AI Assisted Coding

Lab Assignment – 11.1

NAME : Neha Shazneen

ROLL NO : 2403A510A1

BATCH : 05

DEPARTMENT : CSE

TASK 1 :

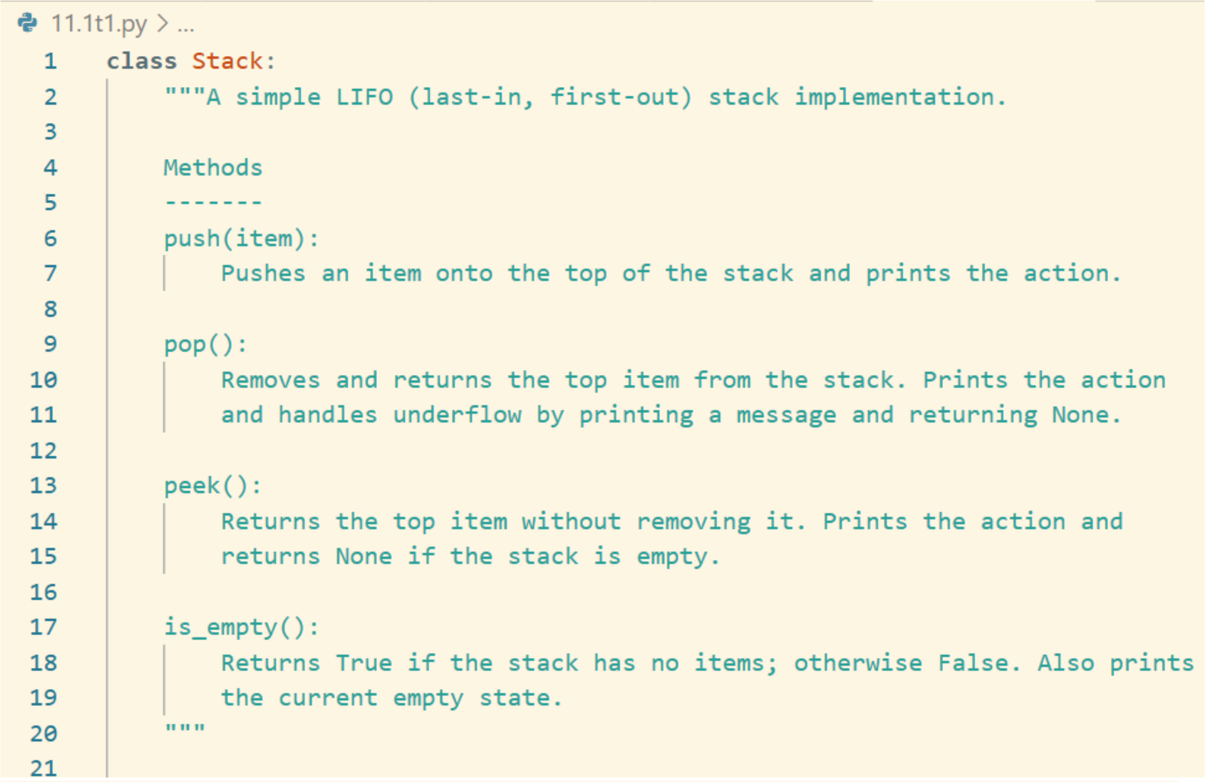
Prompt :

Generate a Stack class with push, pop, peek, and is_empty

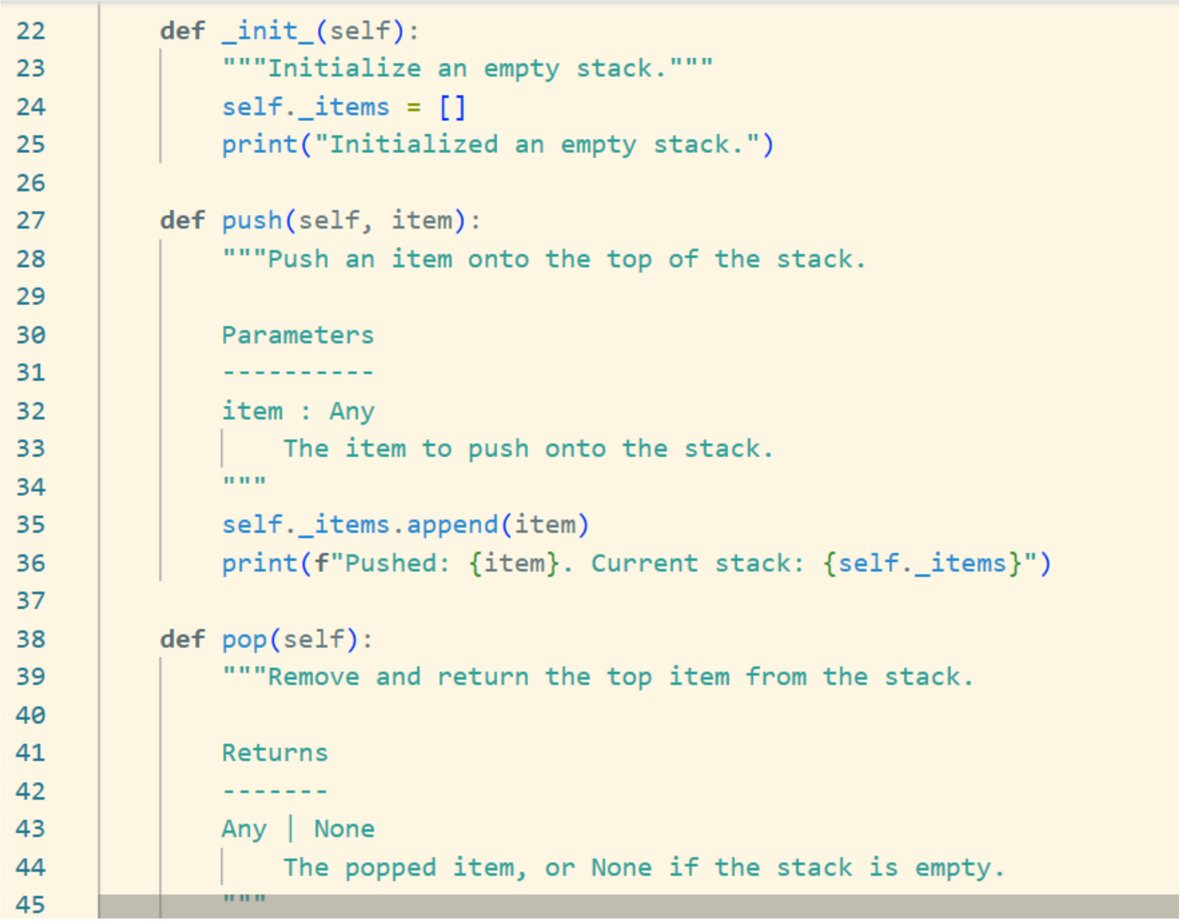
methods.Sample input should be like class Stack:pass

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 .

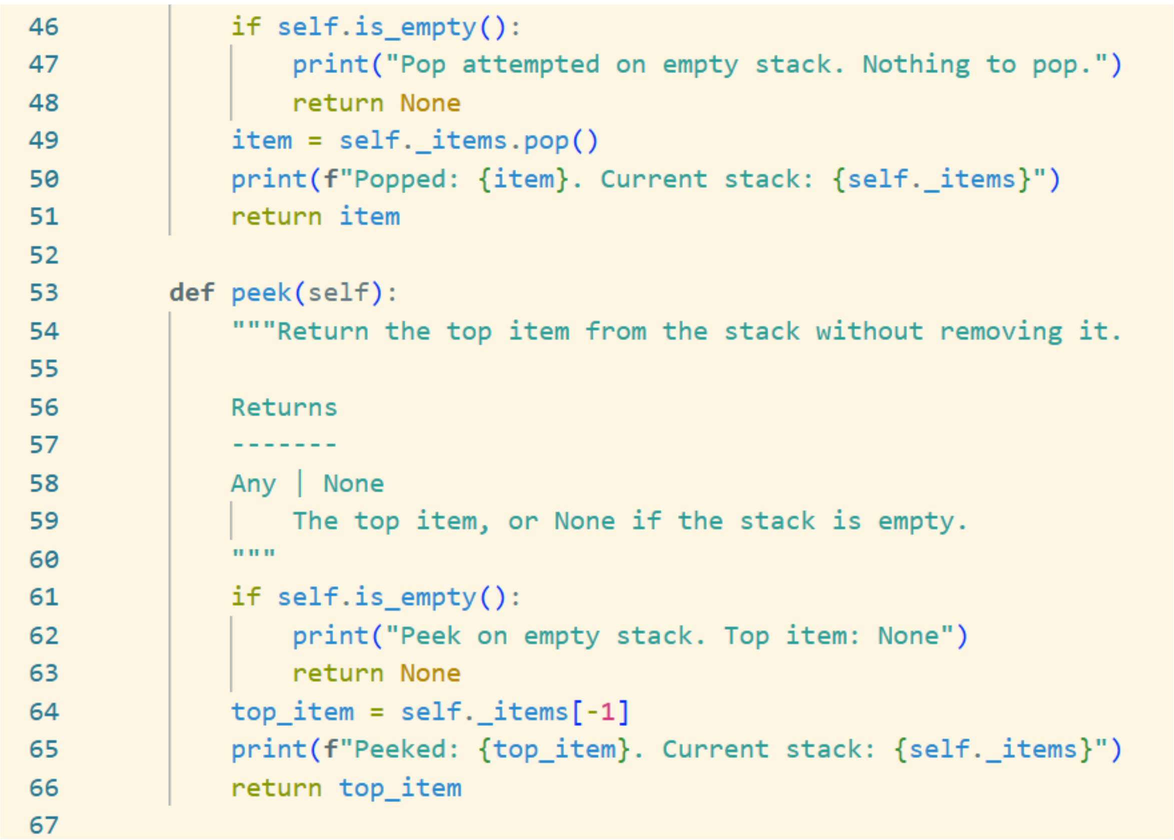
Code Generated :



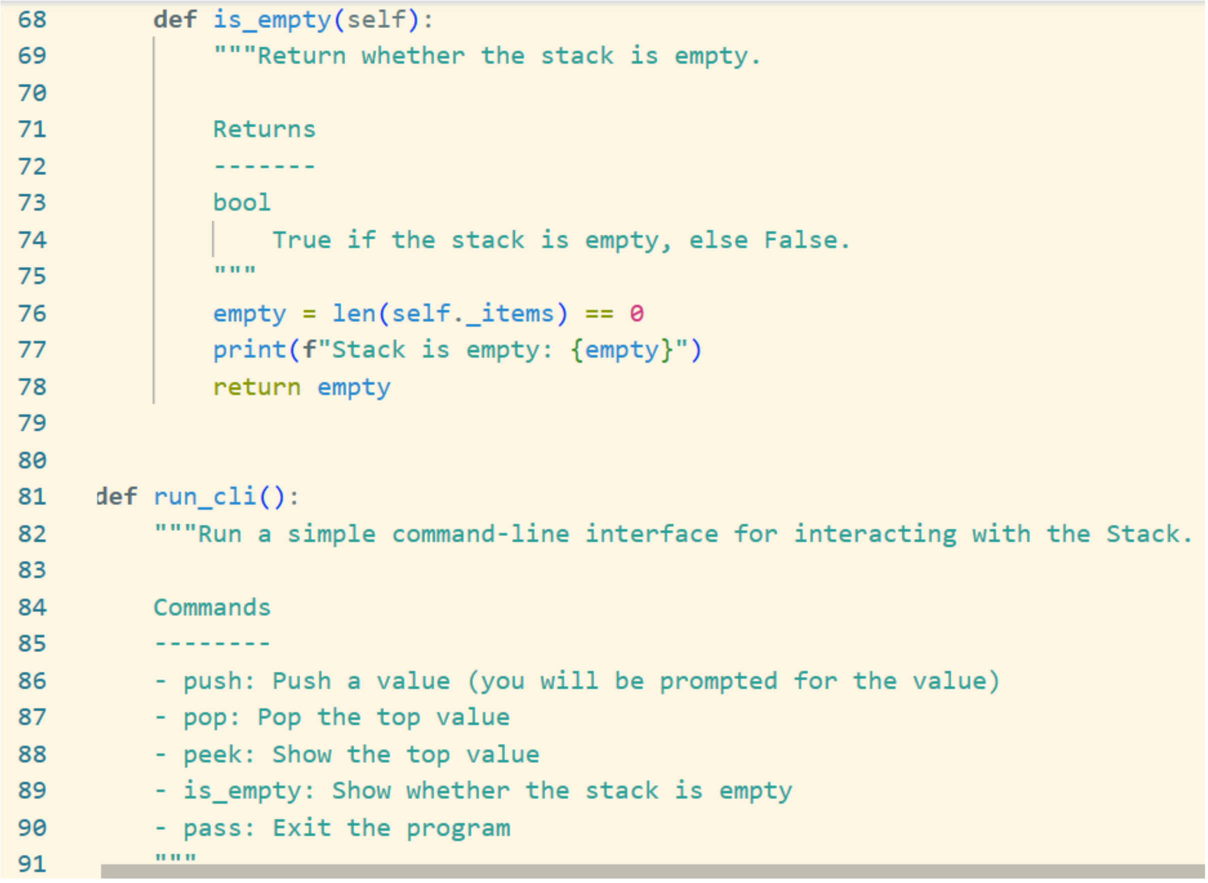
```python
class Stack:
    """A simple LIFO (last-in, first-out) stack implementation.

    Methods
    -------
    push(item):
        Pushes an item onto the top of the stack and prints the action.

    pop():
        Removes and returns the top item from the stack. Prints the action
        and handles underflow by printing a message and returning None.

    peek():
        Returns the top item without removing it. Prints the action and
        returns None if the stack is empty.

    is_empty():
        Returns True if the stack has no items; otherwise False. Also prints
        the current empty state.
    """

