ASSIGNMENT 11.1

Data Structures with AI: Implementing Fundamental Structures

Atla Sreeja

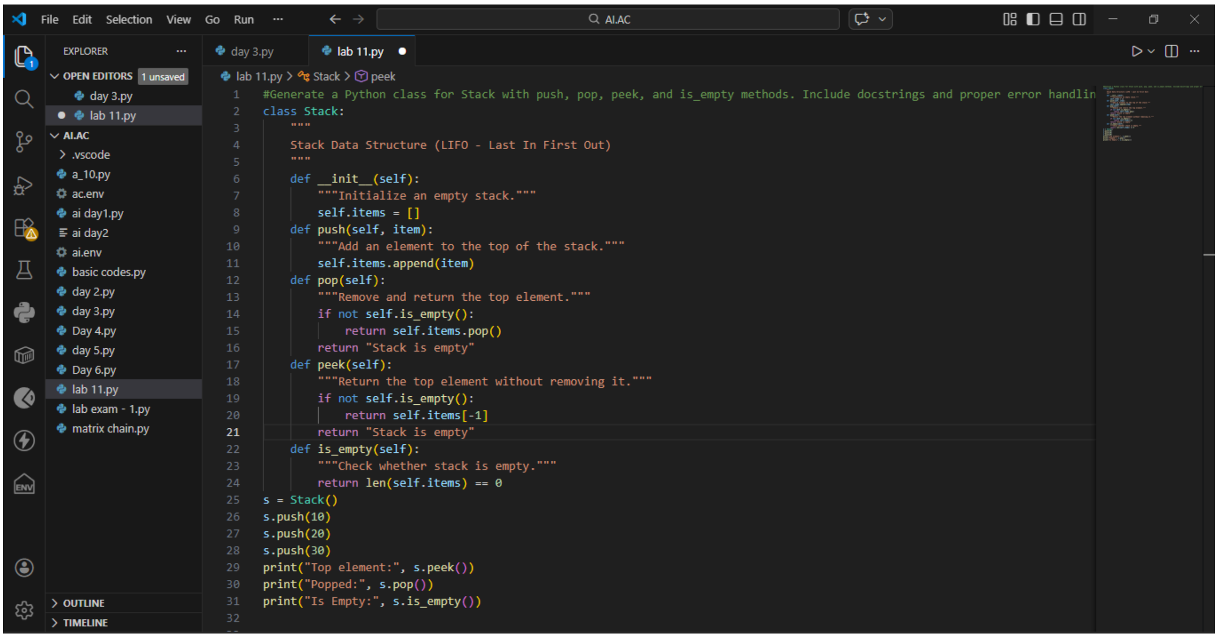
2403A51L02

B-51

Task 1: Stack Implementation

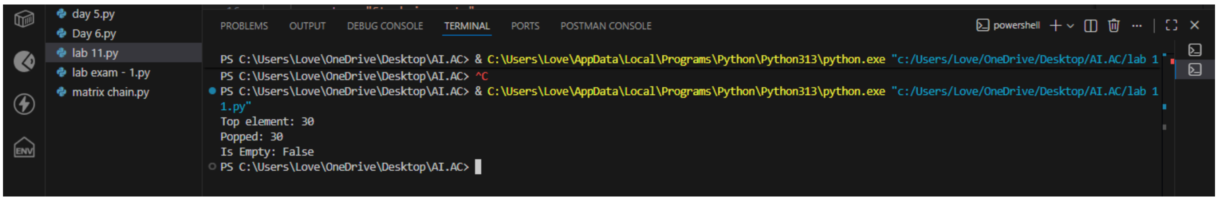
Task: Use AI to generate a Stack class with push, pop, peek, and is_empty methods.

Prompt: Generate a Python class for Stack with push, pop, peek, and is_empty methods. Include docstrings and proper error handling.



```python
#Generate a Python class for Stack with push, pop, peek, and is_empty methods. Include docstrings and proper error handlin
class Stack:
    """
    Stack Data Structure (LIFO - Last In First Out)
    """
    def __init__(self):
        """Initialize an empty stack."""
        self.items = []
    def push(self, item):
        """Add an element to the top of the stack."""
        self.items.append(item)
    def pop(self):
        """Remove and return the top element."""
        if not self.is_empty():
            return self.items.pop()
        return "Stack is empty"
    def peek(self):
        """Return the top element without removing it."""
        if not self.is_empty():
            return self.items[-1]
        return "Stack is empty"
    def is_empty(self):
        """Check whether stack is empty."""
        return len(self.items) == 0
s = Stack()
s.push(10)
s.push(20)
s.push(30)
print("Top element:", s.peek())
print("Popped:", s.pop())
print("Is Empty:", s.is_empty())
```

OUTPUT:



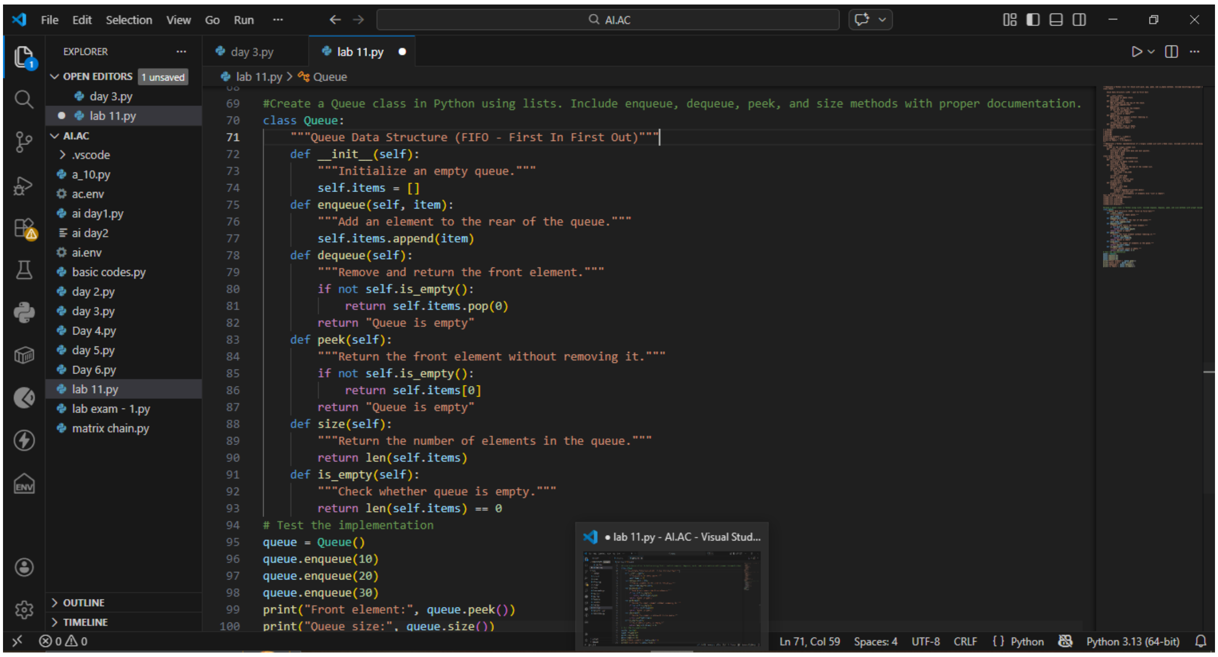
```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 1
PS C:\Users\Love\OneDrive\Desktop\AI.AC> ^C
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 1
1.py"
Top element: 30
Popped: 30
Is Empty: False
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

Explanation: A Stack is a linear data structure that follows the LIFO (Last In First Out) principle, where the last element inserted is the first one removed. Operations such as push, pop, and peek are performed at one end called the top. It is commonly used in function calls, undo operations, and expression evaluation.

Task Description #2: Queue Implementation

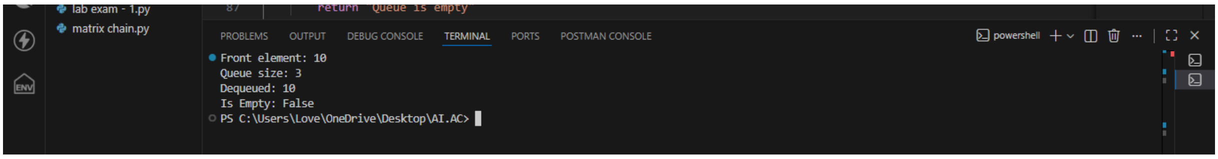
Task: Use AI to implement a Queue using Python lists.

Prompt: Create a Queue class in Python using lists. Include enqueue,dequeue, peek, and size methods with proper documentation.



```python
#Create a Queue class in Python using lists. Include enqueue, dequeue, peek, and size methods with proper documentation.
class Queue:
    """Queue Data Structure (FIFO - First In First Out)"""
    def __init__(self):
        """Initialize an empty queue."""
        self.items = []
    def enqueue(self, item):
        """Add an element to the rear of the queue."""
        self.items.append(item)
    def dequeue(self):
        """Remove and return the front element."""
        if not self.is_empty():
            return self.items.pop(0)
        return "Queue is empty"
    def peek(self):
        """Return the front element without removing it."""
        if not self.is_empty():
            return self.items[0]
        return "Queue is empty"
    def size(self):
        """Return the number of elements in the queue."""
        return len(self.items)
    def is_empty(self):
        """Check whether queue is empty."""
        return len(self.items) == 0
# Test the implementation
queue = Queue()
queue.enqueue(10)
queue.enqueue(20)
queue.enqueue(30)
print("Front element:", queue.peek())
print("Queue size:", queue.size())
```

OUTPUT:



lab exam - 1.py
matrix chain.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
powershell

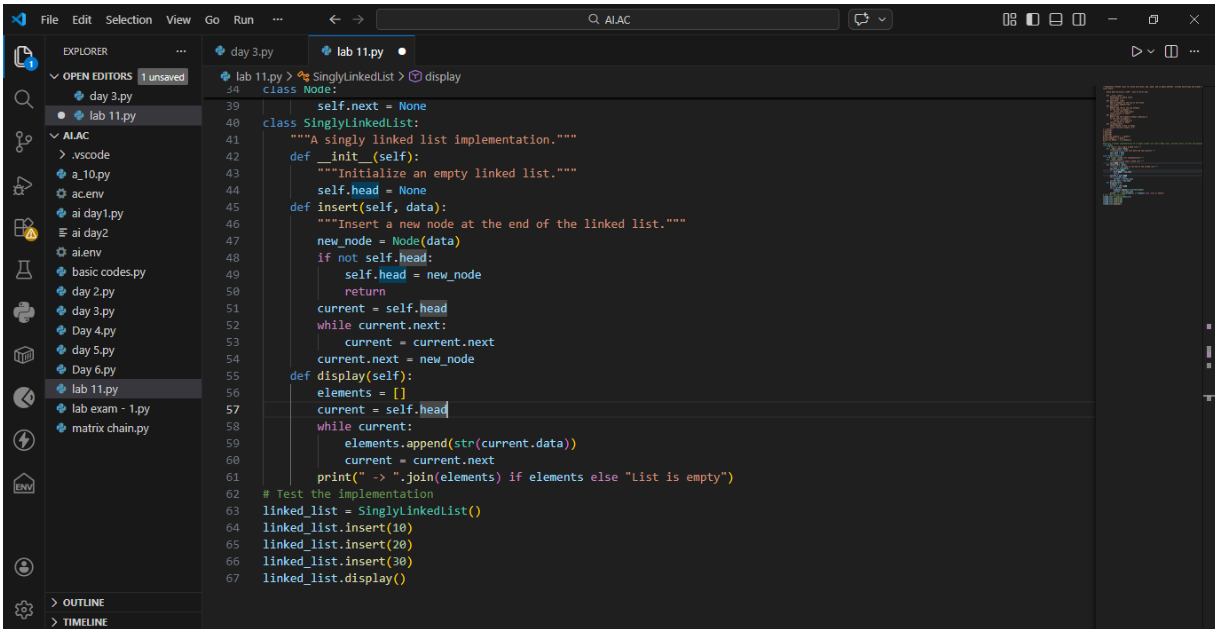
```
Front element: 10
Queue size: 3
Dequeued: 10
Is Empty: False
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

Explanation: A Queue is a linear data structure that follows the FIFO (First In First Out) principle. This means the first element inserted is the first one removed.

