ASSIGNMENT-11.2

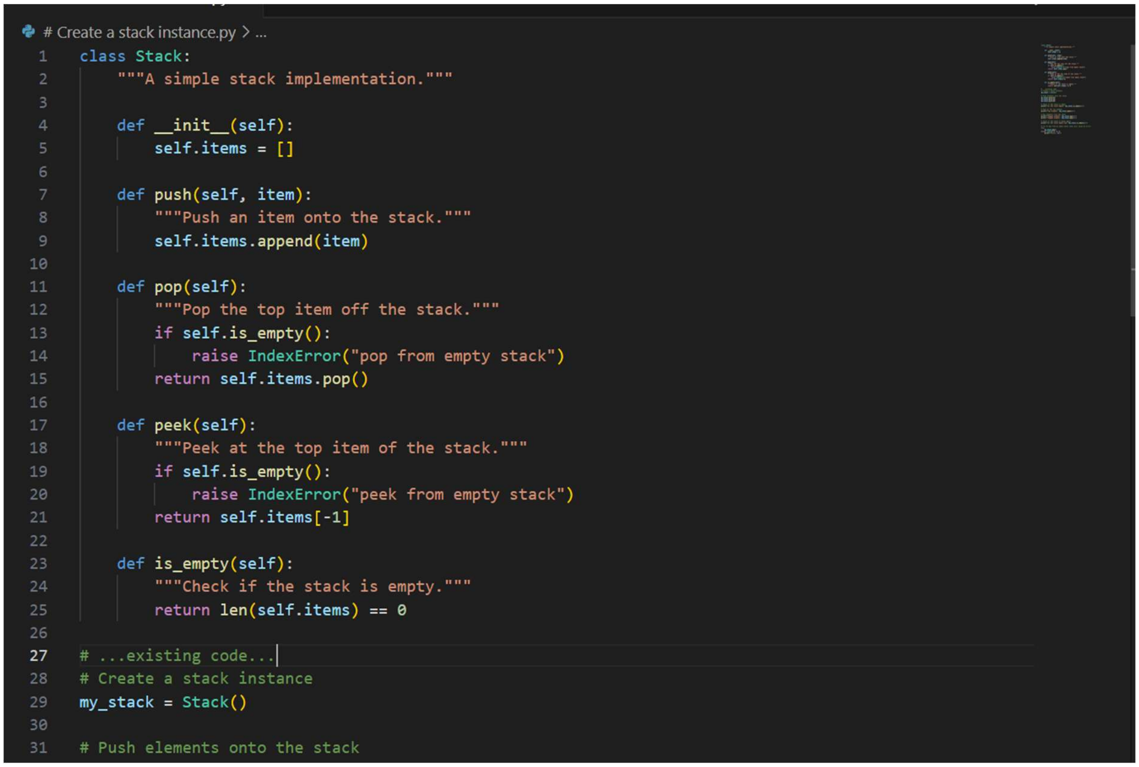
NAME:DEVI PRIYA.G

2403A52067

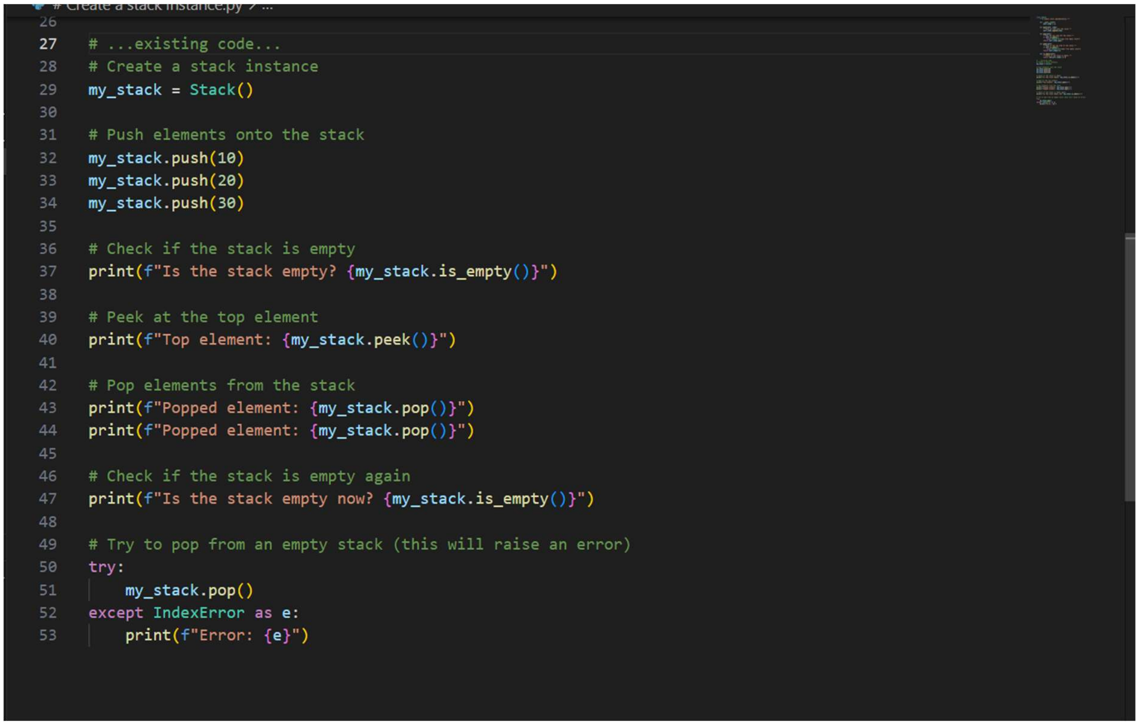
BATCH-04

TASK-1:

PROMPT: Use AI to generate a Stack class with push, pop, peek, and is_empty methods.

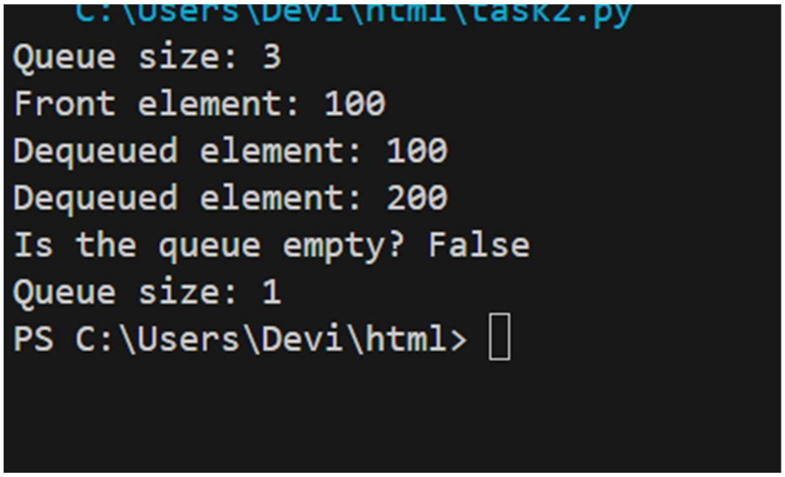


```python
class Stack:
    """A simple stack implementation."""

    def __init__(self):
        self.items = []

    def push(self, item):
        """Push an item onto the stack."""
        self.items.append(item)

    def pop(self):
        """Pop the top item off the stack."""
        if self.is_empty():
            raise IndexError("pop from empty stack")
        return self.items.pop()

    def peek(self):
        """Peek at the top item of the stack."""
        if self.is_empty():
            raise IndexError("peek from empty stack")
        return self.items[-1]

    def is_empty(self):
        """Check if the stack is empty."""
        return len(self.items) == 0

# ...existing code...
# Create a stack instance
my_stack = Stack()

# Push elements onto the stack
```



```python
# ...existing code...
# Create a stack instance
my_stack = Stack()

# Push elements onto the stack
my_stack.push(10)
my_stack.push(20)
my_stack.push(30)

# Check if the stack is empty
print(f"Is the stack empty? {my_stack.is_empty()}")

# Peek at the top element
print(f"Top element: {my_stack.peek()}")

# Pop elements from the stack
print(f"Popped element: {my_stack.pop()}")
print(f"Popped element: {my_stack.pop()}")

# Check if the stack is empty again
print(f"Is the stack empty now? {my_stack.is_empty()}")

# Try to pop from an empty stack (this will raise an error)
try:
    my_stack.pop()
except IndexError as e:
    print(f"Error: {e}")
```