```



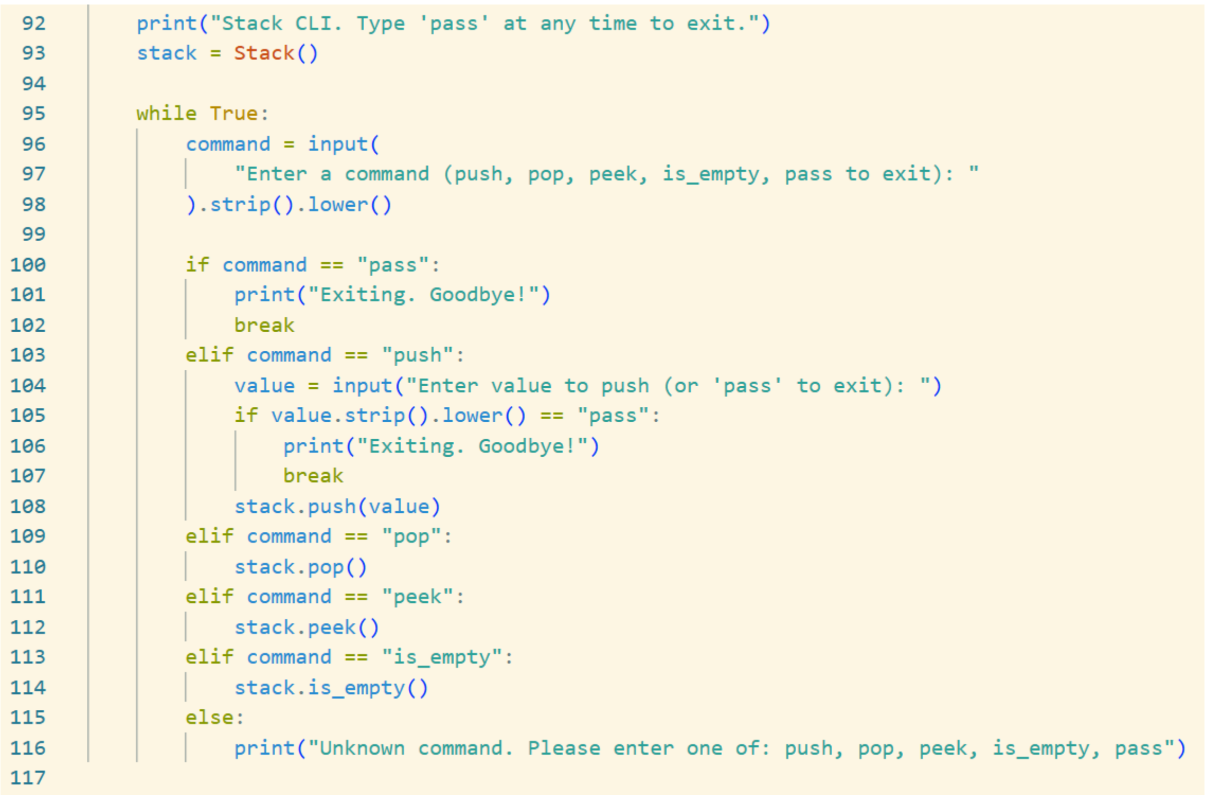
```python
    def _init_(self):
        """Initialize an empty stack."""
        self._items = []
        print("Initialized an empty stack.")

    def push(self, item):
        """Push an item onto the top of the stack.

        Parameters
        ----------
        item : Any
            The item to push onto the stack.
        """
        self._items.append(item)
        print(f"Pushed: {item}. Current stack: {self._items}")

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

        Returns
        -------
        Any | None
            The popped item, or None if the stack is empty.
        """
```



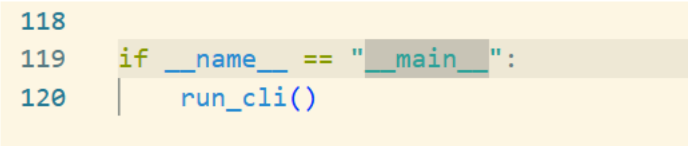
```python
        if self.is_empty():
            print("Pop attempted on empty stack. Nothing to pop.")
            return None
        item = self._items.pop()
        print(f"Popped: {item}. Current stack: {self._items}")
        return item

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

        Returns
        -------
        Any | None
            The top item, or None if the stack is empty.
        """
        if self.is_empty():
            print("Peek on empty stack. Top item: None")
            return None
        top_item = self._items[-1]
        print(f"Peeked: {top_item}. Current stack: {self._items}")
        return top_item

```



```python
    def is_empty(self):
        """Return whether the stack is empty.

        Returns
        -------
        bool
            True if the stack is empty, else False.
        """
        empty = len(self._items) == 0
        print(f"Stack is empty: {empty}")
        return empty


def run_cli():
    """Run a simple command-line interface for interacting with the Stack.

    Commands
    --------
    - push: Push a value (you will be prompted for the value)
    - pop: Pop the top value
    - peek: Show the top value
    - is_empty: Show whether the stack is empty
    - pass: Exit the program
    """
```

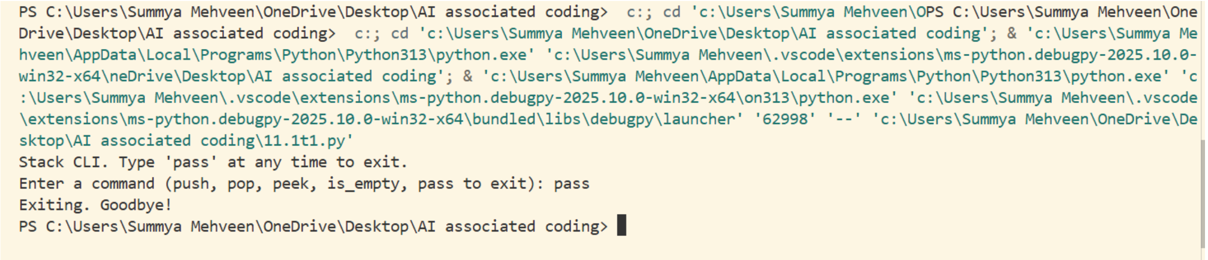


```python
    print("Stack CLI. Type 'pass' at any time to exit.")
    stack = Stack()

    while True:
        command = input(
            "Enter a command (push, pop, peek, is_empty, pass to exit): "
        ).strip().lower()

        if command == "pass":
            print("Exiting. Goodbye!")
            break
        elif command == "push":
            value = input("Enter value to push (or 'pass' to exit): ")
            if value.strip().lower() == "pass":
                print("Exiting. Goodbye!")
                break
            stack.push(value)
        elif command == "pop":
            stack.pop()
        elif command == "peek":
            stack.peek()
        elif command == "is_empty":
            stack.is_empty()
        else:
            print("Unknown command. Please enter one of: push, pop, peek, is_empty, pass")

```



```python

if __name__ == "__main__":
    run_cli()
```

Output :



```
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>  c:; cd 'c:\Users\Summya Mehveen\OPS C:\Users\Summya Mehveen\One
Drive\Desktop\AI associated coding>  c:; cd 'c:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding'; & 'c:\Users\Summya Me
hveen\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\Summya Mehveen\.vscode\extensions\ms-python.debugpy-2025.10.0-
win32-x64\neDrive\Desktop\AI associated coding'; & 'c:\Users\Summya Mehveen\AppData\Local\Programs\Python\Python313\python.exe' 'c
:\Users\Summya Mehveen\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\on313\python.exe' 'c:\Users\Summya Mehveen\.vscode
\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '62998' '--' 'c:\Users\Summya Mehveen\OneDrive\De
sktop\AI associated coding\11.1t1.py'
Stack CLI. Type 'pass' at any time to exit.
Enter a command (push, pop, peek, is_empty, pass to exit): pass
Exiting. Goodbye!
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding> 
```

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 :

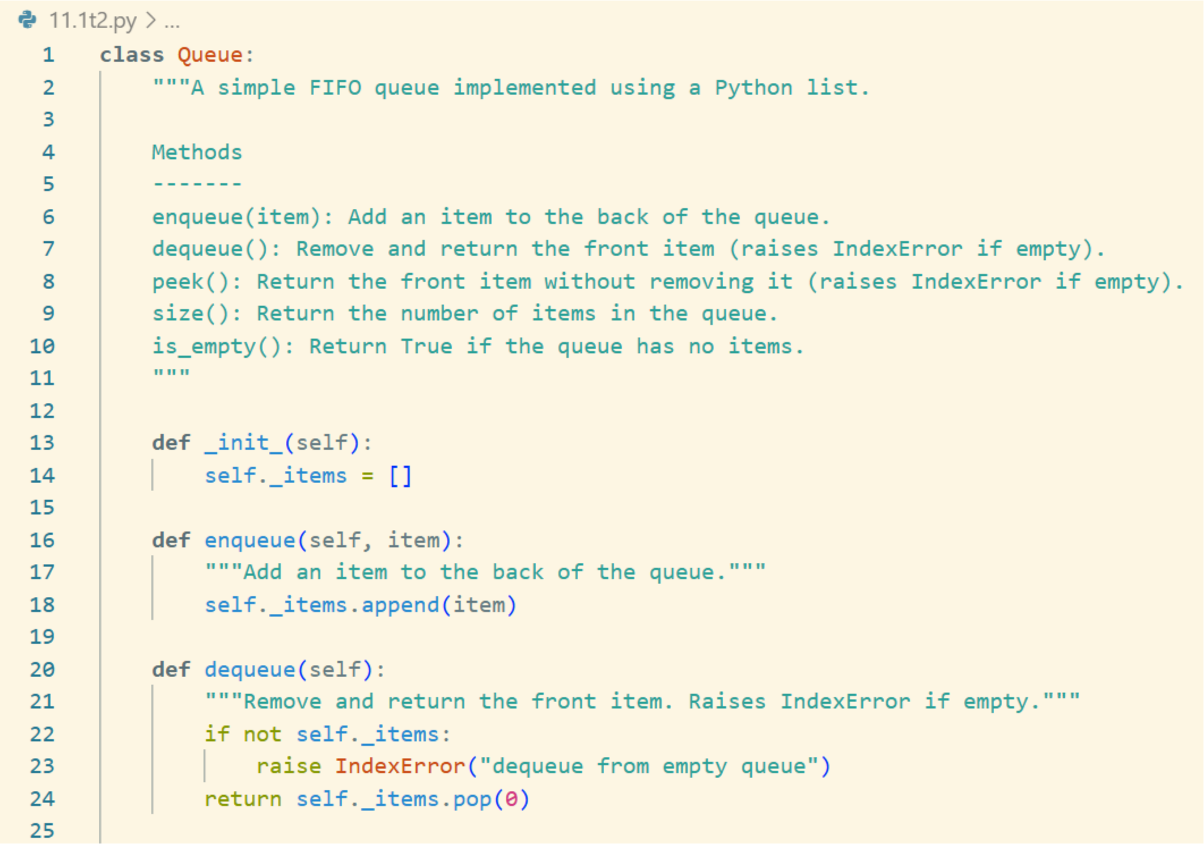
Prompt :

Use AI to implement a Queue using Python lists.

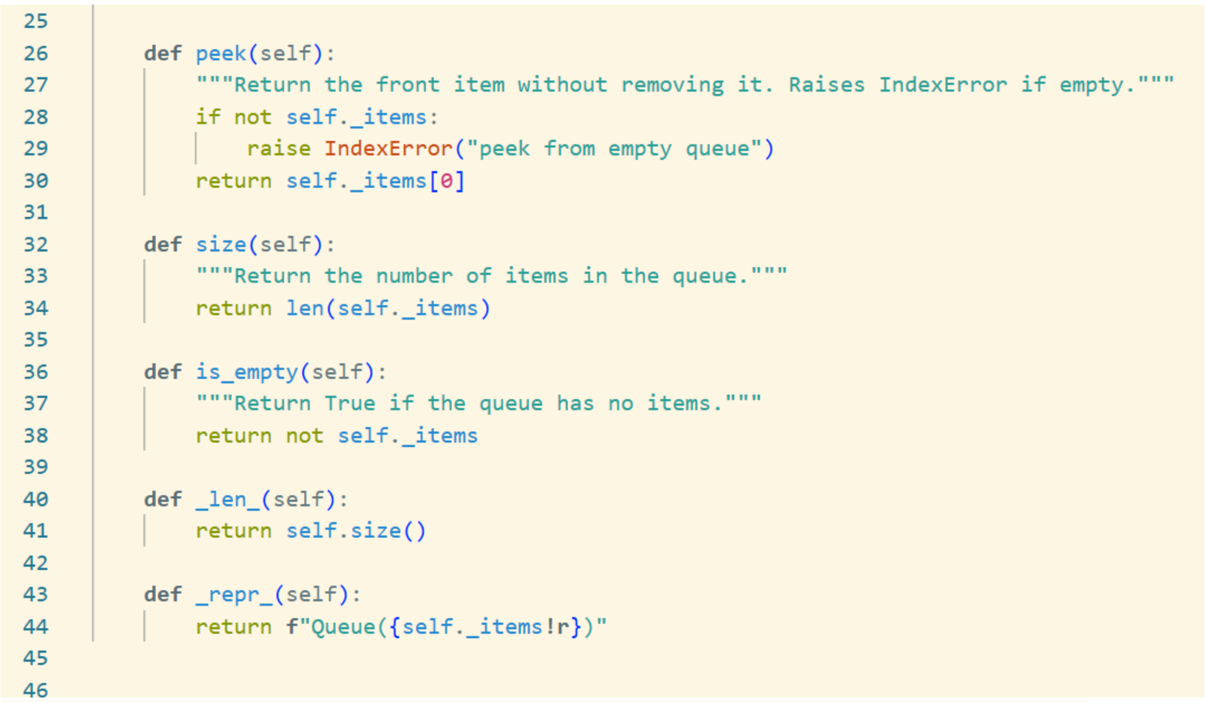
Sample Input Code should be like

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.