Task Description #3: Linked List

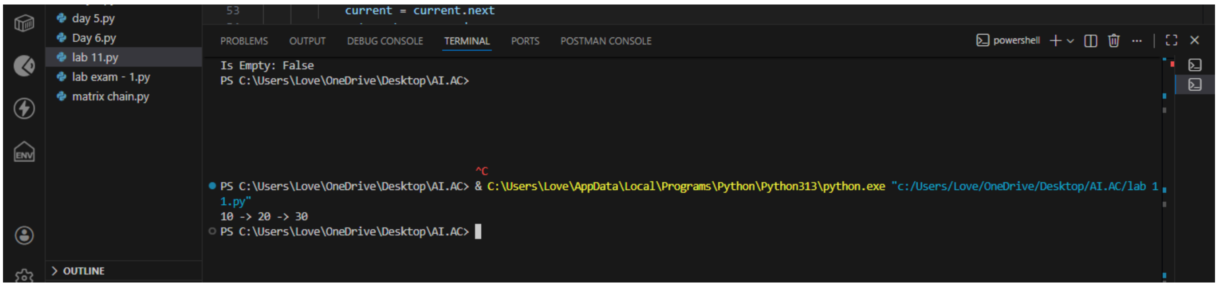
Task: Use AI to generate a Singly Linked List with insert and display methods.

Prompt : Generate a Python implementation of a Singly Linked List with a Node class. Include insert (at end) and display methods with docstrings.



```python
class Node:
        self.next = None
class SinglyLinkedList:
    """A singly linked list implementation."""
    def __init__(self):
        """Initialize an empty linked list."""
        self.head = None
    def insert(self, data):
        """Insert a new node at the end of the linked list."""
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node
    def display(self):
        elements = []
        current = self.head
        while current:
            elements.append(str(current.data))
            current = current.next
        print(" -> ".join(elements) if elements else "List is empty")
# Test the implementation
linked_list = SinglyLinkedList()
linked_list.insert(10)
linked_list.insert(20)
linked_list.insert(30)
linked_list.display()
```

OUTPUT:



```
        current = current.next
```

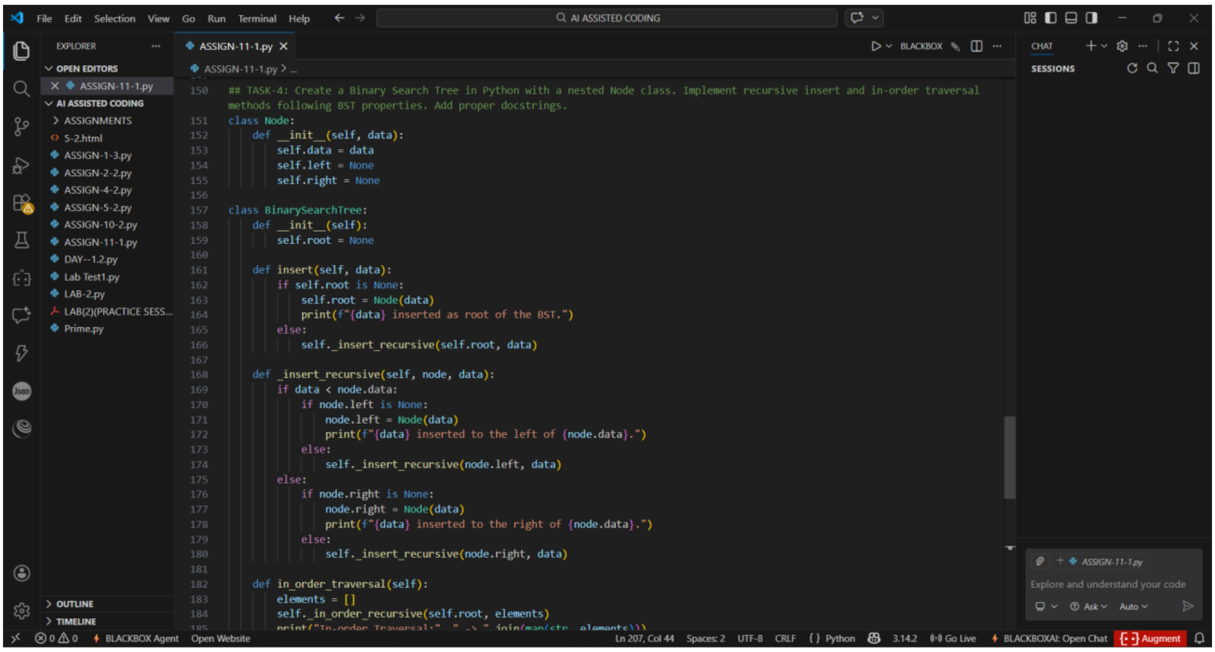
```
Is Empty: False
PS C:\Users\Love\OneDrive\Desktop\AI.AC>



                                        ^C
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 1
1.py"
10 -> 20 -> 30
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

Explanation: A Singly Linked List is a dynamic data structure where elements (nodes) are connected using pointers. Linked Lists are useful when frequent insertions and deletions are required, as they do not require shifting elements like arrays.

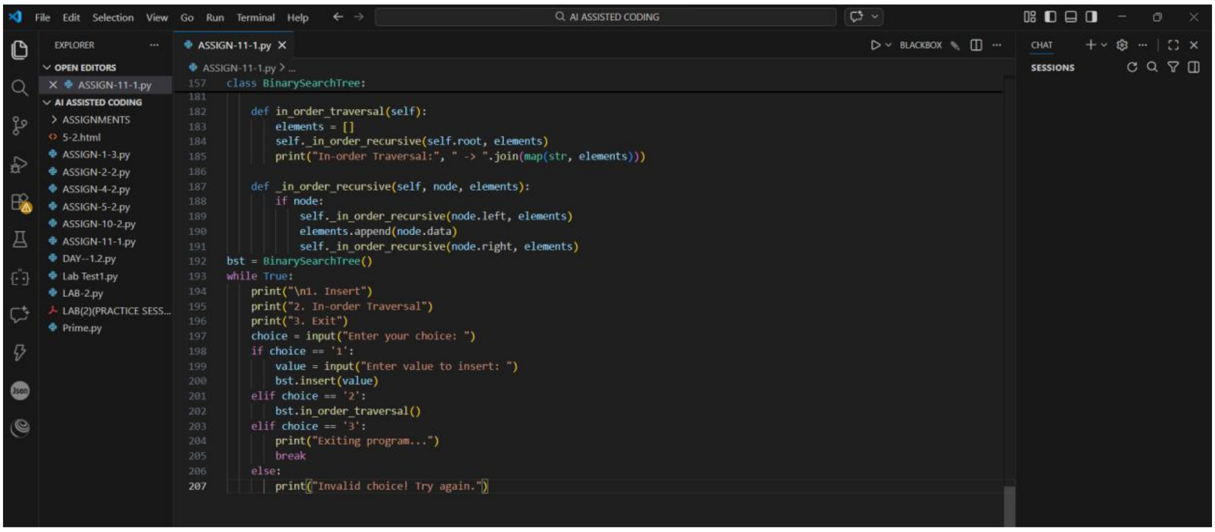
Task Description #4: Binary Search Tree (BST)

Task: Use AI to create a BST with insert and in-order traversal methods.

Prompt: Create a Binary Search Tree in Python with recursive insert and inorder traversal methods. Include proper class structure and documentation.

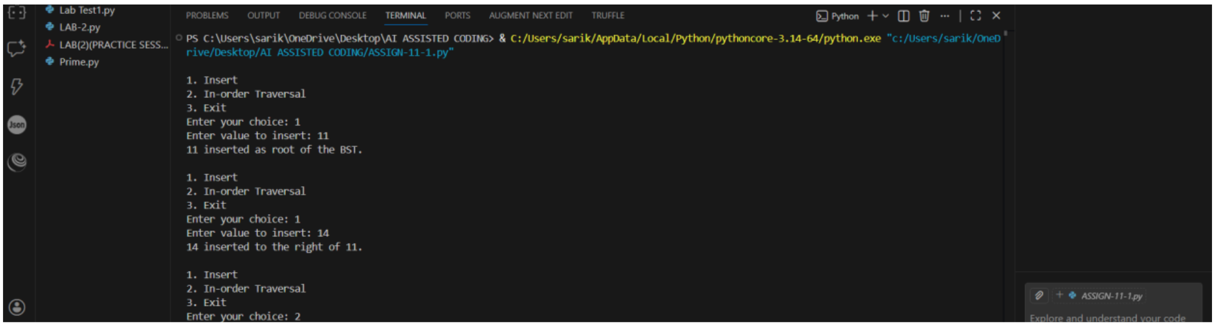


