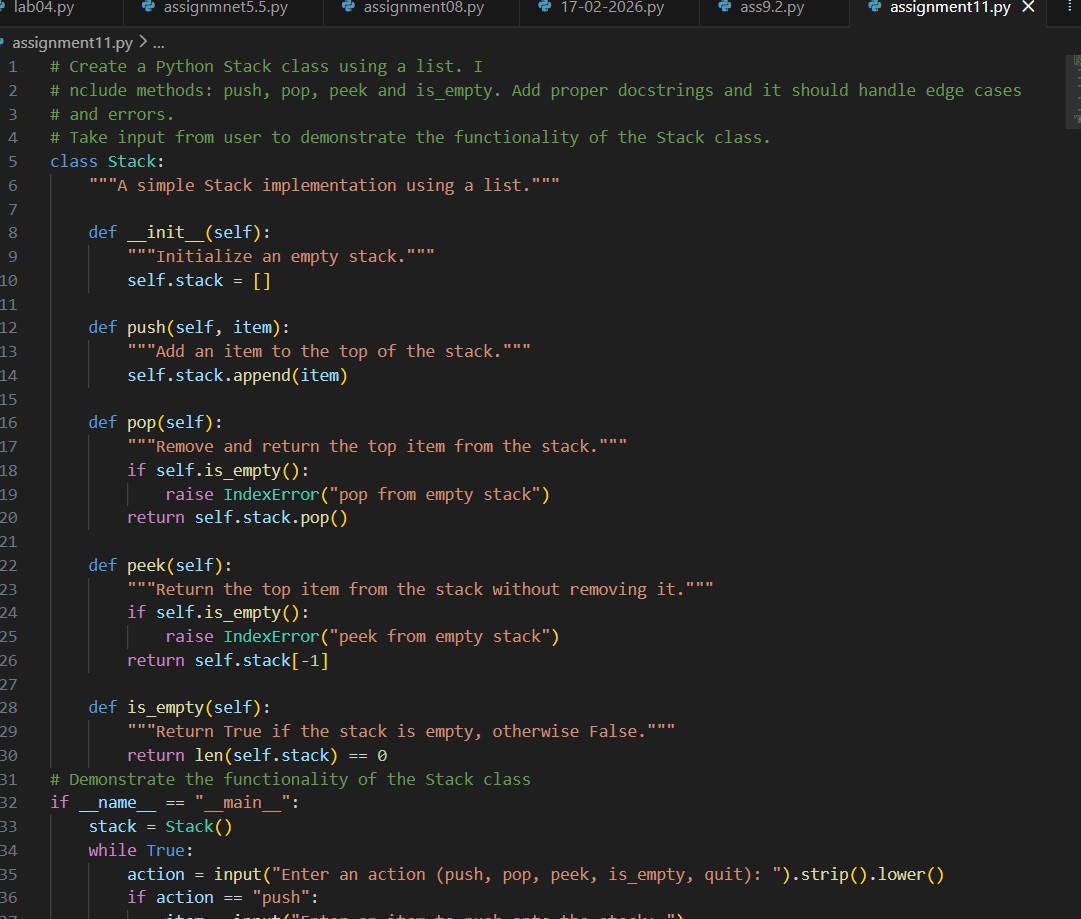
**Ht.no: 2303A51956**

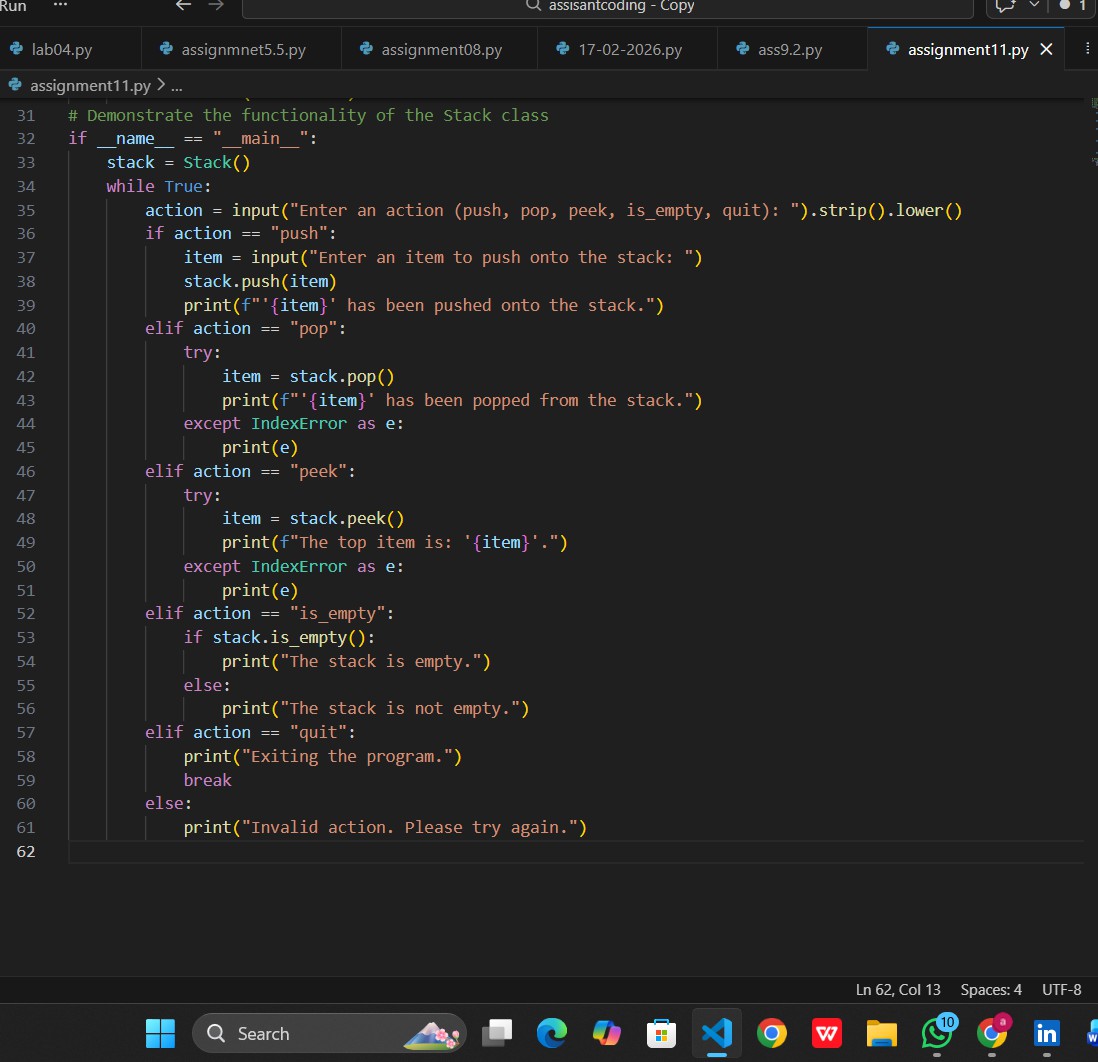
**Batch. No: 22**

**Task Description #1 – Stack Implementation Prompt:**

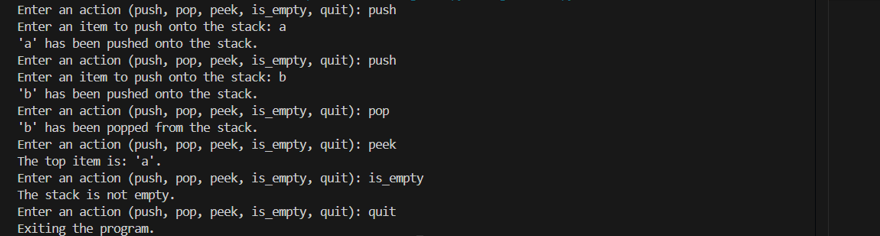
Create a Python Stack class using a list. Include methods: push, pop, peek and is\_empty. Add proper docstrings and it should handle edge cases and errors. Take input from user to demonstrate the functionality of the Stack class.

