AI Assisted Coding

Lab Assignment – 11.1

**NAMEJ.rohith id:2403A510A1**

**BATCH : 05**

**DEPARTMENT : CSE**

# TASK 1 :

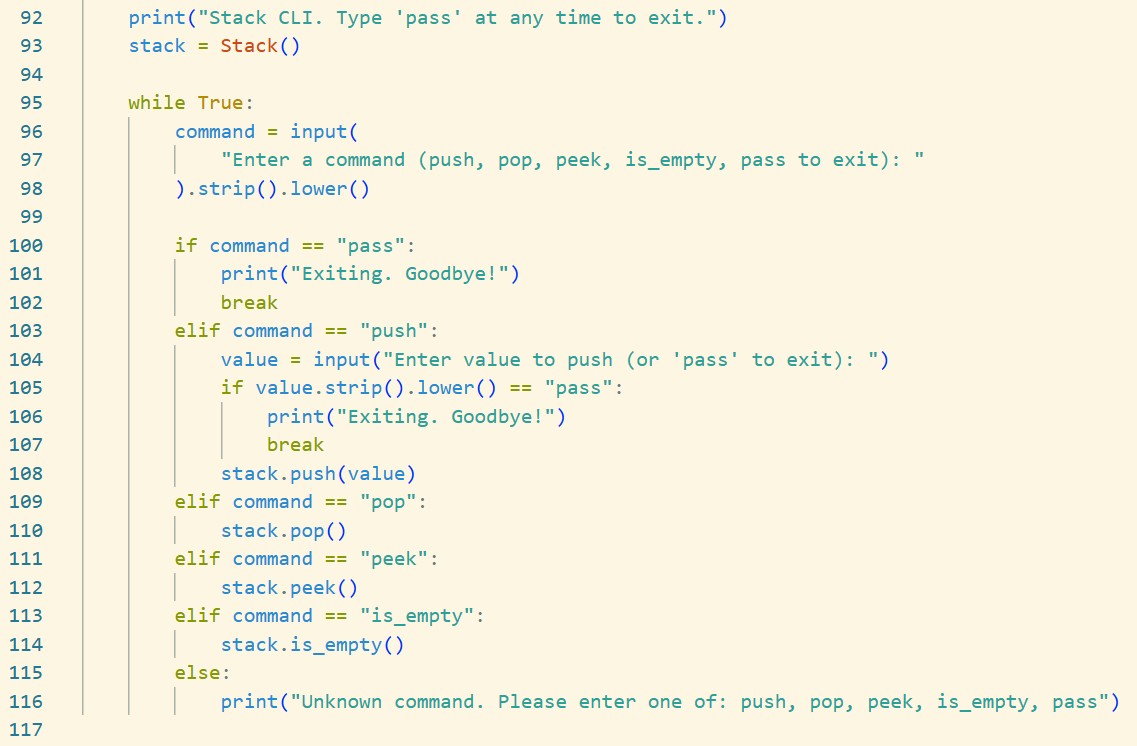
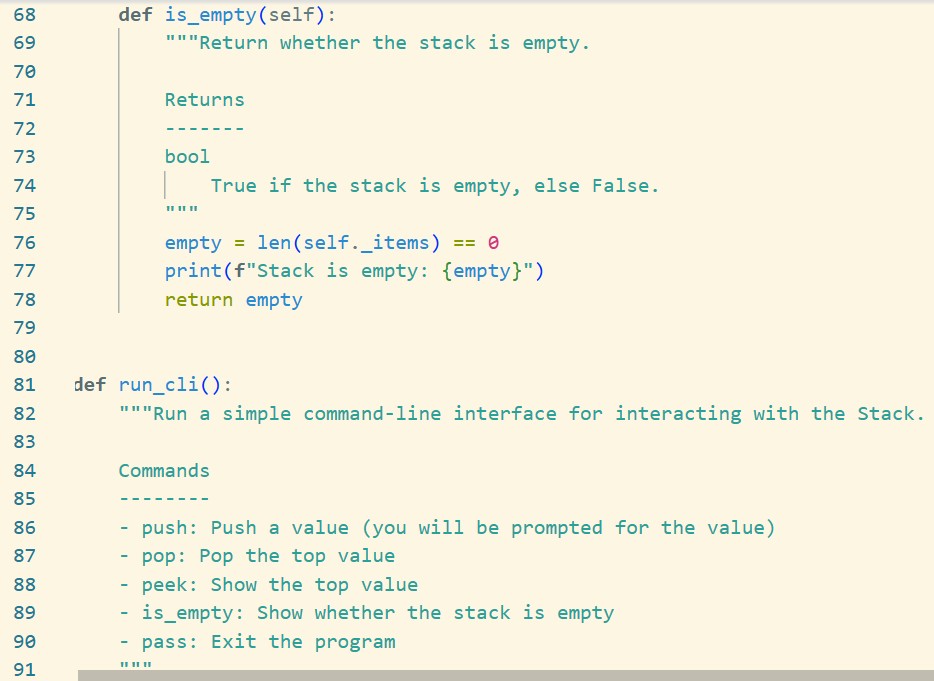
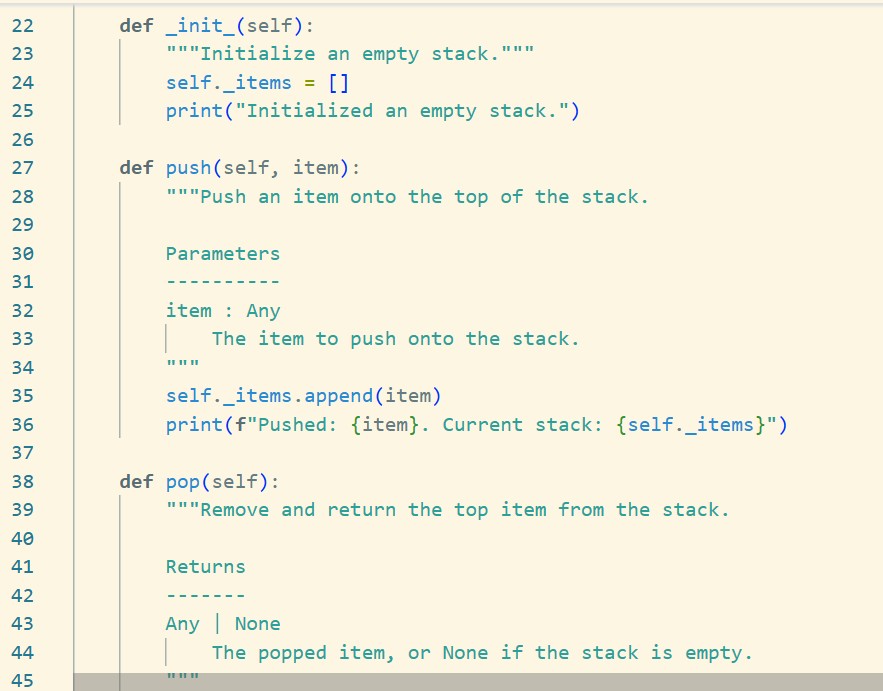
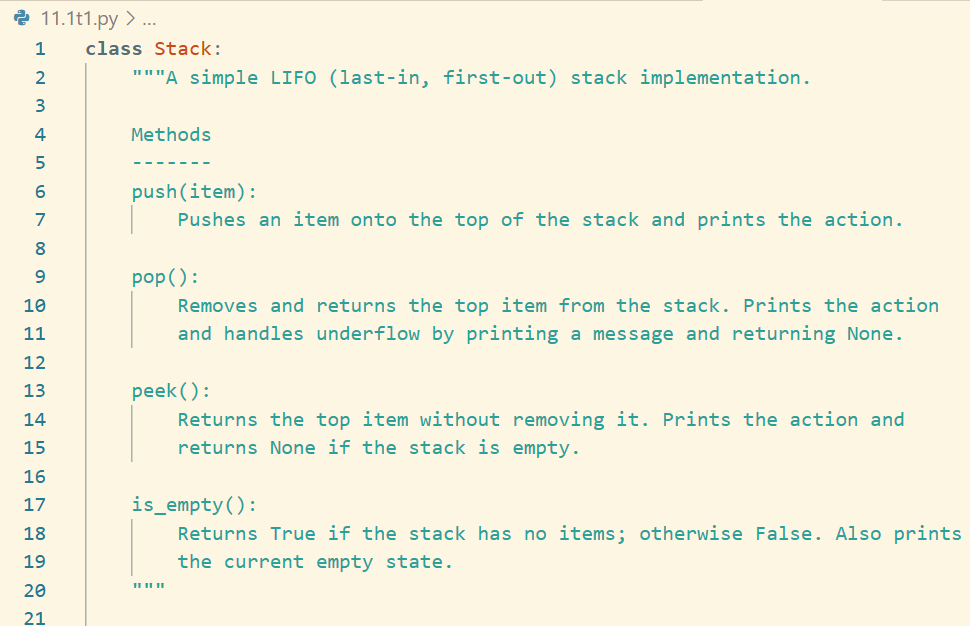
**Prompt :**