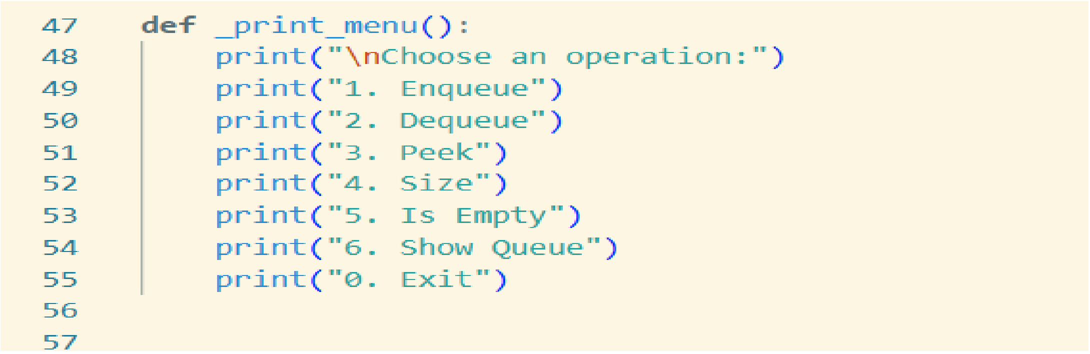
Code Generated :



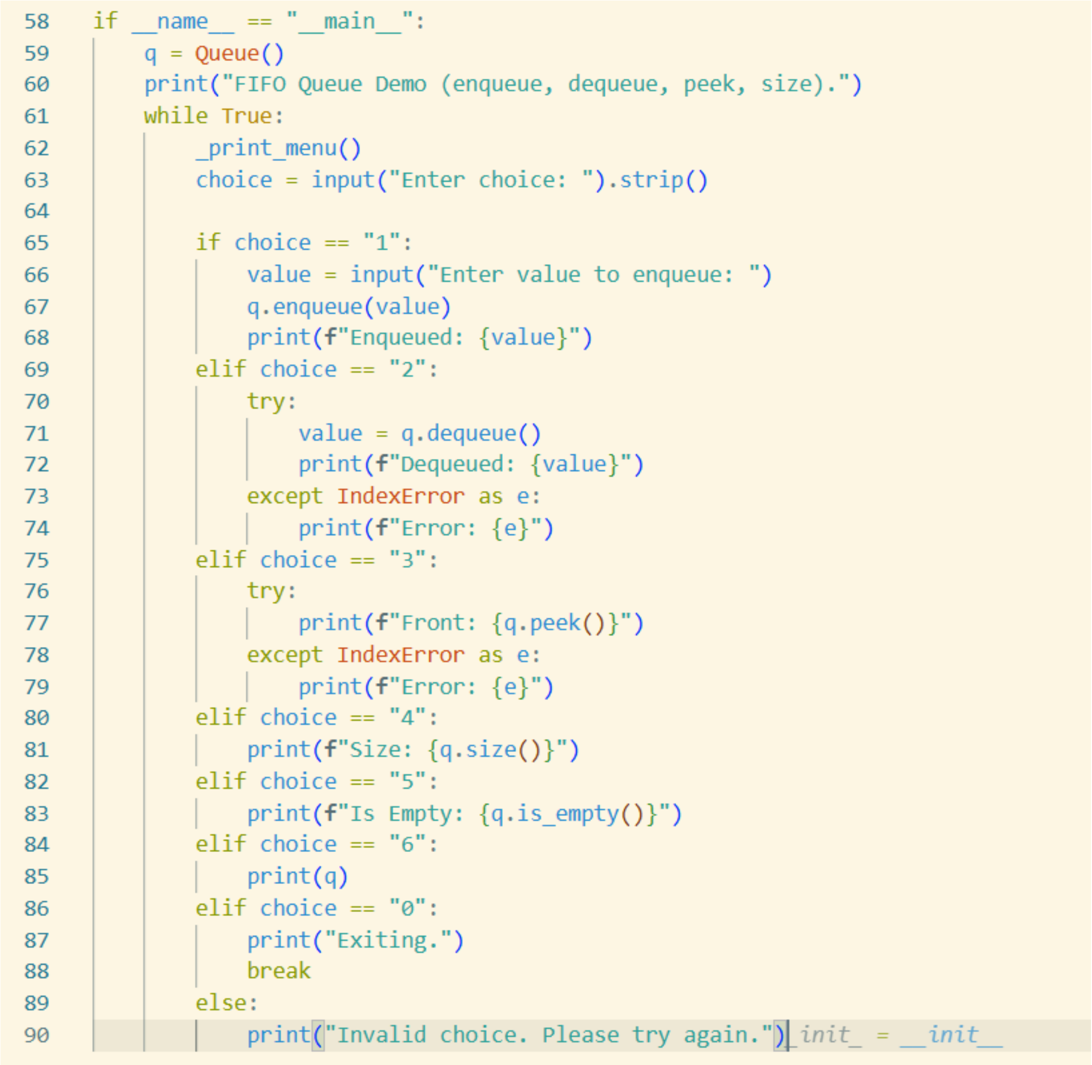
```python
class Queue:
    """A simple FIFO queue implemented using a Python list.

    Methods
    -------
    enqueue(item): Add an item to the back of the queue.
    dequeue(): Remove and return the front item (raises IndexError if empty).
    peek(): Return the front item without removing it (raises IndexError if empty).
    size(): Return the number of items in the queue.
    is_empty(): Return True if the queue has no items.
    """

    def _init_(self):
        self._items = []

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

    def dequeue(self):
        """Remove and return the front item. Raises IndexError if empty."""
        if not self._items:
            raise IndexError("dequeue from empty queue")
        return self._items.pop(0)

```



```python

    def peek(self):
        """Return the front item without removing it. Raises IndexError if empty."""
        if not self._items:
            raise IndexError("peek from empty queue")
        return self._items[0]

    def size(self):
        """Return the number of items in the queue."""
        return len(self._items)

    def is_empty(self):
        """Return True if the queue has no items."""
        return not self._items

    def _len_(self):
        return self.size()

    def _repr_(self):
        return f"Queue({self._items!r})"


```

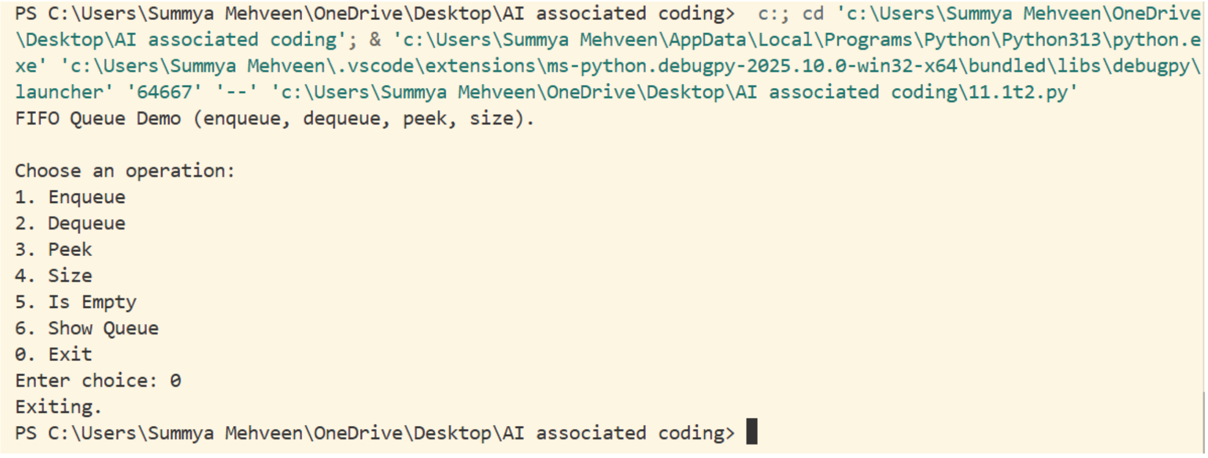


```python
def _print_menu():
    print("\nChoose an operation:")
    print("1. Enqueue")
    print("2. Dequeue")
    print("3. Peek")
    print("4. Size")
    print("5. Is Empty")
    print("6. Show Queue")
    print("0. Exit")


```



```python
if __name__ == "__main__":
    q = Queue()
    print("FIFO Queue Demo (enqueue, dequeue, peek, size).")
    while True:
        _print_menu()
        choice = input("Enter choice: ").strip()

        if choice == "1":
            value = input("Enter value to enqueue: ")
            q.enqueue(value)
            print(f"Enqueued: {value}")
        elif choice == "2":
            try:
                value = q.dequeue()
                print(f"Dequeued: {value}")
            except IndexError as e:
                print(f"Error: {e}")
        elif choice == "3":
            try:
                print(f"Front: {q.peek()}")
            except IndexError as e:
                print(f"Error: {e}")
        elif choice == "4":
            print(f"Size: {q.size()}")
        elif choice == "5":
            print(f"Is Empty: {q.is_empty()}")
        elif choice == "6":
            print(q)
        elif choice == "0":
            print("Exiting.")
            break
        else:
            print("Invalid choice. Please try again.")_init_ = __init__
```

Output :



```
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>  c:; cd 'c:\Users\Summya Mehveen\OneDrive
\Desktop\AI associated coding'; & 'c:\Users\Summya Mehveen\AppData\Local\Programs\Python\Python313\python.e
xe' 'c:\Users\Summya Mehveen\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\
launcher' '64667' '--' 'c:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding\11.1t2.py'
FIFO Queue Demo (enqueue, dequeue, peek, size).

Choose an operation:
1. Enqueue
2. Dequeue
3. Peek
4. Size
5. Is Empty
6. Show Queue
0. Exit
Enter choice: 0
Exiting.
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding> █
```

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

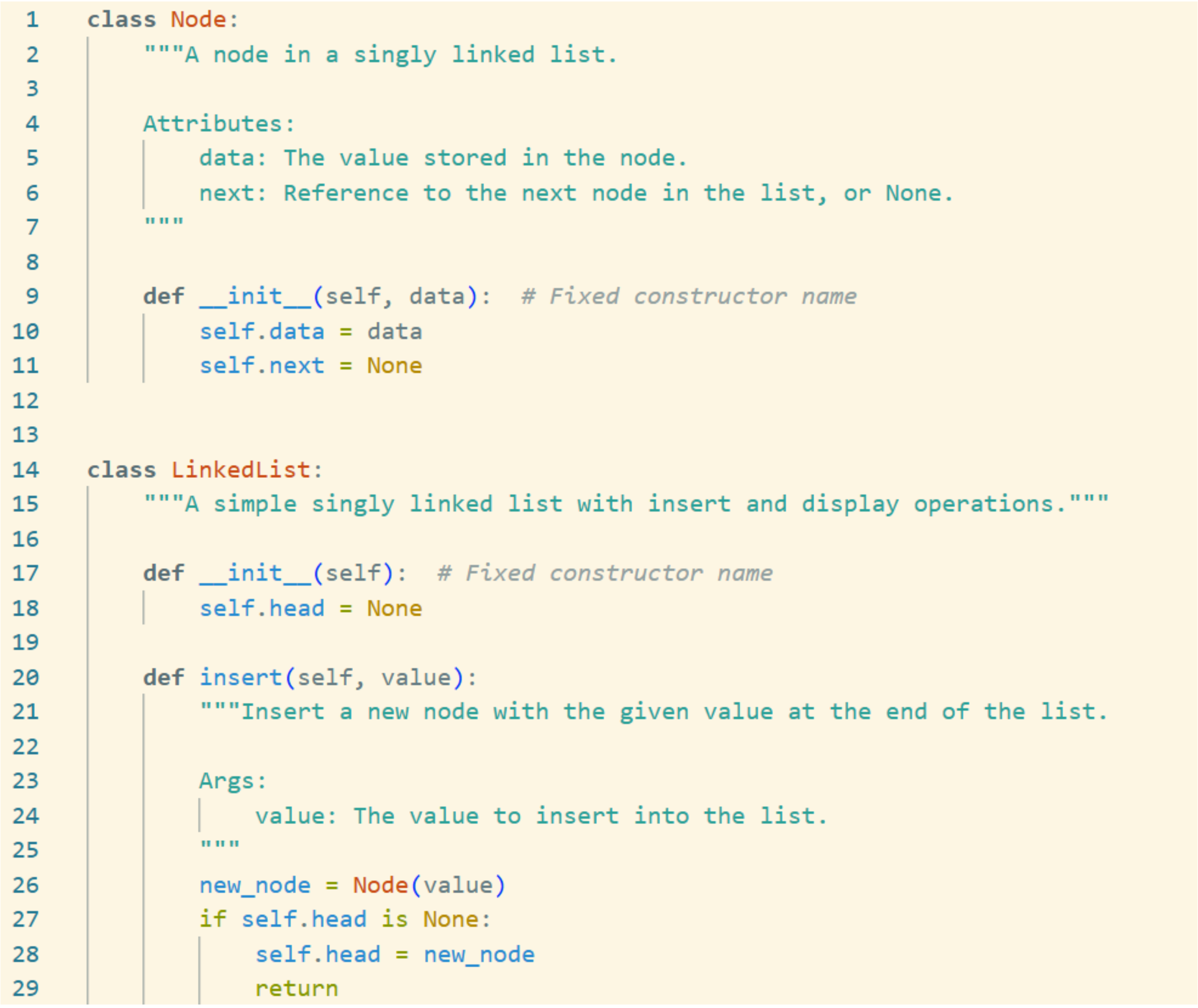
Sample Input Code should be like class Node:pass class Linked: List

Expected Output:

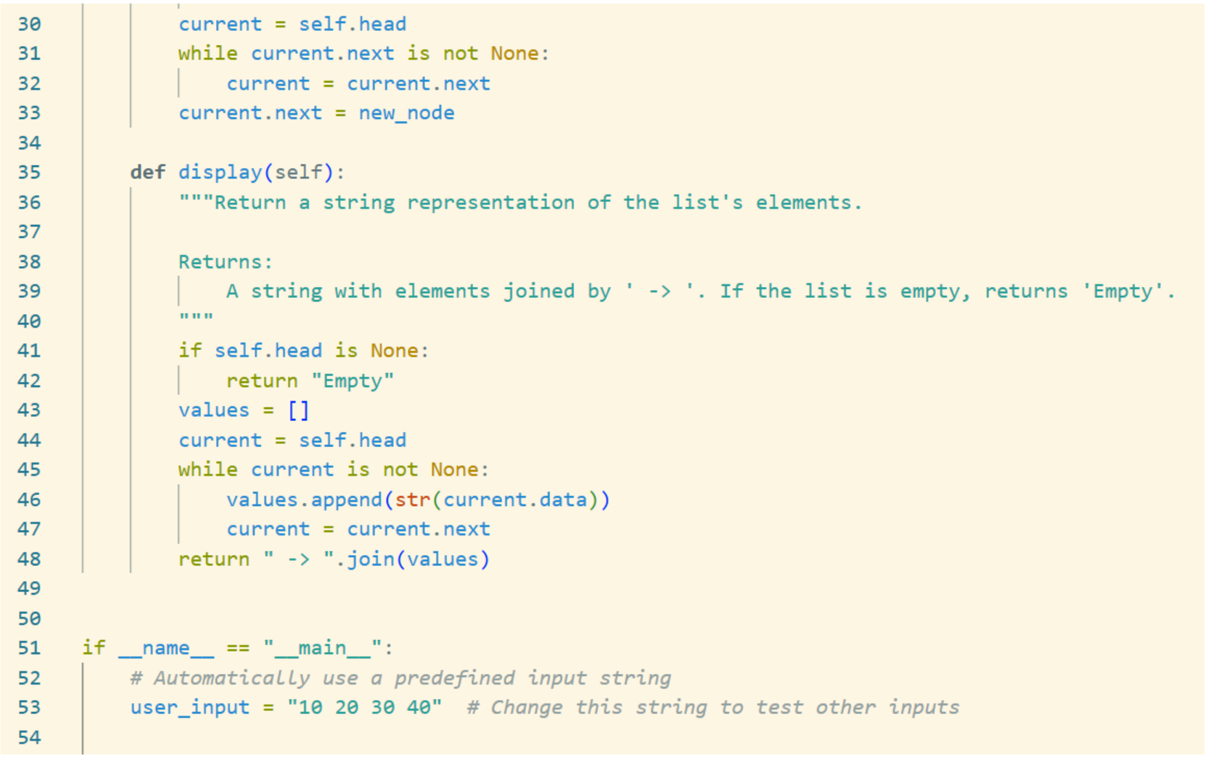
• A working linked list implementation with clear method

documentation.

