SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENTOF COMPUTER SCIENCE ENGINEERING	
ProgramName:M. Tech/MCA/MSC		**AssignmentType: Lab**	**AcademicYear:**2025-2026
CourseCoordinatorName		Venkataramana Veeramsetty	
CourseCode		**CourseTitle**	AI Assisted Problem Solving Using Python
Year/Sem	II/I	**Regulation**	R24
DateandDay of Assignment	Week5- Monday	**Time(s)**	
Duration	2 Hours	**Applicableto Batches**	
AssignmentNumber:11.3(Presentassignmentnumber)/**24**(Totalnumberofassignments)			

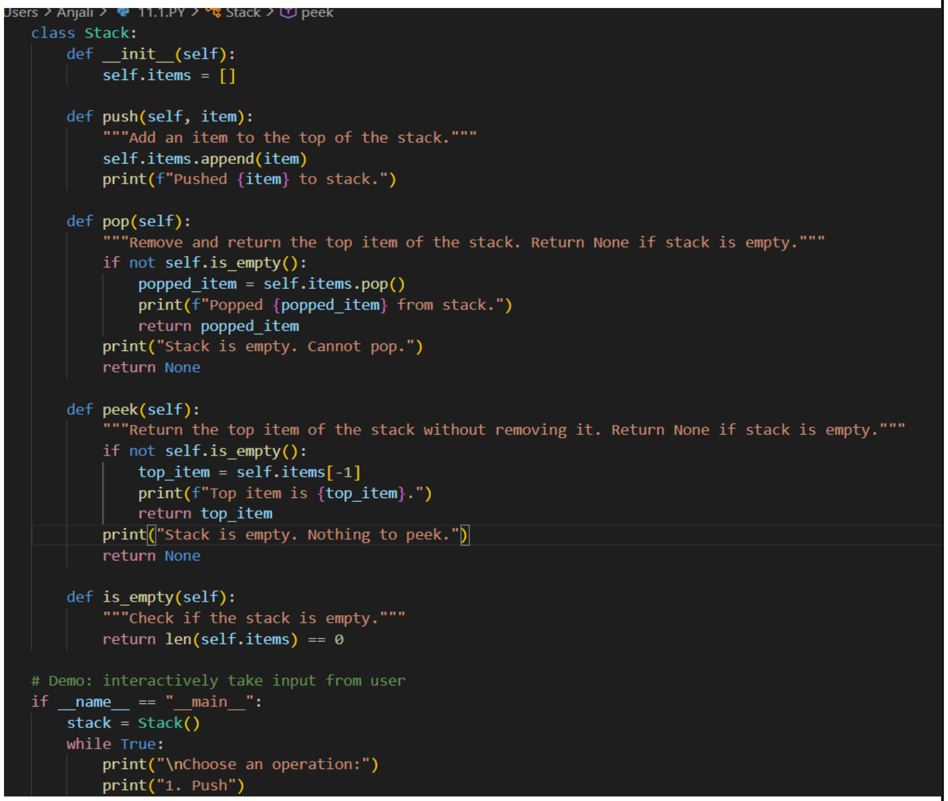
Q.No.	**Question**	***ExpectedTime to complete***
1	**Lab 11 – Data Structures with AI: Implementing Fundamental Structures** **Lab Objectives** ● To implement fundamental data structures with the assistance of AI tools. ● To understand how AI suggests different implementations and optimizations. ● To analyze the readability, correctness, and performance of AI-generated code. ● To reinforce problem-solving skills using AI-powered coding assistance. **Learning Outcomes** After completing this lab, students will be able to: 1. Implement stack, queue, and linked list using Python with AI support. 2. Use AI tools to optimize and refactor basic data structure operations. 3. Compare multiple AI-suggested implementations for the same structure. 4. Apply AI assistance to generate test cases for verifying data structure behavior.	Week5 - Monday

