AI ASSISTED CODING

LAB-11.1

Thotapally Pranathi

2303A51733

Batch-11

Task Description #1 – Stack Implementation

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

methods.

Sample Input Code:

class Stack:

pass

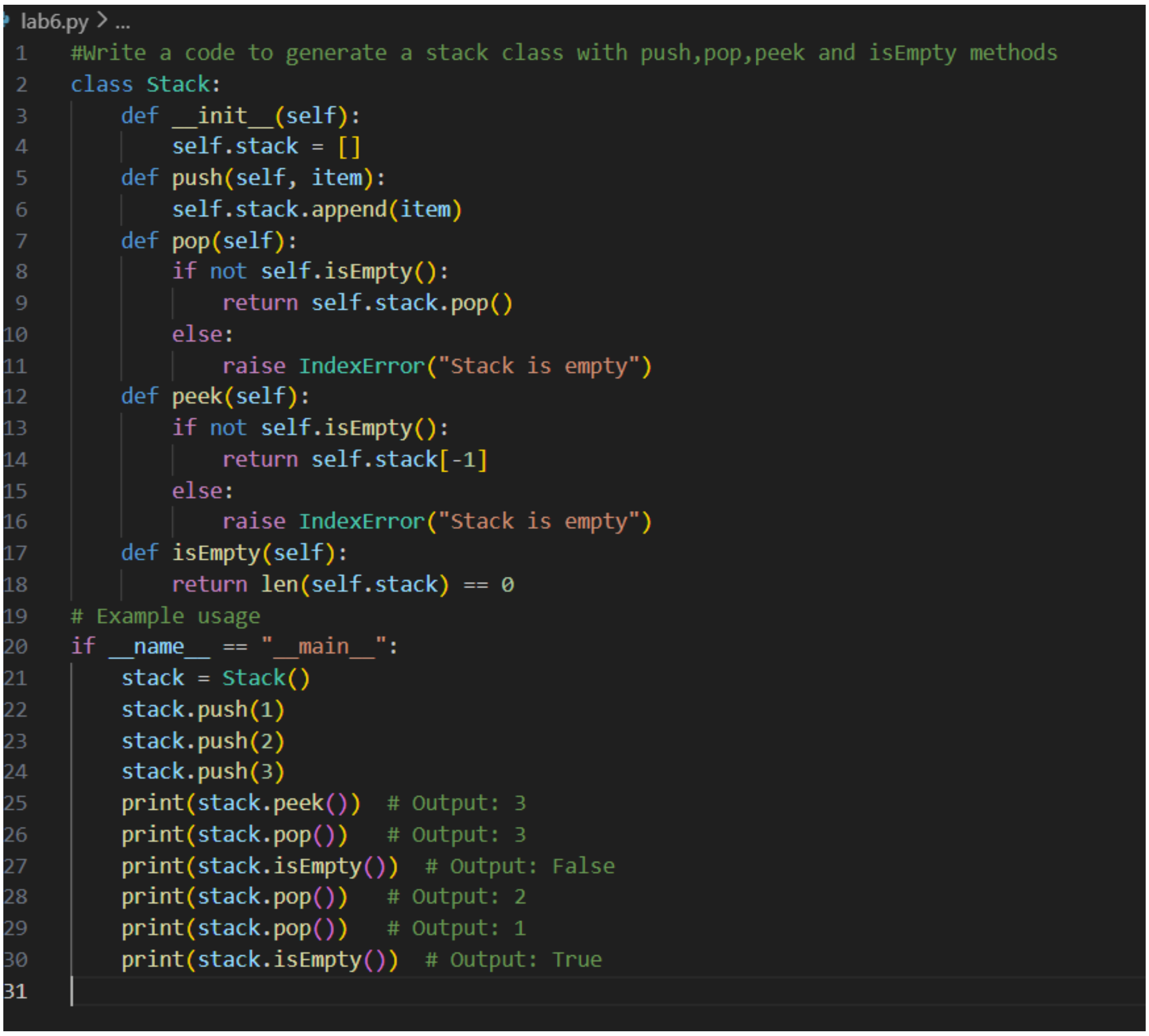
Expected Output:

• A functional stack implementation with all required methods and docstrings

PROMPT:

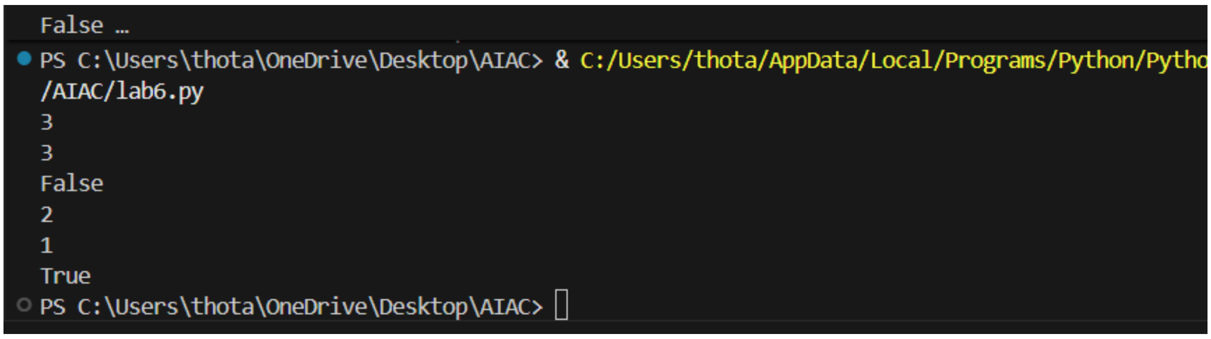
#Write a code to generate a stack class with push,pop,peek and isEmpty methods

CODE:



```
lab6.py > ...
#Write a code to generate a stack class with push,pop,peek and isEmpty methods
class Stack:
    def __init__(self):
        self.stack = []
    def push(self, item):
        self.stack.append(item)
    def pop(self):
        if not self.isEmpty():
            return self.stack.pop()
        else:
            raise IndexError("Stack is empty")
    def peek(self):
        if not self.isEmpty():
            return self.stack[-1]
        else:
            raise IndexError("Stack is empty")
    def isEmpty(self):
        return len(self.stack) == 0
# Example usage
if __name__ == "__main__":
    stack = Stack()
    stack.push(1)
    stack.push(2)
    stack.push(3)
    print(stack.peek())  # Output: 3
    print(stack.pop())   # Output: 3
    print(stack.isEmpty())  # Output: False
    print(stack.pop())   # Output: 2
    print(stack.pop())   # Output: 1
    print(stack.isEmpty())  # Output: True
```

OUTPUT:



```
False ...
PS C:\Users\thota\OneDrive\Desktop\AIAC> & C:/Users/thota/AppData/Local/Programs/Python/Pytho
/AIAC/lab6.py
3
3
False
2
1
True
PS C:\Users\thota\OneDrive\Desktop\AIAC> 
```

Task Description #2 – Queue Implementation

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

Sample Input Code:

class Queue:

pass

Expected Output:

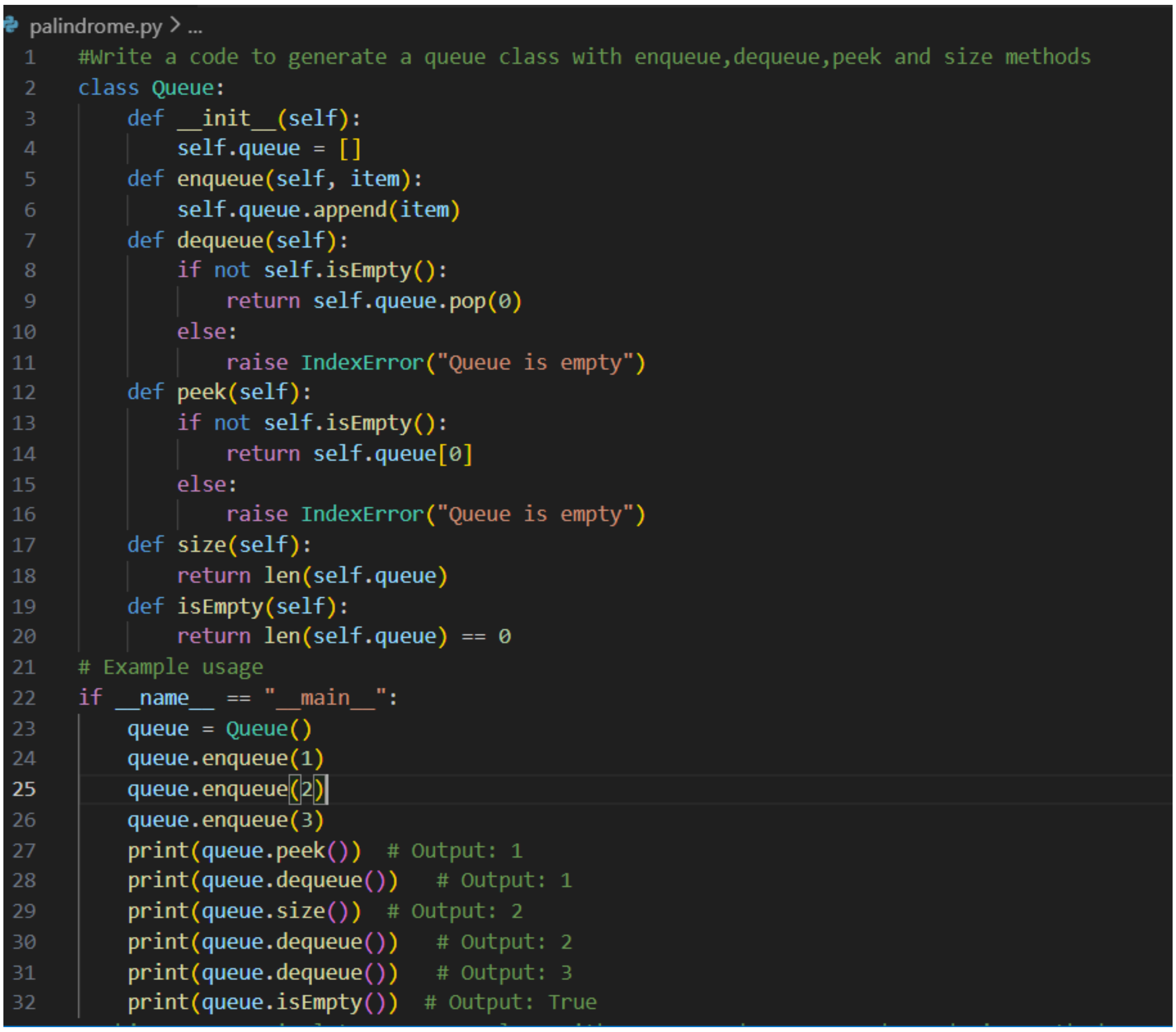
• FIFO-based queue class with enqueue, dequeue, peek, and size

methods.

PROMPT:

#Write a code to generate a queue class with enqueue,dequeue,peek and size methods

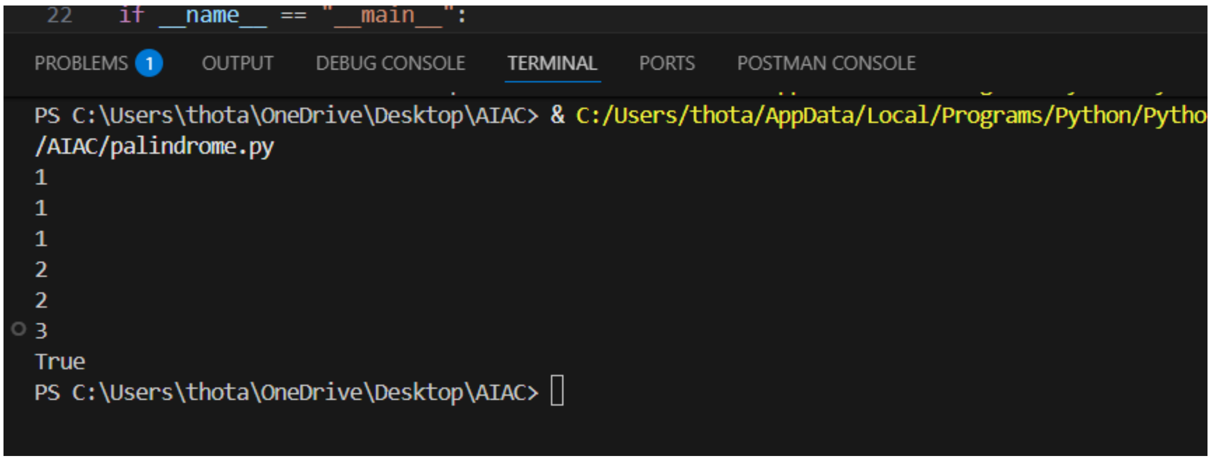
CODE:



palindrome.py > ...

```python
#Write a code to generate a queue class with enqueue,dequeue,peek and size methods
class Queue:
    def __init__(self):
        self.queue = []
    def enqueue(self, item):
        self.queue.append(item)
    def dequeue(self):
        if not self.isEmpty():
            return self.queue.pop(0)
        else:
            raise IndexError("Queue is empty")
    def peek(self):
        if not self.isEmpty():
            return self.queue[0]
        else:
            raise IndexError("Queue is empty")
    def size(self):
        return len(self.queue)
    def isEmpty(self):
        return len(self.queue) == 0
# Example usage
if __name__ == "__main__":
    queue = Queue()
    queue.enqueue(1)
    queue.enqueue(2)
    queue.enqueue(3)
    print(queue.peek())  # Output: 1
    print(queue.dequeue())   # Output: 1
    print(queue.size())  # Output: 2
    print(queue.dequeue())   # Output: 2
    print(queue.dequeue())   # Output: 3
    print(queue.isEmpty())  # Output: True
```

OUTPUT:



```
PS C:\Users\thota\OneDrive\Desktop\AIAC> & C:/Users/thota/AppData/Local/Programs/Python/Pytho
/AIAC/palindrome.py
1
1
1
2
2
3
True
PS C:\Users\thota\OneDrive\Desktop\AIAC>
```

Task Description #3 – Linked List

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

methods.

Sample Input Code:

class Node:

pass

class LinkedList:

pass

Expected Output:

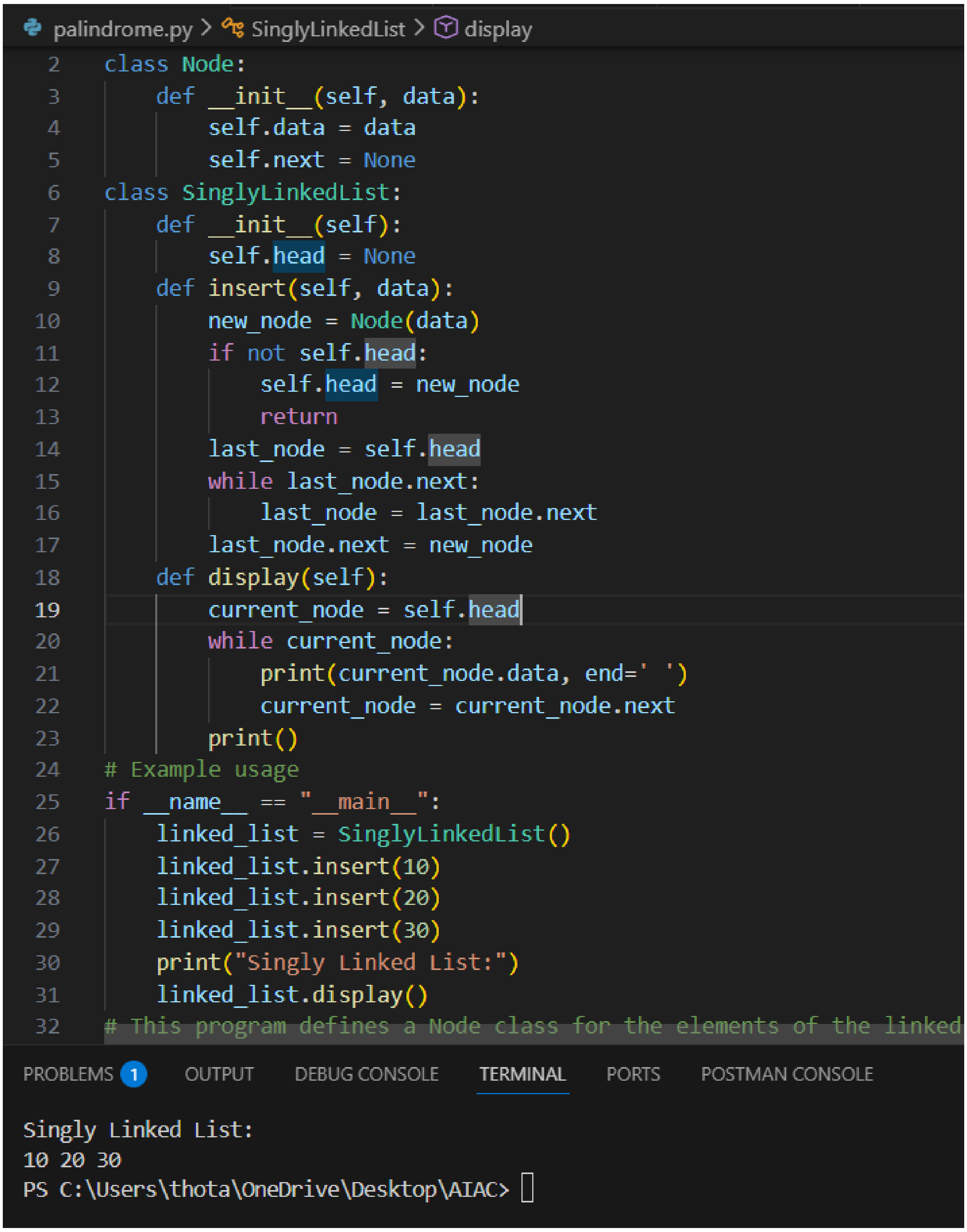
• A working linked list implementation with clear method

documentation

PROMPT:

#Write a code to generate a singly linkedlist with insert and display methods

CODE AND OUTPUT:



```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class SinglyLinkedList:
    def __init__(self):
        self.head = None
    def insert(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
        last_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node
    def display(self):
        current_node = self.head
        while current_node:
            print(current_node.data, end=' ')
            current_node = current_node.next
        print()
# Example usage
if __name__ == "__main__":
    linked_list = SinglyLinkedList()
    linked_list.insert(10)
    linked_list.insert(20)
    linked_list.insert(30)
    print("Singly Linked List:")
    linked_list.display()
# This program defines a Node class for the elements of the linked
```

```
Singly Linked List:
10 20 30
PS C:\Users\thota\OneDrive\Desktop\AIAC>
```

Task Description #4 – Binary Search Tree (BST)

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

Sample Input Code:

class BST:

pass

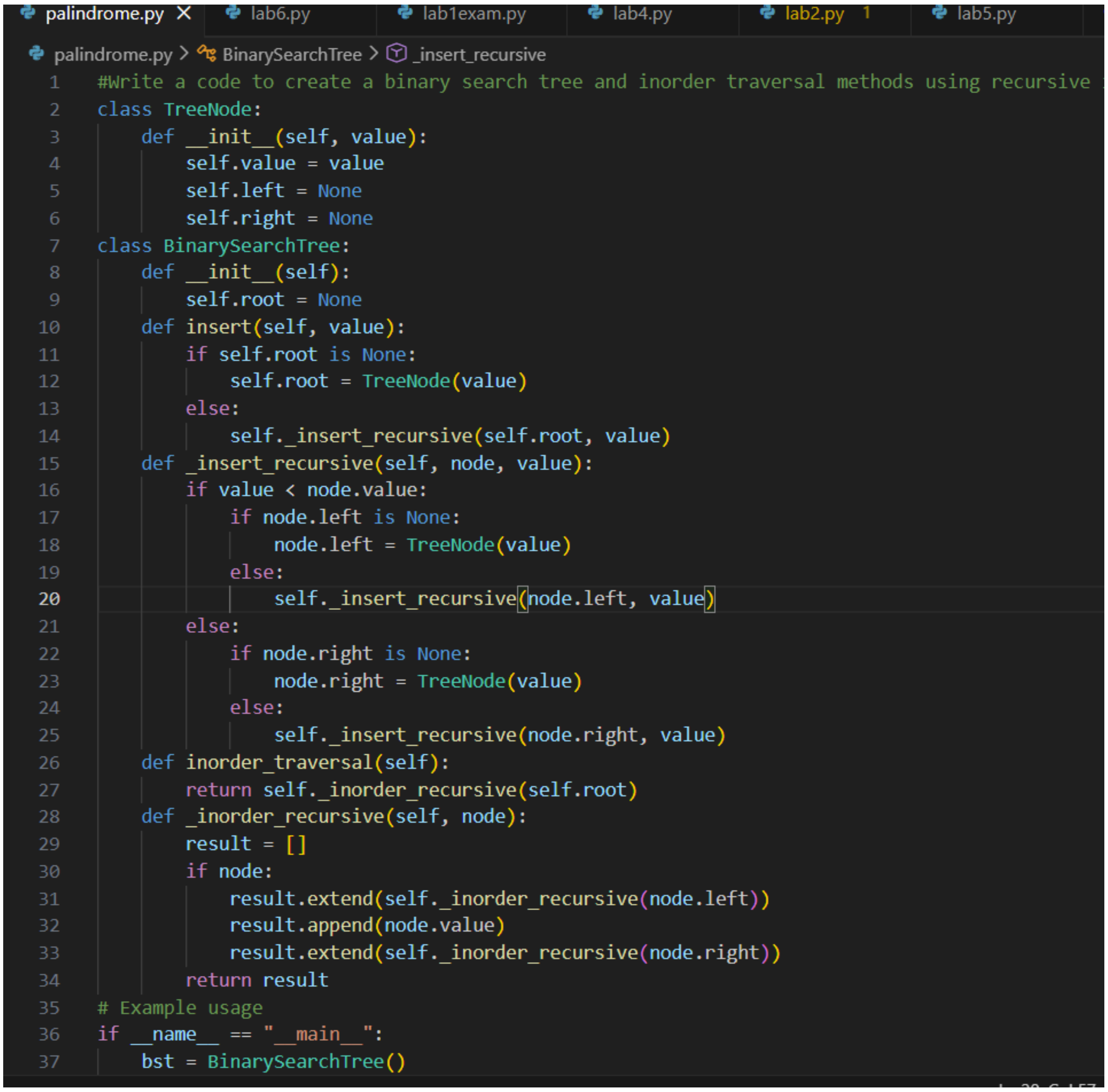
Expected Output:

• BST implementation with recursive insert and traversal methods.