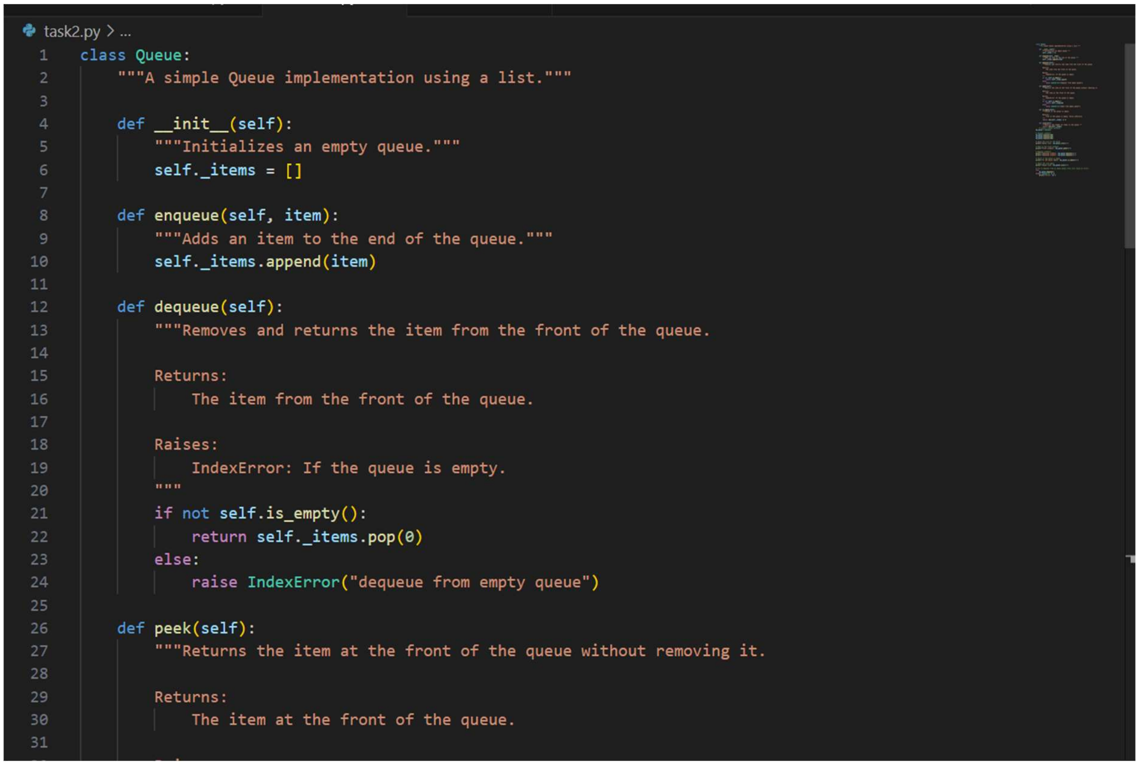
OUTPUT:



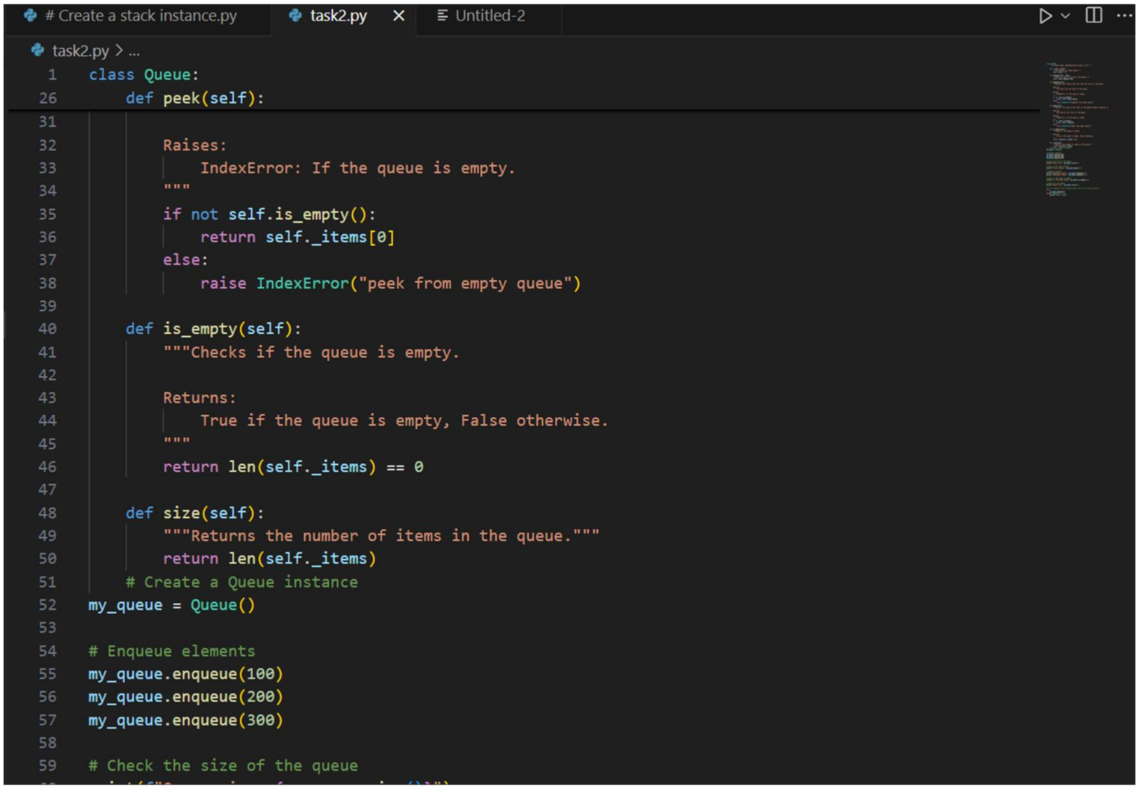
```
   C:\Users\Devi\html\task2.py
Queue size: 3
Front element: 100
Dequeued element: 100
Dequeued element: 200
Is the queue empty? False
Queue size: 1
PS C:\Users\Devi\html> 
```

TASK-2:

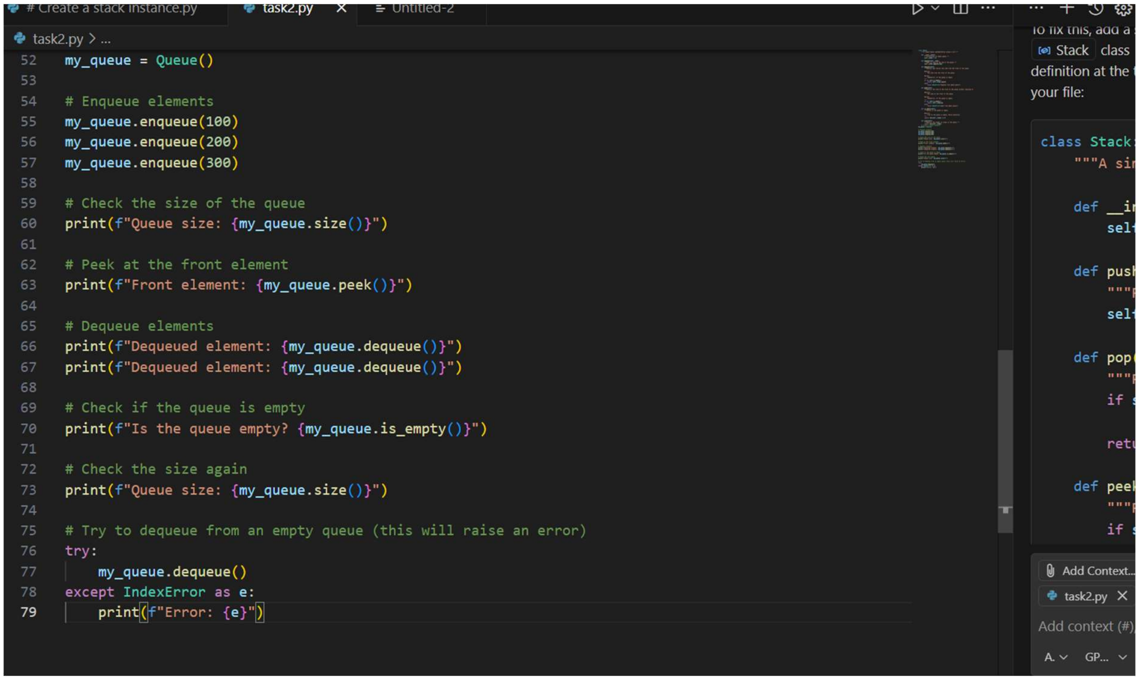
PROMPT: Use AI to implement a Queue using Python lists.



```python
class Queue:
    """A simple Queue implementation using a list."""

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

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

    def dequeue(self):
        """Removes and returns the item from the front of the queue.

        Returns:
            The item from the front of the queue.

        Raises:
            IndexError: If the queue is empty.
        """
        if not self.is_empty():
            return self._items.pop(0)
        else:
            raise IndexError("dequeue from empty queue")

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

        Returns:
            The item at the front of the queue.