5. Demonstrate understanding of trade-offs in AI-generated solutions.

Task Description #1 – Stack class implementation

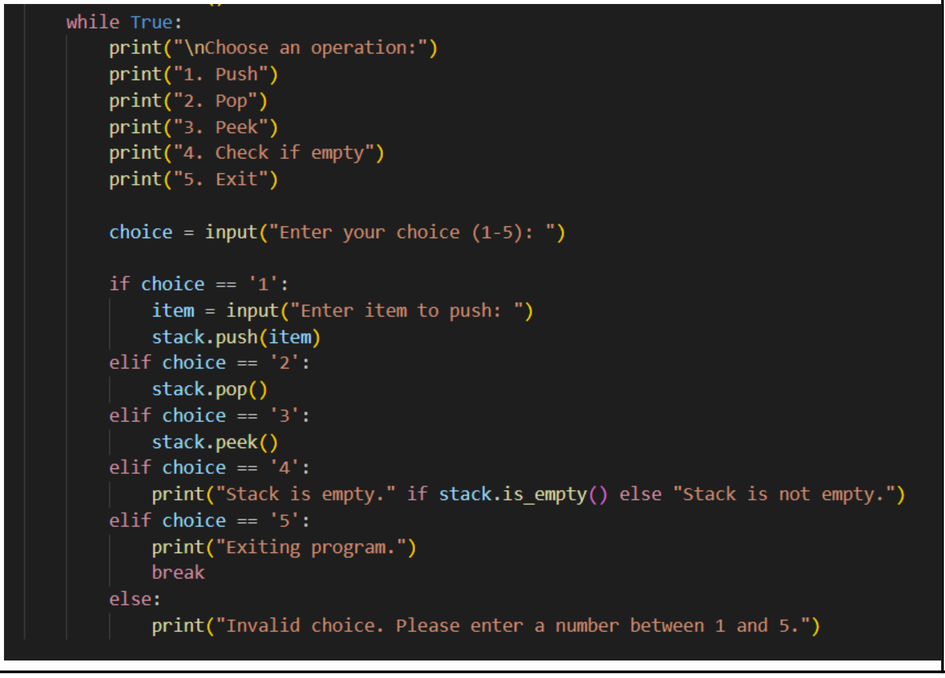
Task: Ask AI to implement a stack class with push(), pop(), peek() and is_empty() methods

CODE:

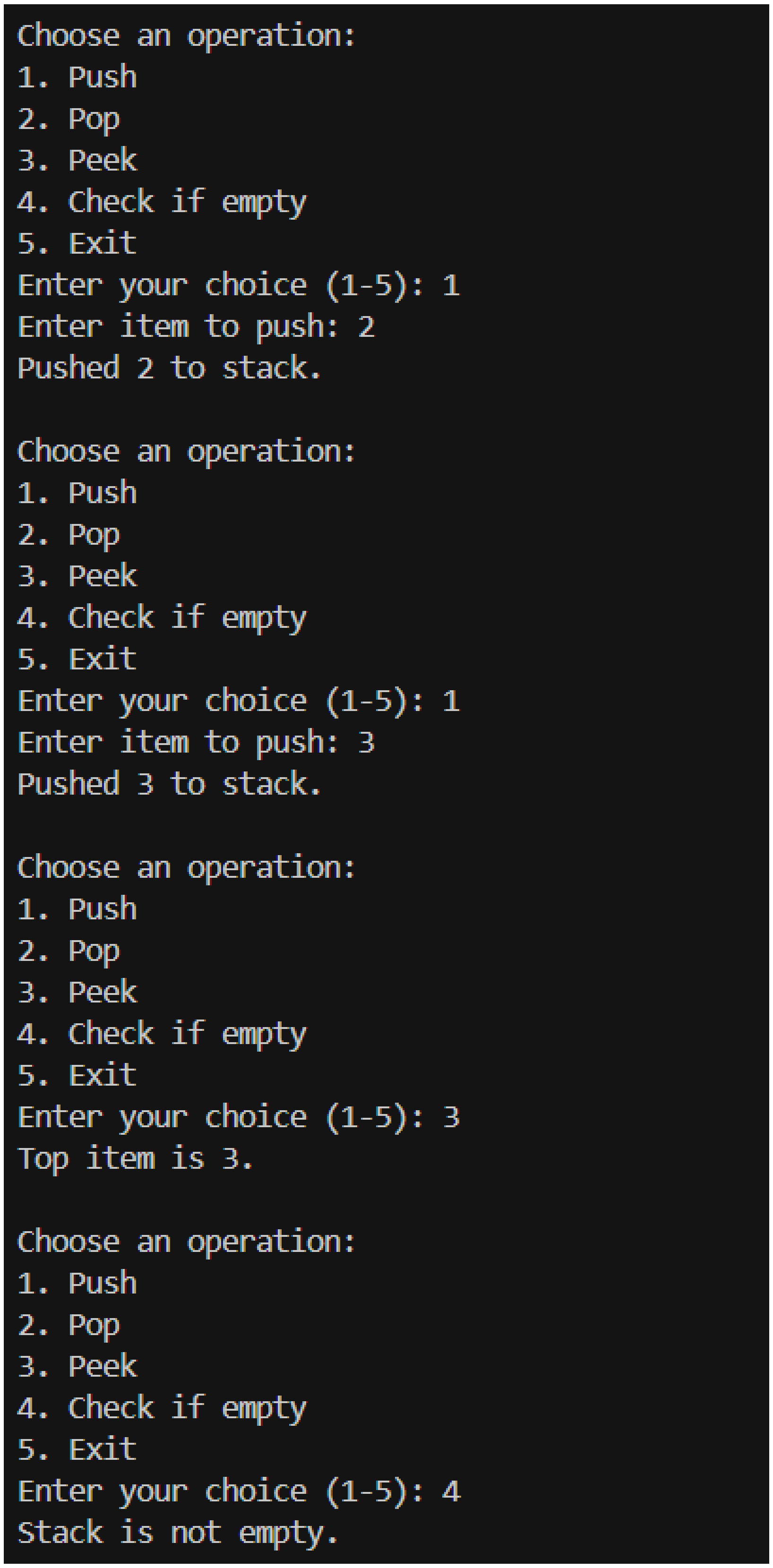


Users > Anjali > 11.1.PY > Stack > peek

```python
class Stack:
    def __init__(self):
        self.items = []

    def push(self, item):
        """Add an item to the top of the stack."""
        self.items.append(item)
        print(f"Pushed {item} to stack.")

    def pop(self):
        """Remove and return the top item of the stack. Return None if stack is empty."""
        if not self.is_empty():
            popped_item = self.items.pop()
            print(f"Popped {popped_item} from stack.")
            return popped_item
        print("Stack is empty. Cannot pop.")
        return None

    def peek(self):
        """Return the top item of the stack without removing it. Return None if stack is empty."""
        if not self.is_empty():
            top_item = self.items[-1]
            print(f"Top item is {top_item}.")
            return top_item
        print("Stack is empty. Nothing to peek.")
        return None

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

# Demo: interactively take input from user
if __name__ == "__main__":
    stack = Stack()
    while True:
        print("\nChoose an operation:")
        print("1. Push")
```



```python
    while True:
        print("\nChoose an operation:")
        print("1. Push")
        print("2. Pop")
        print("3. Peek")
        print("4. Check if empty")
        print("5. Exit")

        choice = input("Enter your choice (1-5): ")

        if choice == '1':
            item = input("Enter item to push: ")
            stack.push(item)
        elif choice == '2':
            stack.pop()
        elif choice == '3':
            stack.peek()
        elif choice == '4':
            print("Stack is empty." if stack.is_empty() else "Stack is not empty.")
        elif choice == '5':
            print("Exiting program.")
            break
        else:
            print("Invalid choice. Please enter a number between 1 and 5.")
```

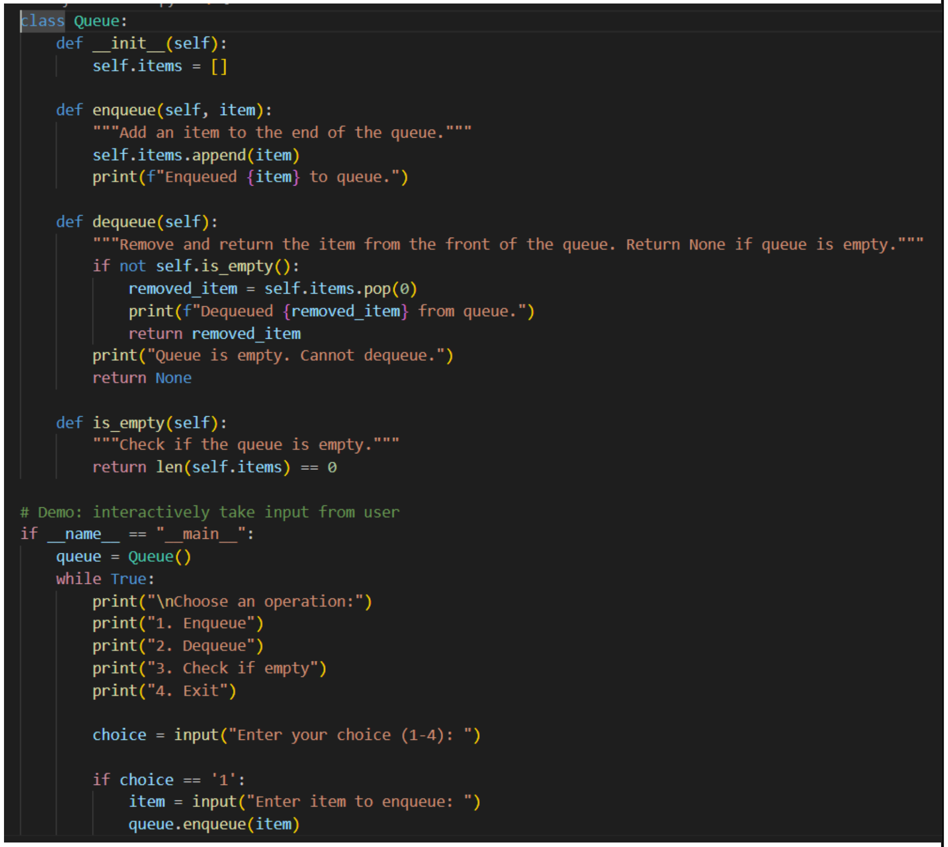
	OUTPUT: 	

```
Choose an operation:
1. Push
2. Pop
3. Peek
4. Check if empty
5. Exit
Enter your choice (1-5): 1
Enter item to push: 2
Pushed 2 to stack.

Choose an operation:
1. Push
2. Pop
3. Peek
4. Check if empty
5. Exit
Enter your choice (1-5): 1
Enter item to push: 3
Pushed 3 to stack.

Choose an operation:
1. Push
2. Pop
3. Peek
4. Check if empty
5. Exit
Enter your choice (1-5): 3
Top item is 3.

Choose an operation:
1. Push
2. Pop
3. Peek
4. Check if empty
5. Exit
Enter your choice (1-5): 4
Stack is not empty.
```