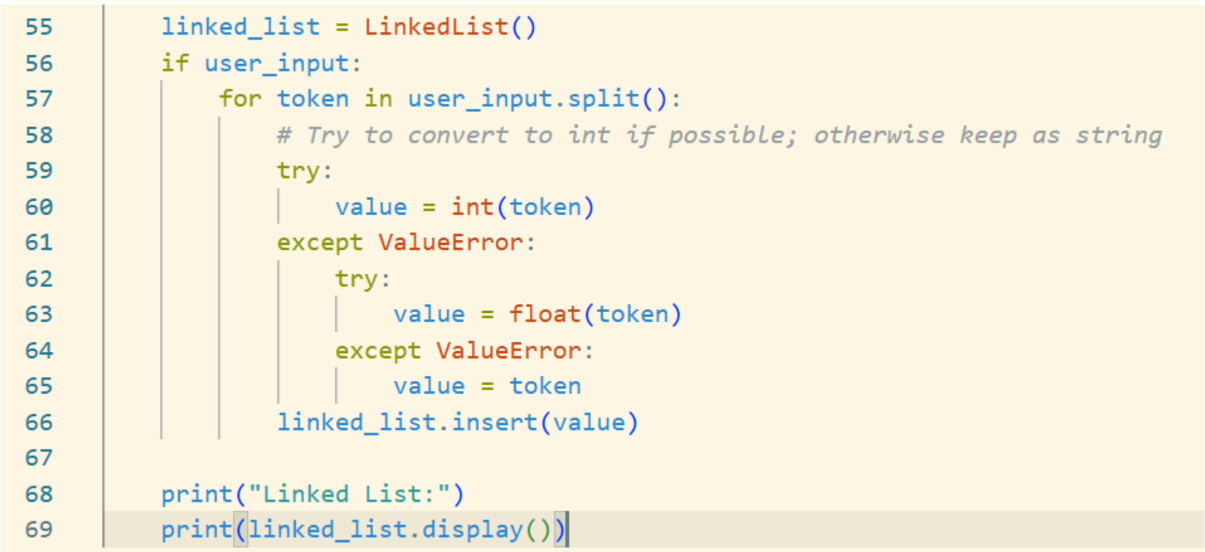
Code Generated :



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

    Attributes:
        data: The value stored in the node.
        next: Reference to the next node in the list, or None.
    """

    def __init__(self, data):  # Fixed constructor name
        self.data = data
        self.next = None


class LinkedList:
    """A simple singly linked list with insert and display operations."""

    def __init__(self):  # Fixed constructor name
        self.head = None

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

        Args:
            value: The value to insert into the list.
        """
        new_node = Node(value)
        if self.head is None:
            self.head = new_node
            return
```

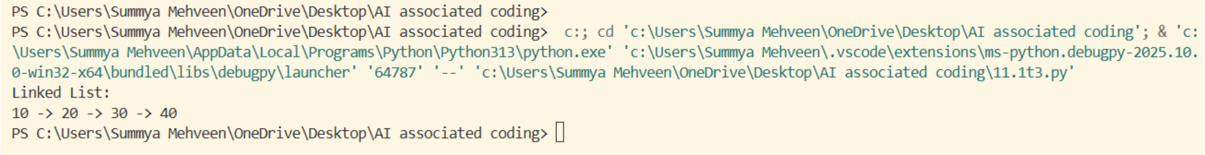


```python
        current = self.head
        while current.next is not None:
            current = current.next
        current.next = new_node

    def display(self):
        """Return a string representation of the list's elements.

        Returns:
            A string with elements joined by ' -> '. If the list is empty, returns 'Empty'.
        """
        if self.head is None:
            return "Empty"
        values = []
        current = self.head
        while current is not None:
            values.append(str(current.data))
            current = current.next
        return " -> ".join(values)


if __name__ == "__main__":
    # Automatically use a predefined input string
    user_input = "10 20 30 40"  # Change this string to test other inputs

```



```python
    linked_list = LinkedList()
    if user_input:
        for token in user_input.split():
            # Try to convert to int if possible; otherwise keep as string
            try:
                value = int(token)
            except ValueError:
                try:
                    value = float(token)
                except ValueError:
                    value = token
            linked_list.insert(value)

    print("Linked List:")
    print(linked_list.display())
```

Output :



```
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>  c:; cd 'c:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding'; & 'c:
\Users\Summya Mehveen\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\Summya Mehveen\.vscode\extensions\ms-python.debugpy-2025.10.
0-win32-x64\bundled\libs\debugpy\launcher' '64787' '--' 'c:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding\11.1t3.py'
Linked List:
10 -> 20 -> 30 -> 40
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding> ▯
```

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 :

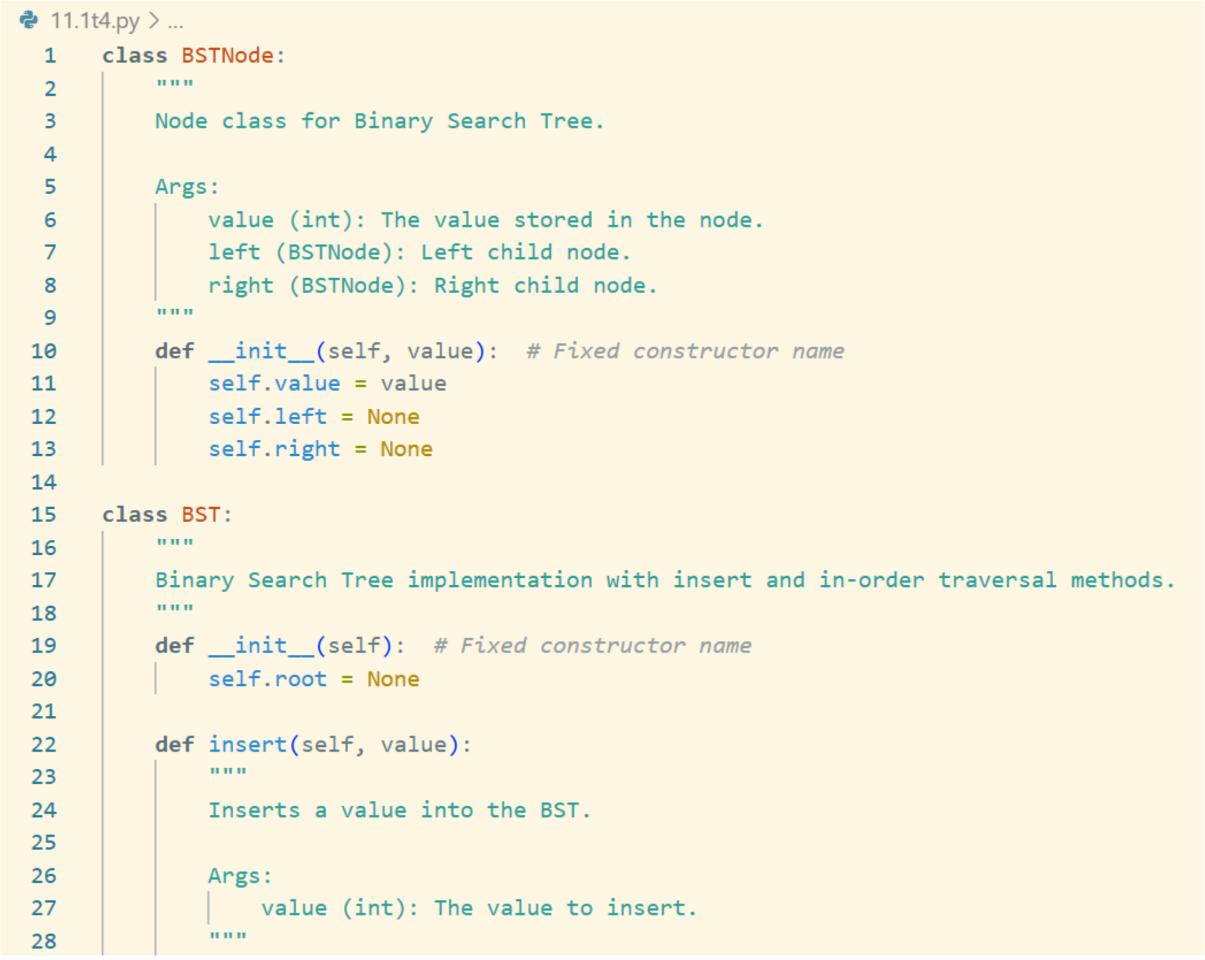
create a BST with insert and in-order traversal methods.

Sample Input Code:

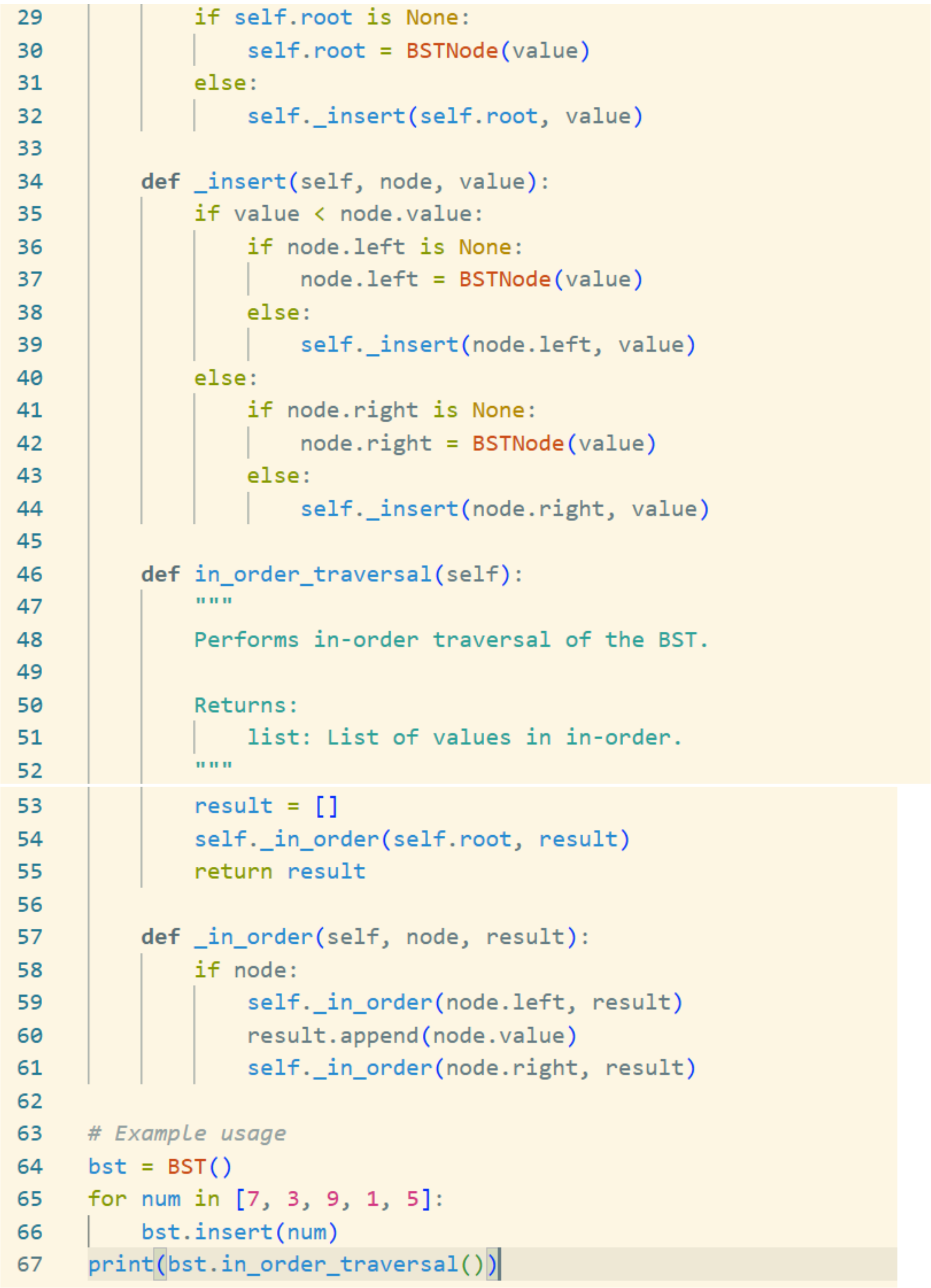
class BST:

pass

Code Generated :

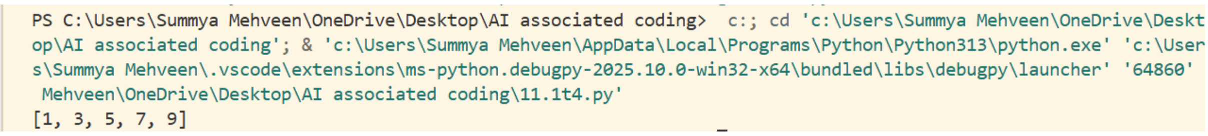


```python
class BSTNode:
    """
    Node class for Binary Search Tree.

    Args:
        value (int): The value stored in the node.
        left (BSTNode): Left child node.
        right (BSTNode): Right child node.
    """
    def __init__(self, value):  # Fixed constructor name
        self.value = value
        self.left = None
        self.right = None

class BST:
    """
    Binary Search Tree implementation with insert and in-order traversal methods.
    """
    def __init__(self):  # Fixed constructor name
        self.root = None

    def insert(self, value):
        """
        Inserts a value into the BST.

        Args:
            value (int): The value to insert.
        """
```



```python
        if self.root is None:
            self.root = BSTNode(value)
        else:
            self._insert(self.root, value)

    def _insert(self, node, value):
        if value < node.value:
            if node.left is None:
                node.left = BSTNode(value)
            else:
                self._insert(node.left, value)
        else:
            if node.right is None:
                node.right = BSTNode(value)
            else:
                self._insert(node.right, value)

    def in_order_traversal(self):
        """
        Performs in-order traversal of the BST.