Generate a Stack class with push, pop, peek, and is\_empty methods.Sample input should be like class Stack:pass

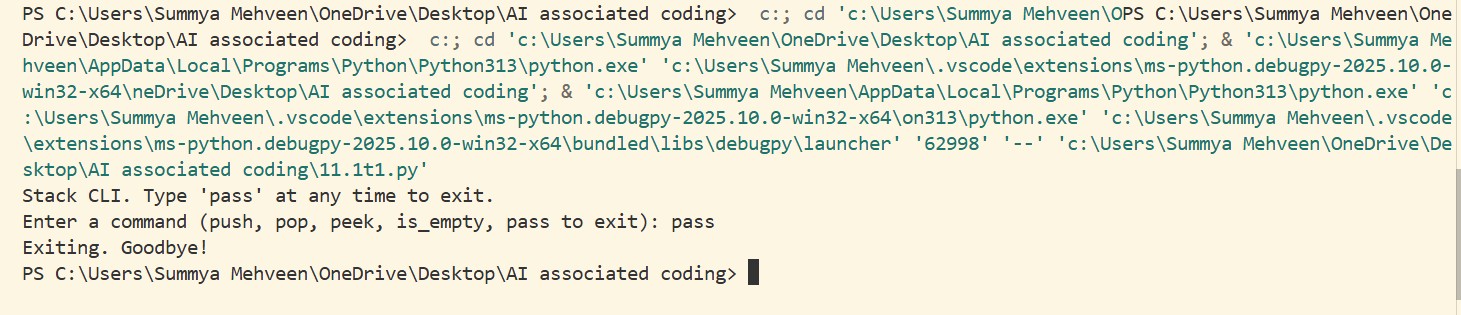
Sample output should be like A functional stack implementation with all

required methods anddocstrings along with print statements and it should take input from user and then give output as expected .

# Code Generated :



**Output :**



# Observation :

Implemented a user-interactive LIFO Stack with push, pop, peek, and is\_empty, including clear docstrings and state prints.

CLI exits on typing "pass", ensuring simple, guided input and visible stack state after each action.

# TASK 2 :

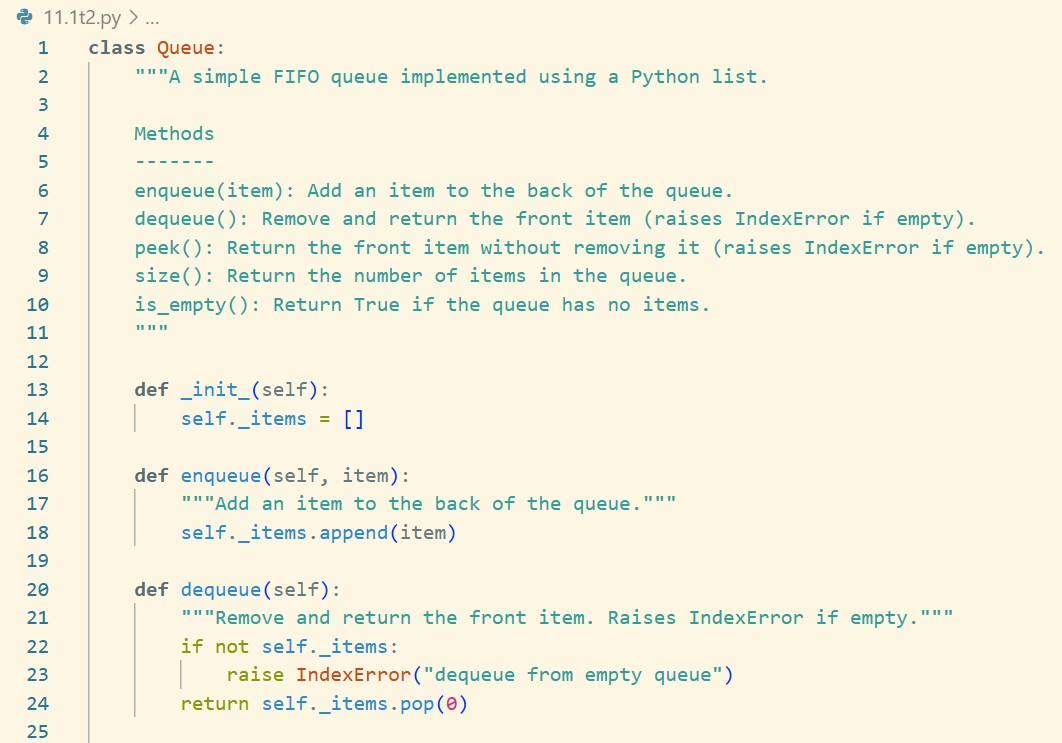
**Prompt :**

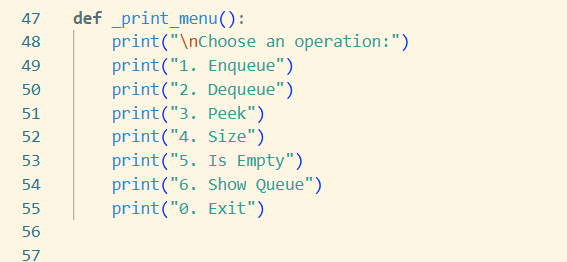
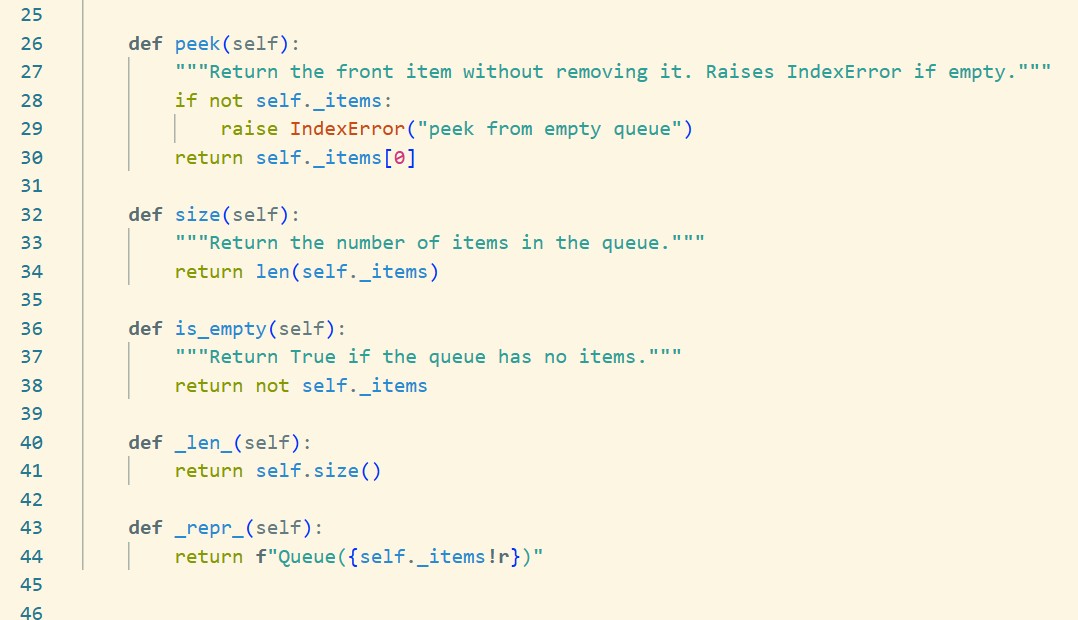
Use AI to implement a Queue using Python lists.