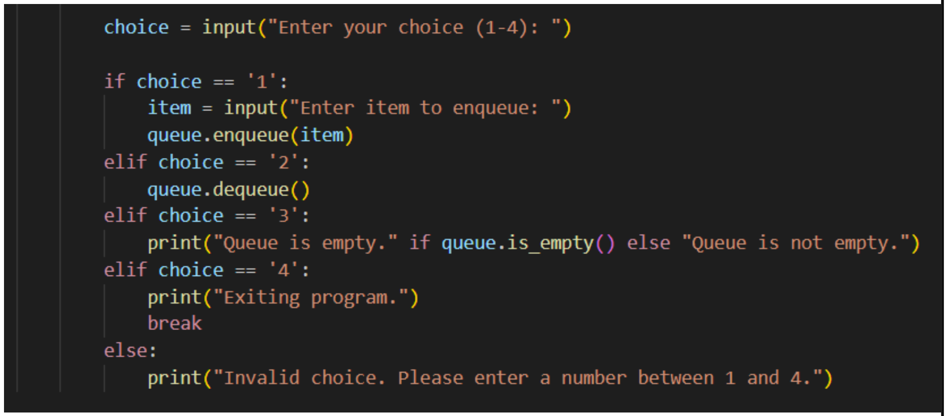
Task Description #2 – Queue Implementation

Task: Use AI to generate a Queue class with enqueue(), dequeue(), and is_empty().

CODE:

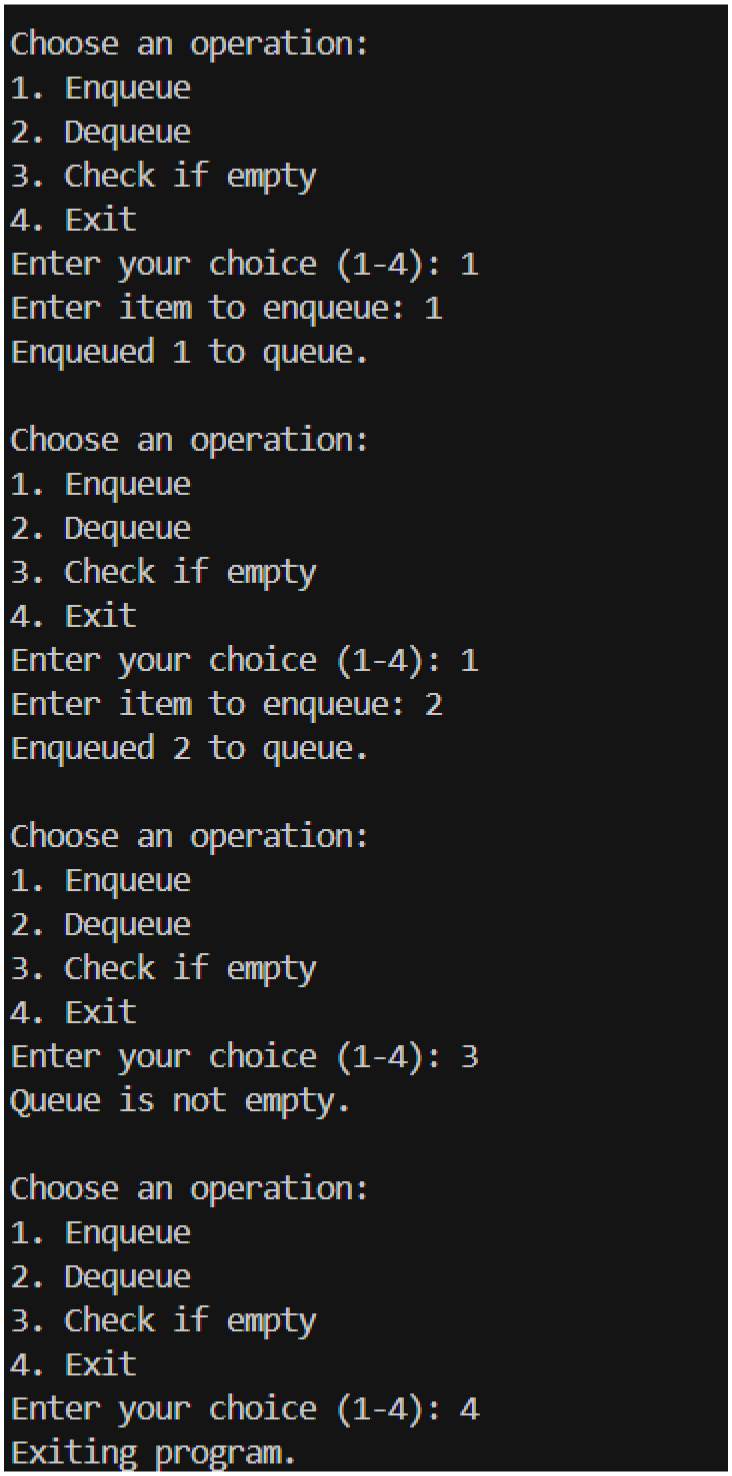


```python
class Queue:
    def __init__(self):
        self.items = []

    def enqueue(self, item):
        """Add an item to the end of the queue."""
        self.items.append(item)
        print(f"Enqueued {item} to queue.")

    def dequeue(self):
        """Remove and return the item from the front of the queue. Return None if queue is empty."""
        if not self.is_empty():
            removed_item = self.items.pop(0)
            print(f"Dequeued {removed_item} from queue.")
            return removed_item
        print("Queue is empty. Cannot dequeue.")
        return None

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

# Demo: interactively take input from user
if __name__ == "__main__":
    queue = Queue()
    while True:
        print("\nChoose an operation:")
        print("1. Enqueue")
        print("2. Dequeue")
        print("3. Check if empty")
        print("4. Exit")

        choice = input("Enter your choice (1-4): ")

        if choice == '1':
            item = input("Enter item to enqueue: ")
            queue.enqueue(item)
```



```python
        choice = input("Enter your choice (1-4): ")

        if choice == '1':
            item = input("Enter item to enqueue: ")
            queue.enqueue(item)
        elif choice == '2':
            queue.dequeue()
        elif choice == '3':
            print("Queue is empty." if queue.is_empty() else "Queue is not empty.")
        elif choice == '4':
            print("Exiting program.")
            break
        else:
            print("Invalid choice. Please enter a number between 1 and 4.")
```