```



```python
class Queue:
    def peek(self):

        Raises:
            IndexError: If the queue is empty.
        """
        if not self.is_empty():
            return self._items[0]
        else:
            raise IndexError("peek from empty queue")

    def is_empty(self):
        """Checks if the queue is empty.

        Returns:
            True if the queue is empty, False otherwise.
        """
        return len(self._items) == 0

    def size(self):
        """Returns the number of items in the queue."""
        return len(self._items)
    # Create a Queue instance
my_queue = Queue()

# Enqueue elements
my_queue.enqueue(100)
my_queue.enqueue(200)
my_queue.enqueue(300)

# Check the size of the queue
```

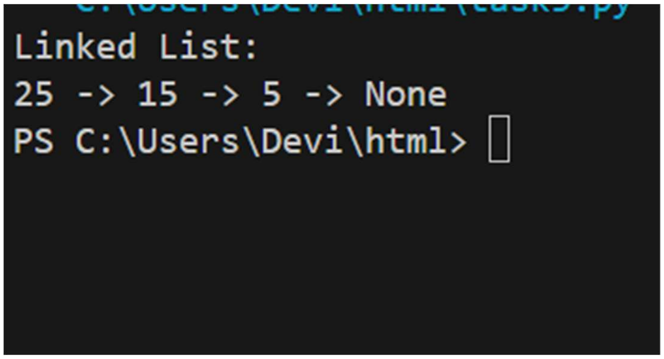


```python
my_queue = Queue()

# Enqueue elements
my_queue.enqueue(100)
my_queue.enqueue(200)
my_queue.enqueue(300)

# Check the size of the queue
print(f"Queue size: {my_queue.size()}")

# Peek at the front element
print(f"Front element: {my_queue.peek()}")

# Dequeue elements
print(f"Dequeued element: {my_queue.dequeue()}")
print(f"Dequeued element: {my_queue.dequeue()}")

# Check if the queue is empty
print(f"Is the queue empty? {my_queue.is_empty()}")

# Check the size again
print(f"Queue size: {my_queue.size()}")

# Try to dequeue from an empty queue (this will raise an error)
try:
    my_queue.dequeue()
except IndexError as e:
    print(f"Error: {e}")
```

To fix this, add a Stack class definition at the your file:

```
class Stack
    """A si

    def __i
        sel

    def pus
        """
        sel

    def pop
        """
        if 

        ret

    def pee
        """
        if 
```

OUTPUT:

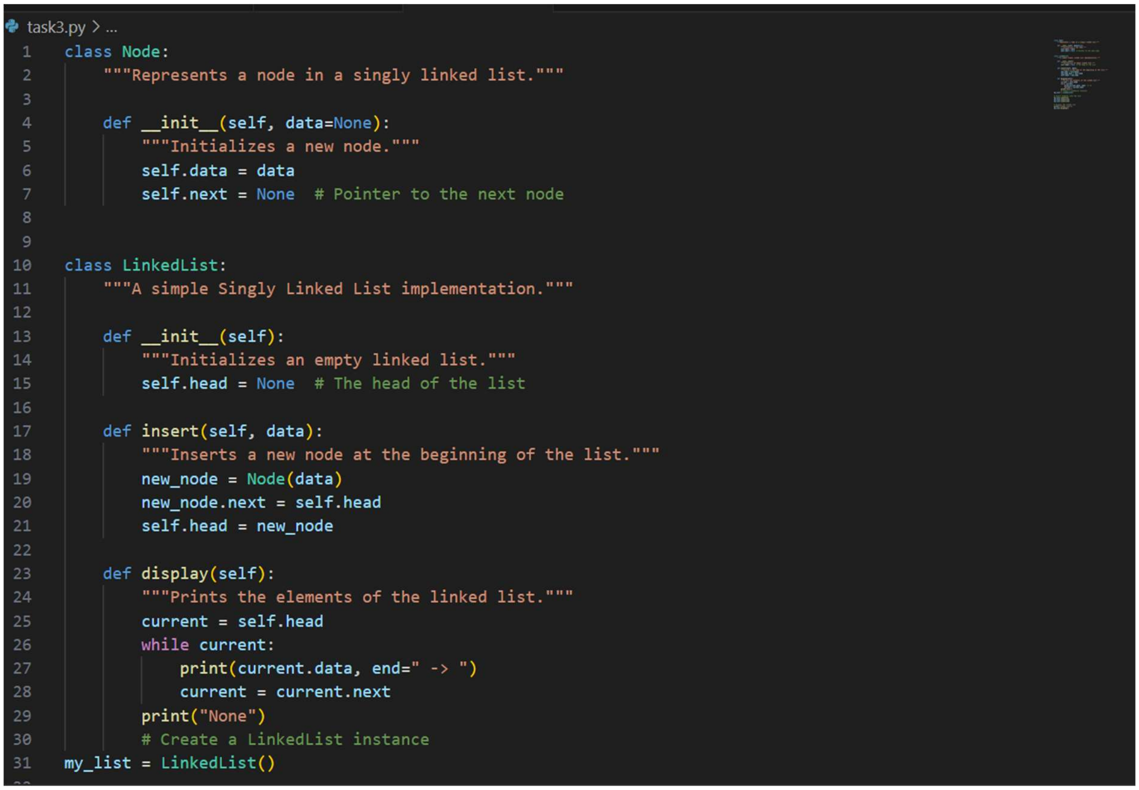


```
C:\Users\Devi\html\task3.py
Linked List:
25 -> 15 -> 5 -> None
PS C:\Users\Devi\html> 
```

TASK-3:

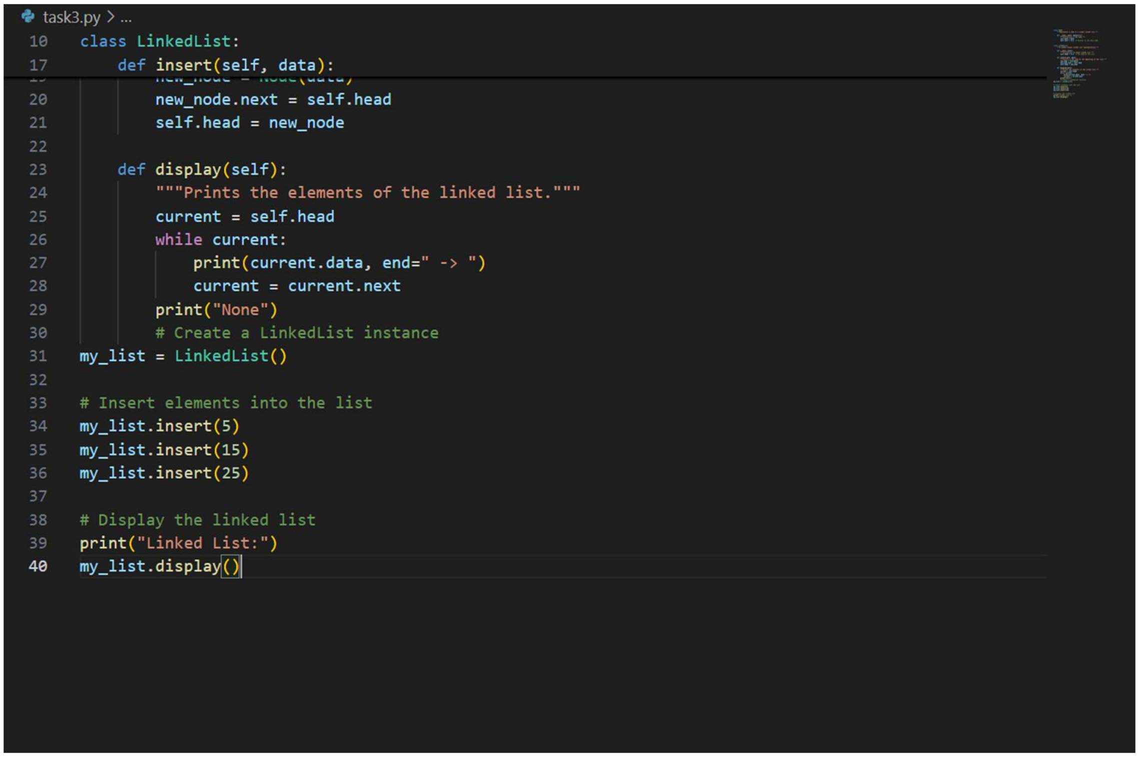
PROMPT:

Use AI to generate a Singly Linked List with insert and display methods



task3.py > ...

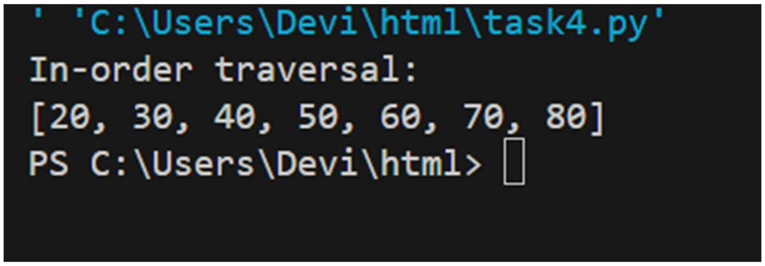
```python
class Node:
    """Represents a node in a singly linked list."""

    def __init__(self, data=None):
        """Initializes a new node."""
        self.data = data
        self.next = None  # Pointer to the next node


class LinkedList:
    """A simple Singly Linked List implementation."""

    def __init__(self):
        """Initializes an empty linked list."""
        self.head = None  # The head of the list

    def insert(self, data):
        """Inserts a new node at the beginning of the list."""
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node

    def display(self):
        """Prints the elements of the linked list."""
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")
        # Create a LinkedList instance
my_list = LinkedList()
```



task3.py > ...

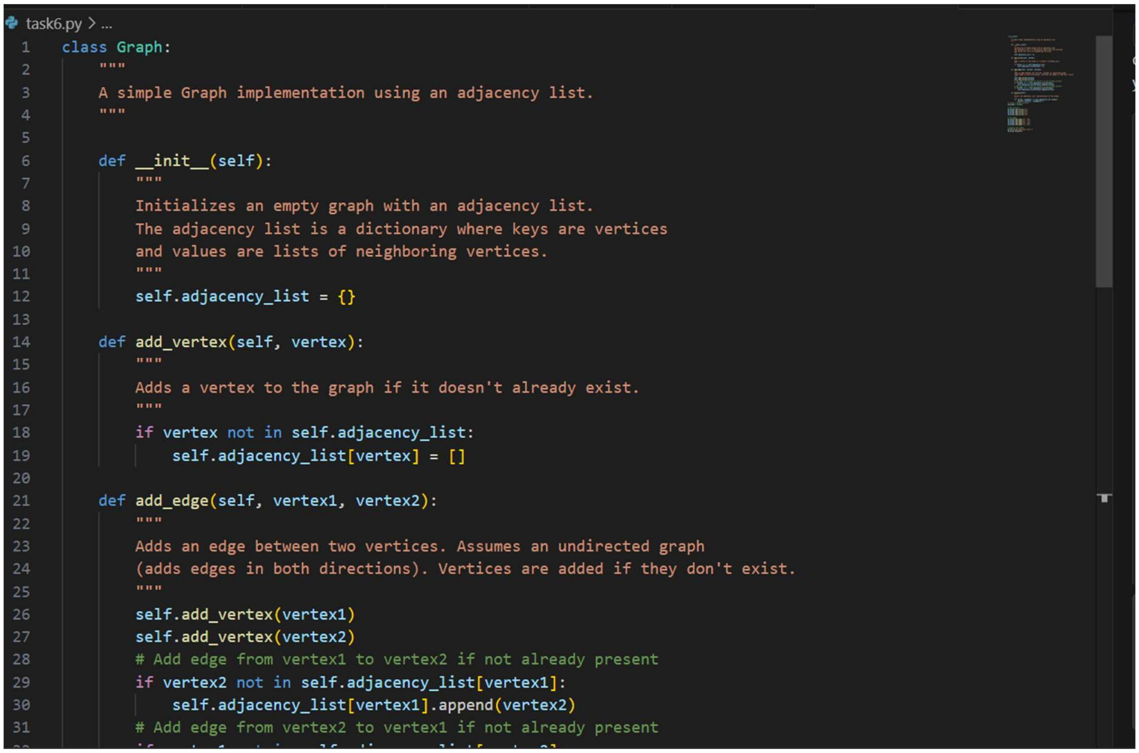
```python
class LinkedList:
    def insert(self, data):
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node

    def display(self):
        """Prints the elements of the linked list."""
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")
        # Create a LinkedList instance
my_list = LinkedList()

# Insert elements into the list
my_list.insert(5)
my_list.insert(15)
my_list.insert(25)

# Display the linked list
print("Linked List:")
my_list.display()
```

OUTPUT:



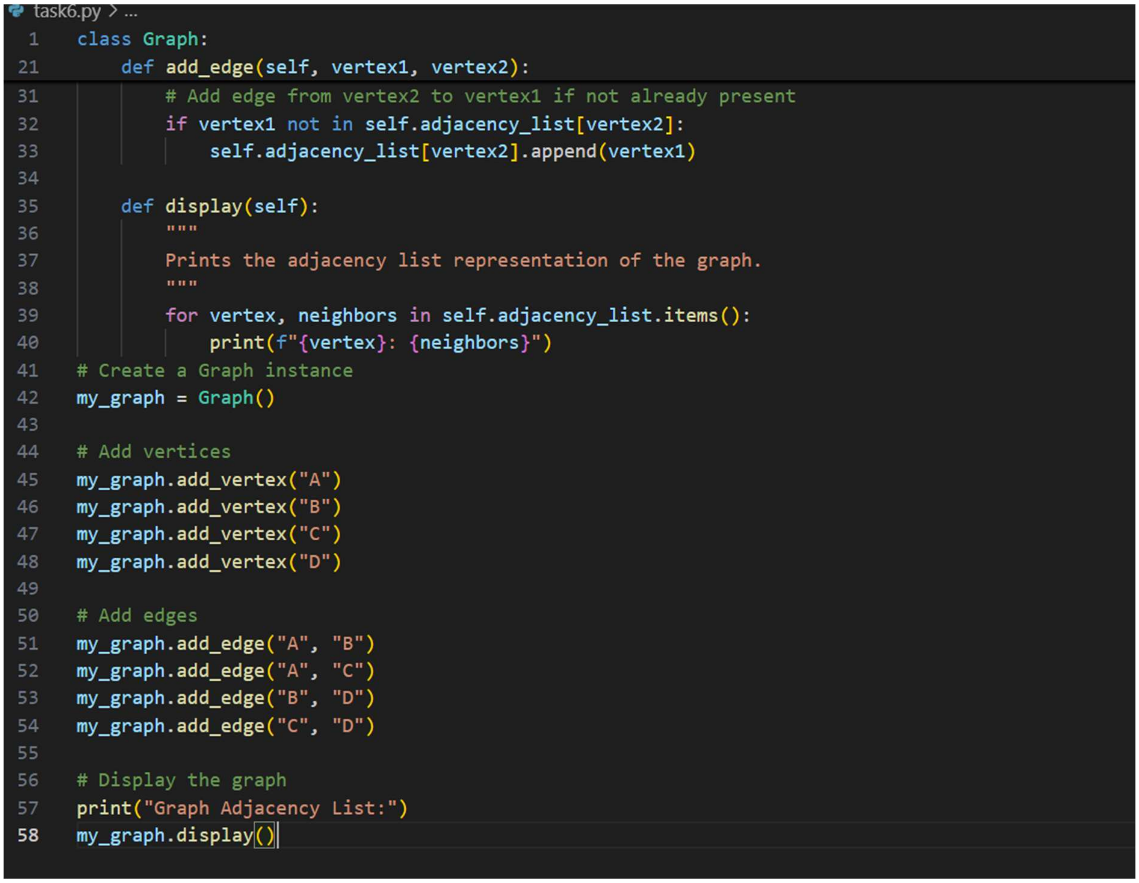
```
' 'C:\Users\Devi\html\task4.py'
In-order traversal:
[20, 30, 40, 50, 60, 70, 80]
PS C:\Users\Devi\html> 
```