**Code:**





**Output:**



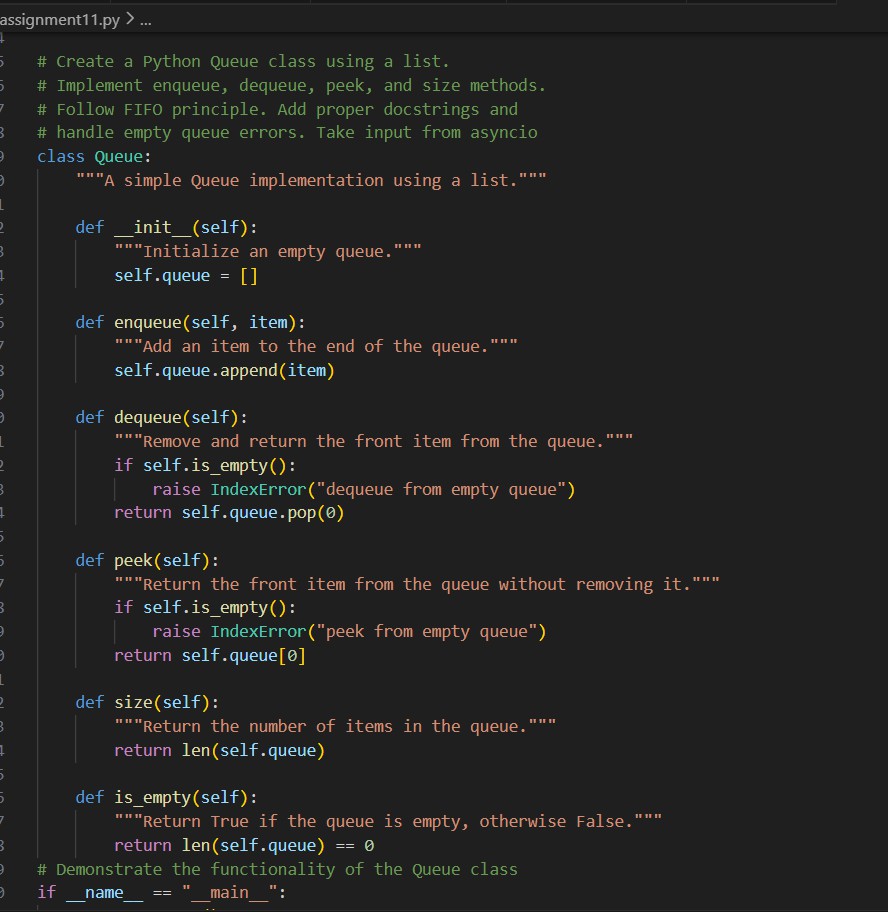
**Observation:**

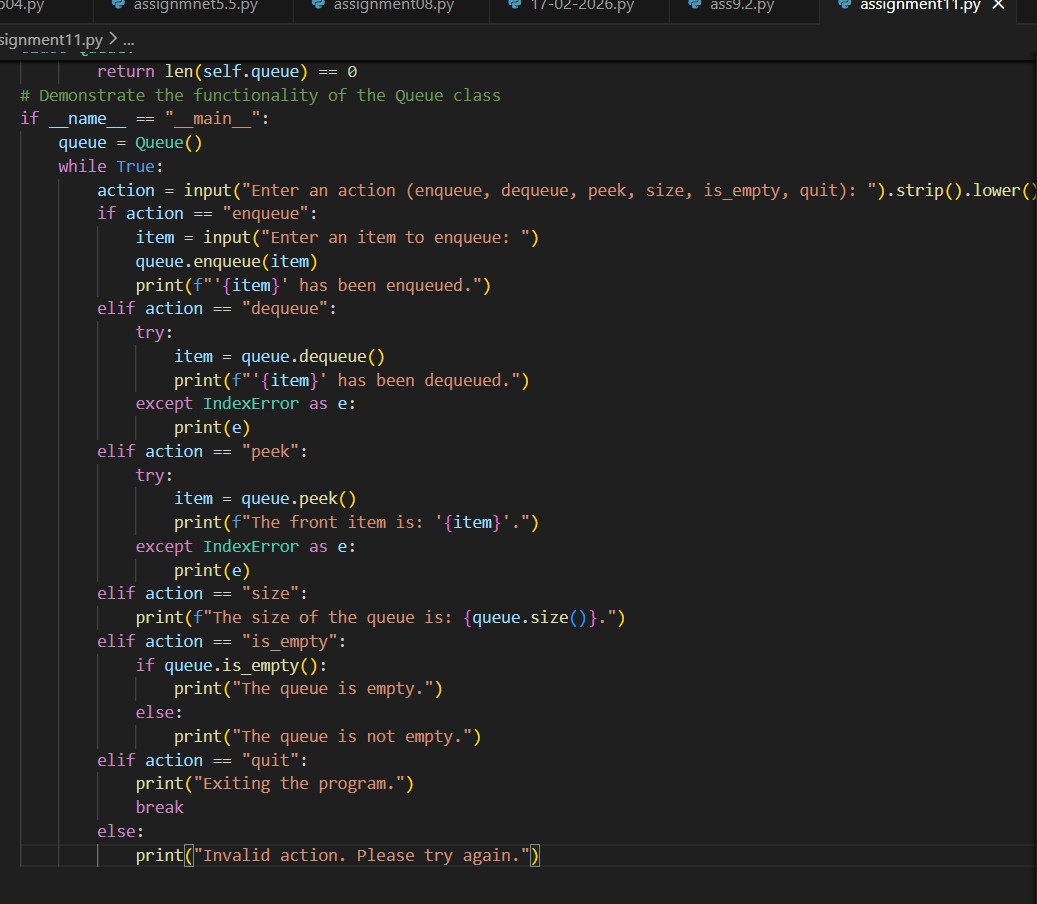
The Stack class works correctly by following the Last In, First Out (LIFO) principle, where the most recently pushed element is removed first. The push, pop, peek, and is\_empty methods perform as expected, including proper handling of edge cases like popping or peeking from an empty stack. The interactive user input successfully demonstrates the functionality and error handling of the stack implementation.

**Task Description #2 – Queue Implementation Prompt:**

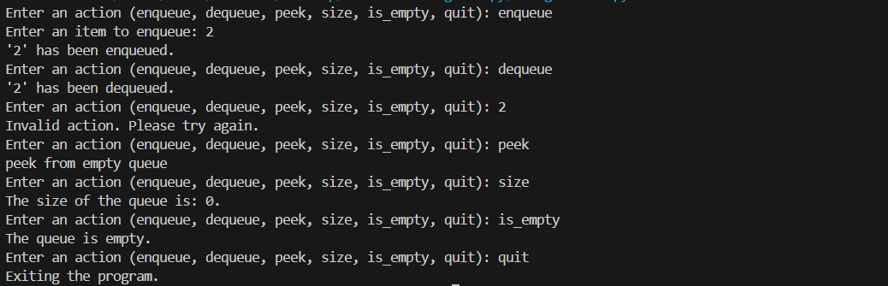