        Returns:
            list: List of values in in-order.
        """
        result = []
        self._in_order(self.root, result)
        return result

    def _in_order(self, node, result):
        if node:
            self._in_order(node.left, result)
            result.append(node.value)
            self._in_order(node.right, result)

# Example usage
bst = BST()
for num in [7, 3, 9, 1, 5]:
    bst.insert(num)
print(bst.in_order_traversal())
```

Output :



```
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>  c:; cd 'c:\Users\Summya Mehveen\OneDrive\Deskt
op\AI associated coding'; & 'c:\Users\Summya Mehveen\AppData\Local\Programs\Python\Python313\python.exe' 'c:\User
s\Summya Mehveen\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '64860'
 Mehveen\OneDrive\Desktop\AI associated coding\11.1t4.py'
[1, 3, 5, 7, 9]
```

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 :

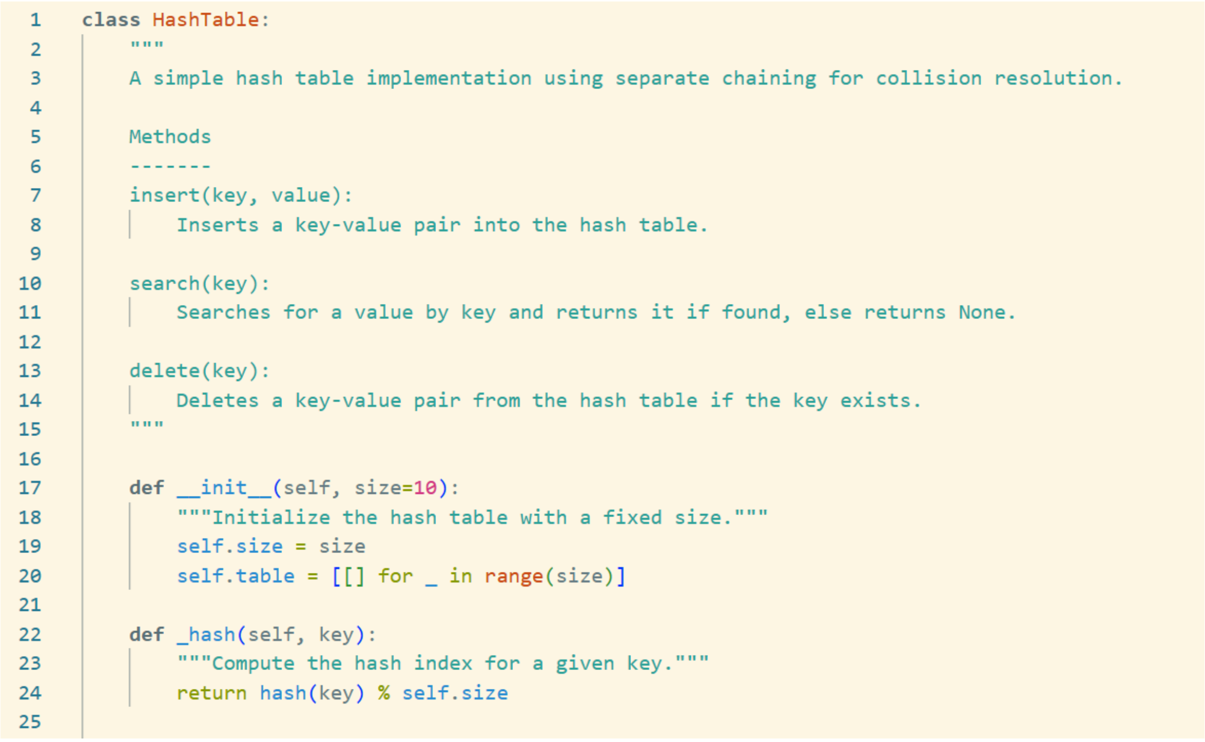
Prompt :

Implement a hash table with basic insert, search, and delete methods.

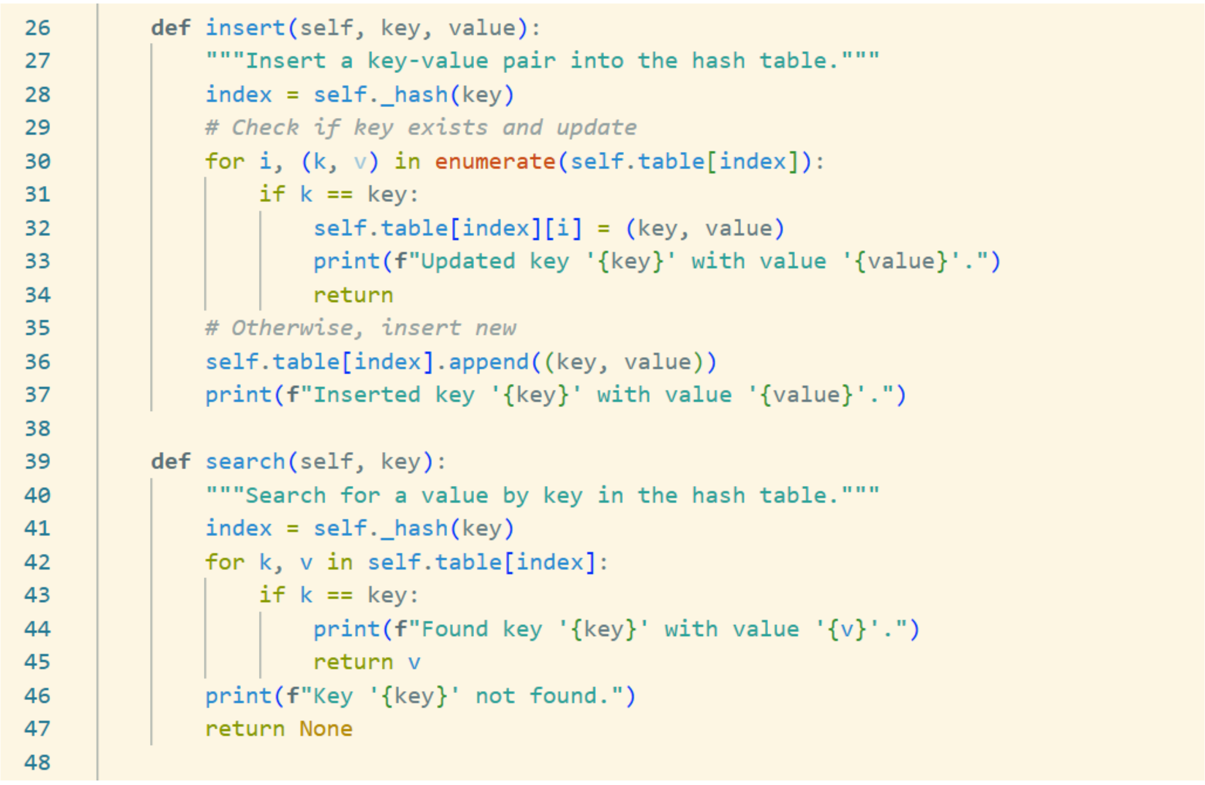
Sample Input Code:

```
class HashTable:

pass
```

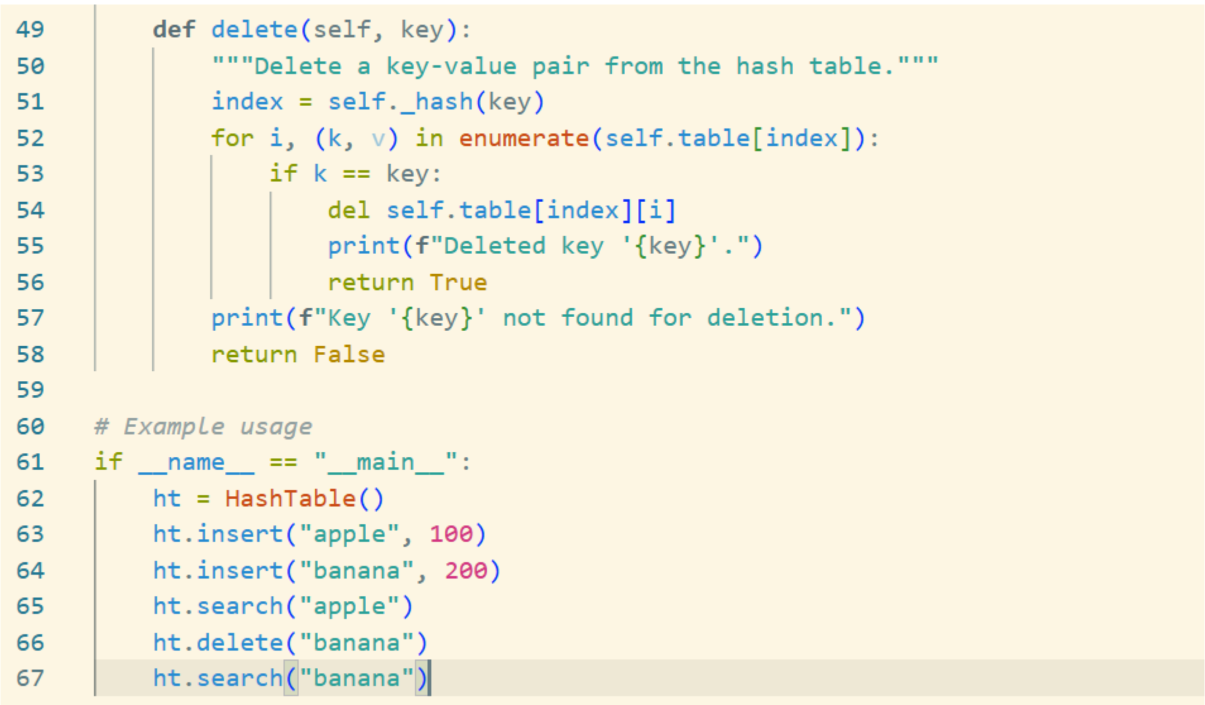
Code Generated :



```python
class HashTable:
    """
    A simple hash table implementation using separate chaining for collision resolution.

    Methods
    -------
    insert(key, value):
        Inserts a key-value pair into the hash table.

    search(key):
        Searches for a value by key and returns it if found, else returns None.

    delete(key):
        Deletes a key-value pair from the hash table if the key exists.
    """

    def __init__(self, size=10):
        """Initialize the hash table with a fixed size."""
        self.size = size
        self.table = [[] for _ in range(size)]

    def _hash(self, key):
        """Compute the hash index for a given key."""
        return hash(key) % self.size

```

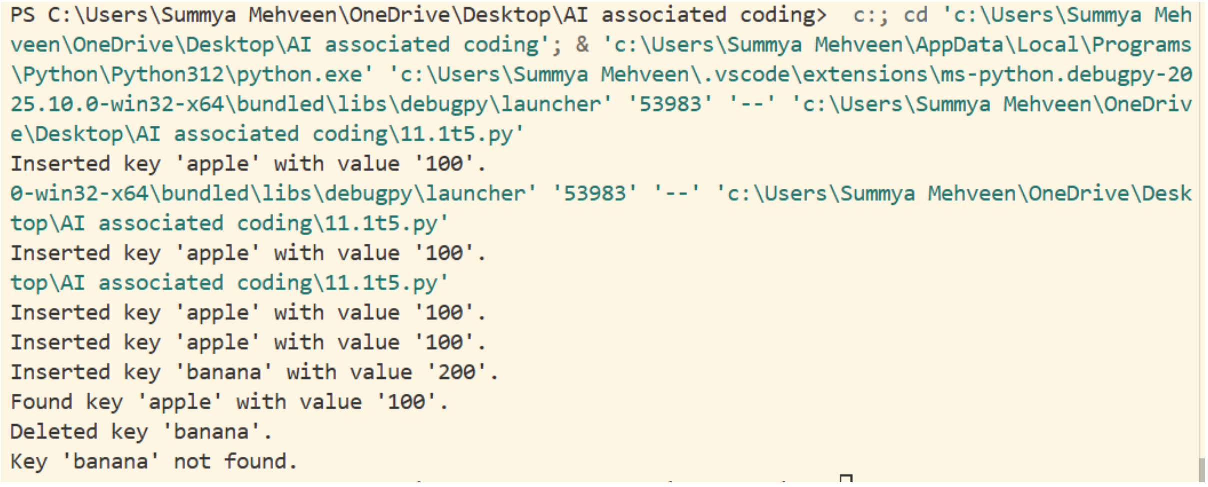


```python
    def insert(self, key, value):
        """Insert a key-value pair into the hash table."""
        index = self._hash(key)
        # Check if key exists and update
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                self.table[index][i] = (key, value)
                print(f"Updated key '{key}' with value '{value}'.")
                return
        # Otherwise, insert new
        self.table[index].append((key, value))
        print(f"Inserted key '{key}' with value '{value}'.")

    def search(self, key):
        """Search for a value by key in the hash table."""
        index = self._hash(key)
        for k, v in self.table[index]:
            if k == key:
                print(f"Found key '{key}' with value '{v}'.")
                return v
        print(f"Key '{key}' not found.")
        return None

```



```
    def delete(self, key):
        """Delete a key-value pair from the hash table."""
        index = self._hash(key)
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                del self.table[index][i]
                print(f"Deleted key '{key}'.")
                return True
        print(f"Key '{key}' not found for deletion.")
        return False

# Example usage
if __name__ == "__main__":
    ht = HashTable()
    ht.insert("apple", 100)
    ht.insert("banana", 200)
    ht.search("apple")
    ht.delete("banana")
    ht.search("banana")
```

Output :