TASK-4:



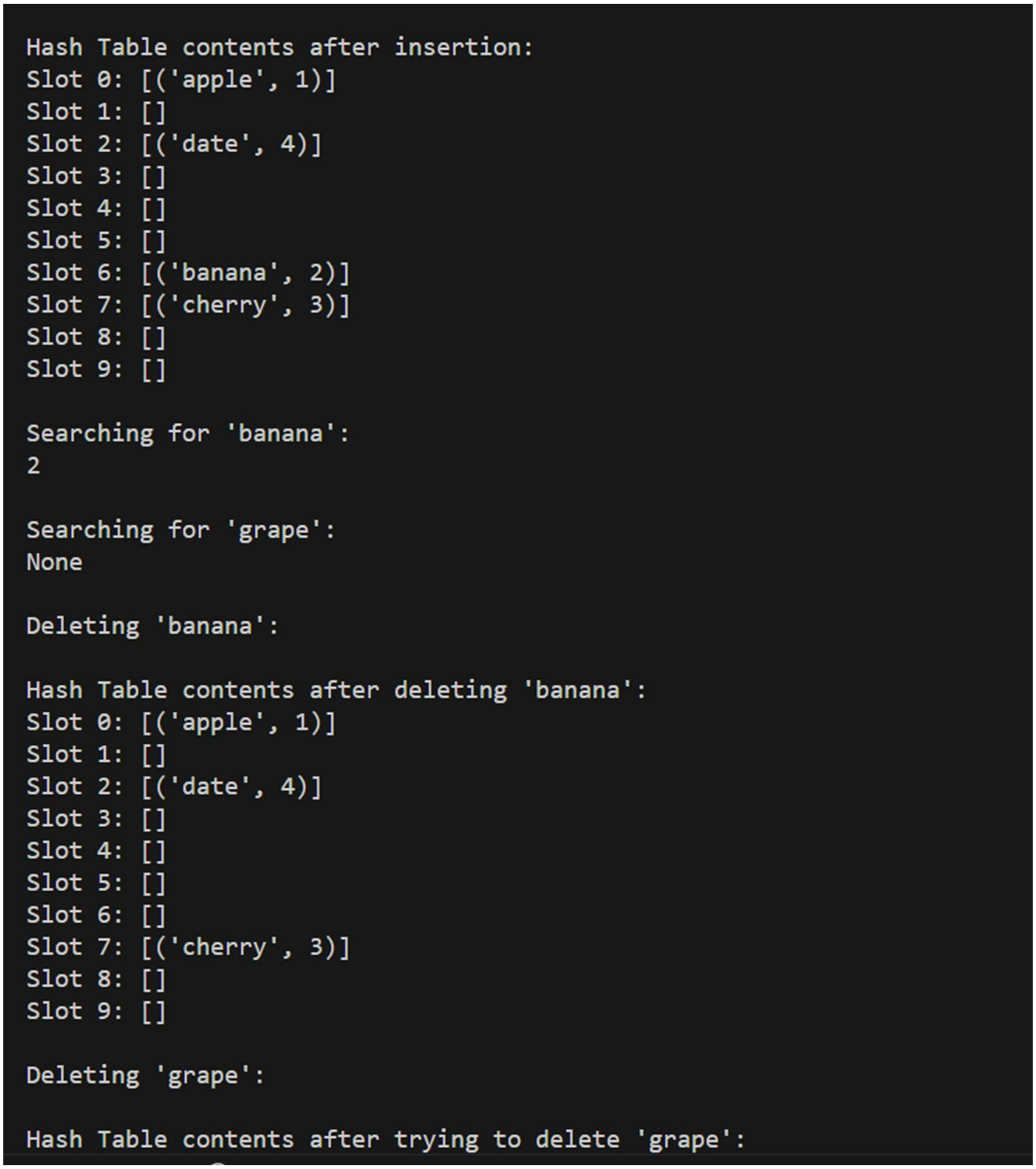
task6.py > ...

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

    def __init__(self):
        """
        Initializes an empty graph with an adjacency list.
        The adjacency list is a dictionary where keys are vertices
        and values are lists of neighboring vertices.
        """
        self.adjacency_list = {}

    def add_vertex(self, vertex):
        """
        Adds a vertex to the graph if it doesn't already exist.
        """
        if vertex not in self.adjacency_list:
            self.adjacency_list[vertex] = []

    def add_edge(self, vertex1, vertex2):
        """
        Adds an edge between two vertices. Assumes an undirected graph
        (adds edges in both directions). Vertices are added if they don't exist.
        """
        self.add_vertex(vertex1)
        self.add_vertex(vertex2)
        # Add edge from vertex1 to vertex2 if not already present
        if vertex2 not in self.adjacency_list[vertex1]:
            self.adjacency_list[vertex1].append(vertex2)
        # Add edge from vertex2 to vertex1 if not already present
```

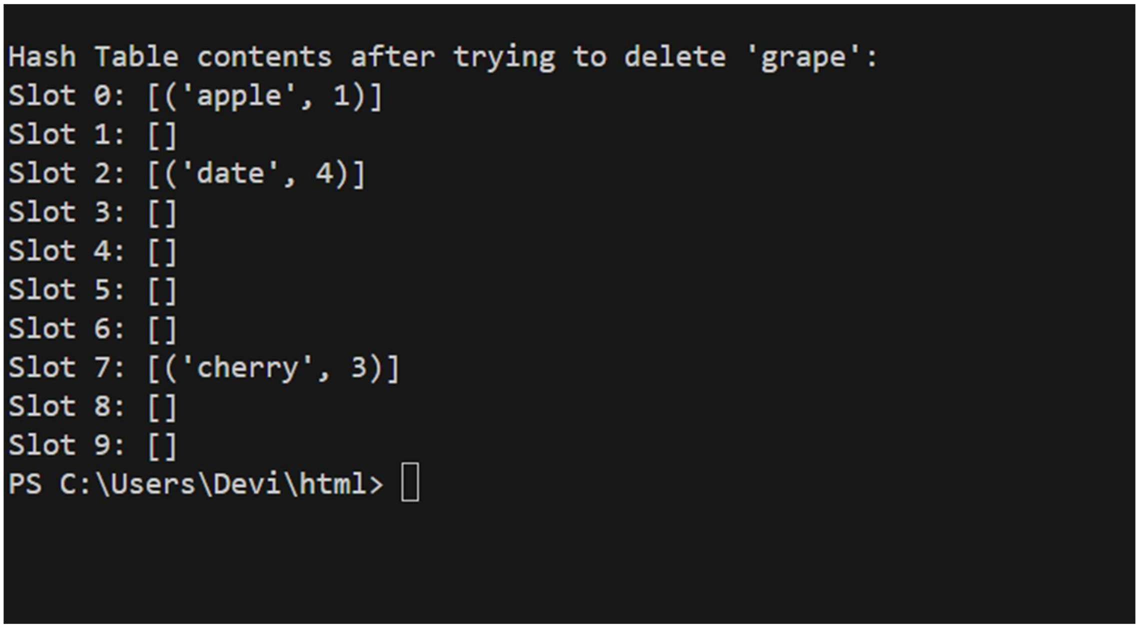


task6.py > ...