Create a Python Queue class using a list. Implement enqueue, dequeue, peek, and size methods. Follow FIFO principle. Add proper docstrings and handle empty queue errors. Take input from user to demonstrate the functionality of the Queue class.

**Code:**





**Output:**

****

**Observation:**

The Queue class correctly follows the First In, First Out (FIFO) principle, ensuring that elements are removed in the same order they were added. All methods including enqueue, dequeue, peek, and size function properly and handle edge cases like empty queue operations. The implementation effectively demonstrates queue behavior using Python lists.

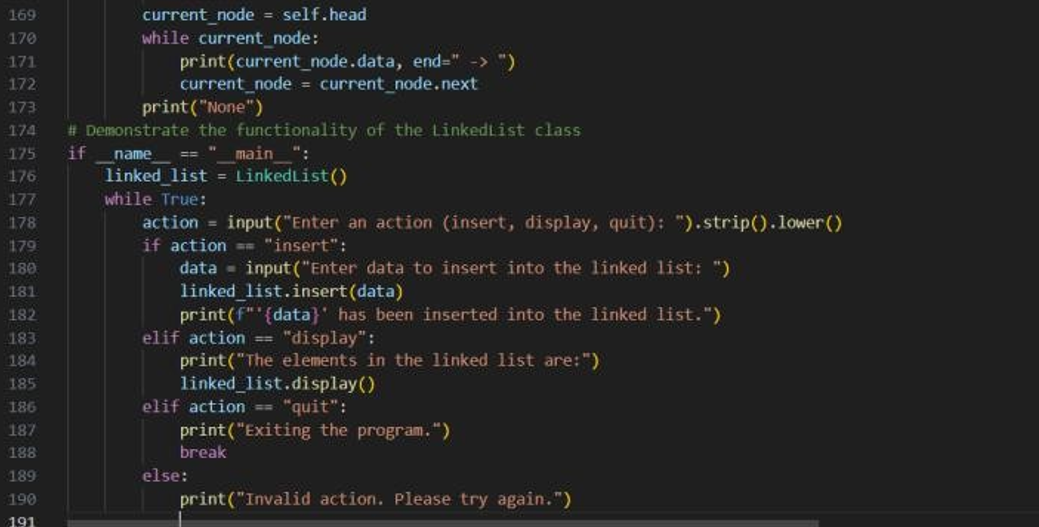
**Task Description #3 – Linked List Prompt:**

Create a Python implementation of a Singly Linked List. Define a Node class and a LinkedList class.