OUTPUT:

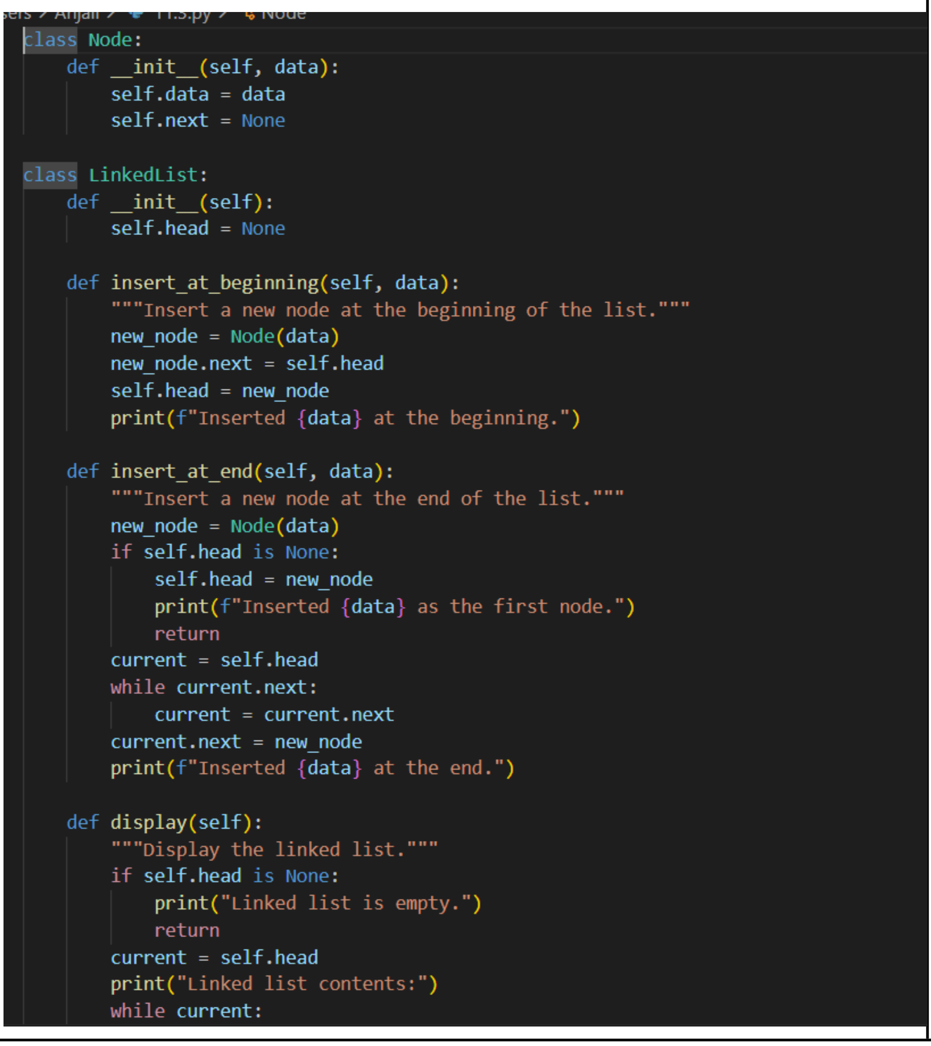


```
Choose an operation:
1. Enqueue
2. Dequeue
3. Check if empty
4. Exit
Enter your choice (1-4): 1
Enter item to enqueue: 1
Enqueued 1 to queue.

Choose an operation:
1. Enqueue
2. Dequeue
3. Check if empty
4. Exit
Enter your choice (1-4): 1
Enter item to enqueue: 2
Enqueued 2 to queue.

Choose an operation:
1. Enqueue
2. Dequeue
3. Check if empty
4. Exit
Enter your choice (1-4): 3
Queue is not empty.

Choose an operation:
1. Enqueue
2. Dequeue
3. Check if empty
4. Exit
Enter your choice (1-4): 4
Exiting program.
```