```python
class Graph:
    def add_edge(self, vertex1, vertex2):
        # Add edge from vertex2 to vertex1 if not already present
        if vertex1 not in self.adjacency_list[vertex2]:
            self.adjacency_list[vertex2].append(vertex1)

    def display(self):
        """
        Prints the adjacency list representation of the graph.
        """
        for vertex, neighbors in self.adjacency_list.items():
            print(f"{vertex}: {neighbors}")
# Create a Graph instance
my_graph = Graph()

# Add vertices
my_graph.add_vertex("A")
my_graph.add_vertex("B")
my_graph.add_vertex("C")
my_graph.add_vertex("D")

# Add edges
my_graph.add_edge("A", "B")
my_graph.add_edge("A", "C")
my_graph.add_edge("B", "D")
my_graph.add_edge("C", "D")

# Display the graph
print("Graph Adjacency List:")
my_graph.display()
```

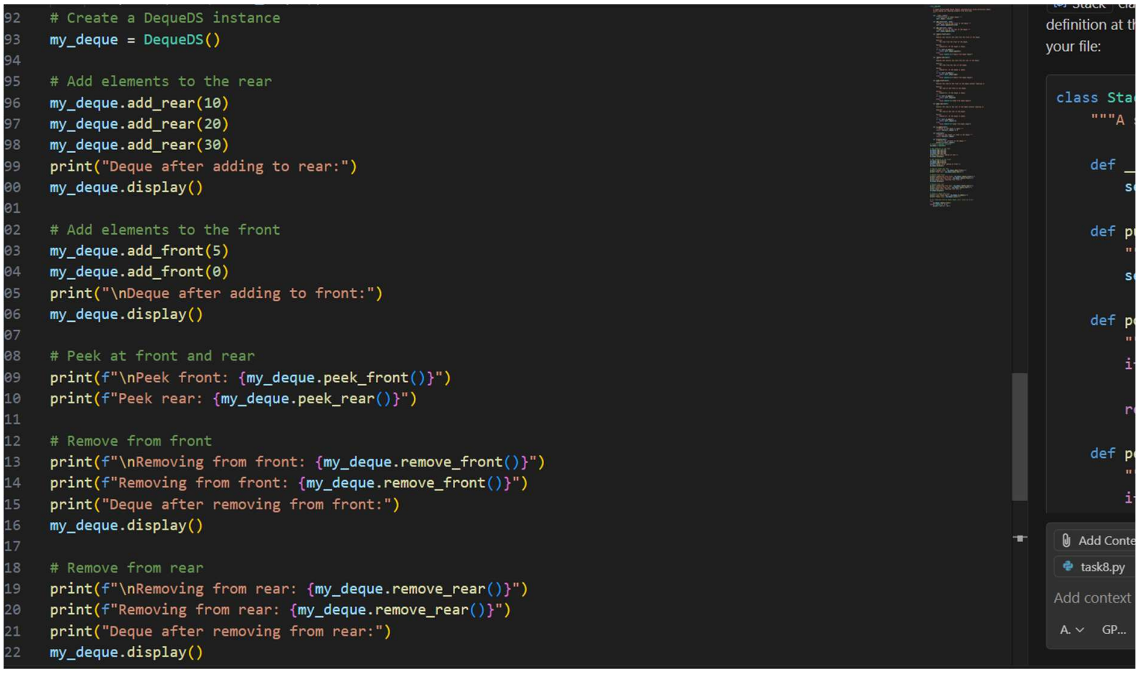


```
Hash Table contents after insertion:
Slot 0: [('apple', 1)]
Slot 1: []
Slot 2: [('date', 4)]
Slot 3: []
Slot 4: []
Slot 5: []
Slot 6: [('banana', 2)]
Slot 7: [('cherry', 3)]
Slot 8: []
Slot 9: []

Searching for 'banana':
2

Searching for 'grape':
None

Deleting 'banana':

Hash Table contents after deleting 'banana':
Slot 0: [('apple', 1)]
Slot 1: []
Slot 2: [('date', 4)]
Slot 3: []
Slot 4: []
Slot 5: []
Slot 6: []
Slot 7: [('cherry', 3)]
Slot 8: []
Slot 9: []

Deleting 'grape':

Hash Table contents after trying to delete 'grape':
```

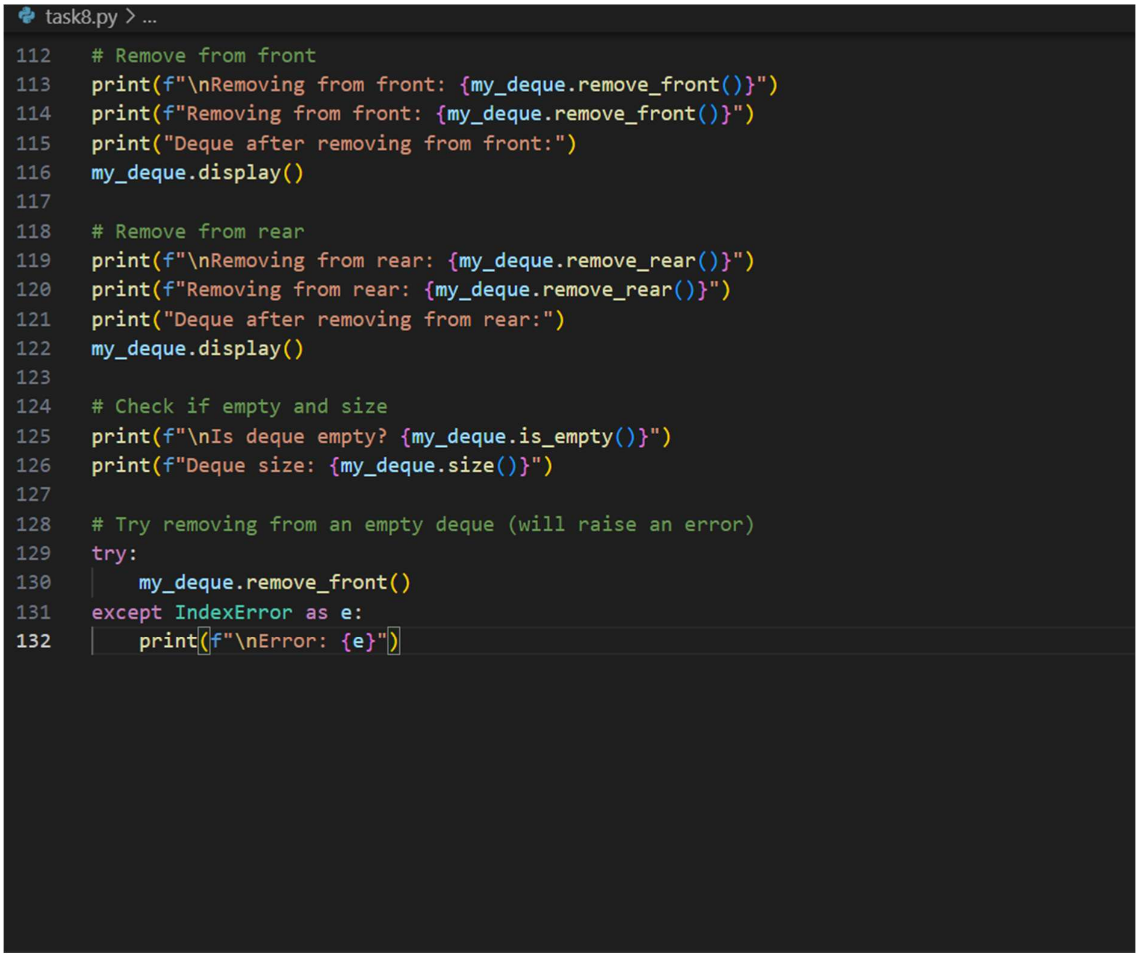


