.AI ASSISTED CODING.

LAB ASSIGNMENT -11.3

Program	: B-TECH
Specialization	: AIML
Name of Student	: MD.Heena
Enrollment-No	: 2403A54067
Batch No	: 01
Date	: 08-10-2025

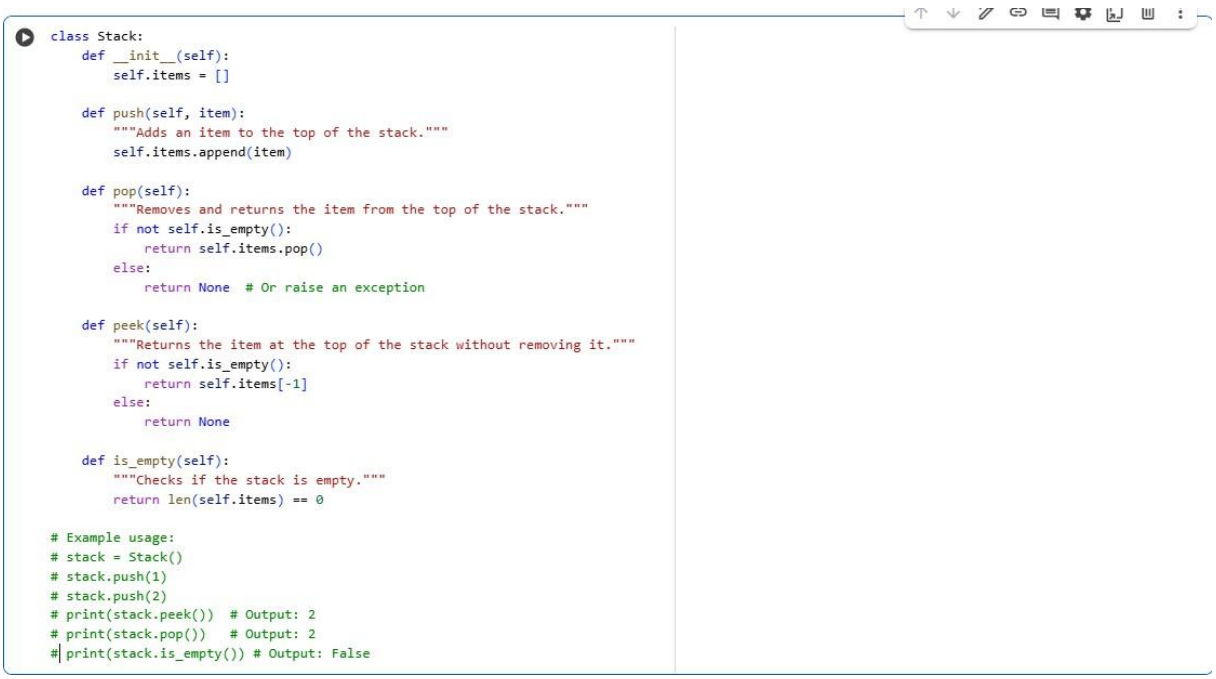
TASK DESCRIPTION-1

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

PROMPT:-

Implement a stack class with push(), pop(), peek() and
is_empty() methods.

Code Generated:-



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

    def push(self, item):
        """Adds an item to the top of the stack."""
        self.items.append(item)

    def pop(self):
        """Removes and returns the item from the top of the stack."""
        if not self.is_empty():
            return self.items.pop()
        else:
            return None  # Or raise an exception

    def peek(self):
        """Returns the item at the top of the stack without removing it."""
        if not self.is_empty():
            return self.items[-1]
        else:
            return None

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

# Example usage:
# stack = Stack()
# stack.push(1)
# stack.push(2)
# print(stack.peek())  # Output: 2
# print(stack.pop())   # Output: 2
# print(stack.is_empty()) # Output: False
```

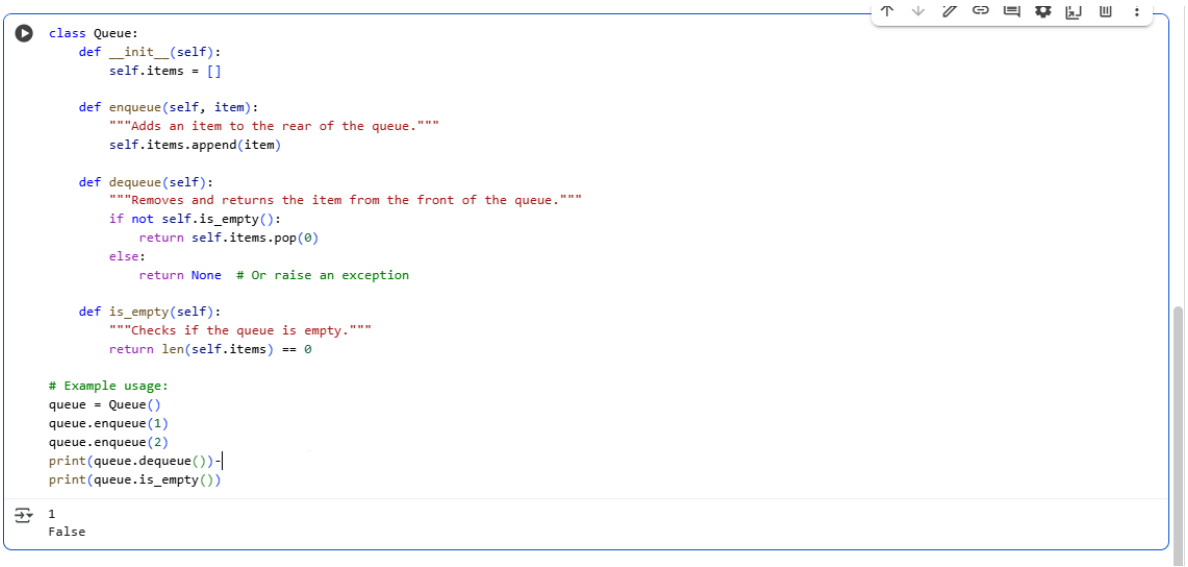
TASK DESCRIPTION-2:

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

PROMPT:-

generate a Queue class with enqueue(), dequeue(), and is_empty().

GENERATED CODE:-



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

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

    def dequeue(self):
        """Removes and returns the item from the front of the queue."""
        if not self.is_empty():
            return self.items.pop(0)
        else:
            return None  # Or raise an exception

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

# Example usage:
queue = Queue()
queue.enqueue(1)
queue.enqueue(2)
print(queue.dequeue())
print(queue.is_empty())
```

```
1
False
```