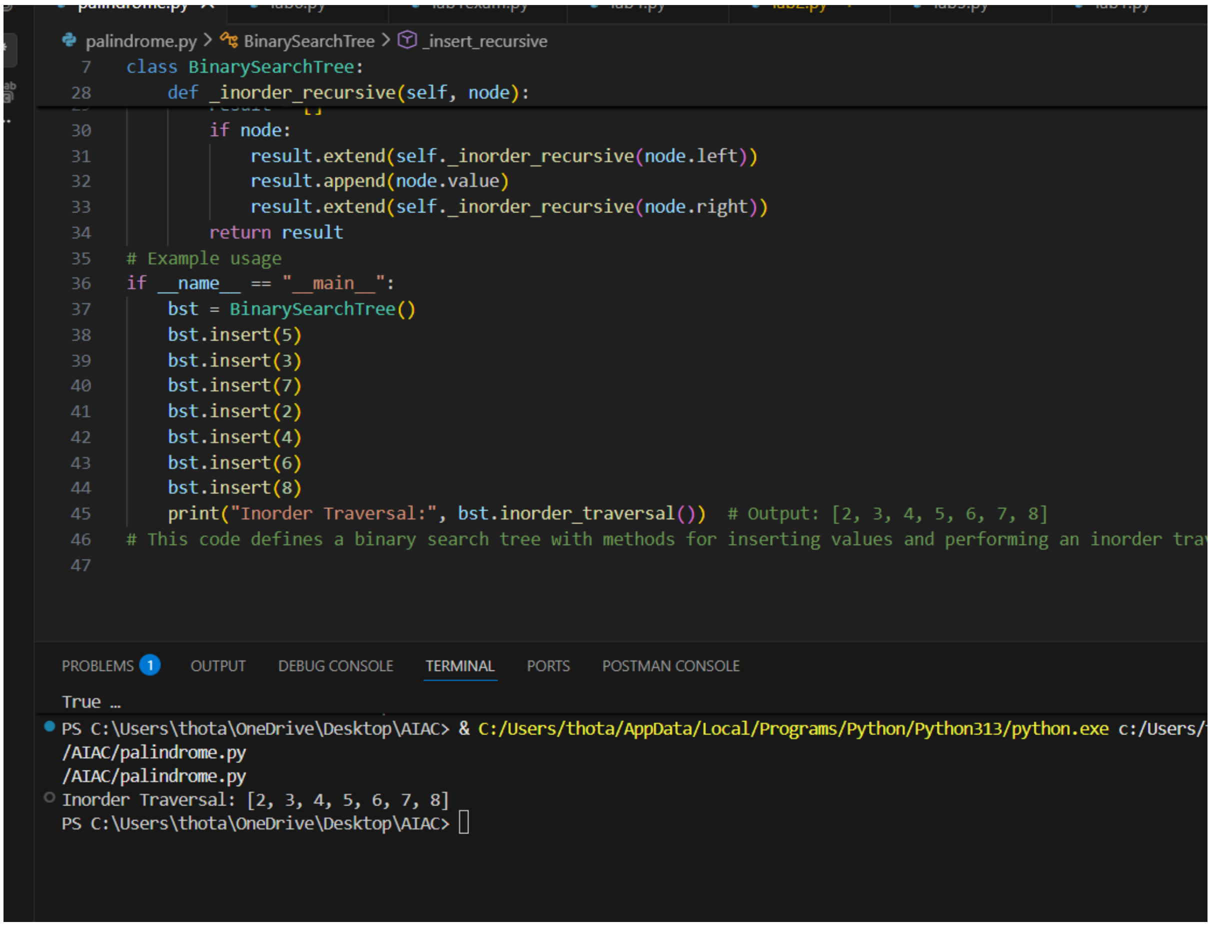
PROMPT:

#Write a code to create a binary search tree and inorder traversal methods using recursive insert and travesal methods

CODE AND OUTPUT:



```python
#Write a code to create a binary search tree and inorder traversal methods using recursive
class TreeNode:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None
class BinarySearchTree:
    def __init__(self):
        self.root = None
    def insert(self, value):
        if self.root is None:
            self.root = TreeNode(value)
        else:
            self._insert_recursive(self.root, value)
    def _insert_recursive(self, node, value):
        if value < node.value:
            if node.left is None:
                node.left = TreeNode(value)
            else:
                self._insert_recursive(node.left, value)
        else:
            if node.right is None:
                node.right = TreeNode(value)
            else:
                self._insert_recursive(node.right, value)
    def inorder_traversal(self):
        return self._inorder_recursive(self.root)
    def _inorder_recursive(self, node):
        result = []
        if node:
            result.extend(self._inorder_recursive(node.left))
            result.append(node.value)
            result.extend(self._inorder_recursive(node.right))
        return result
# Example usage
if __name__ == "__main__":
    bst = BinarySearchTree()
```



```
class BinarySearchTree:
    def _inorder_recursive(self, node):
        if node:
            result.extend(self._inorder_recursive(node.left))
            result.append(node.value)
            result.extend(self._inorder_recursive(node.right))
        return result
# Example usage
if __name__ == "__main__":
    bst = BinarySearchTree()
    bst.insert(5)
    bst.insert(3)
    bst.insert(7)
    bst.insert(2)
    bst.insert(4)
    bst.insert(6)
    bst.insert(8)
    print("Inorder Traversal:", bst.inorder_traversal())  # Output: [2, 3, 4, 5, 6, 7, 8]
# This code defines a binary search tree with methods for inserting values and performing an inorder tra
```

```
True …
PS C:\Users\thota\OneDrive\Desktop\AIAC> & C:/Users/thota/AppData/Local/Programs/Python/Python313/python.exe c:/Users/
/AIAC/palindrome.py
/AIAC/palindrome.py
Inorder Traversal: [2, 3, 4, 5, 6, 7, 8]
PS C:\Users\thota\OneDrive\Desktop\AIAC> 
```

Task Description #5 – Hash Table

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

delete methods.

Sample Input Code:

class HashTable:

pass

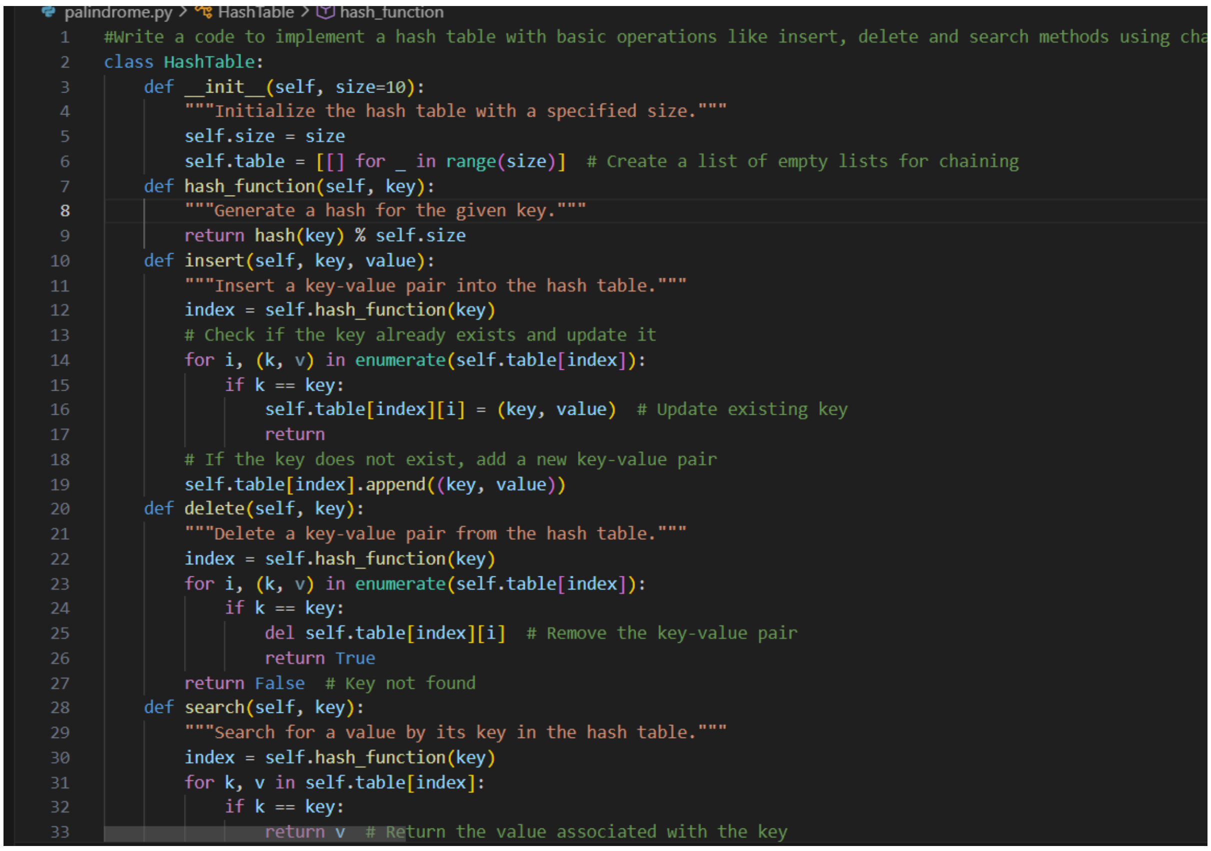
Expected Output:

• Collision handling using chaining, with well-commented methods.