```
Hash Table contents after trying to delete 'grape':
Slot 0: [('apple', 1)]
Slot 1: []
Slot 2: [('date', 4)]
Slot 3: []
Slot 4: []
Slot 5: []
Slot 6: []
Slot 7: [('cherry', 3)]
Slot 8: []
Slot 9: []
PS C:\Users\Devi\html> 
```

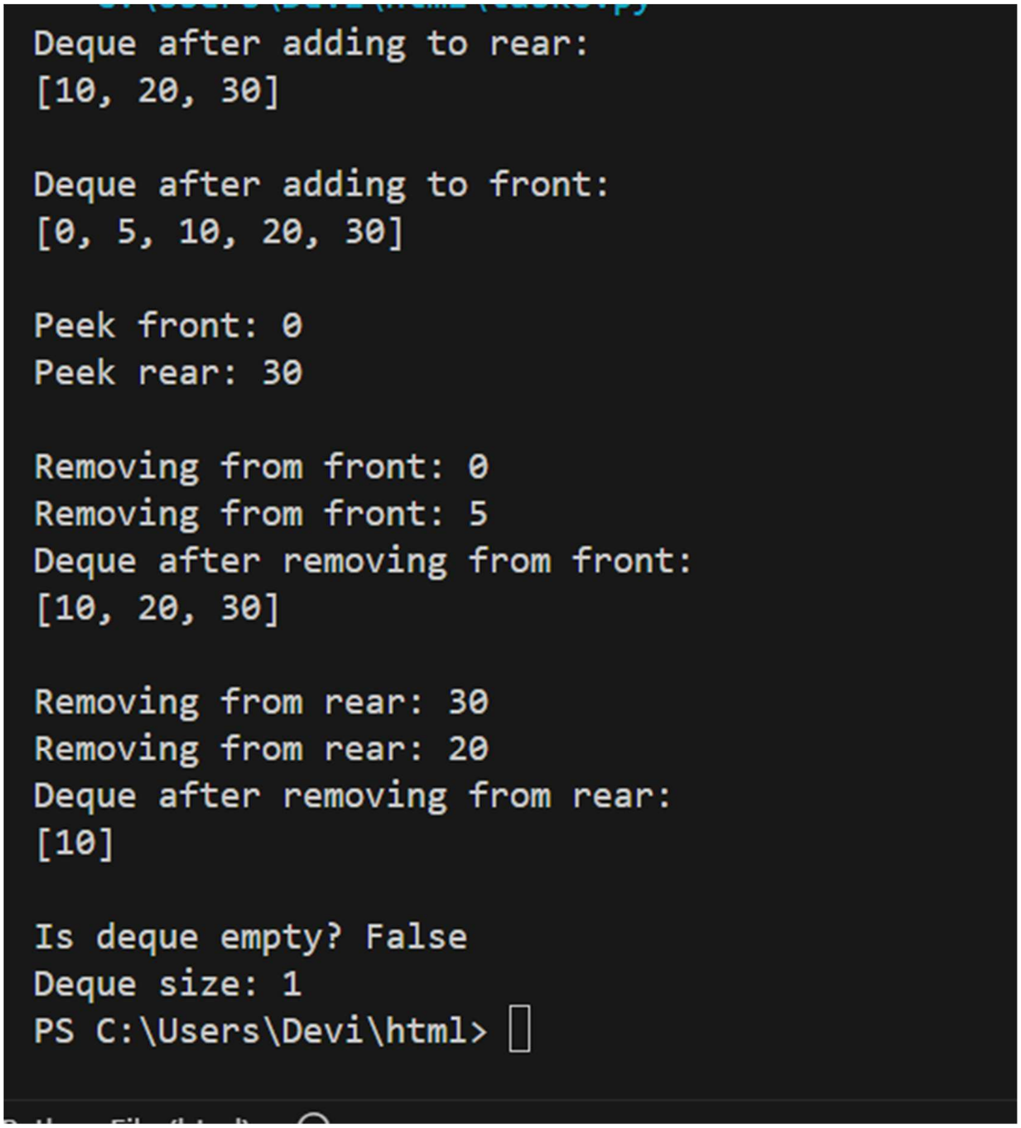
Task-5:



```python
# Create a DequeDS instance
my_deque = DequeDS()

# Add elements to the rear
my_deque.add_rear(10)
my_deque.add_rear(20)
my_deque.add_rear(30)
print("Deque after adding to rear:")
my_deque.display()

# Add elements to the front
my_deque.add_front(5)
my_deque.add_front(0)
print("\nDeque after adding to front:")
my_deque.display()

# Peek at front and rear
print(f"\nPeek front: {my_deque.peek_front()}")
print(f"Peek rear: {my_deque.peek_rear()}")

# Remove from front
print(f"\nRemoving from front: {my_deque.remove_front()}")
print(f"Removing from front: {my_deque.remove_front()}")
print("Deque after removing from front:")
my_deque.display()

# Remove from rear
print(f"\nRemoving from rear: {my_deque.remove_rear()}")
print(f"Removing from rear: {my_deque.remove_rear()}")
print("Deque after removing from rear:")
my_deque.display()
```

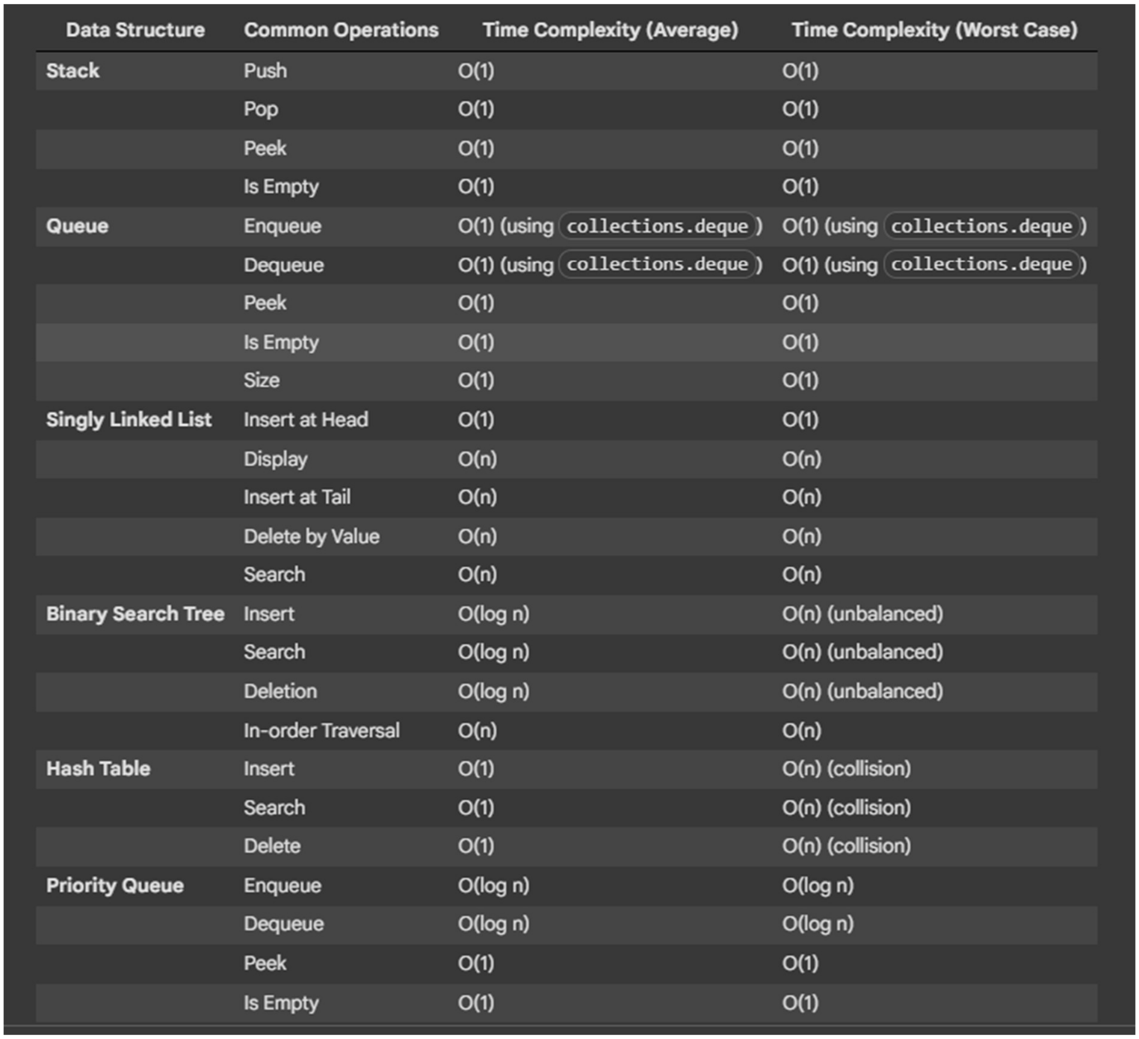