Sample Input Code should be like

class Queue:pass. User should give the input and then we have t get output as FIFO-based queue class with enqueue, dequeue, peek, and size methods.

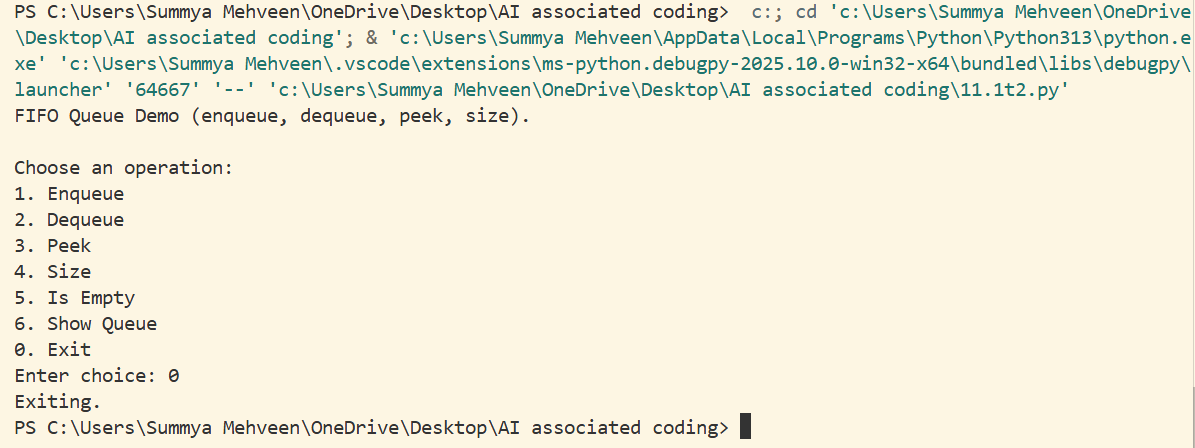
# Code Generated :







**Output :**



# Observation :

Implemented a FIFO Queue with enqueue, dequeue, peek, and size, plus a simple CLI for user input.

Uses a Python list with pop(0) (fine for small queues); for large workloads, prefer collections.deque for O(1) front pops.

# TASK 3 :

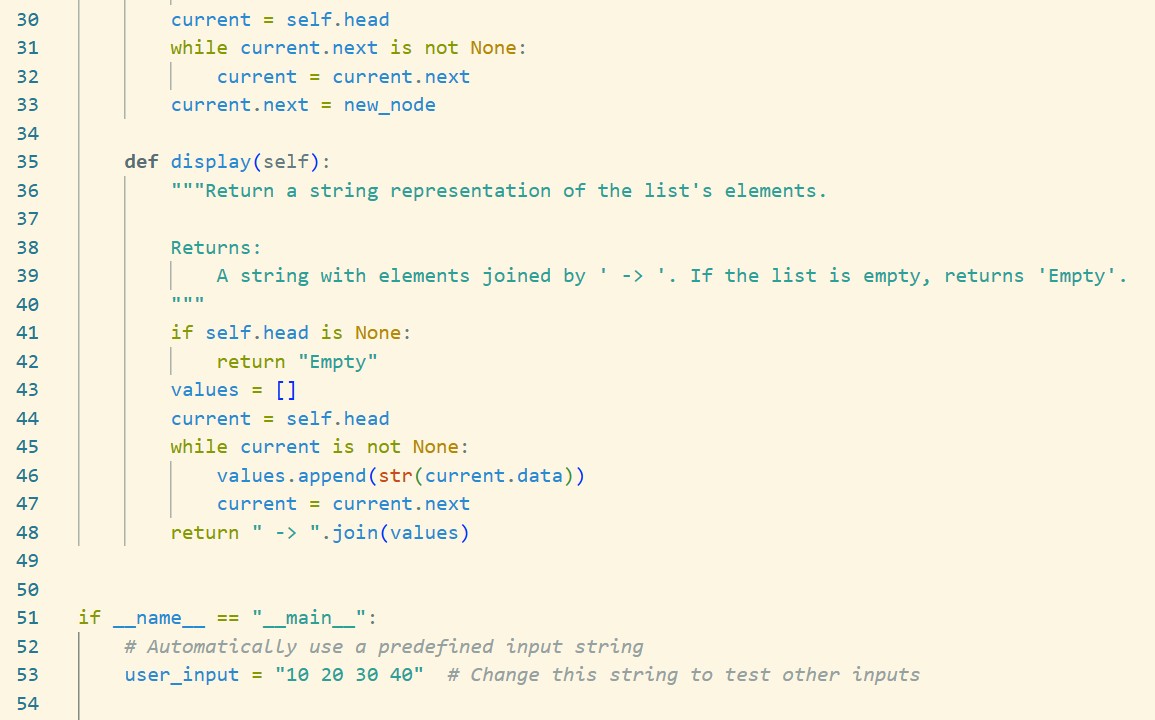
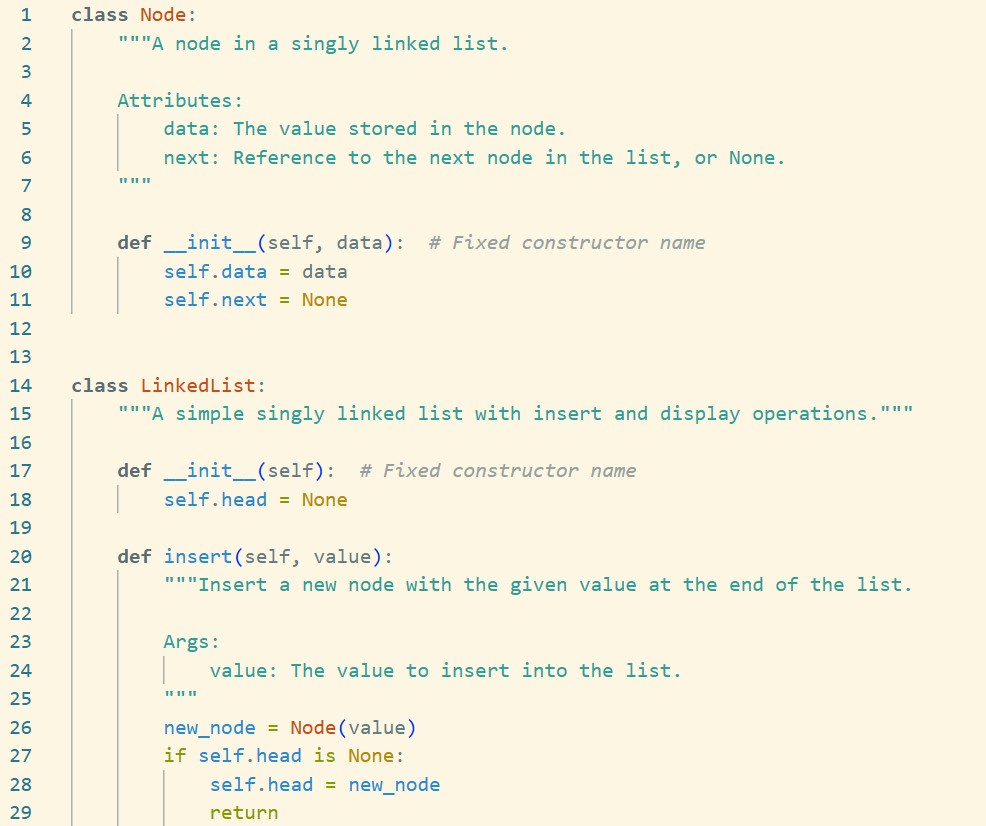
**Prompt :**