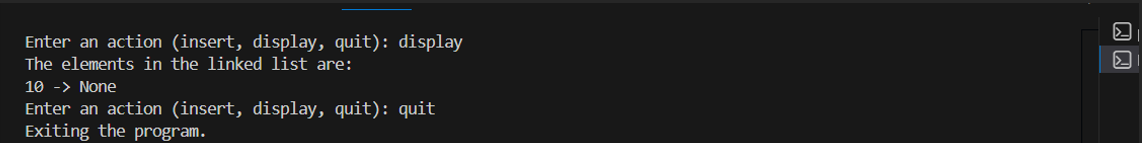
Include methods: insert(data) to add at the end and display() to print all elements. Add proper docstrings and basic error handling. Take input from user to demonstrate the functionality of the LinkedList class.

**Code:**





**Output:**



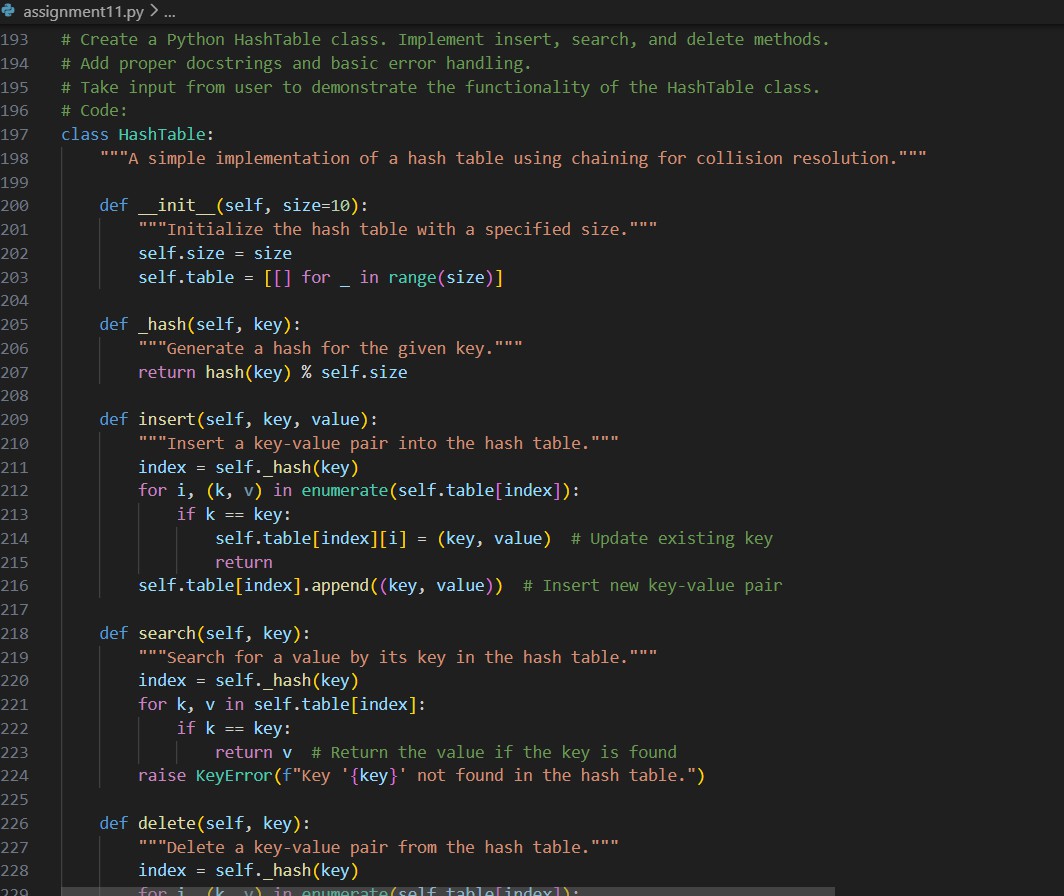
**Observation:**

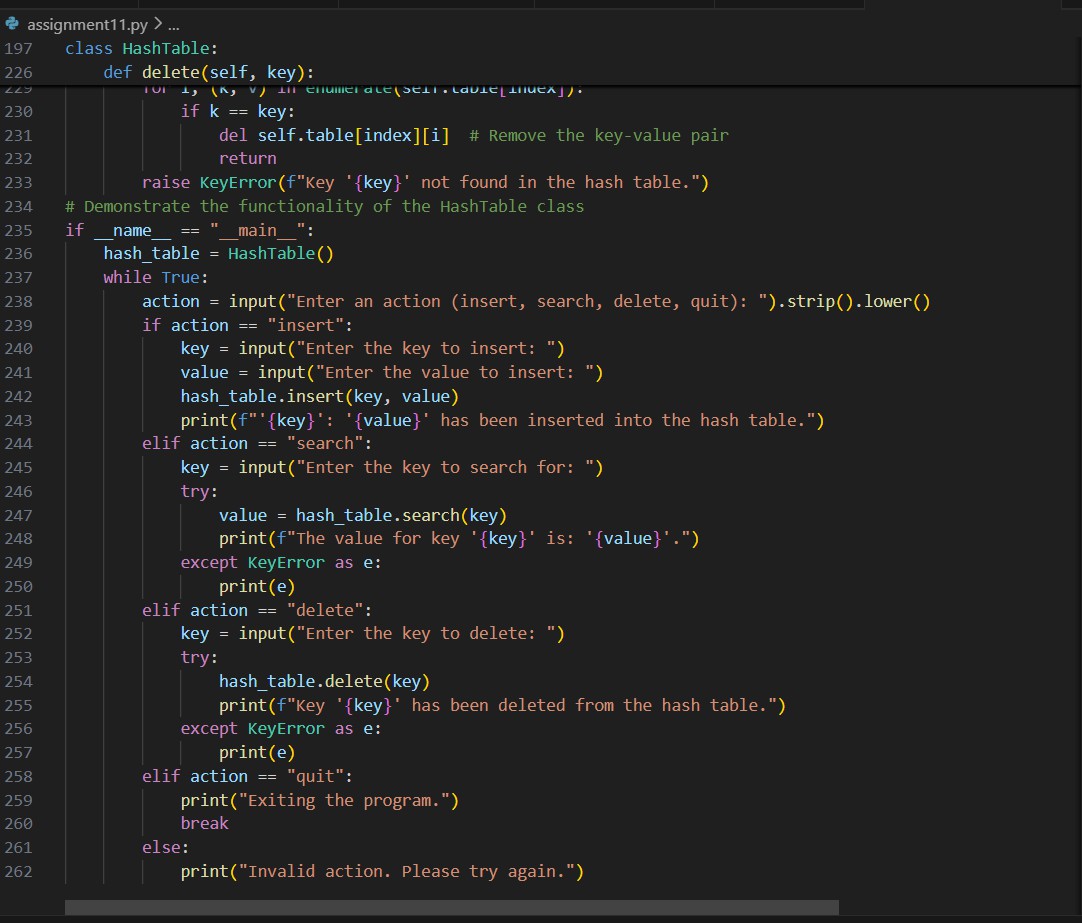
The Singly Linked List correctly stores elements in sequential order using node connections. The insert method successfully adds elements at the end of the list, and the display method prints all nodes clearly. The implementation also handles the empty list case properly without errors.