```python
# Remove from front
print(f"\nRemoving from front: {my_deque.remove_front()}")
print(f"Removing from front: {my_deque.remove_front()}")
print("Deque after removing from front:")
my_deque.display()

# Remove from rear
print(f"\nRemoving from rear: {my_deque.remove_rear()}")
print(f"Removing from rear: {my_deque.remove_rear()}")
print("Deque after removing from rear:")
my_deque.display()

# Check if empty and size
print(f"\nIs deque empty? {my_deque.is_empty()}")
print(f"Deque size: {my_deque.size()}")

# Try removing from an empty deque (will raise an error)
try:
    my_deque.remove_front()
except IndexError as e:
    print(f"\nError: {e}")
```

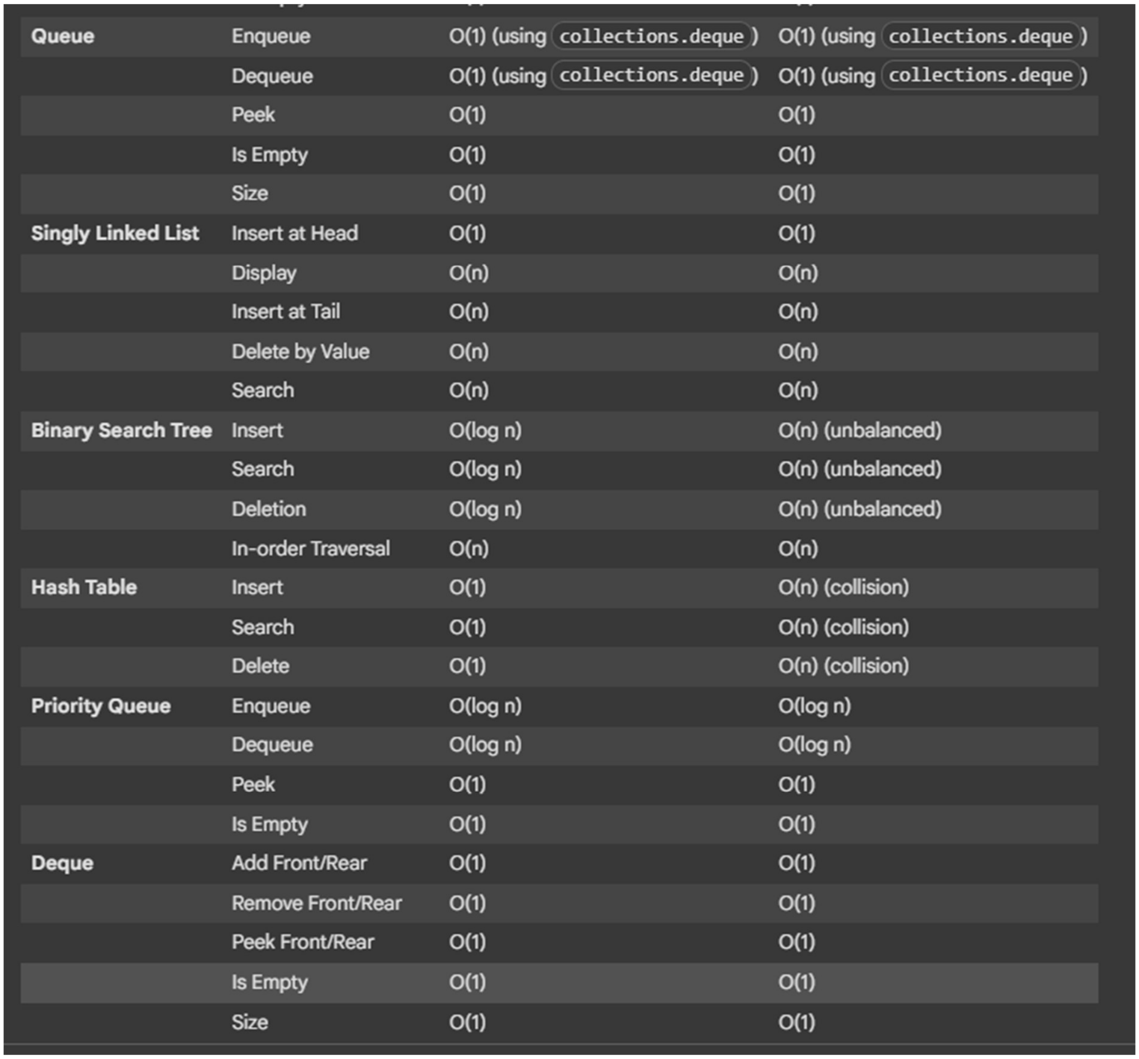


```
Deque after adding to rear:
[10, 20, 30]

Deque after adding to front:
[0, 5, 10, 20, 30]

Peek front: 0
Peek rear: 30

Removing from front: 0
Removing from front: 5
Deque after removing from front:
[10, 20, 30]

Removing from rear: 30
Removing from rear: 20
Deque after removing from rear:
[10]

Is deque empty? False
Deque size: 1
PS C:\Users\Devi\html> 
```

Task-9:



Data Structure	Common Operations	Time Complexity (Average)	Time Complexity (Worst Case)
Stack	Push	O(1)	O(1)
	Pop	O(1)	O(1)
	Peek	O(1)	O(1)
	Is Empty	O(1)	O(1)
Queue	Enqueue	O(1) (using `collections.deque`)	O(1) (using `collections.deque`)
	Dequeue	O(1) (using `collections.deque`)	O(1) (using `collections.deque`)
	Peek	O(1)	O(1)
	Is Empty	O(1)	O(1)
	Size	O(1)	O(1)
Singly Linked List	Insert at Head	O(1)	O(1)
	Display	O(n)	O(n)
	Insert at Tail	O(n)	O(n)
	Delete by Value	O(n)	O(n)
	Search	O(n)	O(n)
Binary Search Tree	Insert	O(log n)	O(n) (unbalanced)
	Search	O(log n)	O(n) (unbalanced)
	Deletion	O(log n)	O(n) (unbalanced)
	In-order Traversal	O(n)	O(n)
Hash Table	Insert	O(1)	O(n) (collision)
	Search	O(1)	O(n) (collision)
	Delete	O(1)	O(n) (collision)
Priority Queue	Enqueue	O(log n)	O(log n)
	Dequeue	O(log n)	O(log n)
	Peek	O(1)	O(1)
	Is Empty	O(1)	O(1)



Queue	Enqueue	O(1) (using `collections.deque`)	O(1) (using `collections.deque`)
	Dequeue	O(1) (using `collections.deque`)	O(1) (using `collections.deque`)
	Peek	O(1)	O(1)
	Is Empty	O(1)	O(1)
	Size	O(1)	O(1)
Singly Linked List	Insert at Head	O(1)	O(1)
	Display	O(n)	O(n)
	Insert at Tail	O(n)	O(n)
	Delete by Value	O(n)	O(n)
	Search	O(n)	O(n)
Binary Search Tree	Insert	O(log n)	O(n) (unbalanced)
	Search	O(log n)	O(n) (unbalanced)
	Deletion	O(log n)	O(n) (unbalanced)
	In-order Traversal	O(n)	O(n)
Hash Table	Insert	O(1)	O(n) (collision)
	Search	O(1)	O(n) (collision)
	Delete	O(1)	O(n) (collision)
Priority Queue	Enqueue	O(log n)	O(log n)
	Dequeue	O(log n)	O(log n)
	Peek	O(1)	O(1)
	Is Empty	O(1)	O(1)
Deque	Add Front/Rear	O(1)	O(1)
	Remove Front/Rear	O(1)	O(1)
	Peek Front/Rear	O(1)	O(1)
	Is Empty	O(1)	O(1)
	Size	O(1)	O(1)