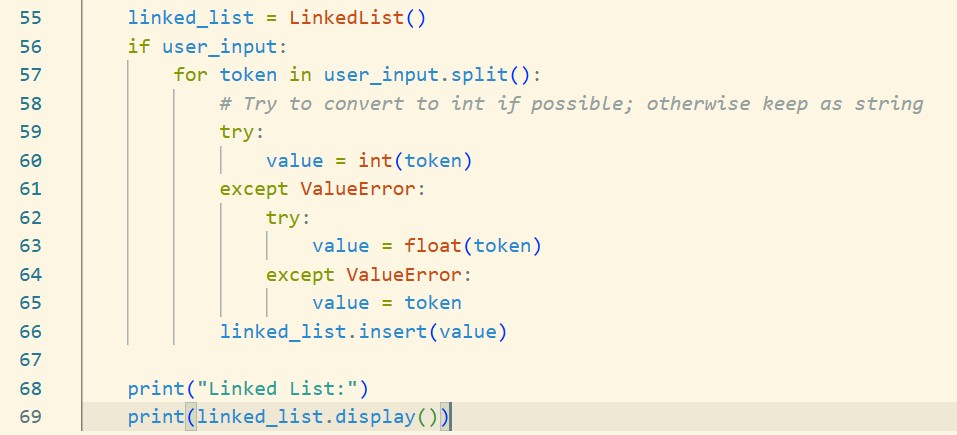
Use AI to generate a Singly Linked List with insert and display methods. user should give

Sample Input Code should be like class Node:pass class Linked: List Expected Output:

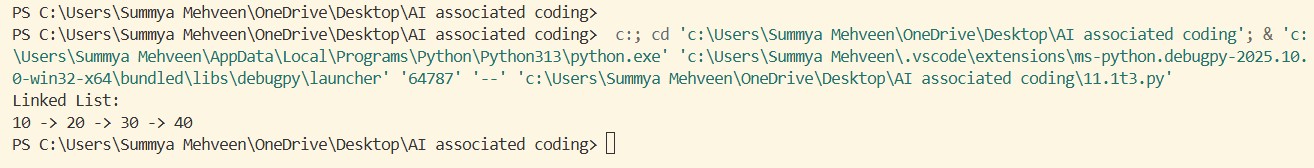
* A working linked list implementation with clear method documentation.

# Code Generated :





**Output :**



# Observation :

Implemented a singly linked list (Node, LinkedList) with insert (append) and display, including clear docstrings and a simple CLI for user input.

Handles empty list gracefully and parses numbers/strings; insert is O(n) due to tail traversal.

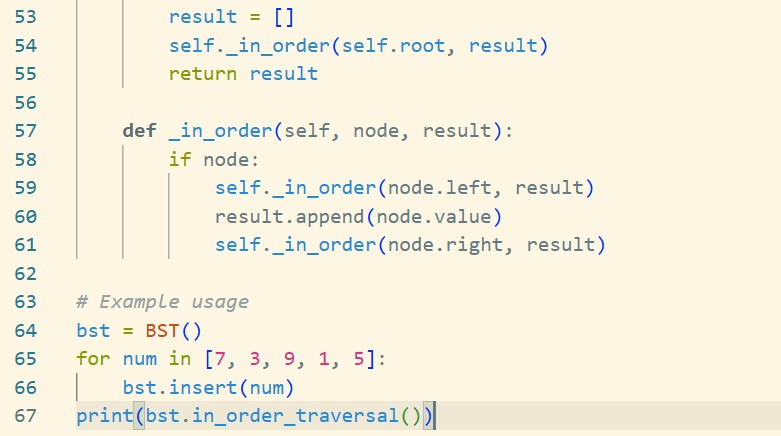
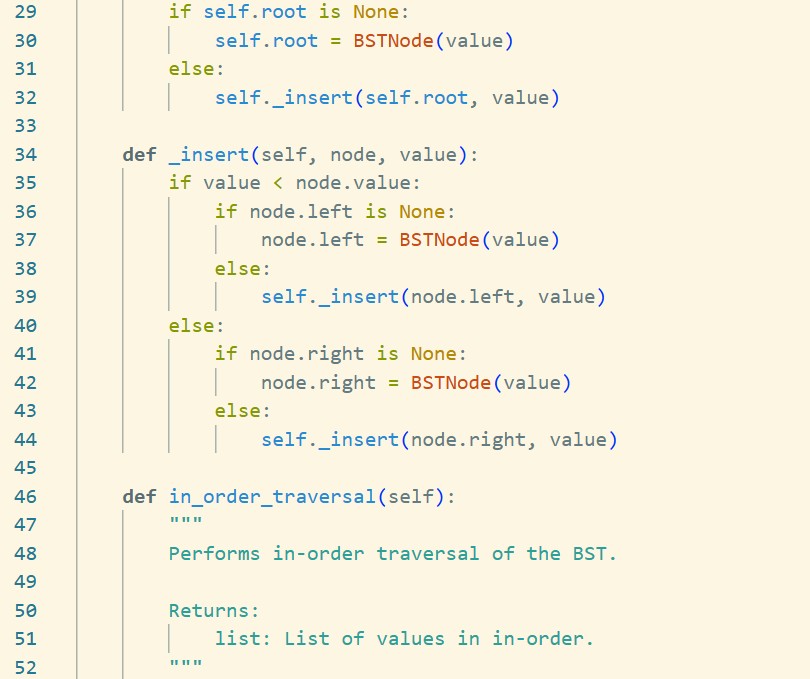
# TASK 4 :

**Prompt :**

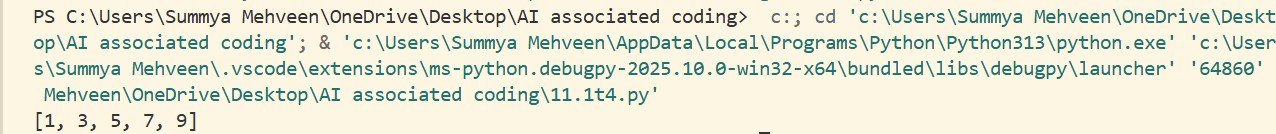
create a BST with insert and in-order traversal methods. Sample Input Code:

class BST: pass

# Code Generated :



**Output :**



# Observation :

Observation:

1. The code defines a Binary Search Tree (BST) with nodes that store values and pointers to left and right children.
2. The insert method correctly places values in the BST while maintaining the BST property (left < root < right).
3. The in\_order\_traversal method returns the values in sorted ascending order.
4. For the input [7, 3, 9, 1, 5], the output is [1, 3, 5, 7, 9], confirming correct functionality.