```python
## TASK-4: Create a Binary Search Tree in Python with a nested Node class. Implement recursive insert and in-order traversal methods following BST properties. Add proper docstrings.
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

class BinarySearchTree:
    def __init__(self):
        self.root = None

    def insert(self, data):
        if self.root is None:
            self.root = Node(data)
            print(f"{data} inserted as root of the BST.")
        else:
            self._insert_recursive(self.root, data)

    def _insert_recursive(self, node, data):
        if data < node.data:
            if node.left is None:
                node.left = Node(data)
                print(f"{data} inserted to the left of {node.data}.")
            else:
                self._insert_recursive(node.left, data)
        else:
            if node.right is None:
                node.right = Node(data)
                print(f"{data} inserted to the right of {node.data}.")
            else:
                self._insert_recursive(node.right, data)

    def in_order_traversal(self):
        elements = []
        self._in_order_recursive(self.root, elements)
        print("In-order Traversal:", " -> ".join(map(str, elements)))
```



```python
class BinarySearchTree:

    def in_order_traversal(self):
        elements = []
        self._in_order_recursive(self.root, elements)
        print("In-order Traversal:", " -> ".join(map(str, elements)))

    def _in_order_recursive(self, node, elements):
        if node:
            self._in_order_recursive(node.left, elements)
            elements.append(node.data)
            self._in_order_recursive(node.right, elements)
bst = BinarySearchTree()
while True:
    print("\n1. Insert")
    print("2. In-order Traversal")
    print("3. Exit")
    choice = input("Enter your choice: ")
    if choice == '1':
        value = input("Enter value to insert: ")
        bst.insert(value)
    elif choice == '2':
        bst.in_order_traversal()
    elif choice == '3':
        print("Exiting program...")
        break
    else:
        print("Invalid choice! Try again.")
```

OUTPUT:



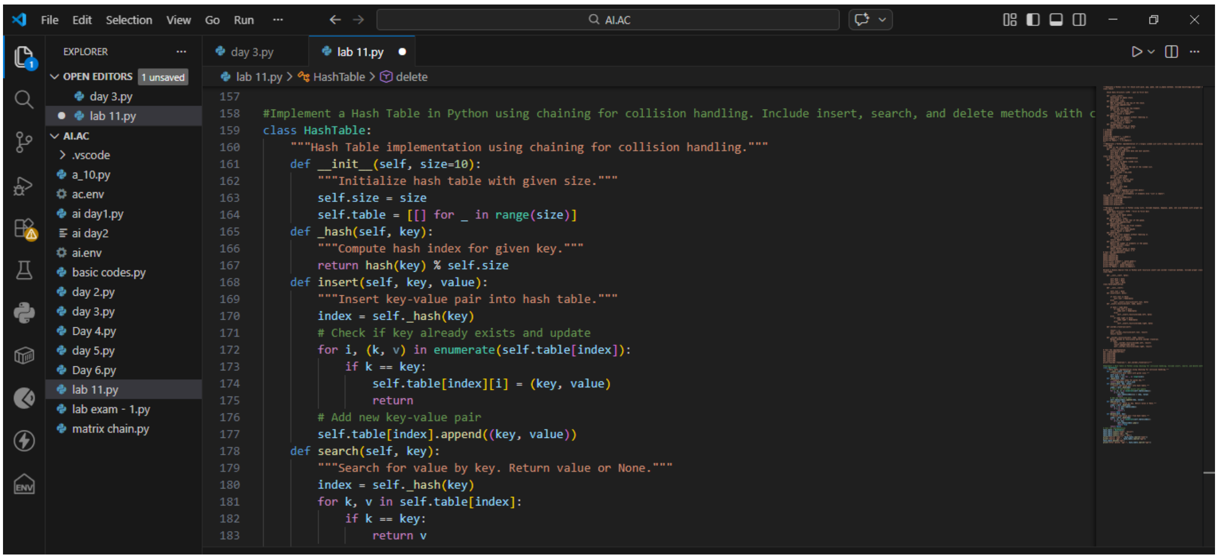
```
PS C:\Users\sarik\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/sarik/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/sarik/OneD
rive/Desktop/AI ASSISTED CODING/ASSIGN-11-1.py"

1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 1
Enter value to insert: 11
11 inserted as root of the BST.