TASK DESCRIPTION-3

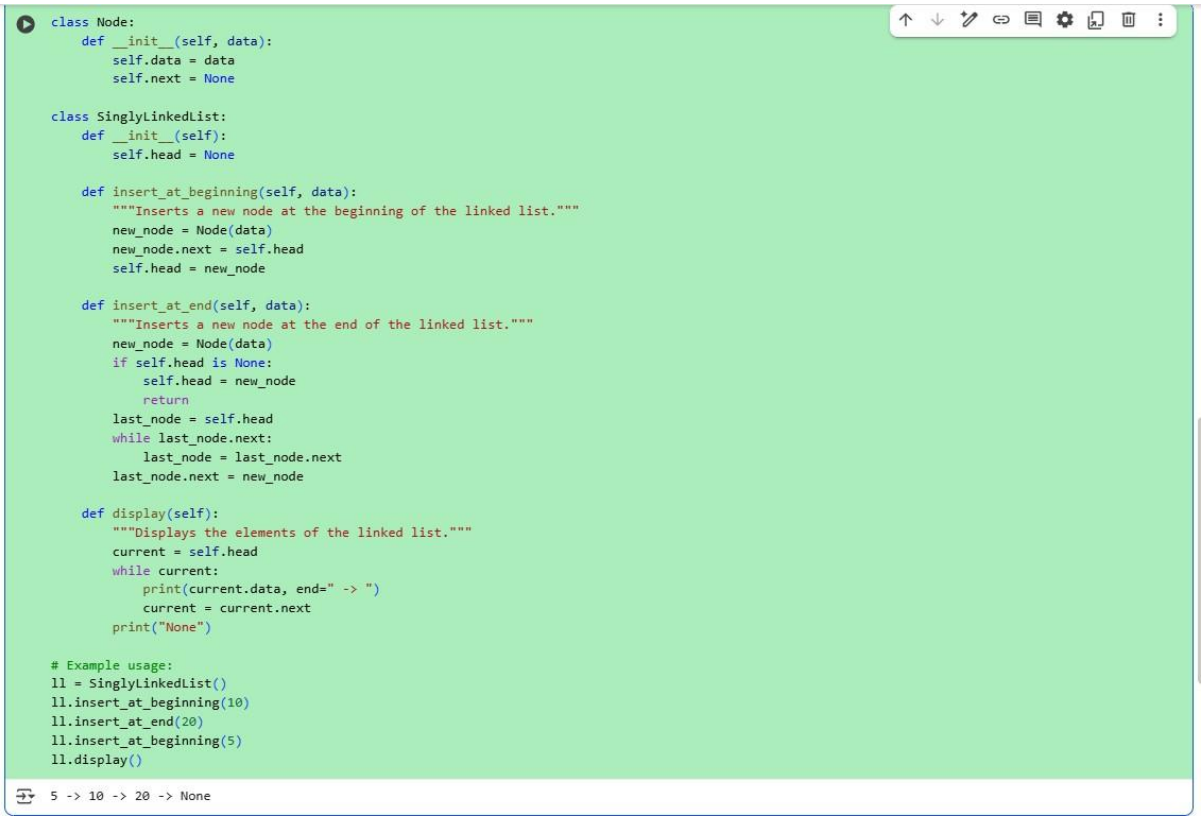
Linked List Implementation

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

PROMPT:-

Create a singly linked list with insert_at_end(), insert_at_beginning(), and display()

GENERATED CODE:



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

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

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

    def insert_at_end(self, data):
        """Inserts a new node at the end of the linked list."""
        new_node = Node(data)
        if self.head is None:
            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):
        """Displays the elements of the linked list."""
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")

# Example usage:
ll = SinglyLinkedList()
ll.insert_at_beginning(10)
ll.insert_at_end(20)
ll.insert_at_beginning(5)
ll.display()
```

```
5 -> 10 -> 20 -> None
```

TASK DESCRIPTION-4:

Binary Search Tree (BST)

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

inorder_traversal().

PROMPT:-

Generate a simple BST with insert() and inorder_traversal().

GENERATED CODE:-

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

    def insert(self, key):
        """Inserts a new node with the given key into the BST."""
        self.root = self._insert_recursive(self.root, key)

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

    def inorder_traversal(self):
        """Performs an in-order traversal of the BST and prints the keys."""
        self._inorder_recursive(self.root)
        print() # Add a newline for cleaner output

    def _inorder_recursive(self, root):
        if root:
            self._inorder_recursive(root.left)
            print(root.key, end=" ")
            self._inorder_recursive(root.right)

# Example usage:
bst = BST()
bst.insert(50)
bst.insert(30)
bst.insert(20)
bst.insert(40)
bst.insert(70)
bst.insert(60)
bst.insert(80)
bst.inorder_traversal()
```

```
20 30 40 50 60 70 80
```

THANK YOU