```
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>  c:; cd 'c:\Users\Summya Meh
veen\OneDrive\Desktop\AI associated coding'; & 'c:\Users\Summya Mehveen\AppData\Local\Programs
\Python\Python312\python.exe' 'c:\Users\Summya Mehveen\.vscode\extensions\ms-python.debugpy-20
25.10.0-win32-x64\bundled\libs\debugpy\launcher' '53983' '--' 'c:\Users\Summya Mehveen\OneDriv
e\Desktop\AI associated coding\11.1t5.py'
Inserted key 'apple' with value '100'.
0-win32-x64\bundled\libs\debugpy\launcher' '53983' '--' 'c:\Users\Summya Mehveen\OneDrive\Desk
top\AI associated coding\11.1t5.py'
Inserted key 'apple' with value '100'.
top\AI associated coding\11.1t5.py'
Inserted key 'apple' with value '100'.
Inserted key 'apple' with value '100'.
Inserted key 'banana' with value '200'.
Found key 'apple' with value '100'.
Deleted key 'banana'.
Key 'banana' not found.
```

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

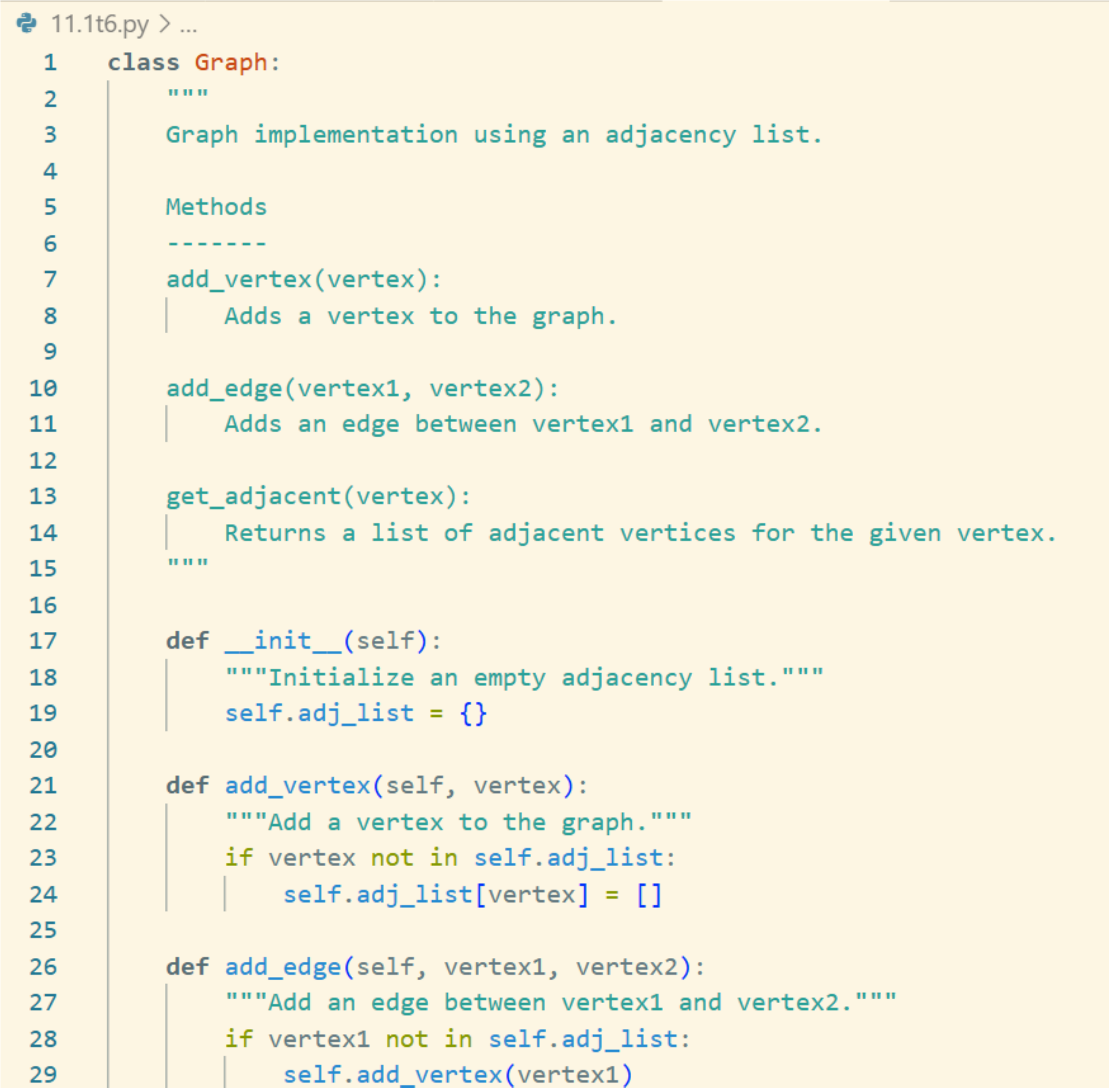
TASK 6 :

Prompt :

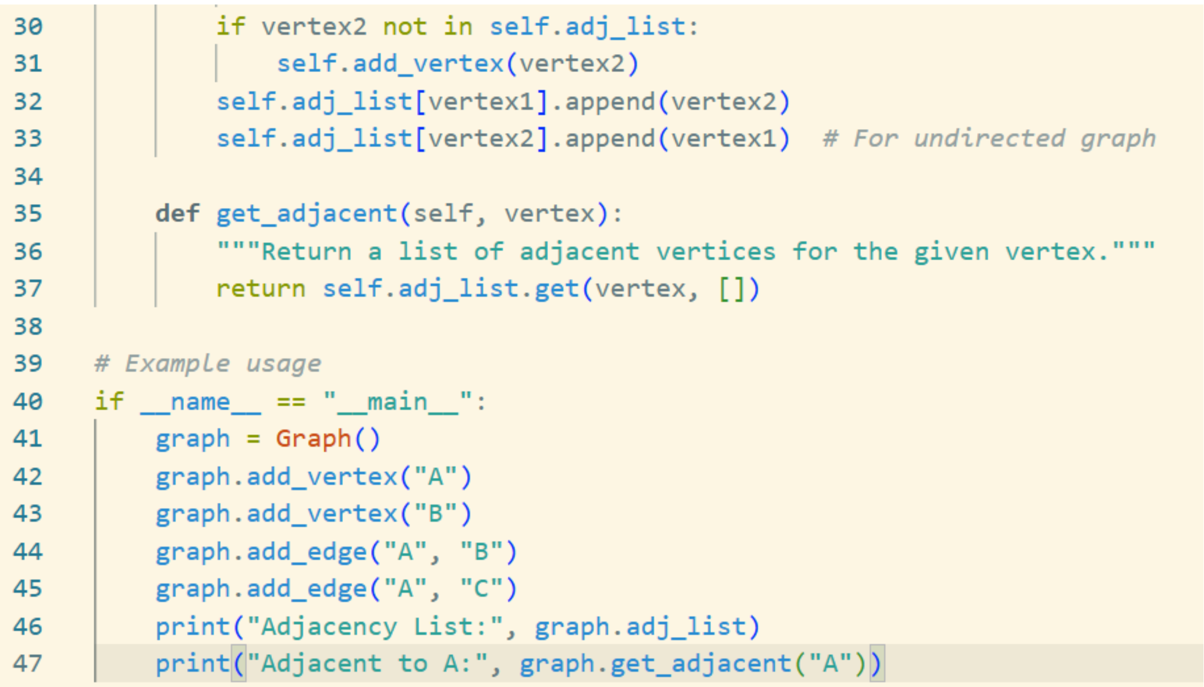
Implement a graph using an adjacency list Sample Input Code:

```
class Graph:

pass
```

Code Generated :

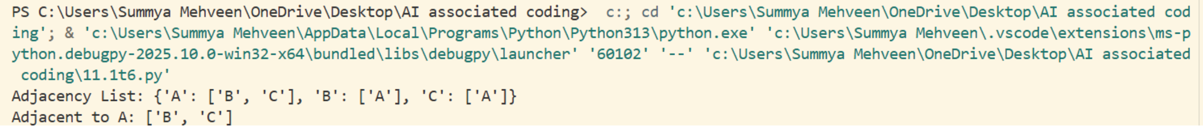


```python
class Graph:
    """
    Graph implementation using an adjacency list.

    Methods
    -------
    add_vertex(vertex):
        Adds a vertex to the graph.

    add_edge(vertex1, vertex2):
        Adds an edge between vertex1 and vertex2.

    get_adjacent(vertex):
        Returns a list of adjacent vertices for the given vertex.
    """

    def __init__(self):
        """Initialize an empty adjacency list."""
        self.adj_list = {}

    def add_vertex(self, vertex):
        """Add a vertex to the graph."""
        if vertex not in self.adj_list:
            self.adj_list[vertex] = []

    def add_edge(self, vertex1, vertex2):
        """Add an edge between vertex1 and vertex2."""
        if vertex1 not in self.adj_list:
            self.add_vertex(vertex1)
```



```python
        if vertex2 not in self.adj_list:
            self.add_vertex(vertex2)
        self.adj_list[vertex1].append(vertex2)
        self.adj_list[vertex2].append(vertex1)  # For undirected graph

    def get_adjacent(self, vertex):
        """Return a list of adjacent vertices for the given vertex."""
        return self.adj_list.get(vertex, [])

# Example usage
if __name__ == "__main__":
    graph = Graph()
    graph.add_vertex("A")
    graph.add_vertex("B")
    graph.add_edge("A", "B")
    graph.add_edge("A", "C")
    print("Adjacency List:", graph.adj_list)
    print("Adjacent to A:", graph.get_adjacent("A"))
```

Output :



```
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>  c:; cd 'c:\Users\Summya Mehveen\OneDrive\Desktop\AI associated cod
ing'; & 'c:\Users\Summya Mehveen\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\Summya Mehveen\.vscode\extensions\ms-p
ython.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '60102' '--' 'c:\Users\Summya Mehveen\OneDrive\Desktop\AI associated
 coding\11.1t6.py'
Adjacency List: {'A': ['B', 'C'], 'B': ['A'], 'C': ['A']}
Adjacent to A: ['B', 'C']
```

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']}.

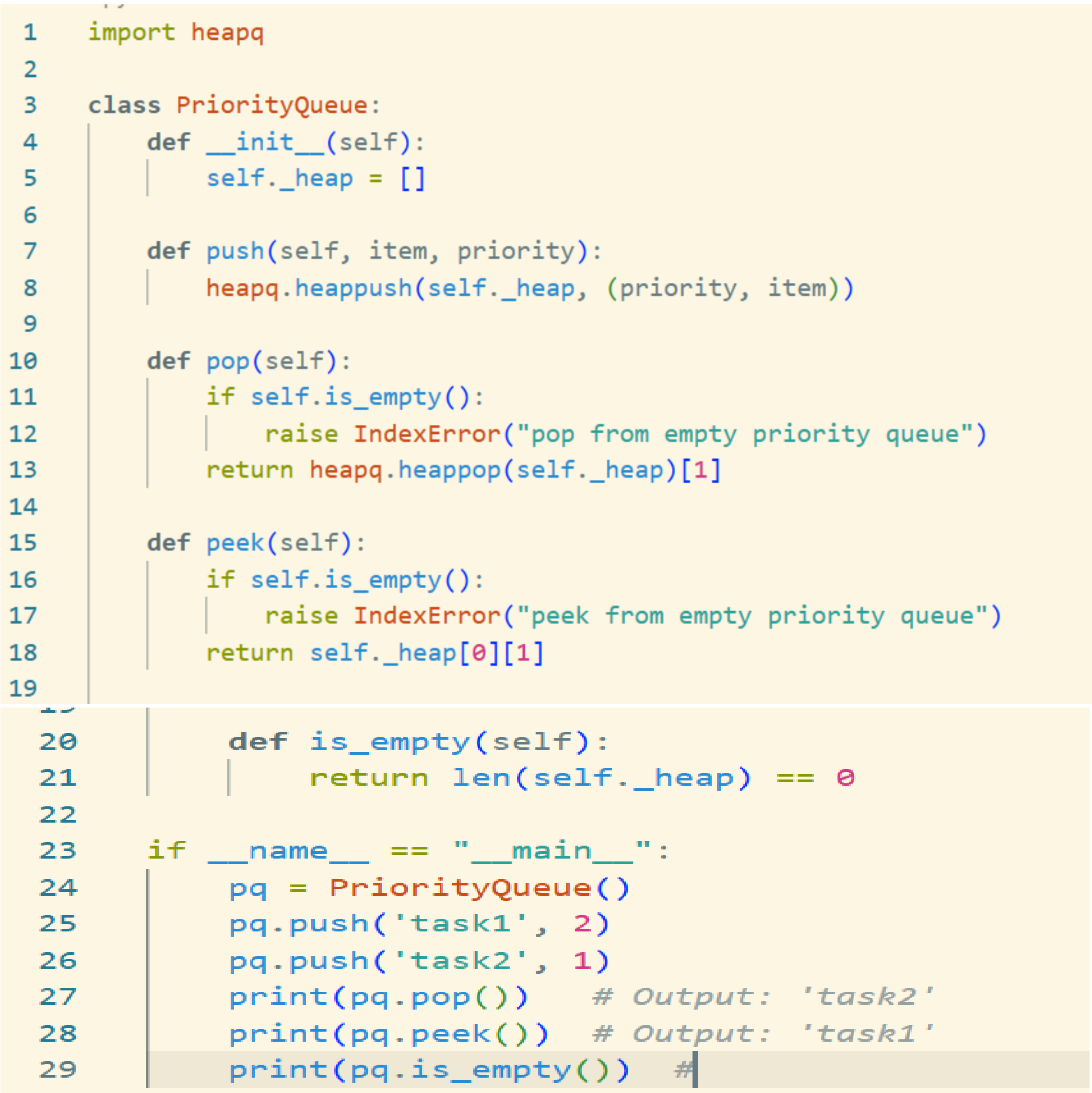
TASK 7 :

Prompt :

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

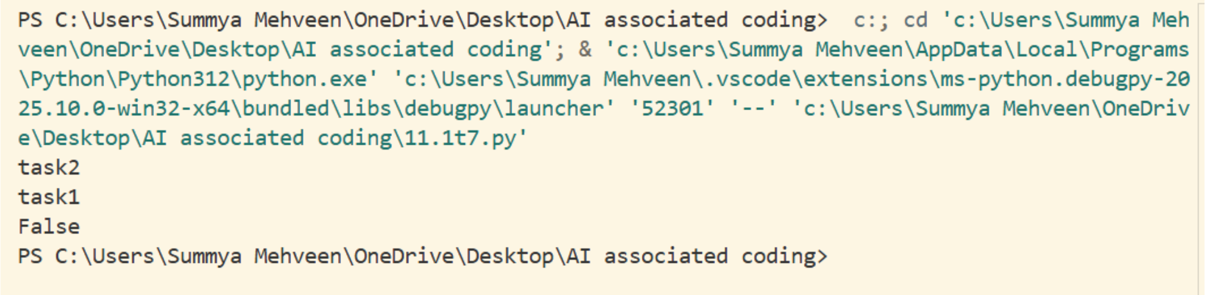
- push(item, priority): Add an item with the given priority to the queue.
- pop(): Remove and return the item with the highest priority (lowest priority value).
- peek(): Return the item with the highest priority without removing it.
- is_empty(): Return True if the queue is empty, otherwise False.

Code Generated :



```python
import heapq

class PriorityQueue:
    def __init__(self):
        self._heap = []

    def push(self, item, priority):
        heapq.heappush(self._heap, (priority, item))

    def pop(self):
        if self.is_empty():
            raise IndexError("pop from empty priority queue")
        return heapq.heappop(self._heap)[1]

    def peek(self):
        if self.is_empty():
            raise IndexError("peek from empty priority queue")
        return self._heap[0][1]

    def is_empty(self):
        return len(self._heap) == 0

if __name__ == "__main__":
    pq = PriorityQueue()
    pq.push('task1', 2)
    pq.push('task2', 1)
    print(pq.pop())   # Output: 'task2'
    print(pq.peek())  # Output: 'task1'
    print(pq.is_empty())  #
```

Output :