1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 1
Enter value to insert: 14
14 inserted to the right of 11.

1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 2
```

Explanation: A Binary Search Tree is a hierarchical data structure where the left child contains smaller values and the right child contains larger values than the root. This property makes searching, insertion, and deletion efficient.

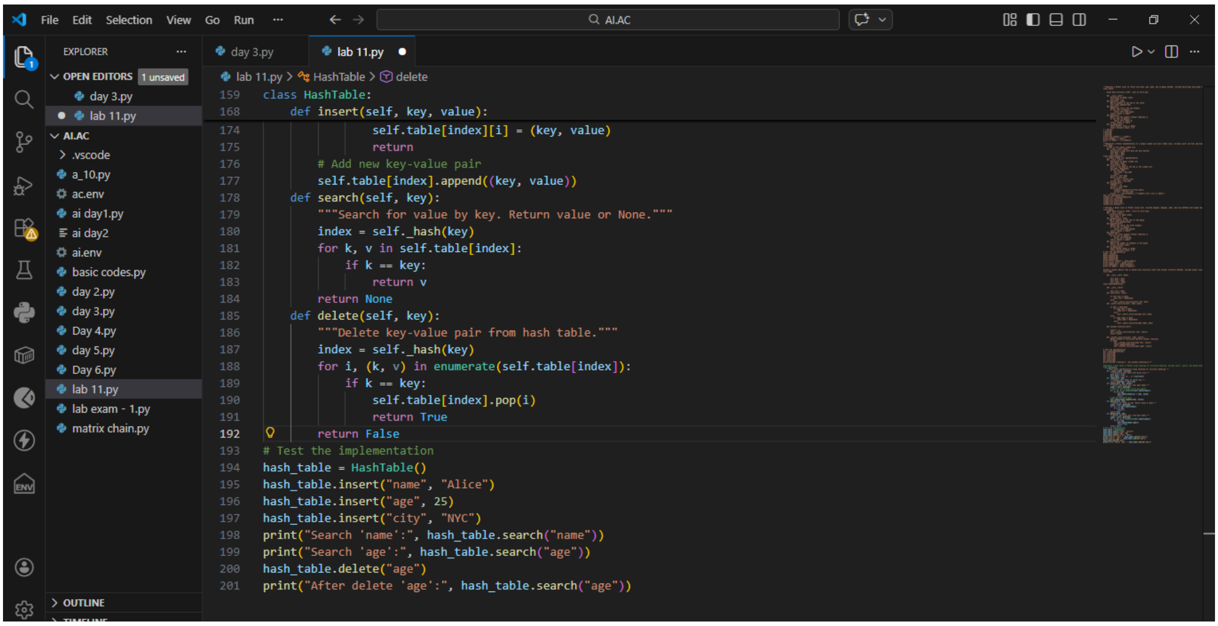
Task Description #5: Hash Table

Task: Use AI to implement a hash table with basic insert, search, and delete methods.

Prompt: Implement a Hash Table in Python using chaining for collision handling. Include insert, search, and delete methods with comments.

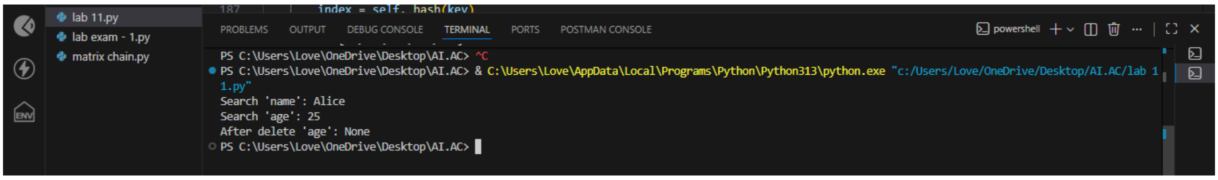


```python

#Implement a Hash Table in Python using chaining for collision handling. Include insert, search, and delete methods with c
class HashTable:
    """Hash Table implementation using chaining for collision handling."""
    def __init__(self, size=10):
        """Initialize hash table with given size."""
        self.size = size
        self.table = [[] for _ in range(size)]
    def _hash(self, key):
        """Compute hash index for given key."""
        return hash(key) % self.size
    def insert(self, key, value):
        """Insert key-value pair into hash table."""
        index = self._hash(key)
        # Check if key already exists and update
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                self.table[index][i] = (key, value)
                return
        # Add new key-value pair
        self.table[index].append((key, value))
    def search(self, key):
        """Search for value by key. Return value or None."""
        index = self._hash(key)
        for k, v in self.table[index]:
            if k == key:
                return v
```