**Task Description #4 – Hash Table Prompt:**

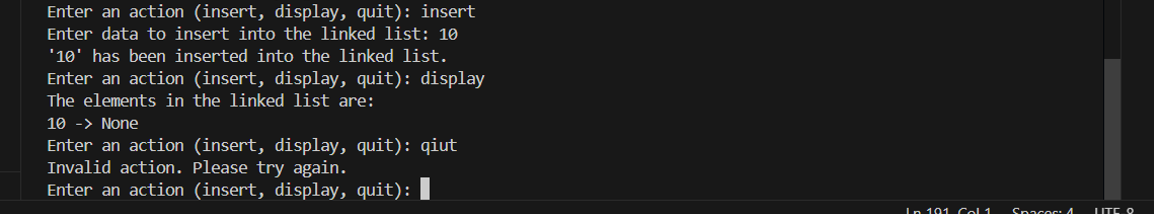
Create a Python HashTable class. Implement insert, search, and delete methods. Add proper docstrings and basic error handling. Take input from user to demonstrate the functionality of the HashTable class.

**Code:**

****



**Output:**



**Observation:**

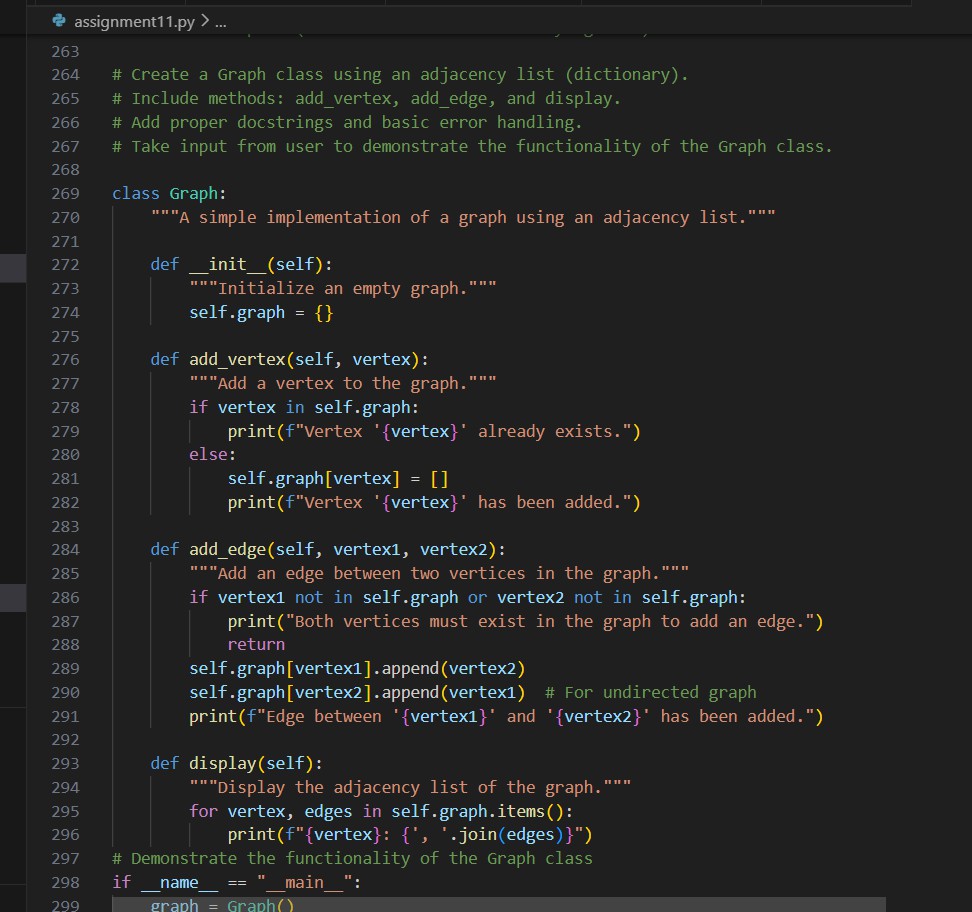
The hash table correctly stores and retrieves key-value pairs using a hash function and chaining for collision handling. The insert, search, and delete operations work efficiently even when multiple keys map to the same index. Edge cases such as deleting or searching for non-existing keys are handled properly without crashing the program.