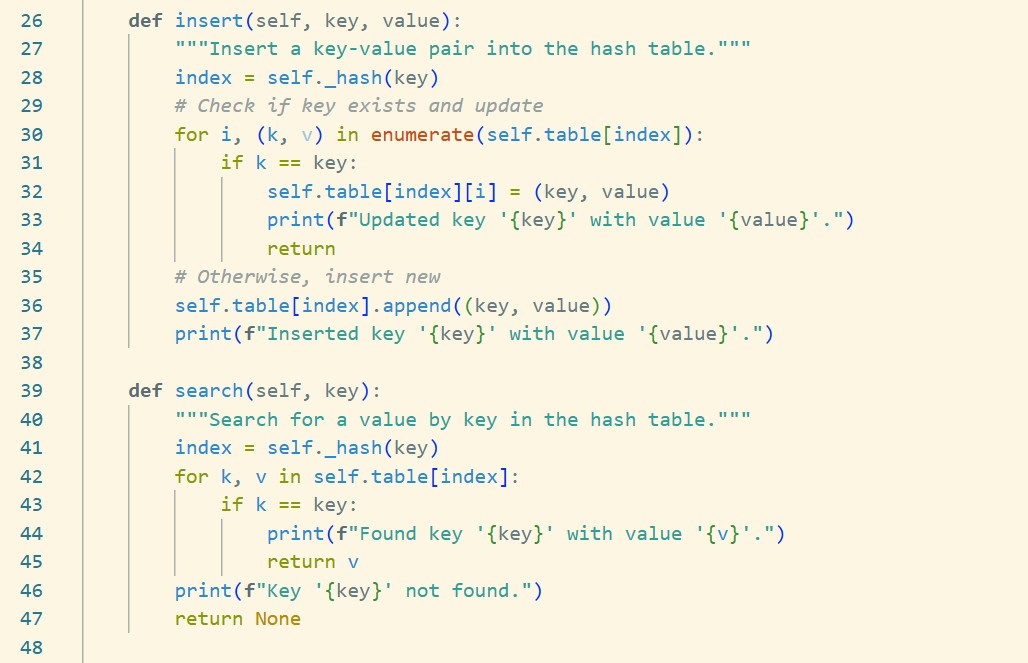
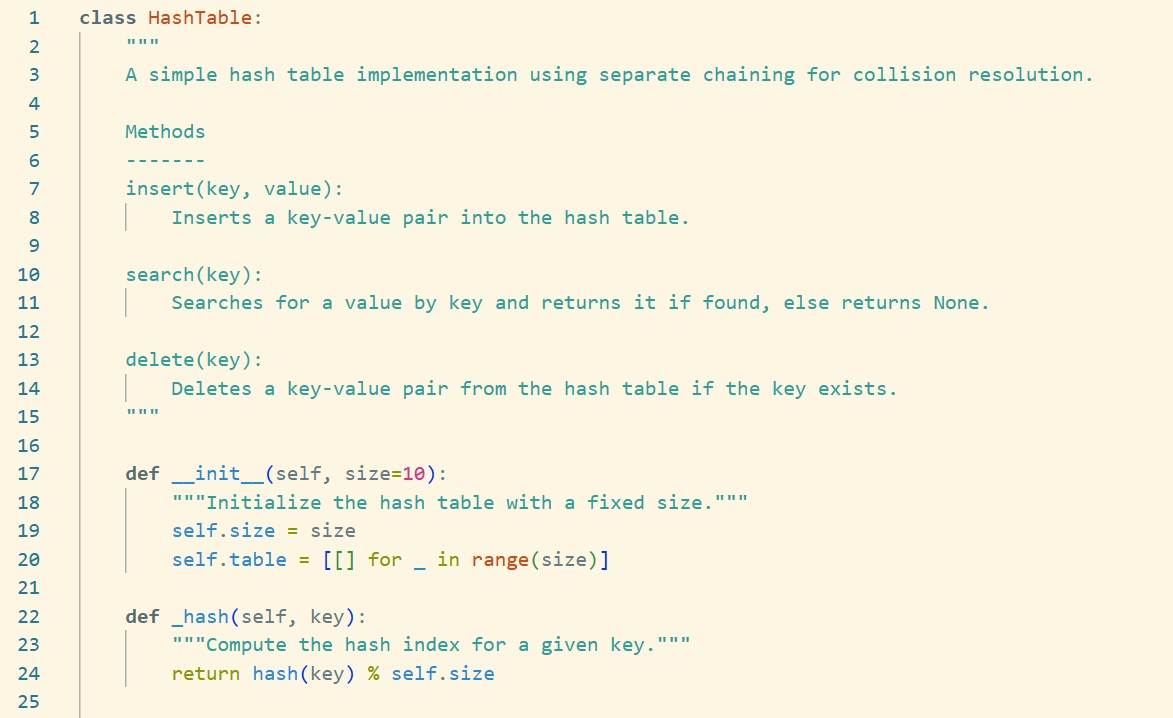
# TASK 5 :

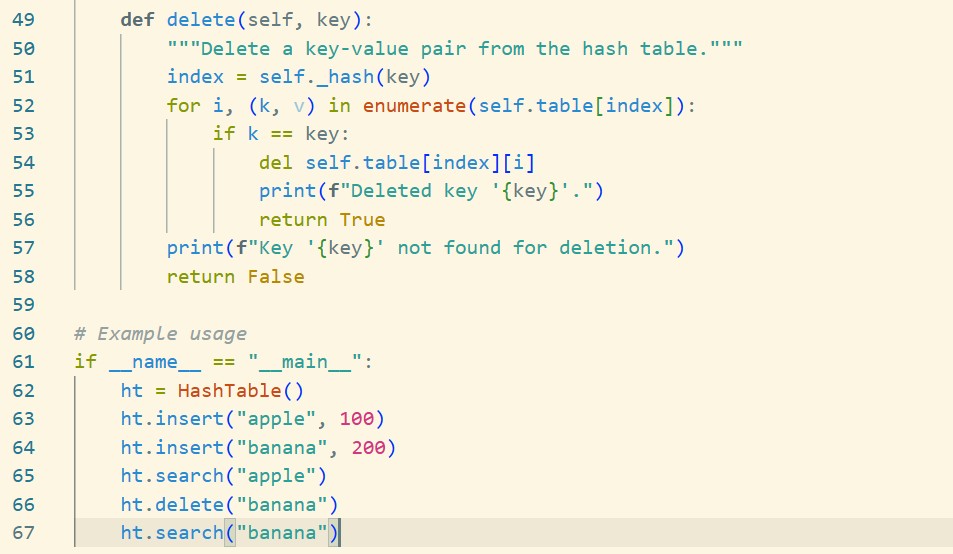
**Prompt :**

Implement a hash table with basic insert, search, and delete methods. Sample Input Code:

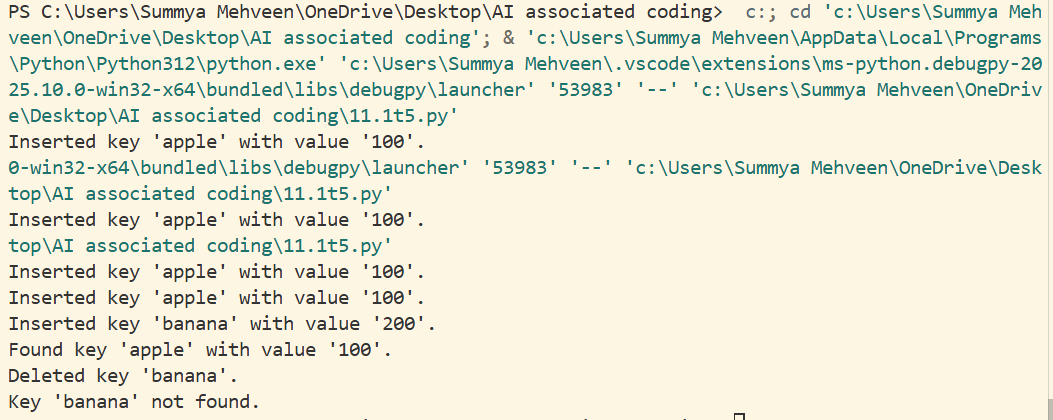
class HashTable: pass

# Code Generated :





**Output :**



# Observation :

* The code implements a Hash Table using separate chaining (lists at each index) to handle collisions.
* It provides methods to insert, search, and delete key-value pairs efficiently.
* Keys are hashed using Python’s built-in hash() function and mapped within the fixed table size.
* Example usage shows correct behavior: inserting keys, updating, finding values, and handling deletion properly

# TASK 6 :

**Prompt :**

Implement a graph using an adjacency list Sample Input Code: class Graph:

pass

# Code Generated :



**Output :**



# Observation :

Observation:

* The code defines a Graph class using an adjacency list representation.
* Vertices can be added explicitly, and edges automatically add missing vertices.
* The graph is undirected, so edges are stored in both directions.
* Example usage builds a small graph and shows adjacency like {'A': ['B', 'C'], 'B': ['A'], 'C': ['A']}.