```python
class HashTable:
    def insert(self, key, value):
                self.table[index][i] = (key, value)
                return
        # Add new key-value pair
        self.table[index].append((key, value))
    def search(self, key):
        """Search for value by key. Return value or None."""
        index = self._hash(key)
        for k, v in self.table[index]:
            if k == key:
                return v
        return None
    def delete(self, key):
        """Delete key-value pair from hash table."""
        index = self._hash(key)
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                self.table[index].pop(i)
                return True
        return False
# Test the implementation
hash_table = HashTable()
hash_table.insert("name", "Alice")
hash_table.insert("age", 25)
hash_table.insert("city", "NYC")
print("Search 'name':", hash_table.search("name"))
print("Search 'age':", hash_table.search("age"))
hash_table.delete("age")
print("After delete 'age':", hash_table.search("age"))
```

OUTPUT:



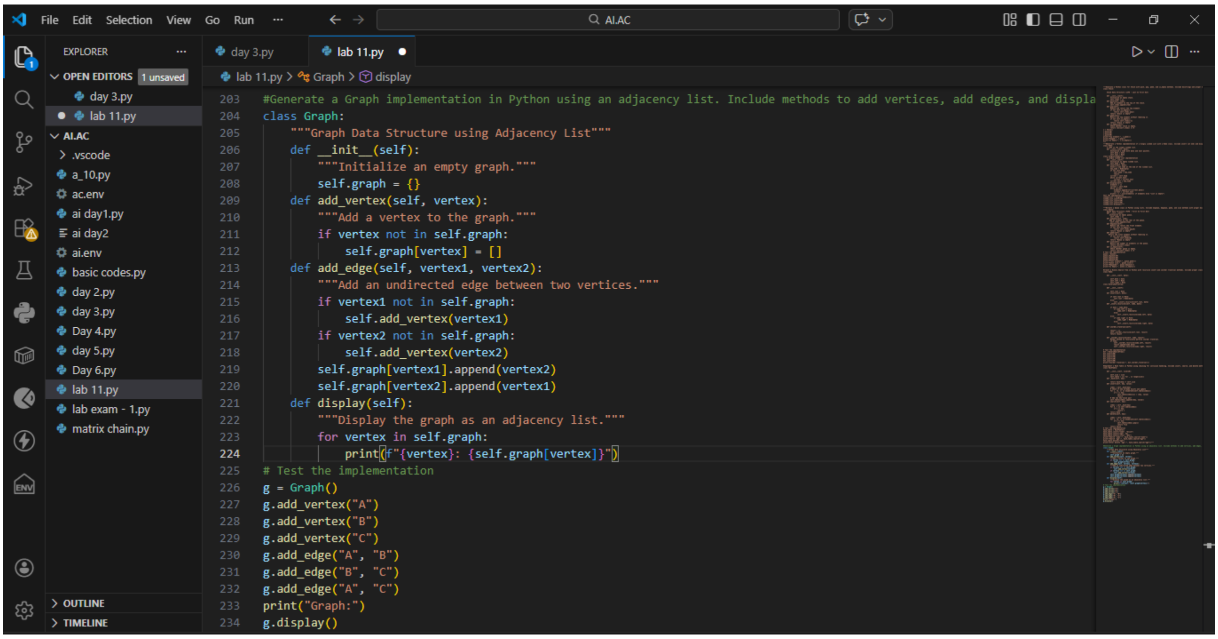
```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> ^C
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 1
1.py"
Search 'name': Alice
Search 'age': 25
After delete 'age': None
PS C:\Users\Love\OneDrive\Desktop\AI.AC> 
```

Explanation: A Hash Table stores data in key-value pairs using a hash function to compute an index. It provides fast average-case time complexity for search, insertion, and deletion operations.

Task Description #6: Graph Representation

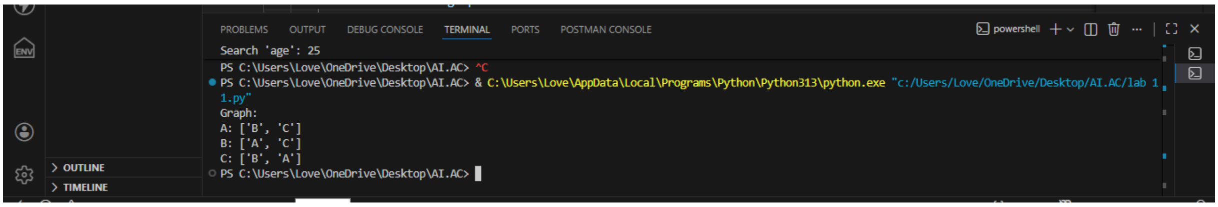
Task: Use AI to implement a graph using an adjacency list.

Prompt: Generate a Graph implementation in Python using an adjacency list. Include methods to add vertices, add edges, and display the graph.



```python
#Generate a Graph implementation in Python using an adjacency list. Include methods to add vertices, add edges, and displa
class Graph:
    """Graph Data Structure using Adjacency List"""
    def __init__(self):
        """Initialize an empty graph."""
        self.graph = {}
    def add_vertex(self, vertex):
        """Add a vertex to the graph."""
        if vertex not in self.graph:
            self.graph[vertex] = []
    def add_edge(self, vertex1, vertex2):
        """Add an undirected edge between two vertices."""
        if vertex1 not in self.graph:
            self.add_vertex(vertex1)
        if vertex2 not in self.graph:
            self.add_vertex(vertex2)
        self.graph[vertex1].append(vertex2)
        self.graph[vertex2].append(vertex1)
    def display(self):
        """Display the graph as an adjacency list."""
        for vertex in self.graph:
            print(f"{vertex}: {self.graph[vertex]}")
# Test the implementation
g = Graph()
g.add_vertex("A")
g.add_vertex("B")
g.add_vertex("C")
g.add_edge("A", "B")
g.add_edge("B", "C")
g.add_edge("A", "C")
print("Graph:")
g.display()
```

Output:



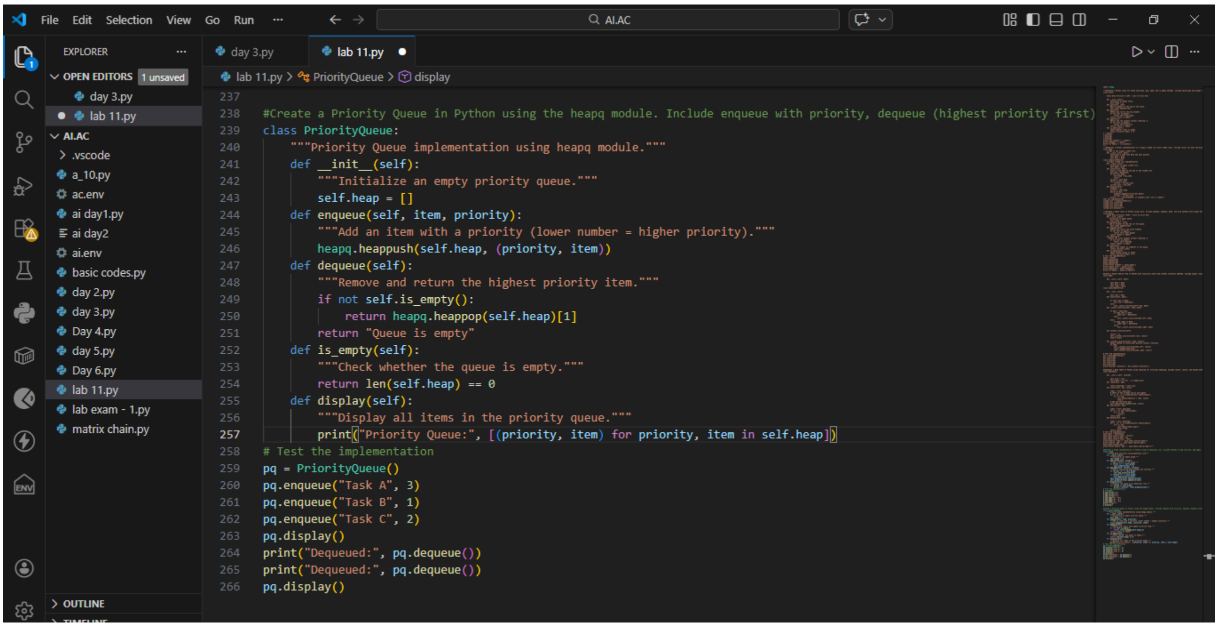
```
Search 'age': 25
PS C:\Users\Love\OneDrive\Desktop\AI.AC> ^C
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 1
1.py"
Graph:
A: ['B', 'C']
B: ['A', 'C']
C: ['B', 'A']
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