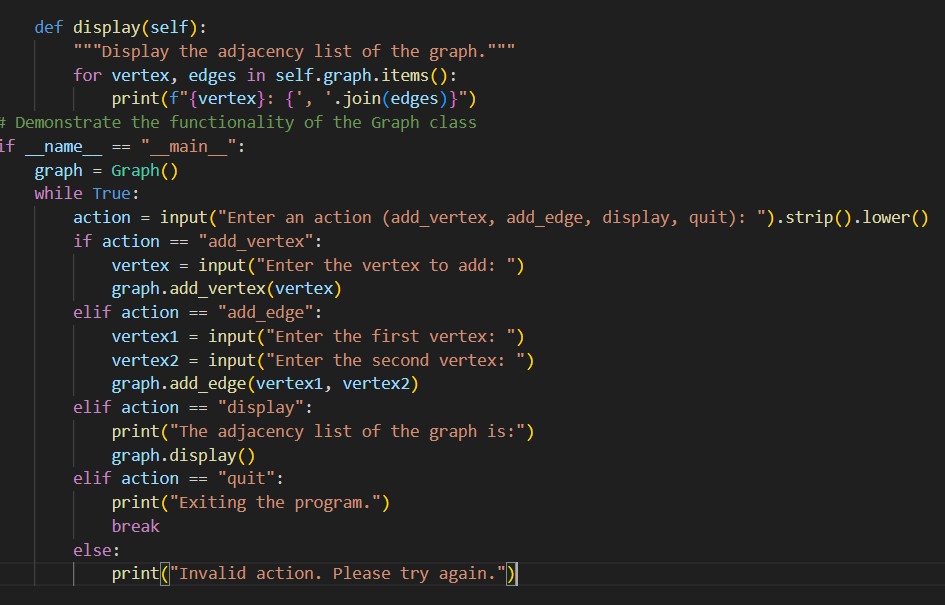
**Task Description #5 – Graph Representation Prompt:**

Create a Graph class using an adjacency list (dictionary).

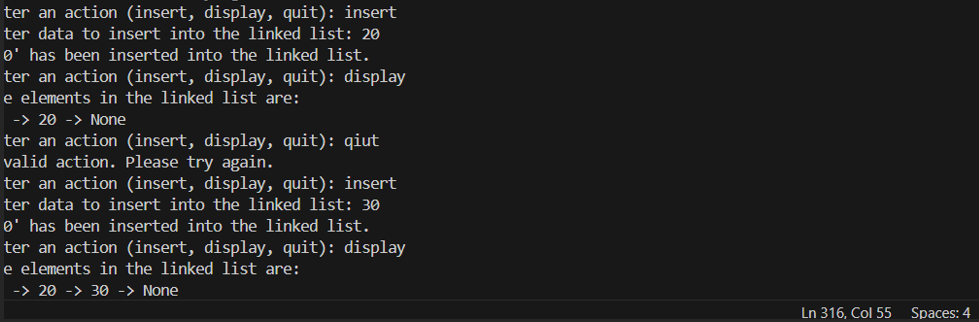
Include methods: add\_vertex, add\_edge, and display. Add proper docstrings and basic error handling. Take input from user to demonstrate the functionality of the Graph class.

**Code:**





**Output:**

****

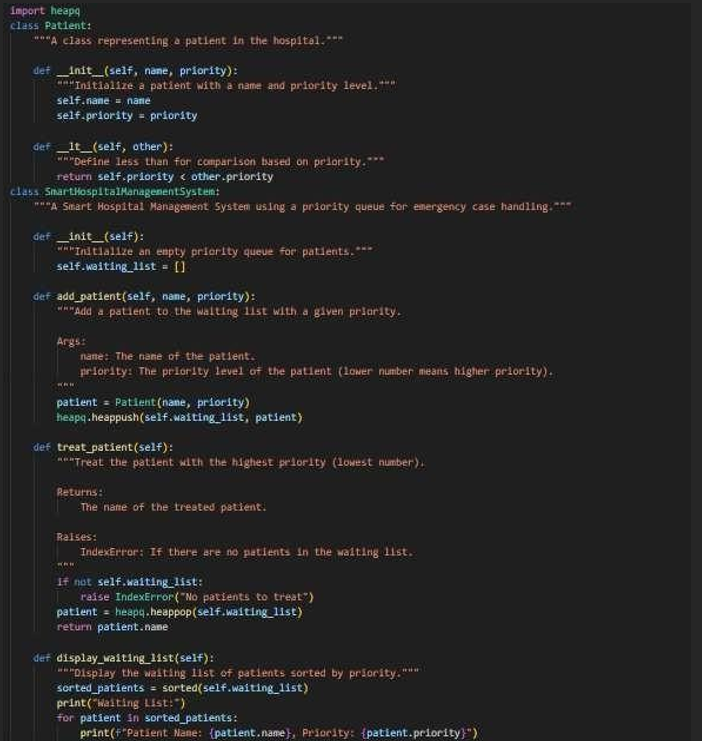
**Observation:**

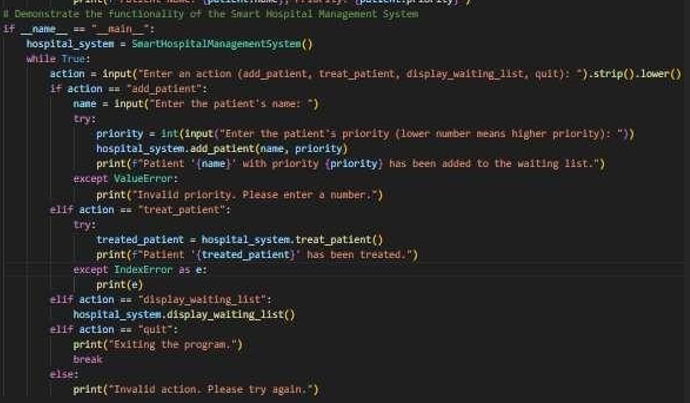
The graph implementation correctly stores vertices and edges using an adjacency list structure. The add\_vertex and add\_edge methods properly update connections between nodes in an undirected manner. The display method successfully shows all vertices along with their connected neighbors, confirming correct functionality.