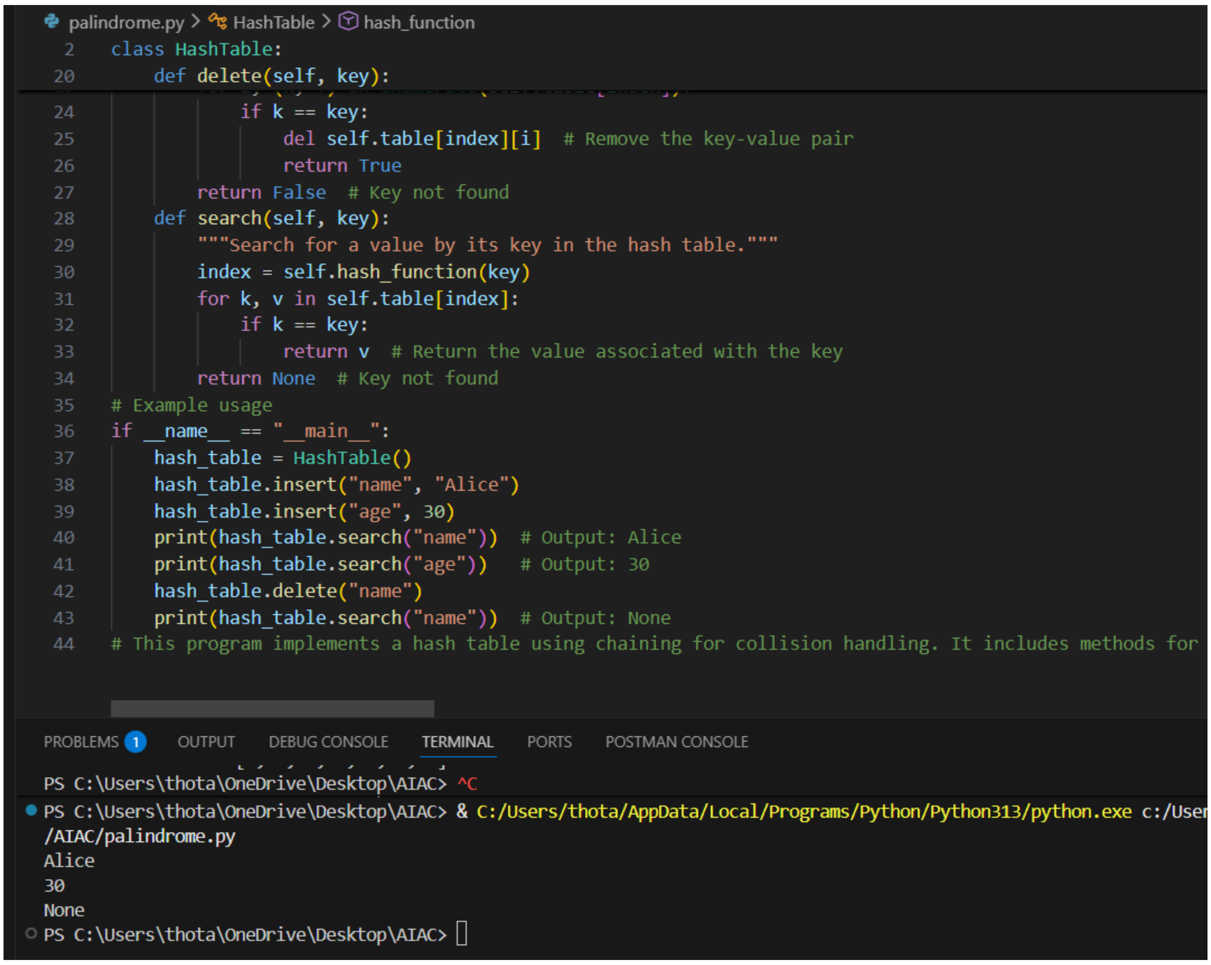
PROMPT:

#Write a code to implement a hash table with basic operations like insert, delete and search methods using chaining for collision handling with well commented methods

CODE AND OUTPUT:



```python
#Write a code to implement a hash table with basic operations like insert, delete and search methods using cha
class HashTable:
    def __init__(self, size=10):
        """Initialize the hash table with a specified size."""
        self.size = size
        self.table = [[] for _ in range(size)]  # Create a list of empty lists for chaining
    def hash_function(self, key):
        """Generate a hash for the given key."""
        return hash(key) % self.size
    def insert(self, key, value):
        """Insert a key-value pair into the hash table."""
        index = self.hash_function(key)
        # Check if the key already exists and update it
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                self.table[index][i] = (key, value)  # Update existing key
                return
        # If the key does not exist, add a new key-value pair
        self.table[index].append((key, value))
    def delete(self, key):
        """Delete a key-value pair from the hash table."""
        index = self.hash_function(key)
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                del self.table[index][i]  # Remove the key-value pair
                return True
        return False  # Key not found
    def search(self, key):
        """Search for a value by its key in the hash table."""
        index = self.hash_function(key)
        for k, v in self.table[index]:
            if k == key:
                return v  # Return the value associated with the key
```



```python
class HashTable:
    def delete(self, key):
            if k == key:
                del self.table[index][i]  # Remove the key-value pair
                return True
        return False  # Key not found
    def search(self, key):
        """Search for a value by its key in the hash table."""
        index = self.hash_function(key)
        for k, v in self.table[index]:
            if k == key:
                return v  # Return the value associated with the key
        return None  # Key not found
# Example usage
if __name__ == "__main__":
    hash_table = HashTable()
    hash_table.insert("name", "Alice")
    hash_table.insert("age", 30)
    print(hash_table.search("name"))  # Output: Alice
    print(hash_table.search("age"))   # Output: 30
    hash_table.delete("name")
    print(hash_table.search("name"))  # Output: None
# This program implements a hash table using chaining for collision handling. It includes methods for
```

```
PS C:\Users\thota\OneDrive\Desktop\AIAC> ^C
PS C:\Users\thota\OneDrive\Desktop\AIAC> & C:/Users/thota/AppData/Local/Programs/Python/Python313/python.exe c:/User
/AIAC/palindrome.py
Alice
30
None
PS C:\Users\thota\OneDrive\Desktop\AIAC>
```

Task Description #6 – Graph Representation

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

Sample Input Code:

class Graph:

pass

Expected Output:

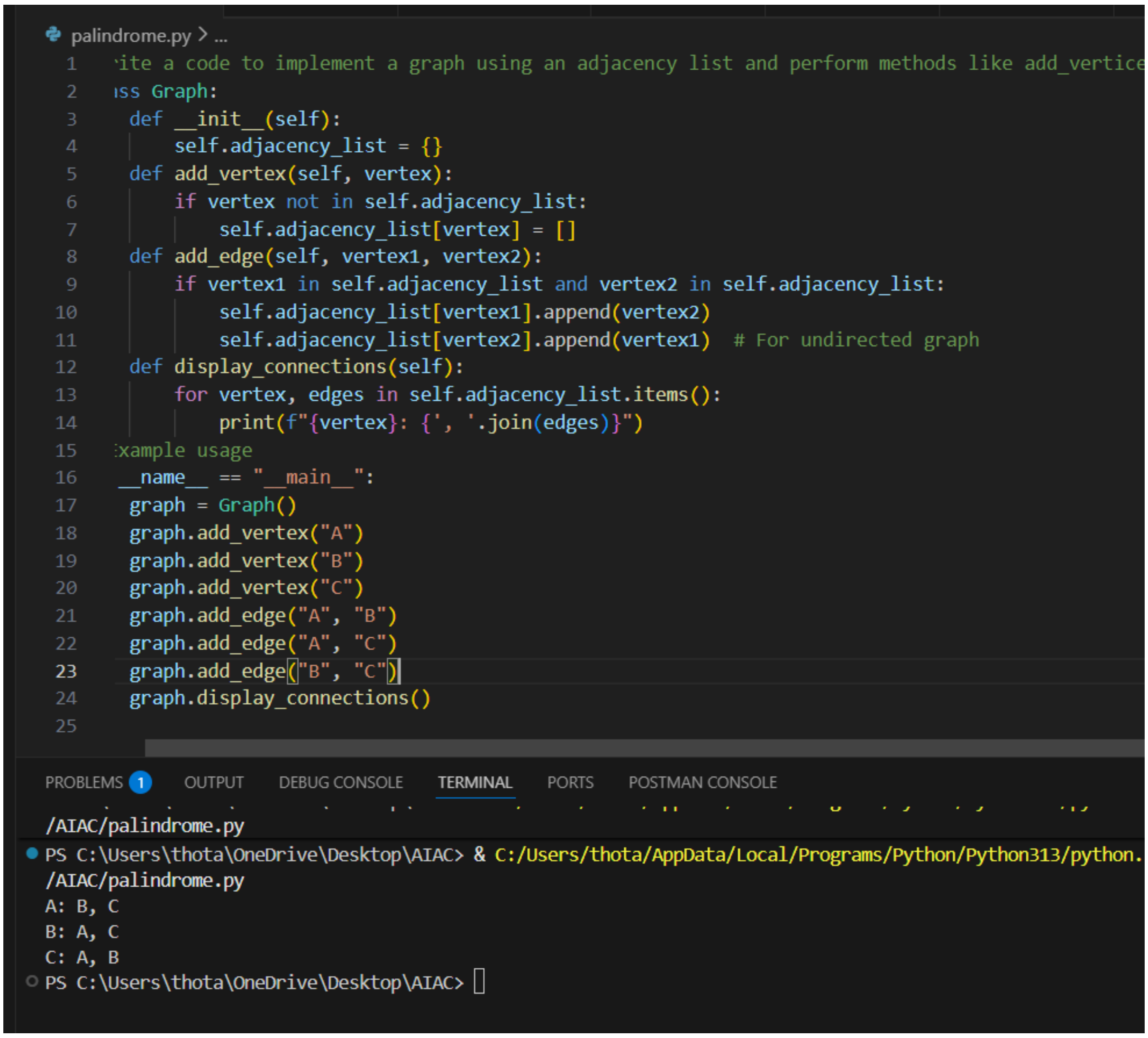
• Graph with methods to add vertices, add edges, and display

connections.

PROMPT:

#Write a code to implement a graph using an adjacency list and perform methods like add_vertices,add_edges and display connections

CODE AND OUTPUT:



palindrome.py > ...

```python
'ite a code to implement a graph using an adjacency list and perform methods like add_vertice
iss Graph:
    def __init__(self):
        self.adjacency_list = {}
    def add_vertex(self, vertex):
        if vertex not in self.adjacency_list:
            self.adjacency_list[vertex] = []
    def add_edge(self, vertex1, vertex2):
        if vertex1 in self.adjacency_list and vertex2 in self.adjacency_list:
            self.adjacency_list[vertex1].append(vertex2)
            self.adjacency_list[vertex2].append(vertex1)  # For undirected graph
    def display_connections(self):
        for vertex, edges in self.adjacency_list.items():
            print(f"{vertex}: {', '.join(edges)}")
:xample usage
 __name__ == "__main__":
    graph = Graph()
    graph.add_vertex("A")
    graph.add_vertex("B")
    graph.add_vertex("C")
    graph.add_edge("A", "B")
    graph.add_edge("A", "C")
    graph.add_edge("B", "C")
    graph.display_connections()
```

```
/AIAC/palindrome.py
PS C:\Users\thota\OneDrive\Desktop\AIAC> & C:/Users/thota/AppData/Local/Programs/Python/Python313/python.
/AIAC/palindrome.py
A: B, C
B: A, C
C: A, B
PS C:\Users\thota\OneDrive\Desktop\AIAC>
```

Task Description #7 – Priority Queue

Task: Use AI to implement a priority queue using Python’s heapq

module.

Sample Input Code:

class PriorityQueue:

pass

Expected Output:

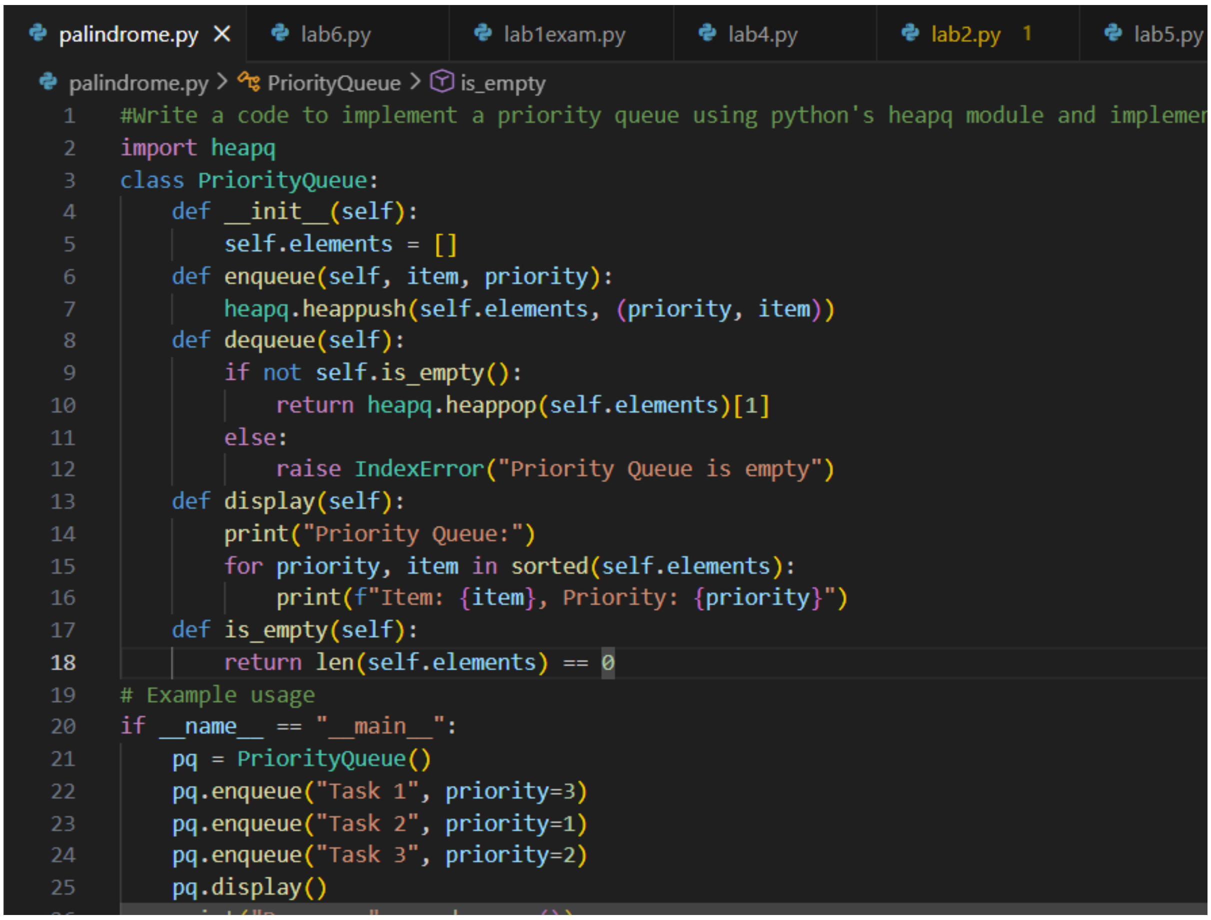
• Implementation with enqueue (priority), dequeue (highest priority),

and display methods.