```
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>  c:; cd 'c:\Users\Summya Meh
veen\OneDrive\Desktop\AI associated coding'; & 'c:\Users\Summya Mehveen\AppData\Local\Programs
\Python\Python312\python.exe' 'c:\Users\Summya Mehveen\.vscode\extensions\ms-python.debugpy-20
25.10.0-win32-x64\bundled\libs\debugpy\launcher' '52301' '--' 'c:\Users\Summya Mehveen\OneDriv
e\Desktop\AI associated coding\11.1t7.py'
task2
task1
False
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>
```

Observation :

- The code defines a PriorityQueue class using Python’s heapq module to maintain a min-heap.
- Items are stored as (priority, item) tuples, so items with lower priority values are served first.
- The push method adds items with their priority to the queue.
- The pop method removes and returns the item with the highest priority (lowest value).
- The peek method returns the highest priority item without removing it.
- The is_empty method checks if the queue is empty.
- In the sample usage, 'task1' (priority 2) and 'task2' (priority 1) are added.
    - pop() returns 'task2' (since priority 1 < 2).
    - peek() returns 'task1'.
    - is_empty() returns False because one item remains.
- The code raises IndexError if pop or peek is called on an empty queue.

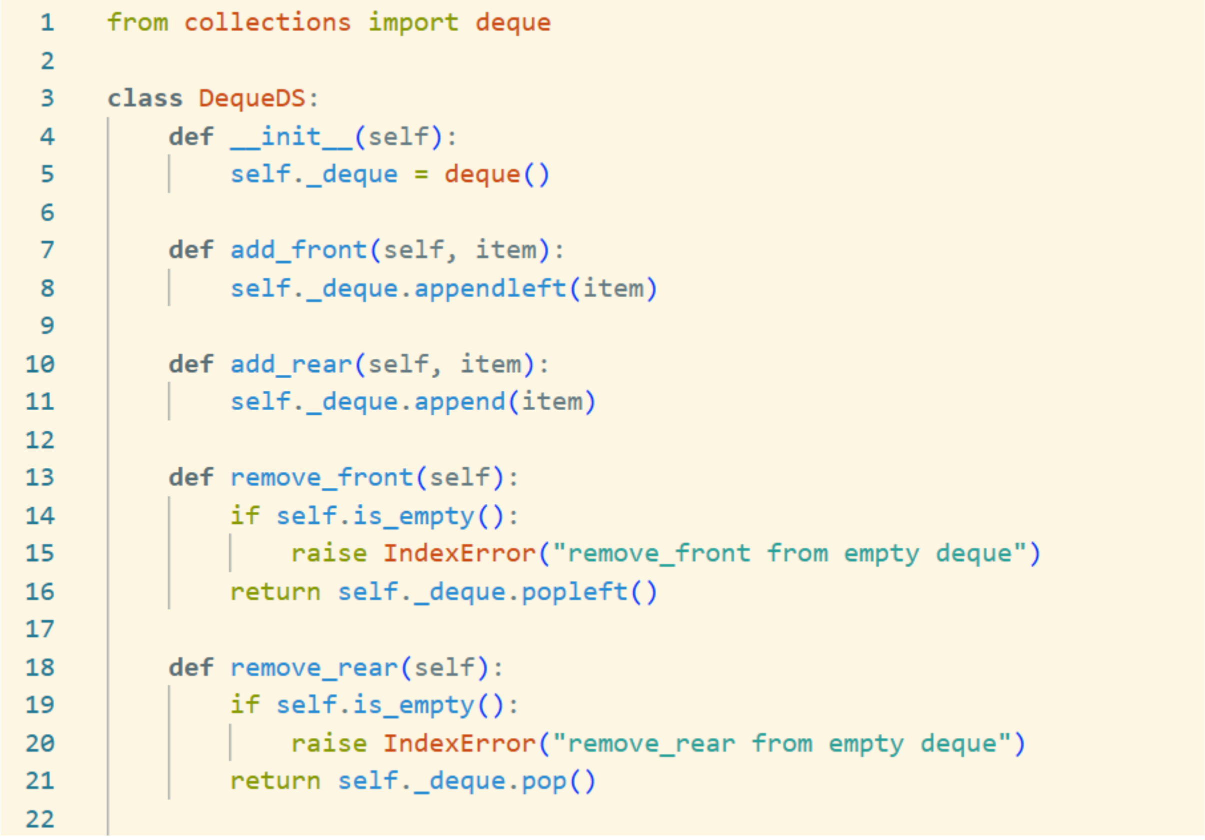
TASK 8 :

Prompt :

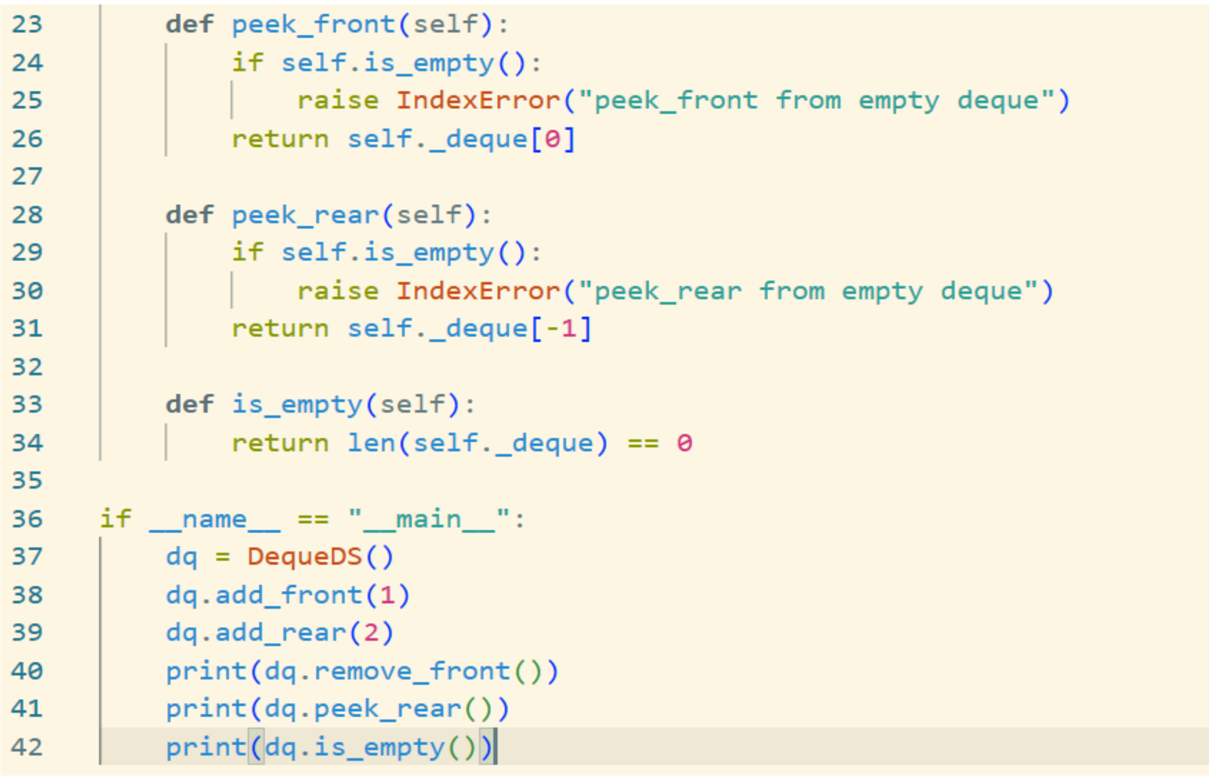
Implement a DequeDS class in Python using the built-in collections.deque module. Your class should support the following methods:

- 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 :

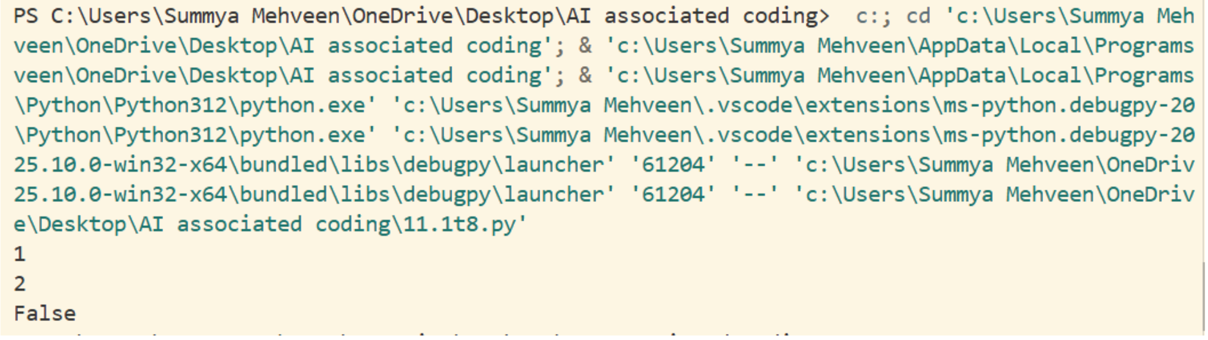


```python
from collections import deque

class DequeDS:
    def __init__(self):
        self._deque = deque()

    def add_front(self, item):
        self._deque.appendleft(item)

    def add_rear(self, item):
        self._deque.append(item)

    def remove_front(self):
        if self.is_empty():
            raise IndexError("remove_front from empty deque")
        return self._deque.popleft()

    def remove_rear(self):
        if self.is_empty():
            raise IndexError("remove_rear from empty deque")
        return self._deque.pop()

```



```python
    def peek_front(self):
        if self.is_empty():
            raise IndexError("peek_front from empty deque")
        return self._deque[0]

    def peek_rear(self):
        if self.is_empty():
            raise IndexError("peek_rear from empty deque")
        return self._deque[-1]

    def is_empty(self):
        return len(self._deque) == 0

if __name__ == "__main__":
    dq = DequeDS()
    dq.add_front(1)
    dq.add_rear(2)
    print(dq.remove_front())
    print(dq.peek_rear())
    print(dq.is_empty())
```

Output :



```
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>  c:; cd 'c:\Users\Summya Meh
veen\OneDrive\Desktop\AI associated coding'; & 'c:\Users\Summya Mehveen\AppData\Local\Programs
veen\OneDrive\Desktop\AI associated coding'; & 'c:\Users\Summya Mehveen\AppData\Local\Programs
\Python\Python312\python.exe' 'c:\Users\Summya Mehveen\.vscode\extensions\ms-python.debugpy-20
\Python\Python312\python.exe' 'c:\Users\Summya Mehveen\.vscode\extensions\ms-python.debugpy-20
25.10.0-win32-x64\bundled\libs\debugpy\launcher' '61204' '--' 'c:\Users\Summya Mehveen\OneDriv
25.10.0-win32-x64\bundled\libs\debugpy\launcher' '61204' '--' 'c:\Users\Summya Mehveen\OneDriv
e\Desktop\AI associated coding\11.1t8.py'
1
2
False
```

Observation :

- The code implements a double-ended queue (DequeDS) using Python’s collections.deque.
- 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:

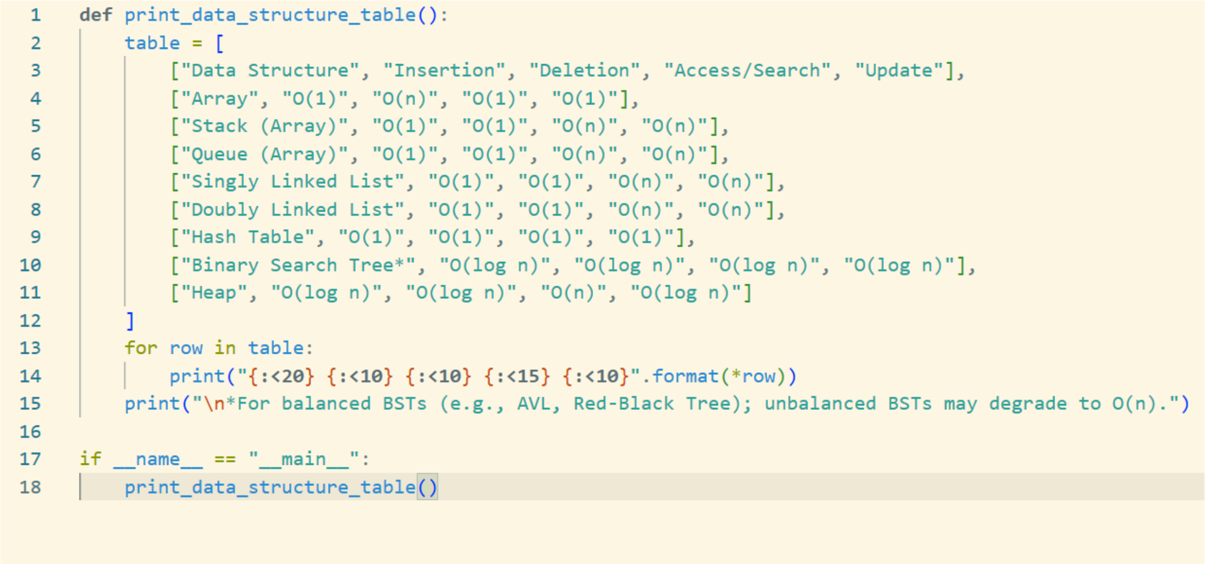
    - add_front(1) adds 1 to the front.
    - add_rear(2) adds 2 to the rear.
    - remove_front() removes and prints 1.
    - peek_rear() prints 2 (the only remaining item).
    - is_empty() prints False since one item remains.
- The code raises IndexError if removal or peek operations are attempted on an empty deque.

TASK 9 :

Prompt :

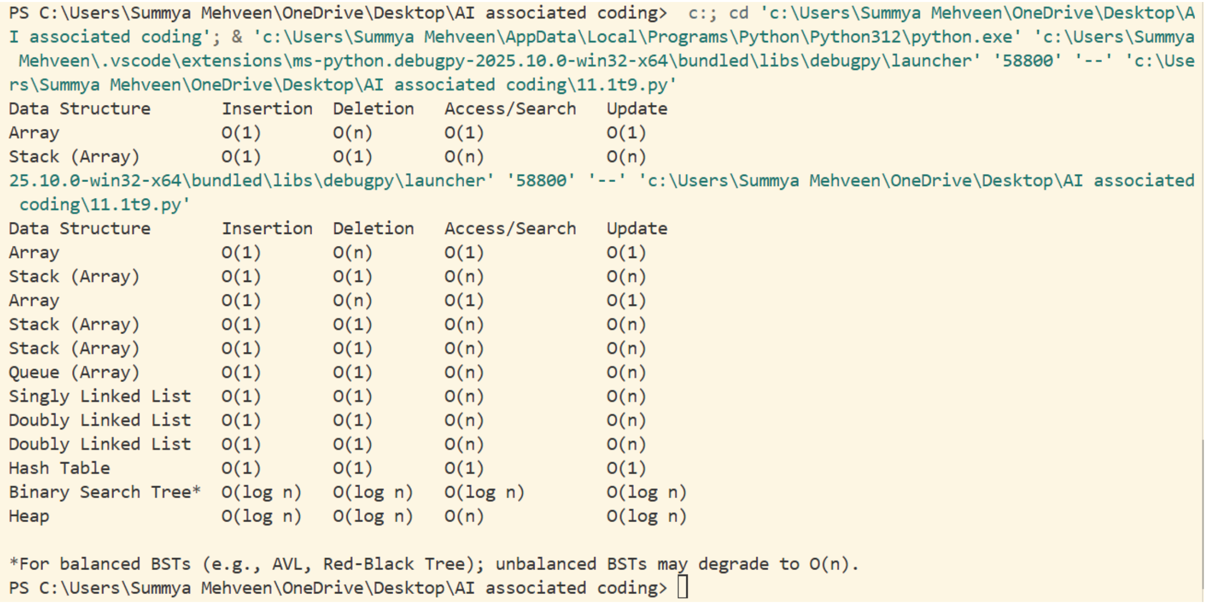
Generate a comparison table of different data structures (such as stack, queue, linked list, array, hash table, binary search tree, heap, etc.) including their typical time complexities for common operations like insertion, deletion, access/search, and update. Present the table in Markdown format.

Code Generated :