**Task Description #6: Smart Hospital Management System – Data Structure Selection Prompt:**

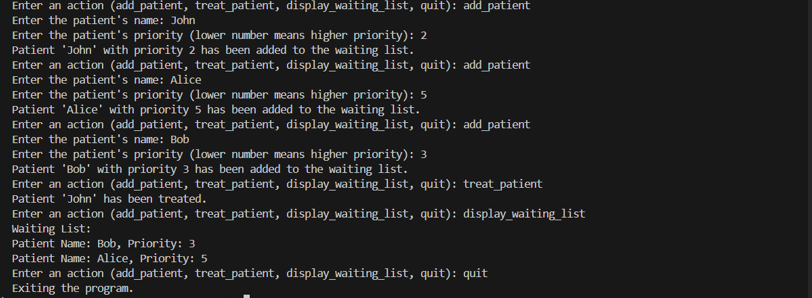
Create a Python program for Smart Hospital Management System. Implement Emergency Case Handling using a Priority Queue. Patients with higher priority (critical level) should be treated first. Include functions to add patient, treat patient, and display waiting list. Add proper docstrings, and basic error handling. Take input from user to demonstrate the functionality of the Smart Hospital Management System.

**Code:**



****

**Output:**

****

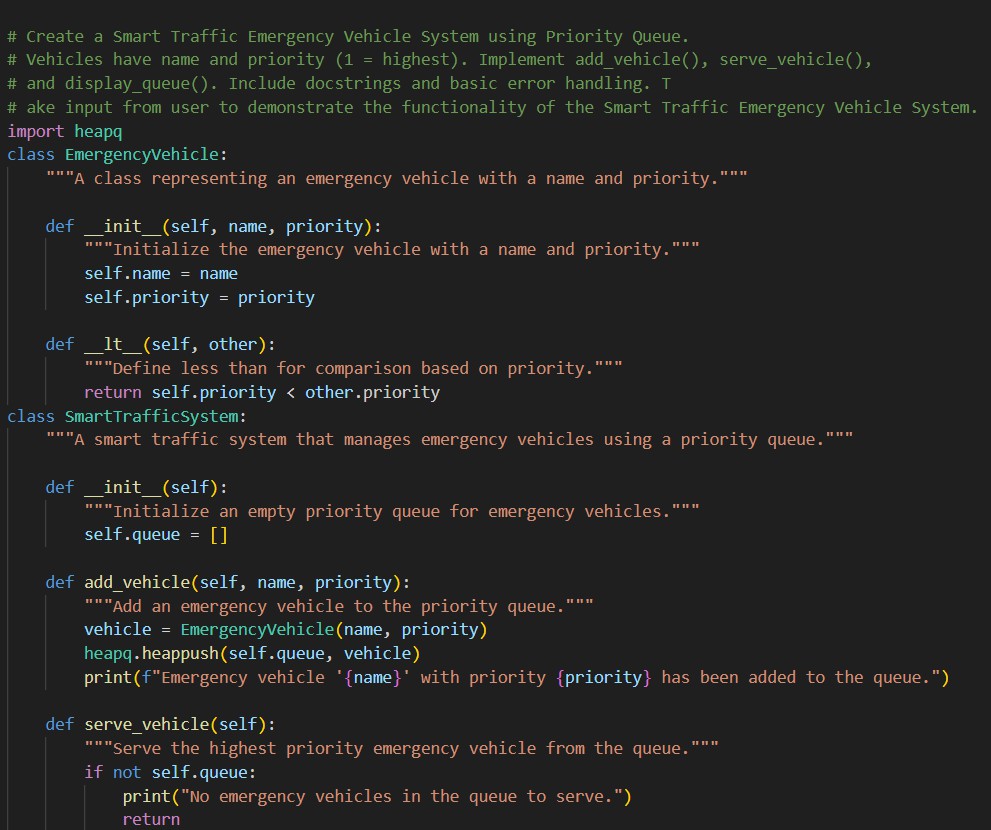
**Observation:**

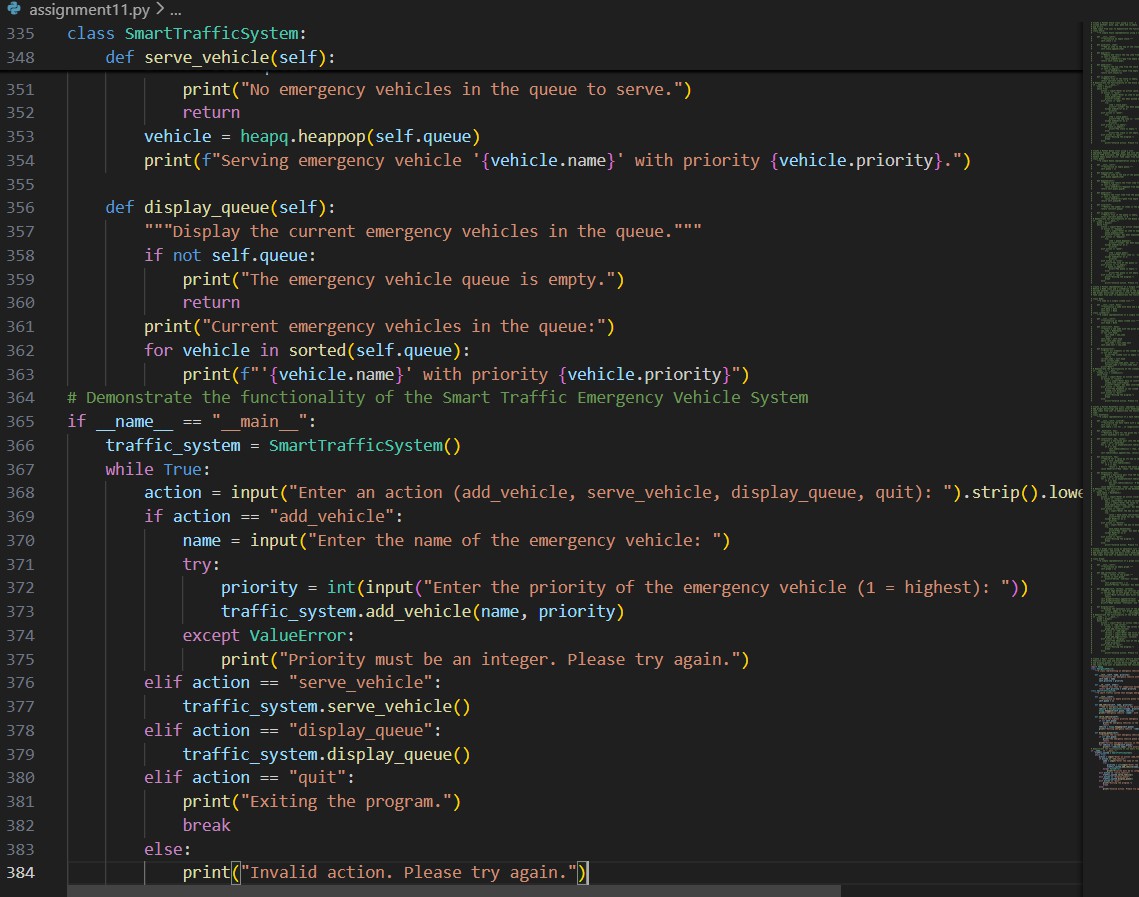
The Priority Queue ensures that patients are treated based on the severity of their condition rather than their arrival time. Patients with critical conditions are given higher priority and attended to first, which supports effective emergency management. The system also properly handles situations when no patients are waiting and maintains efficient performance during patient insertion and treatment.