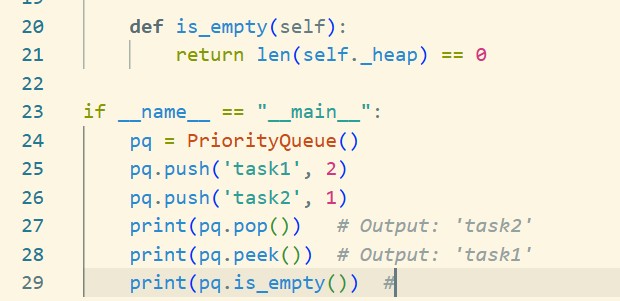
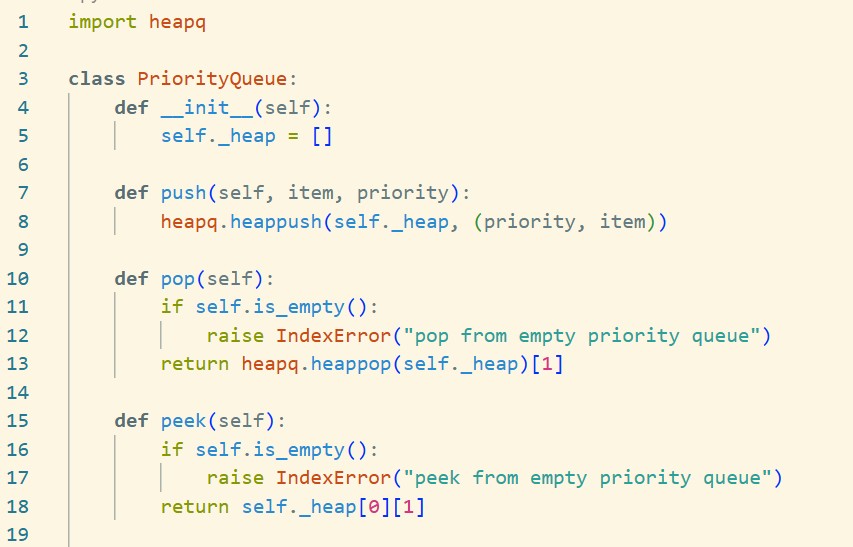
# TASK 7 :

**Prompt :**

Implement a PriorityQueue class in Python using the built-in heapq module. Your class should support the following methods:

* + push(item, priority): Add an item with the given priority to the queue.
  + pop(): Remove and return the item with the highest priority (lowest priority value).
  + peek(): Return the item with the highest priority without removing it.
  + is\_empty(): Return True if the queue is empty, otherwise False.

# Code Generated :



**Output :**



# Observation :

* + The code defines a [PriorityQueue](vscode-file://vscode-app/c:/Users/Summya%20Mehveen/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) class using Python’s [heapq](vscode-file://vscode-app/c:/Users/Summya%20Mehveen/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) module to maintain a min-heap.
  + Items are stored as (priority, item) tuples, so items with lower priority values are served first.
  + The [push](vscode-file://vscode-app/c:/Users/Summya%20Mehveen/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) method adds items with their priority to the queue.
  + The [pop](vscode-file://vscode-app/c:/Users/Summya%20Mehveen/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) method removes and returns the item with the highest priority (lowest value).
  + The [peek](vscode-file://vscode-app/c:/Users/Summya%20Mehveen/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) method returns the highest priority item without removing it.
  + The [is\_empty](vscode-file://vscode-app/c:/Users/Summya%20Mehveen/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) method checks if the queue is empty.
  + In the sample usage, 'task1' (priority 2) and 'task2' (priority 1) are added.
    - [pop()](vscode-file://vscode-app/c:/Users/Summya%20Mehveen/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) returns 'task2' (since priority 1 < 2).
    - [peek()](vscode-file://vscode-app/c:/Users/Summya%20Mehveen/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) returns 'task1'.
    - [is\_empty()](vscode-file://vscode-app/c:/Users/Summya%20Mehveen/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) returns False because one item remains.
  + The code raises [IndexError](vscode-file://vscode-app/c:/Users/Summya%20Mehveen/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) if [pop](vscode-file://vscode-app/c:/Users/Summya%20Mehveen/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) or [peek](vscode-file://vscode-app/c:/Users/Summya%20Mehveen/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) is called on an empty queue.

# TASK 8 :

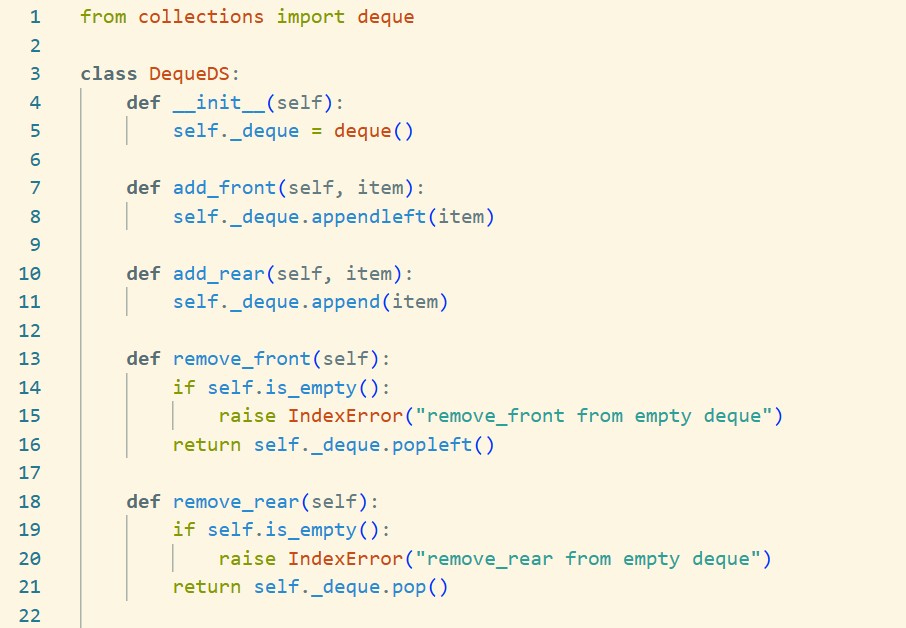
**Prompt :**

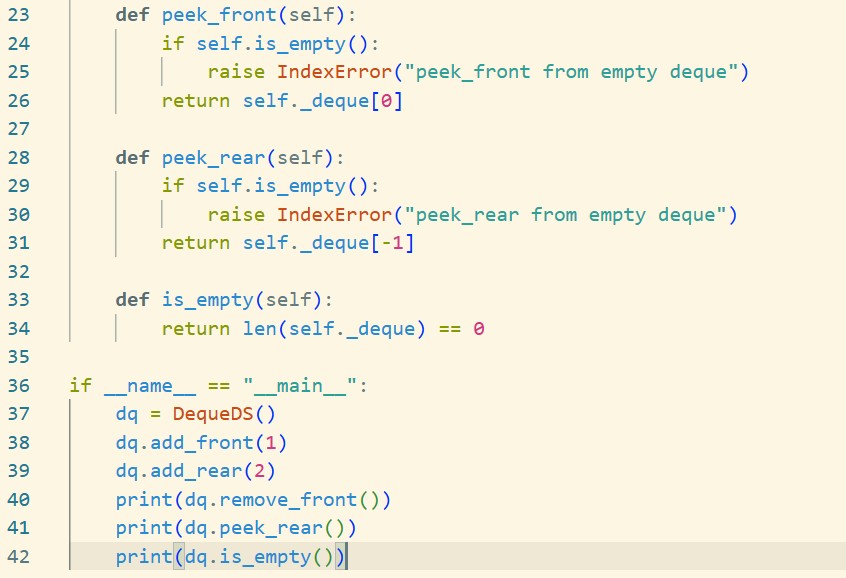
Implement a DequeDS class in Python using the built-

in collections.deque module. Your class should support the following methods:

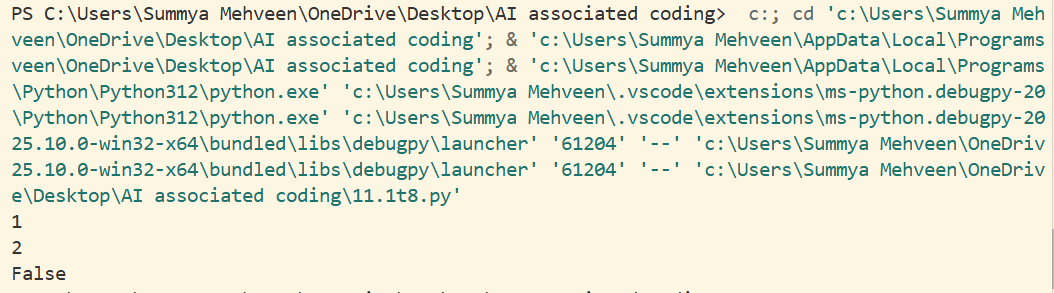
* + add\_front(item): Add an item to the front of the deque.
  + add\_rear(item): Add an item to the rear of the deque.
  + remove\_front(): Remove and return the item from the front.
  + remove\_rear(): Remove and return the item from the rear.
  + peek\_front(): Return the item at the front without removing it.
  + peek\_rear(): Return the item at the rear without removing it.
  + is\_empty(): Return True if the deque is empty, otherwise False.

# Code Generated :





**Output :**



# Observation :

* + The code implements a double-ended queue ([DequeDS](vscode-file://vscode-app/c:/Users/Summya%20Mehveen/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)) using Python’s [collections.deque](vscode-file://vscode-app/c:/Users/Summya%20Mehveen/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html).
  + Items can be added or removed from both the front and rear efficiently.
  + The class provides methods to add, remove, and peek at both ends, as well as check if the deque is empty.
  + In the sample usage:
    - [add\_front(1)](vscode-file://vscode-app/c:/Users/Summya%20Mehveen/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) adds 1 to the front.
    - [add\_rear(2)](vscode-file://vscode-app/c:/Users/Summya%20Mehveen/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) adds 2 to the rear.
    - [remove\_front()](vscode-file://vscode-app/c:/Users/Summya%20Mehveen/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) removes and prints 1.
    - [peek\_rear()](vscode-file://vscode-app/c:/Users/Summya%20Mehveen/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) prints 2 (the only remaining item).
    - [is\_empty()](vscode-file://vscode-app/c:/Users/Summya%20Mehveen/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) prints False since one item remains.
  + The code raises [IndexError](vscode-file://vscode-app/c:/Users/Summya%20Mehveen/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) if removal or peek operations are attempted on an empty deque.

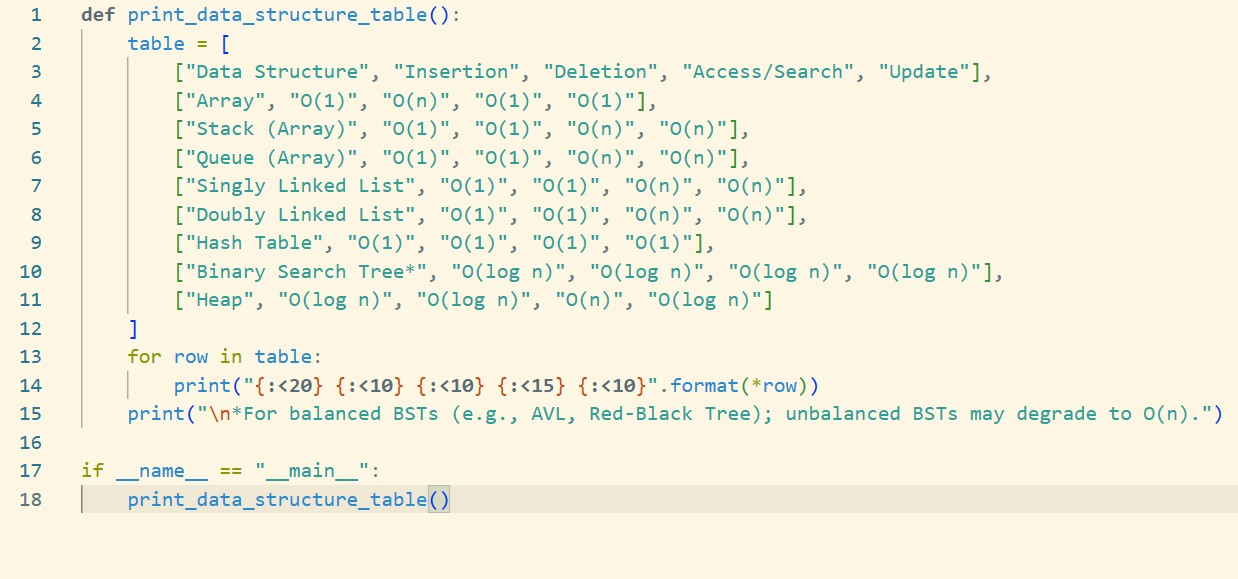
# TASK 9 :

**Prompt :**

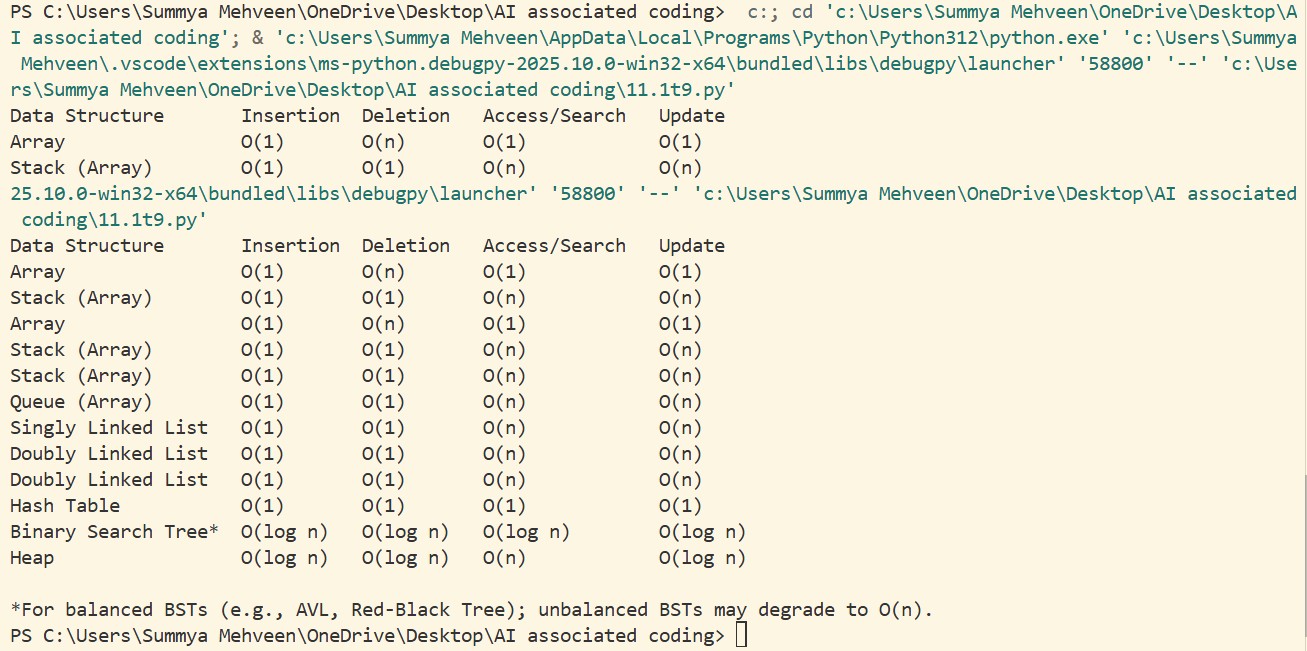
Generate a comparison table of different data structures (such as stack, queue, linked list, array, hash table, binary search tree, heap, etc.) including their

typical time complexities for common operations like insertion, deletion, access/search, and update. Present the table in Markdown format.