Explanation: A Graph is a non-linear data structure used to represent relationships between entities. It consists of vertices (nodes) and edges (connections).

Task Description #7: Priority Queue

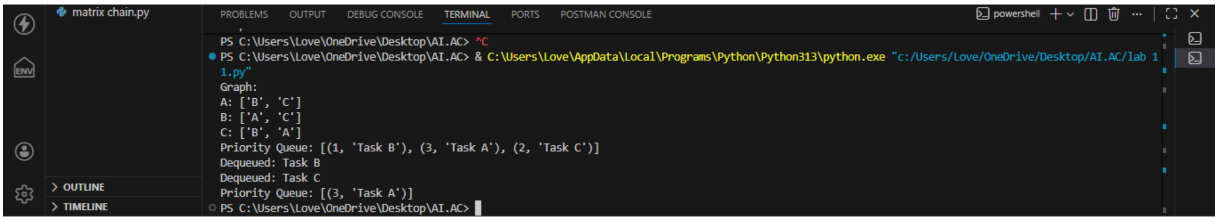
Task: Use AI to implement a priority queue using Python's heap module.

Prompt: Create a Priority Queue in Python using the heapq module. Include enqueue with priority, dequeue (highest priority first), and display methods.



```python

#Create a Priority Queue in Python using the heapq module. Include enqueue with priority, dequeue (highest priority first)
class PriorityQueue:
    """Priority Queue implementation using heapq module."""
    def __init__(self):
        """Initialize an empty priority queue."""
        self.heap = []
    def enqueue(self, item, priority):
        """Add an item with a priority (lower number = higher priority)."""
        heapq.heappush(self.heap, (priority, item))
    def dequeue(self):
        """Remove and return the highest priority item."""
        if not self.is_empty():
            return heapq.heappop(self.heap)[1]
        return "Queue is empty"
    def is_empty(self):
        """Check whether the queue is empty."""
        return len(self.heap) == 0
    def display(self):
        """Display all items in the priority queue."""
        print("Priority Queue:", [(priority, item) for priority, item in self.heap])
# Test the implementation
pq = PriorityQueue()
pq.enqueue("Task A", 3)
pq.enqueue("Task B", 1)
pq.enqueue("Task C", 2)
pq.display()
print("Dequeued:", pq.dequeue())
print("Dequeued:", pq.dequeue())
pq.display()
```

Output:



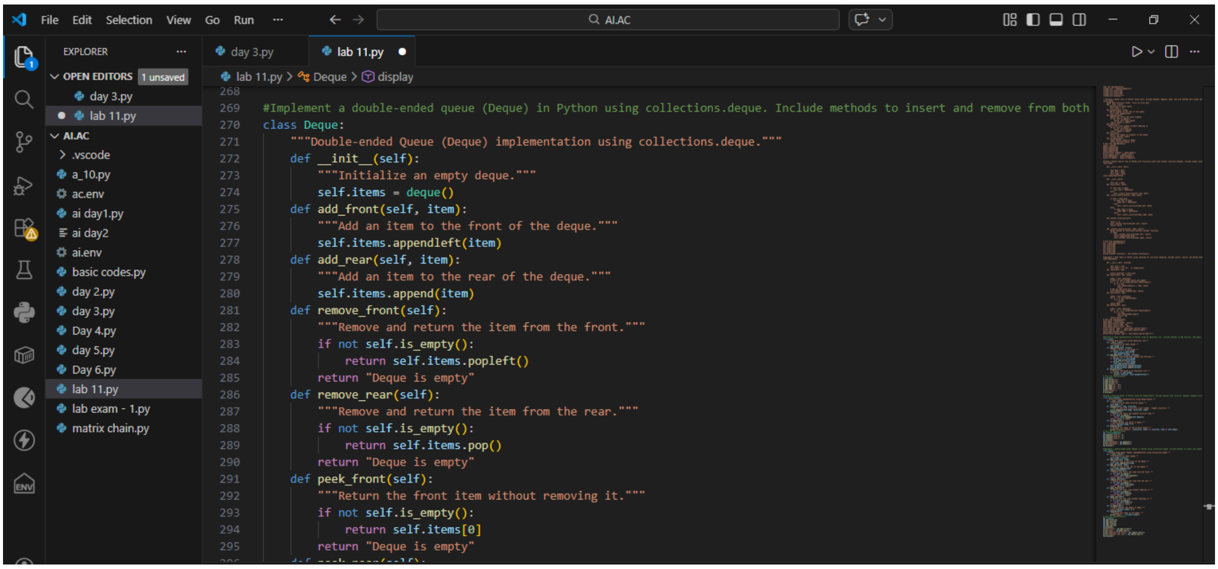
```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> ^C
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 1
1.py"
Graph:
A: ['B', 'C']
B: ['A', 'C']
C: ['B', 'A']
Priority Queue: [(1, 'Task B'), (3, 'Task A'), (2, 'Task C')]
Dequeued: Task B
Dequeued: Task C
Priority Queue: [(3, 'Task A')]
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

Explanation: A Priority Queue is a special type of queue where elements are removed based on priority rather than order of insertion. Higher priority elements are processed first. It is typically implemented using a heap for efficiency.

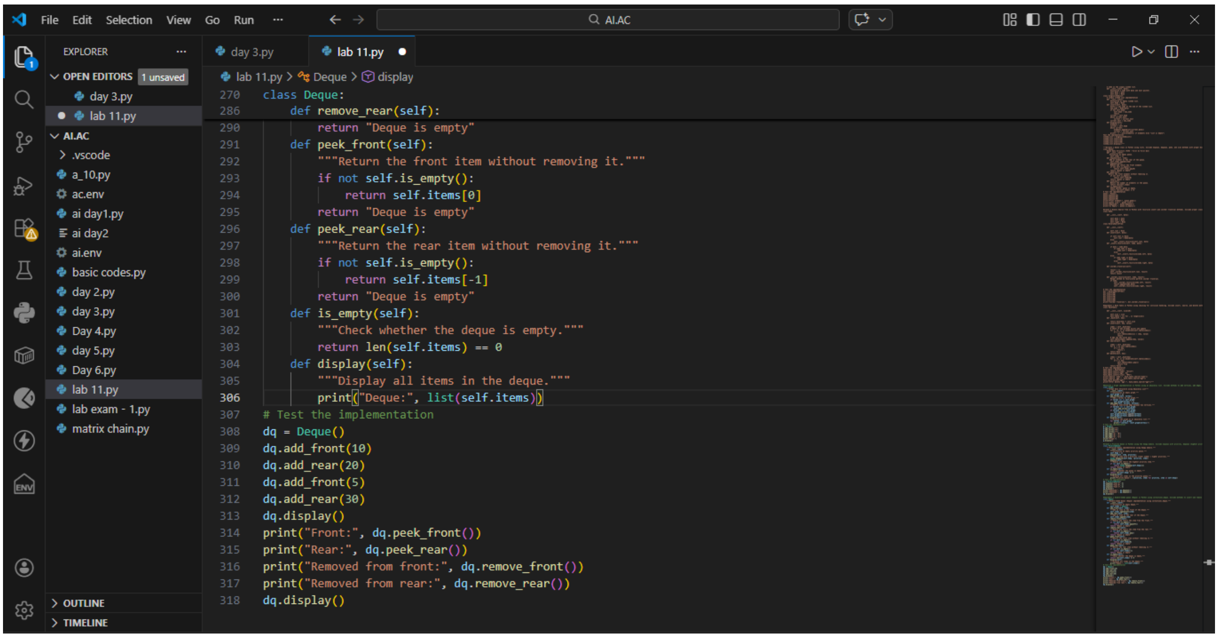
Task Description #8 – Deque

Task: Use AI to implement a double-ended queue using collections.deque.

Prompt: Implement a double-ended queue (Deque) in Python using collections, deque. Include methods to insert and remove from both ends with documentation.

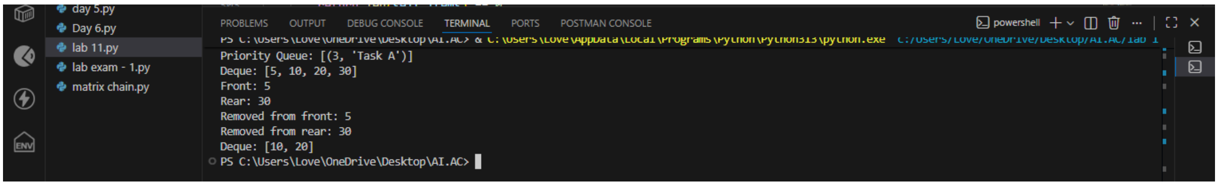


```python

#Implement a double-ended queue (Deque) in Python using collections.deque. Include methods to insert and remove from both
class Deque:
    """Double-ended Queue (Deque) implementation using collections.deque."""
    def __init__(self):
        """Initialize an empty deque."""
        self.items = deque()
    def add_front(self, item):
        """Add an item to the front of the deque."""
        self.items.appendleft(item)
    def add_rear(self, item):
        """Add an item to the rear of the deque."""
        self.items.append(item)
    def remove_front(self):
        """Remove and return the item from the front."""
        if not self.is_empty():
            return self.items.popleft()
        return "Deque is empty"
    def remove_rear(self):
        """Remove and return the item from the rear."""
        if not self.is_empty():
            return self.items.pop()
        return "Deque is empty"
    def peek_front(self):
        """Return the front item without removing it."""
        if not self.is_empty():
            return self.items[0]
        return "Deque is empty"
```



```python
class Deque:
    def remove_rear(self):
            return "Deque is empty"
    def peek_front(self):
        """Return the front item without removing it."""
        if not self.is_empty():
            return self.items[0]
        return "Deque is empty"
    def peek_rear(self):
        """Return the rear item without removing it."""
        if not self.is_empty():
            return self.items[-1]
        return "Deque is empty"
    def is_empty(self):
        """Check whether the deque is empty."""
        return len(self.items) == 0
    def display(self):
        """Display all items in the deque."""
        print("Deque:", list(self.items))
# Test the implementation
dq = Deque()
dq.add_front(10)
dq.add_rear(20)
dq.add_front(5)
dq.add_rear(30)
dq.display()
print("Front:", dq.peek_front())
print("Rear:", dq.peek_rear())
print("Removed from front:", dq.remove_front())
print("Removed from rear:", dq.remove_rear())
dq.display()
```

Output:



```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe c:/Users/Love/OneDrive/Desktop/AI.AC/lab 1
Priority Queue: [(3, 'Task A')]
Deque: [5, 10, 20, 30]
Front: 5
Rear: 30
Removed from front: 5
Removed from rear: 30
Deque: [10, 20]
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

Explanation: A Deque (Double Ended Queue) allows insertion and deletion of elements from both the front and rear ends.