**Task Description #7: Smart City Traffic Control System Prompt:**

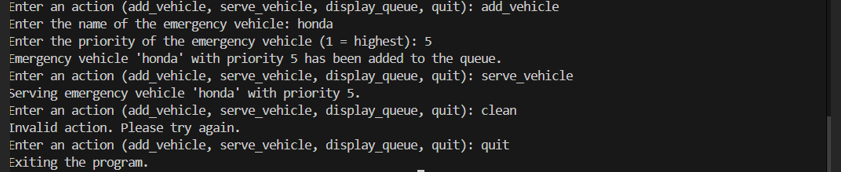
Create a Smart Traffic Emergency Vehicle System using Priority Queue. Vehicles have name and priority (1 = highest). Implement add\_vehicle(), serve\_vehicle(), and display\_queue(). Include docstrings and basic error handling. Take input from user to demonstrate the functionality of the Smart Traffic Emergency Vehicle System.

**Code:**





**Output:**



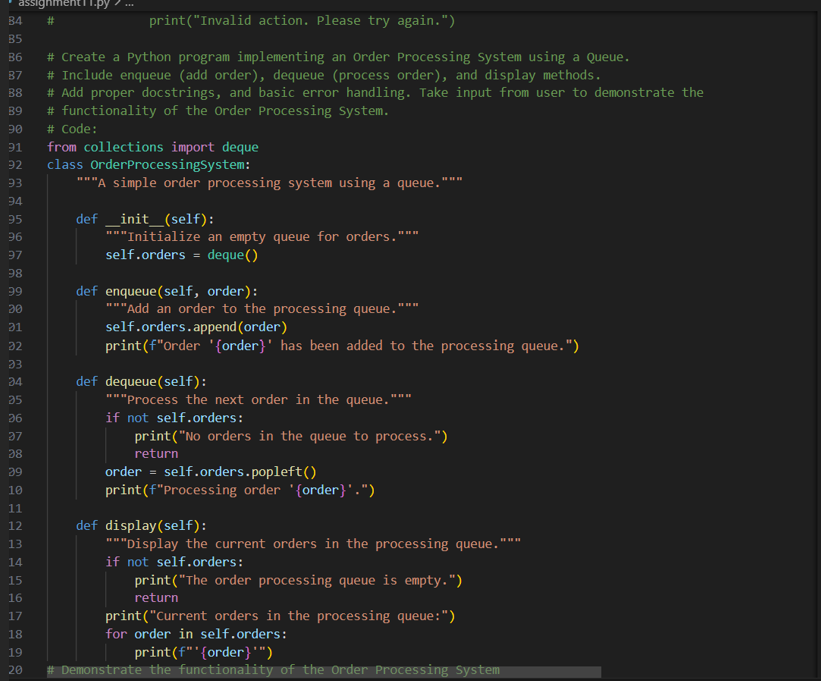
**Observation:**

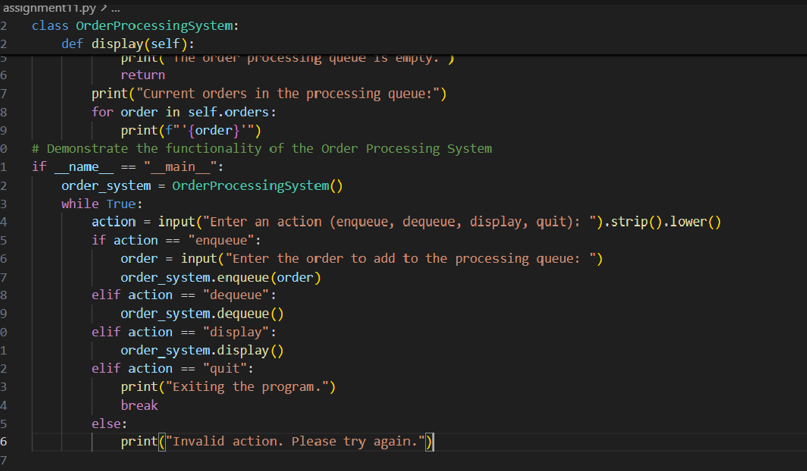
The Priority Queue ensures that emergency vehicles such as ambulances and fire trucks are served before normal vehicles regardless of arrival order. This structure was chosen because traffic management requires priority-based handling rather than simple FIFO processing. The implementation successfully demonstrates efficient insertion and removal based on priority levels.

**Task Description #8: Smart E-Commerce Platform – Data Structure Challenge Prompt:**

Create a Python program implementing an Order Processing System using a Queue. Include enqueue (add order), dequeue (process order), and display methods. Add proper docstrings, and basic error handling. Take input from user to demonstrate the functionality of the Order Processing System.

**Code:**

****



**Output:**

****

**Observation:**

The Order Processing System correctly follows the FIFO principle, ensuring fairness in handling customer orders. The Queue data structure was chosen because it processes elements in the exact order they are inserted. This makes it the most suitable and logical structure for managing sequential order execution in an e-commerce system.