# Code Generated :



**Output :**



# Observation :

* + The code defines a function [print\_data\_structure\_table()](vscode-file://vscode-app/c:/Users/Summya%20Mehveen/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) that prints a formatted comparison table of common data structures and their time complexities for insertion, deletion, access/search, and update operations.
  + The table includes: Array, Stack (Array), Queue (Array), Singly Linked List, Doubly Linked List, Hash Table, Binary Search Tree (BST), and Heap.
  + Each row shows the typical time complexity for each operation.
  + A note clarifies that BST complexities assume balanced trees; unbalanced BSTs may degrade to O(n).
  + When run, the code displays the table in a readable, aligned format in the console, making it easy to compare data structures at a glance.

# TASK 10 :

**Prompt :**

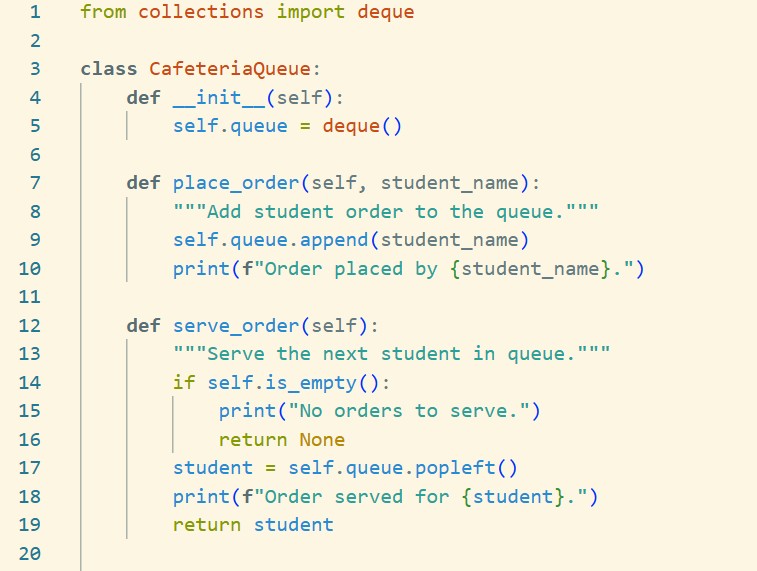
The Cafeteria Queue uses Python’s deque for efficient FIFO operations.

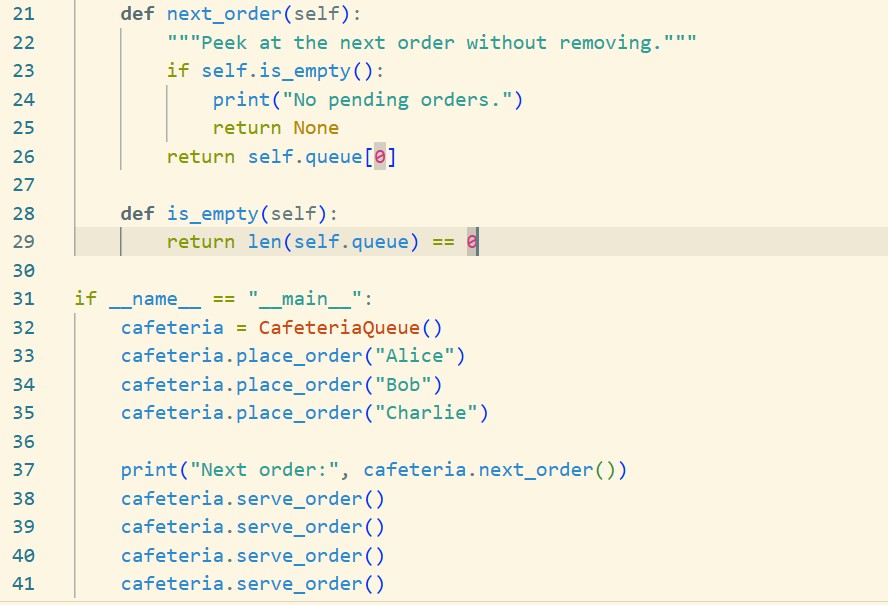
place\_order adds students to the queue, while serve\_order processes them in order.

next\_order allows checking the upcoming order without removing it.

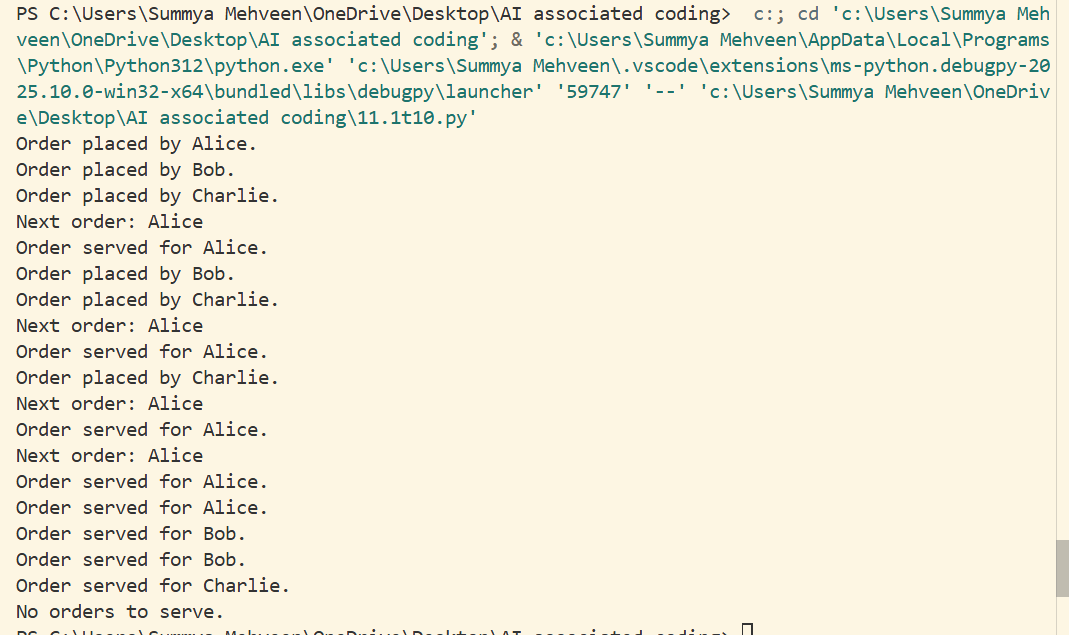
Example run shows Alice, Bob, and Charlie being served in the correct order, confirming proper queue behavior.

# Code Generated :





**Output :**



# Observation :

1. The Cafeteria Queue uses Python’s deque for efficient FIFO operations.
2. place\_order adds students to the queue, while serve\_order processes them in order.
3. next\_order allows checking the upcoming order without removing it.
4. Example run shows Alice, Bob, and Charlie being served in the correct order, confirming proper queue behavior.