```python
def print_data_structure_table():
    table = [
        ["Data Structure", "Insertion", "Deletion", "Access/Search", "Update"],
        ["Array", "O(1)", "O(n)", "O(1)", "O(1)"],
        ["Stack (Array)", "O(1)", "O(1)", "O(n)", "O(n)"],
        ["Queue (Array)", "O(1)", "O(1)", "O(n)", "O(n)"],
        ["Singly Linked List", "O(1)", "O(1)", "O(n)", "O(n)"],
        ["Doubly Linked List", "O(1)", "O(1)", "O(n)", "O(n)"],
        ["Hash Table", "O(1)", "O(1)", "O(1)", "O(1)"],
        ["Binary Search Tree*", "O(log n)", "O(log n)", "O(log n)", "O(log n)"],
        ["Heap", "O(log n)", "O(log n)", "O(n)", "O(log n)"]
    ]
    for row in table:
        print("{:<20} {:<10} {:<10} {:<15} {:<10}".format(*row))
    print("\n*For balanced BSTs (e.g., AVL, Red-Black Tree); unbalanced BSTs may degrade to O(n).")

if __name__ == "__main__":
    print_data_structure_table()
```

Output :



```
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>  c:; cd 'c:\Users\Summya Mehveen\OneDrive\Desktop\A
I associated coding'; & 'c:\Users\Summya Mehveen\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\Summya
 Mehveen\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '58800' '--' 'c:\Use
rs\Summya Mehveen\OneDrive\Desktop\AI associated coding\11.1t9.py'
Data Structure      Insertion  Deletion   Access/Search   Update
Array               O(1)       O(n)       O(1)            O(1)
Stack (Array)       O(1)       O(1)       O(n)            O(n)
25.10.0-win32-x64\bundled\libs\debugpy\launcher' '58800' '--' 'c:\Users\Summya Mehveen\OneDrive\Desktop\AI associated
 coding\11.1t9.py'
Data Structure      Insertion  Deletion   Access/Search   Update
Array               O(1)       O(n)       O(1)            O(1)
Stack (Array)       O(1)       O(1)       O(n)            O(n)
Array               O(1)       O(n)       O(1)            O(1)
Stack (Array)       O(1)       O(1)       O(n)            O(n)
Stack (Array)       O(1)       O(1)       O(n)            O(n)
Queue (Array)       O(1)       O(1)       O(n)            O(n)
Singly Linked List  O(1)       O(1)       O(n)            O(n)
Doubly Linked List  O(1)       O(1)       O(n)            O(n)
Doubly Linked List  O(1)       O(1)       O(n)            O(n)
Hash Table          O(1)       O(1)       O(1)            O(1)
Binary Search Tree* O(log n)   O(log n)   O(log n)        O(log n)
Heap                O(log n)   O(log n)   O(n)            O(log n)

*For balanced BSTs (e.g., AVL, Red-Black Tree); unbalanced BSTs may degrade to O(n).
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding> 
```

Observation :

- The code defines a function print_data_structure_table() 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 :

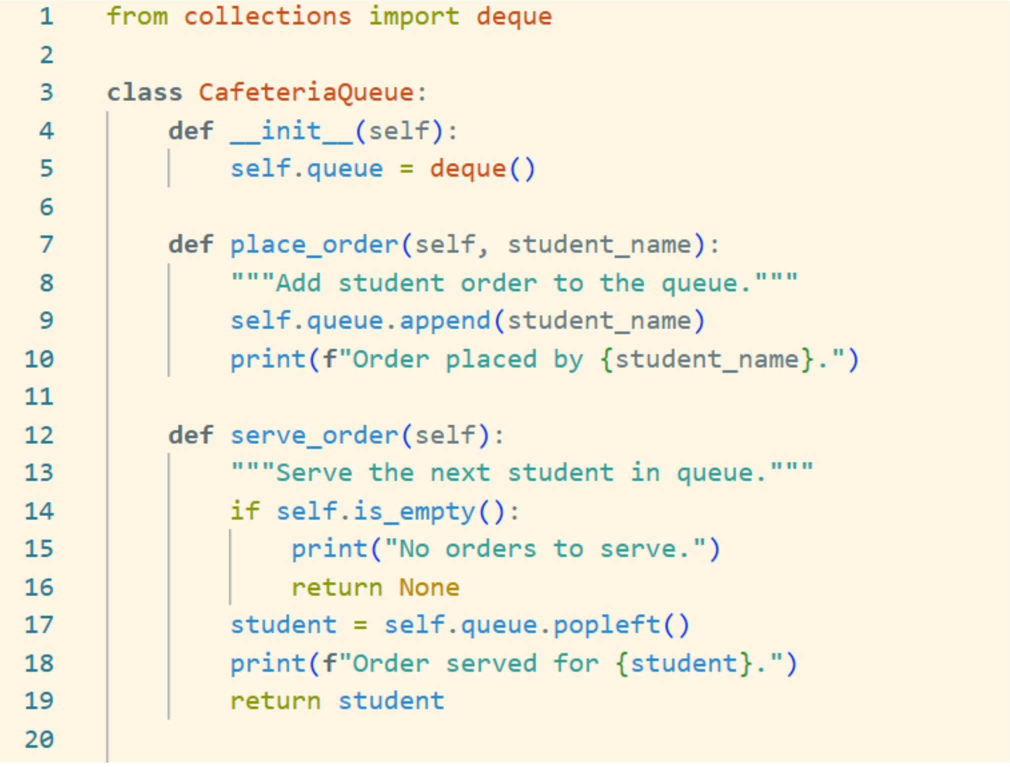
The Cafeteria Queue uses Python’s deque for efficient FIFO operations.

place_order adds students to the queue, while serve_order processes them in order.

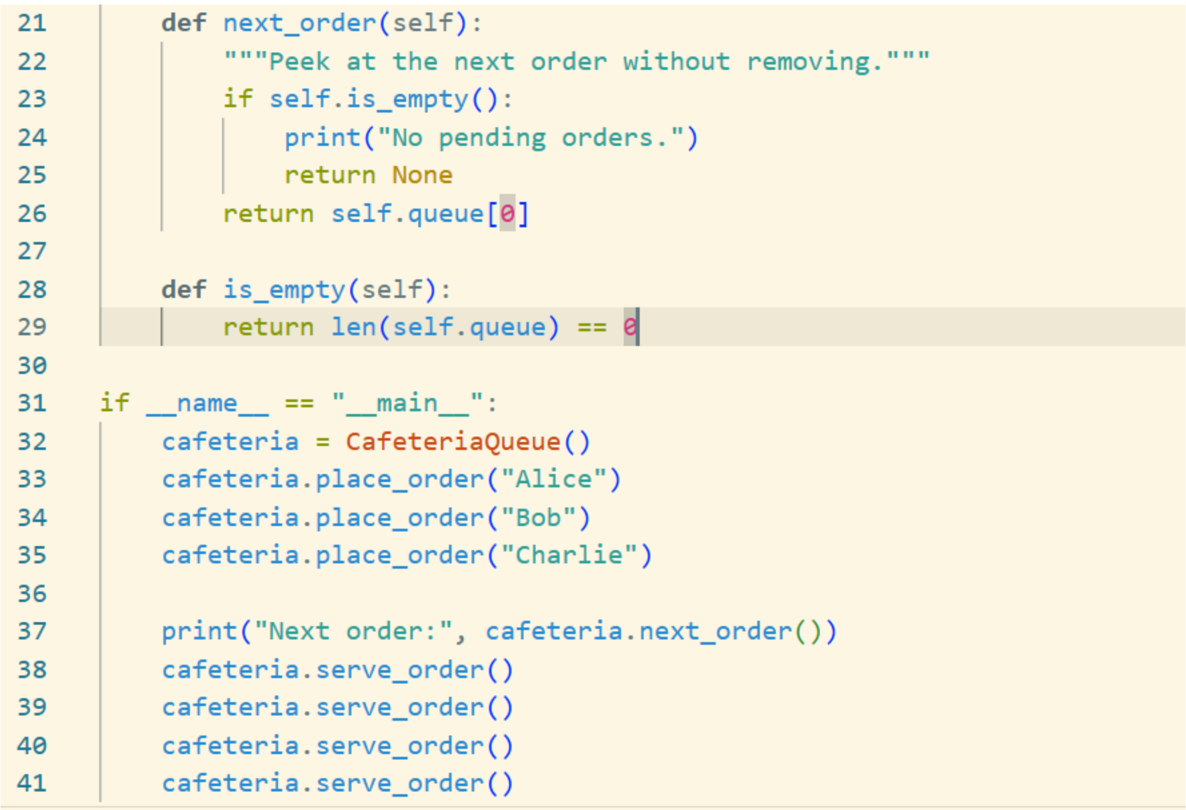
next_order allows checking the upcoming order without removing it.

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

Code Generated :

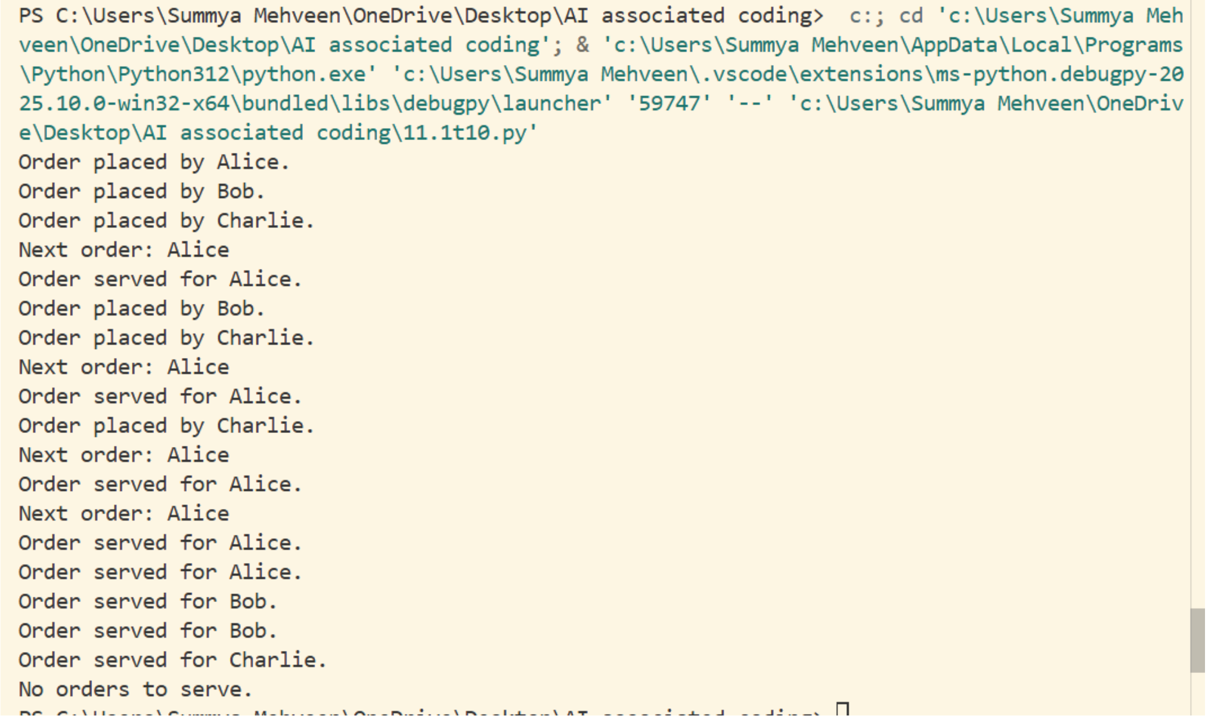


```python
from collections import deque

class CafeteriaQueue:
    def __init__(self):
        self.queue = deque()

    def place_order(self, student_name):
        """Add student order to the queue."""
        self.queue.append(student_name)
        print(f"Order placed by {student_name}.")

    def serve_order(self):
        """Serve the next student in queue."""
        if self.is_empty():
            print("No orders to serve.")
            return None
        student = self.queue.popleft()
        print(f"Order served for {student}.")
        return student

```



```python
    def next_order(self):
        """Peek at the next order without removing."""
        if self.is_empty():
            print("No pending orders.")
            return None
        return self.queue[0]

    def is_empty(self):
        return len(self.queue) == 0

if __name__ == "__main__":
    cafeteria = CafeteriaQueue()
    cafeteria.place_order("Alice")
    cafeteria.place_order("Bob")
    cafeteria.place_order("Charlie")

    print("Next order:", cafeteria.next_order())
    cafeteria.serve_order()
    cafeteria.serve_order()
    cafeteria.serve_order()
    cafeteria.serve_order()
```

Output :



```
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding>  c:; cd 'c:\Users\Summya Meh
veen\OneDrive\Desktop\AI associated coding'; & 'c:\Users\Summya Mehveen\AppData\Local\Programs
\Python\Python312\python.exe' 'c:\Users\Summya Mehveen\.vscode\extensions\ms-python.debugpy-20
25.10.0-win32-x64\bundled\libs\debugpy\launcher' '59747' '--' 'c:\Users\Summya Mehveen\OneDriv
e\Desktop\AI associated coding\11.1t10.py'
Order placed by Alice.
Order placed by Bob.
Order placed by Charlie.
Next order: Alice
Order served for Alice.
Order placed by Bob.
Order placed by Charlie.
Next order: Alice
Order served for Alice.
Order placed by Charlie.
Next order: Alice
Order served for Alice.
Next order: Alice
Order served for Alice.
Order served for Alice.
Order served for Bob.
Order served for Bob.
Order served for Charlie.
No orders to serve.
PS C:\Users\Summya Mehveen\OneDrive\Desktop\AI associated coding> 
```

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.