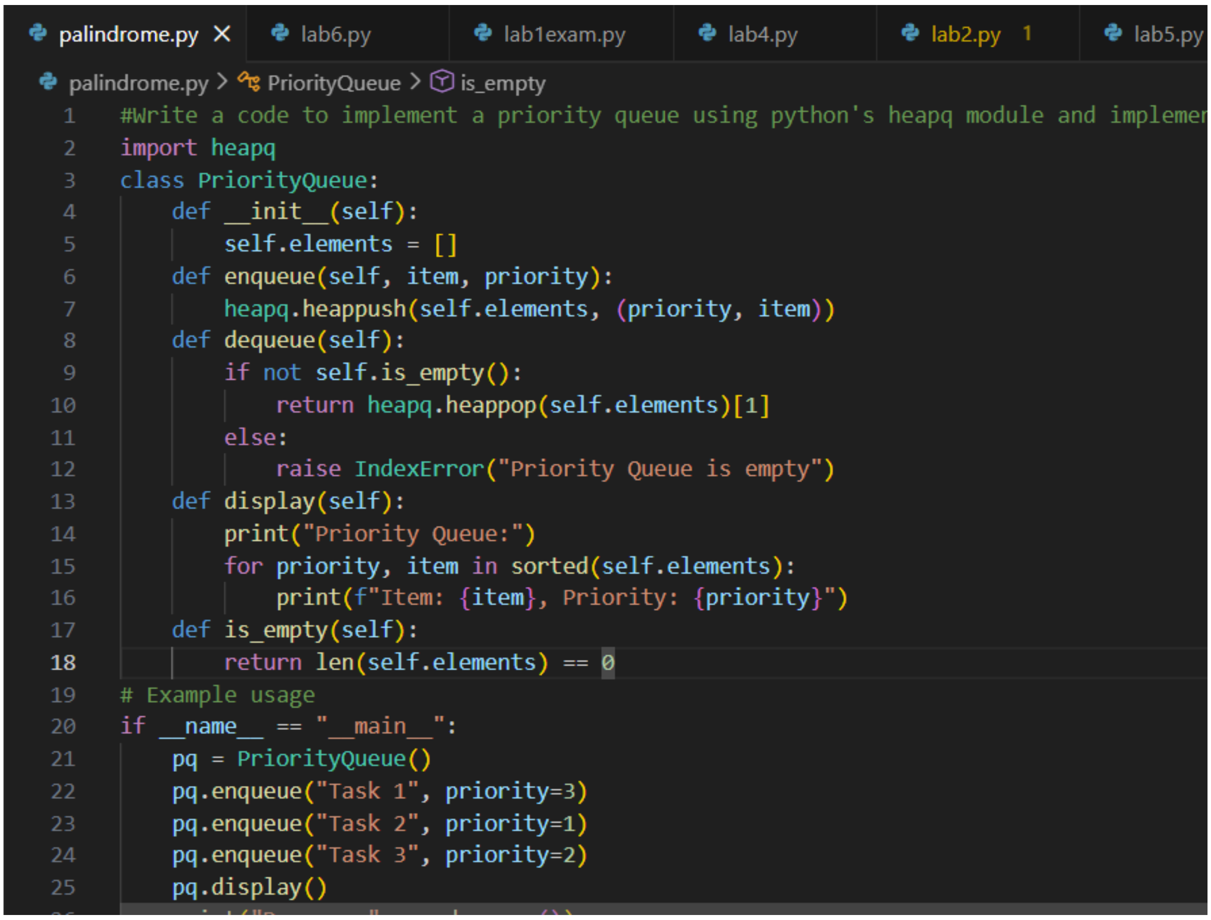
PROMPT:

#Write a code to implement a priority queue using python's heapq module and implement with methods for enqueue,dequeue and display methods

CODE AND OUTPUT:



```python
#Write a code to implement a priority queue using python's heapq module and implemen
import heapq
class PriorityQueue:
    def __init__(self):
        self.elements = []
    def enqueue(self, item, priority):
        heapq.heappush(self.elements, (priority, item))
    def dequeue(self):
        if not self.is_empty():
            return heapq.heappop(self.elements)[1]
        else:
            raise IndexError("Priority Queue is empty")
    def display(self):
        print("Priority Queue:")
        for priority, item in sorted(self.elements):
            print(f"Item: {item}, Priority: {priority}")
    def is_empty(self):
        return len(self.elements) == 0
# Example usage
if __name__ == "__main__":
    pq = PriorityQueue()
    pq.enqueue("Task 1", priority=3)
    pq.enqueue("Task 2", priority=1)
    pq.enqueue("Task 3", priority=2)
    pq.display()
```



```python
#Write a code to implement a priority queue using python's heapq module and implemen
import heapq
class PriorityQueue:
    def __init__(self):
        self.elements = []
    def enqueue(self, item, priority):
        heapq.heappush(self.elements, (priority, item))
    def dequeue(self):
        if not self.is_empty():
            return heapq.heappop(self.elements)[1]
        else:
            raise IndexError("Priority Queue is empty")
    def display(self):
        print("Priority Queue:")
        for priority, item in sorted(self.elements):
            print(f"Item: {item}, Priority: {priority}")
    def is_empty(self):
        return len(self.elements) == 0
# Example usage
if __name__ == "__main__":
    pq = PriorityQueue()
    pq.enqueue("Task 1", priority=3)
    pq.enqueue("Task 2", priority=1)
    pq.enqueue("Task 3", priority=2)
    pq.display()
```

Task Description #8 – Deque

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

Sample Input Code:

class DequeDS:

pass

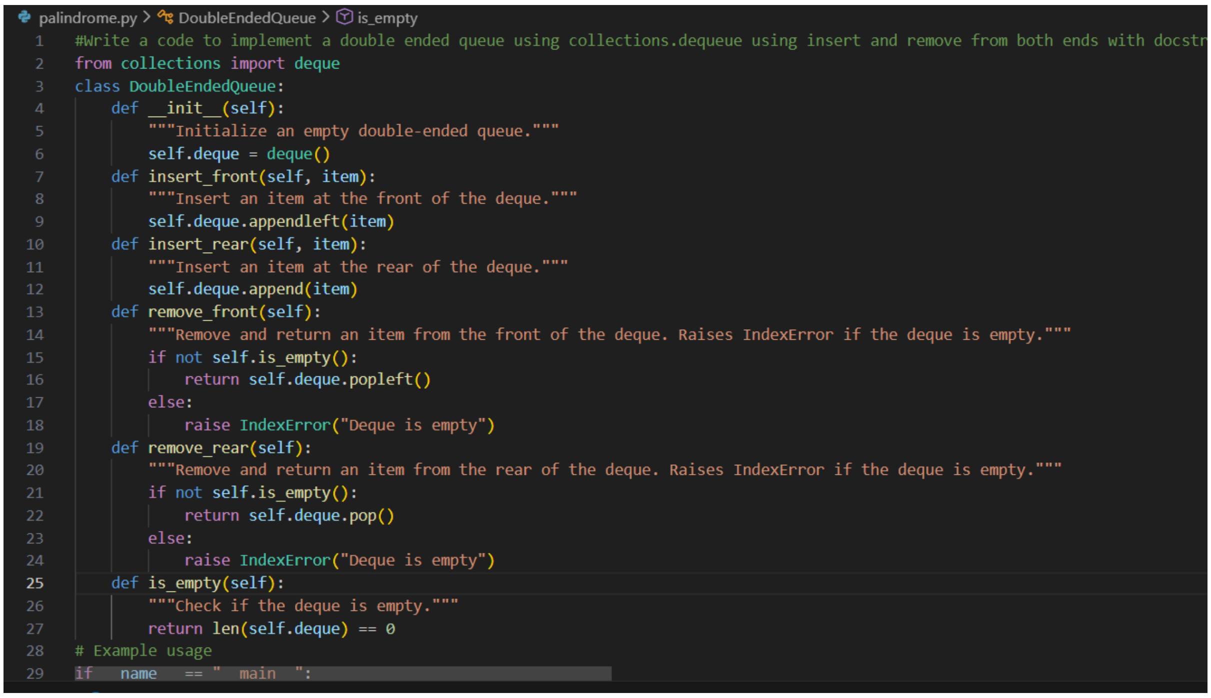
Expected Output:

• Insert and remove from both ends with docstrings.

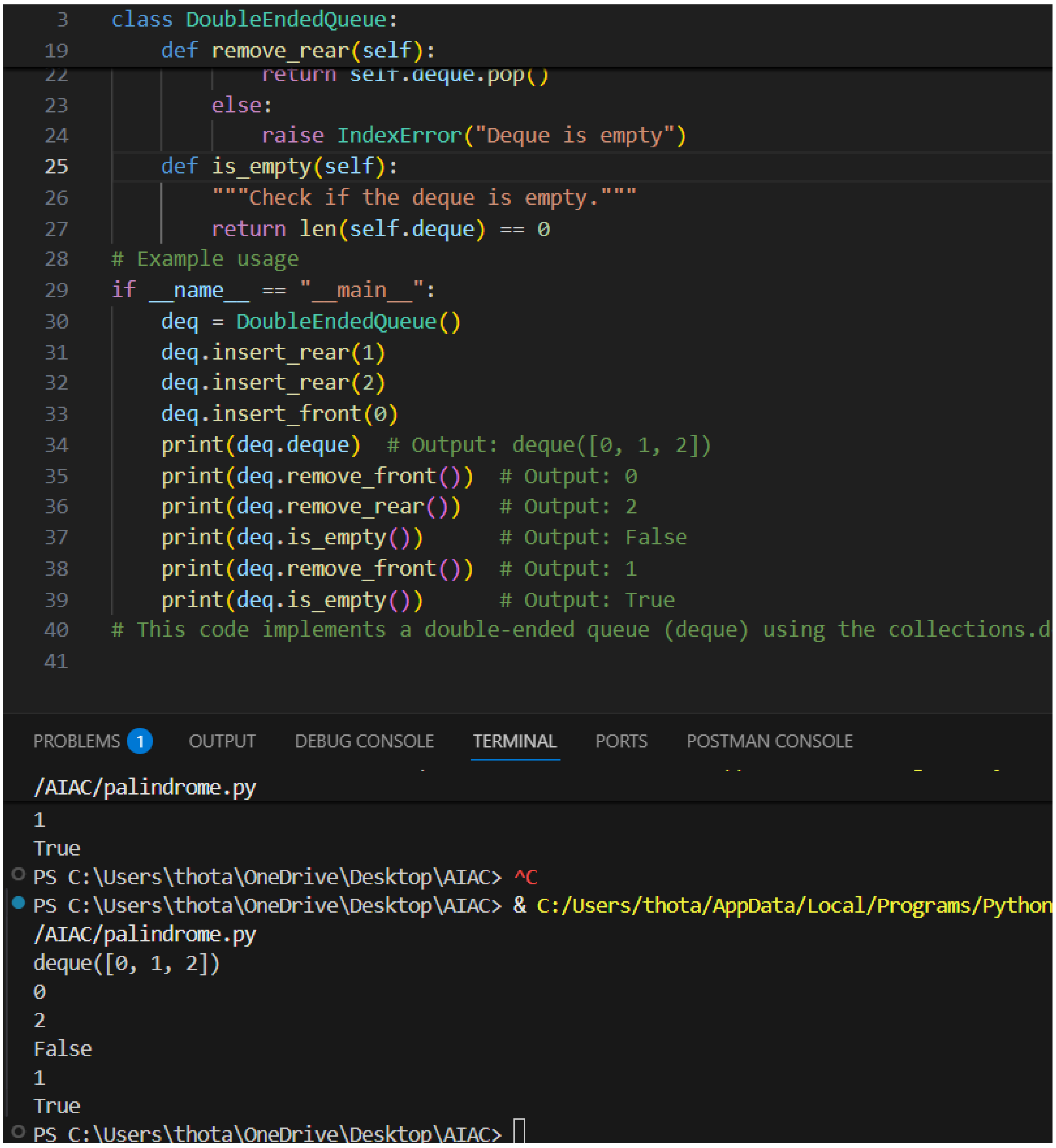
PROMPT:

#Write a code to implement a double ended queue using collections.dequeue using insert and remove from both ends with docstring

CODE AND OUTPUT:



```python
#Write a code to implement a double ended queue using collections.dequeue using insert and remove from both ends with docstr
from collections import deque
class DoubleEndedQueue:
    def __init__(self):
        """Initialize an empty double-ended queue."""
        self.deque = deque()
    def insert_front(self, item):
        """Insert an item at the front of the deque."""
        self.deque.appendleft(item)
    def insert_rear(self, item):
        """Insert an item at the rear of the deque."""
        self.deque.append(item)
    def remove_front(self):
        """Remove and return an item from the front of the deque. Raises IndexError if the deque is empty."""
        if not self.is_empty():
            return self.deque.popleft()
        else:
            raise IndexError("Deque is empty")
    def remove_rear(self):
        """Remove and return an item from the rear of the deque. Raises IndexError if the deque is empty."""
        if not self.is_empty():
            return self.deque.pop()
        else:
            raise IndexError("Deque is empty")
    def is_empty(self):
        """Check if the deque is empty."""
        return len(self.deque) == 0
# Example usage
if   name   == "  main  ":
```



```python
class DoubleEndedQueue:
    def remove_rear(self):
            return self.deque.pop()
        else:
            raise IndexError("Deque is empty")
    def is_empty(self):
        """Check if the deque is empty."""
        return len(self.deque) == 0
# Example usage
if __name__ == "__main__":
    deq = DoubleEndedQueue()
    deq.insert_rear(1)
    deq.insert_rear(2)
    deq.insert_front(0)
    print(deq.deque)  # Output: deque([0, 1, 2])
    print(deq.remove_front())  # Output: 0
    print(deq.remove_rear())   # Output: 2
    print(deq.is_empty())      # Output: False
    print(deq.remove_front())  # Output: 1
    print(deq.is_empty())      # Output: True
# This code implements a double-ended queue (deque) using the collections.d
```

```
/AIAC/palindrome.py
1
True
PS C:\Users\thota\OneDrive\Desktop\AIAC> ^C
PS C:\Users\thota\OneDrive\Desktop\AIAC> & C:/Users/thota/AppData/Local/Programs/Python
/AIAC/palindrome.py
deque([0, 1, 2])
0
2
False
1
True
PS C:\Users\thota\OneDrive\Desktop\AIAC> 
```