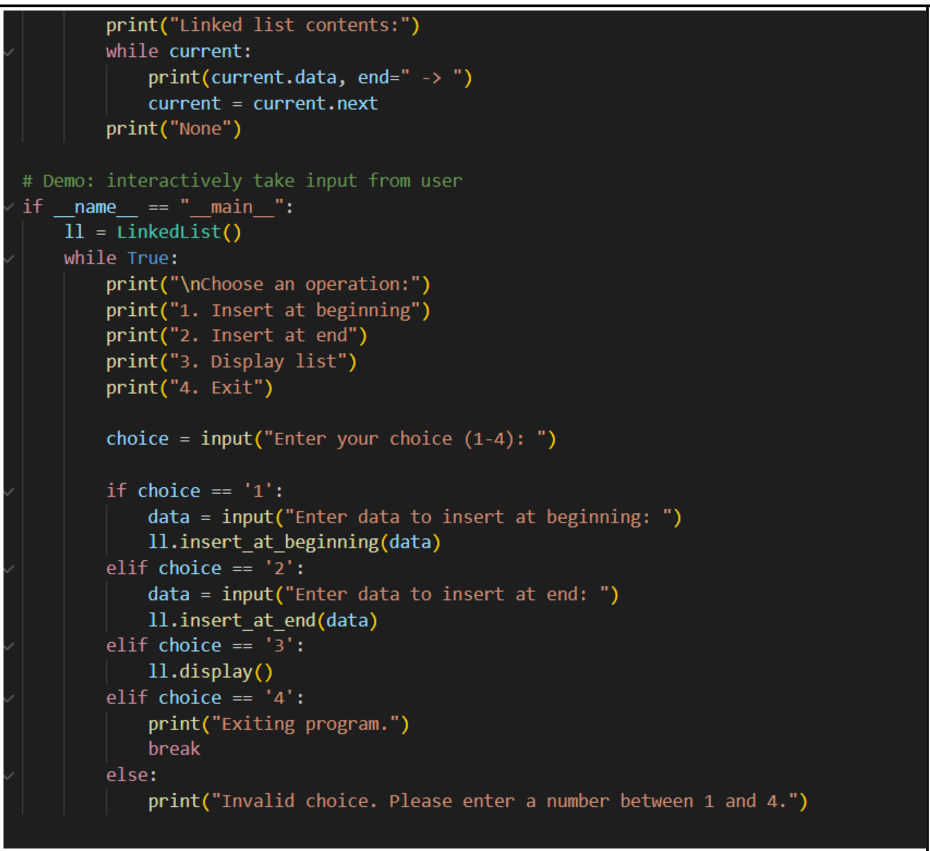
Task Description #3 – Linked List Implementation

Task: Ask AI to create a singly linked list with insert_at_end(), insert_at_beginning(), and display().

CODE:

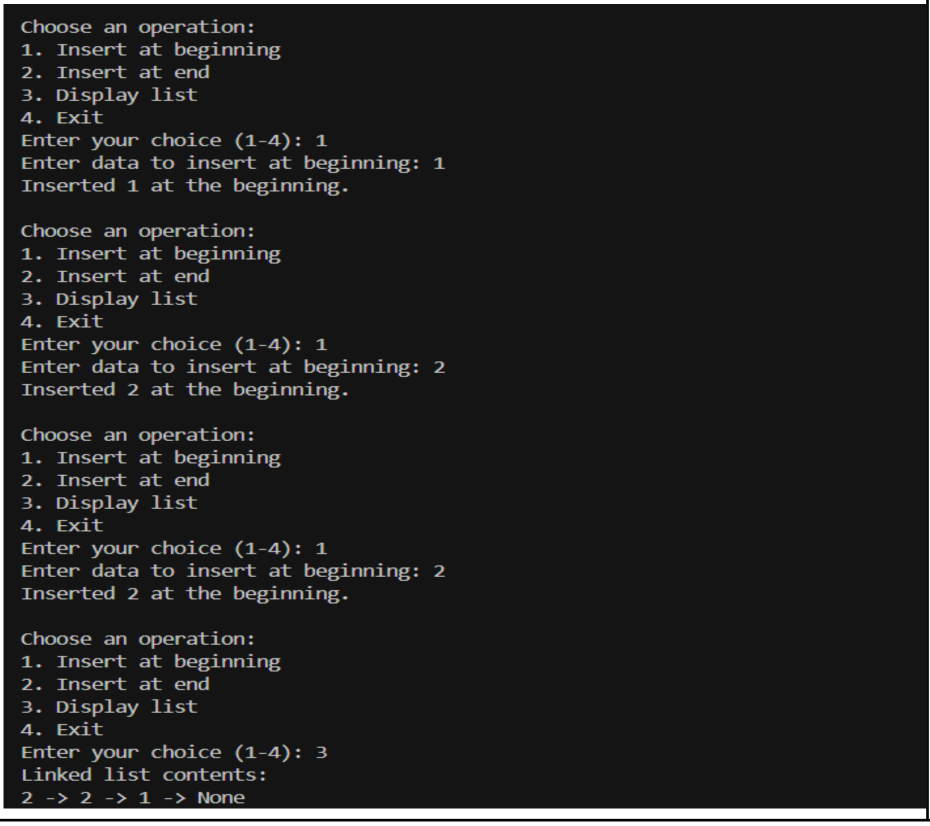


```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def insert_at_beginning(self, data):
        """Insert a new node at the beginning of the list."""
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node
        print(f"Inserted {data} at the beginning.")

    def insert_at_end(self, data):
        """Insert a new node at the end of the list."""
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            print(f"Inserted {data} as the first node.")
            return
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node
        print(f"Inserted {data} at the end.")

    def display(self):
        """Display the linked list."""
        if self.head is None:
            print("Linked list is empty.")
            return
        current = self.head
        print("Linked list contents:")
        while current:
```



```python
        print("Linked list contents:")
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")

# Demo: interactively take input from user
if __name__ == "__main__":
    ll = LinkedList()
    while True:
        print("\nChoose an operation:")
        print("1. Insert at beginning")
        print("2. Insert at end")
        print("3. Display list")
        print("4. Exit")

        choice = input("Enter your choice (1-4): ")

        if choice == '1':
            data = input("Enter data to insert at beginning: ")
            ll.insert_at_beginning(data)
        elif choice == '2':
            data = input("Enter data to insert at end: ")
            ll.insert_at_end(data)
        elif choice == '3':
            ll.display()
        elif choice == '4':
            print("Exiting program.")
            break
        else:
            print("Invalid choice. Please enter a number between 1 and 4.")
```

OUTPUT:

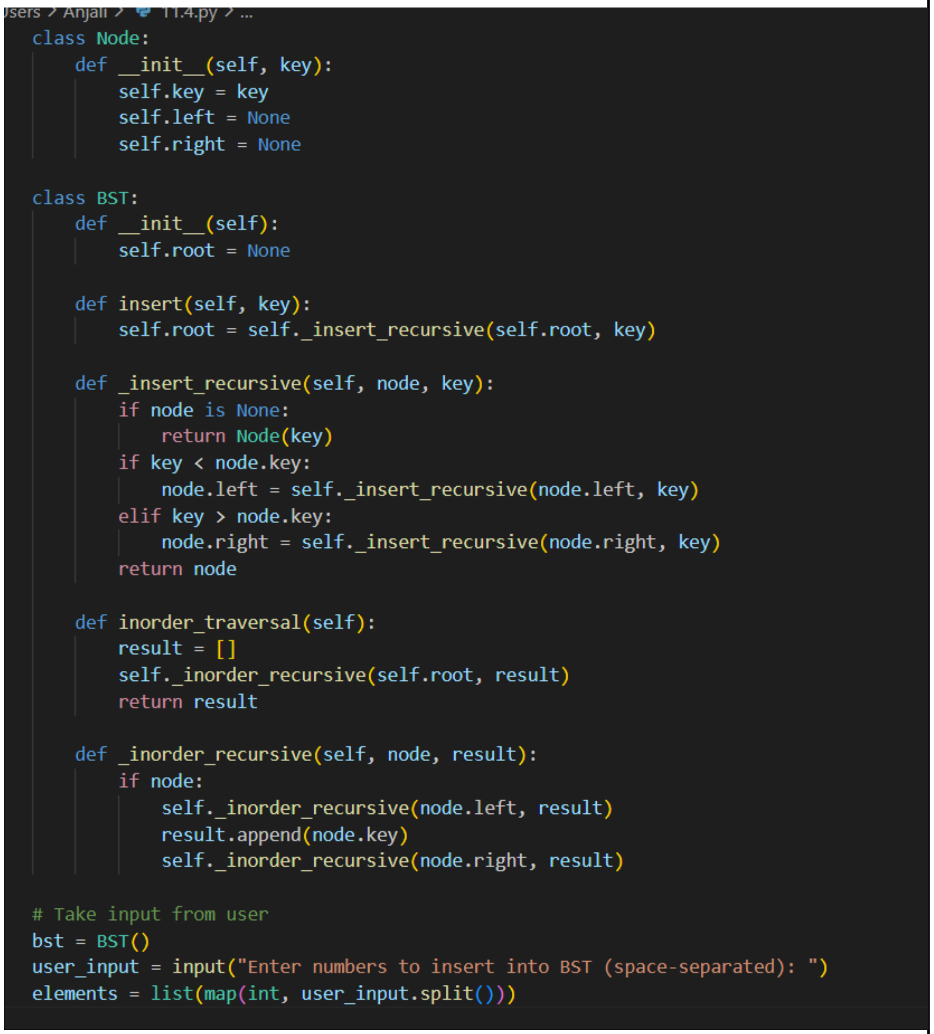


```
Choose an operation:
1. Insert at beginning
2. Insert at end
3. Display list
4. Exit
Enter your choice (1-4): 1
Enter data to insert at beginning: 1
Inserted 1 at the beginning.

Choose an operation:
1. Insert at beginning
2. Insert at end
3. Display list
4. Exit
Enter your choice (1-4): 1
Enter data to insert at beginning: 2
Inserted 2 at the beginning.

Choose an operation:
1. Insert at beginning
2. Insert at end
3. Display list
4. Exit
Enter your choice (1-4): 1
Enter data to insert at beginning: 2
Inserted 2 at the beginning.

Choose an operation:
1. Insert at beginning
2. Insert at end
3. Display list
4. Exit
Enter your choice (1-4): 3
Linked list contents:
2 -> 2 -> 1 -> None
```

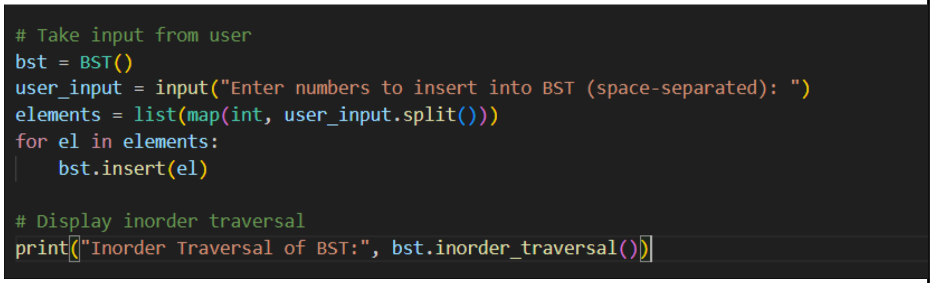
Task Description #4 – Binary Search Tree (BST)

Task: Ask AI to generate a simple BST with insert() and inorder_traversal().

CODE:

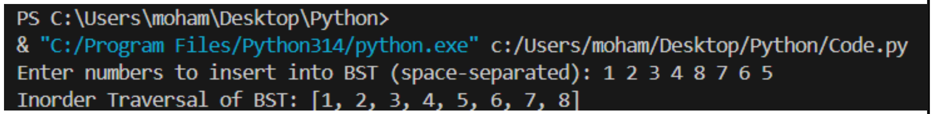


```
Jsers > Anjali > 11.4.py > ...
class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

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

    def insert(self, key):
        self.root = self._insert_recursive(self.root, key)

    def _insert_recursive(self, node, key):
        if node is None:
            return Node(key)
        if key < node.key:
            node.left = self._insert_recursive(node.left, key)
        elif key > node.key:
            node.right = self._insert_recursive(node.right, key)
        return node

    def inorder_traversal(self):
        result = []
        self._inorder_recursive(self.root, result)
        return result

    def _inorder_recursive(self, node, result):
        if node:
            self._inorder_recursive(node.left, result)
            result.append(node.key)
            self._inorder_recursive(node.right, result)

# Take input from user
bst = BST()
user_input = input("Enter numbers to insert into BST (space-separated): ")
elements = list(map(int, user_input.split()))
```



```
# Take input from user
bst = BST()
user_input = input("Enter numbers to insert into BST (space-separated): ")
elements = list(map(int, user_input.split()))
for el in elements:
    bst.insert(el)

# Display inorder traversal
print("Inorder Traversal of BST:", bst.inorder_traversal())
```

OUTPUT:



```
PS C:\Users\moham\Desktop\Python>
& "C:/Program Files/Python314/python.exe" c:/Users/moham/Desktop/Python/Code.py
Enter numbers to insert into BST (space-separated): 1 2 3 4 8 7 6 5
Inorder Traversal of BST: [1, 2, 3, 4, 5, 6, 7, 8]
```