AI ASSISTED CODING LAB 11.3

Lab 11: Data Structures with AI: Implementing Fundamental Structures

ENROLLMENT NO: 2403A52006

NAME: MOHAMMED MUSTAFA

COURSE: CSM(AI/ML)

BATCH: 01

DATE: 8-10-2025

Task Description 1 – Stack class implementation
Task: Ask AI to implement a stack class with push(), pop(), peek() and is_empty() methods

Used Prompt:

Generate a python program to implement a stack class with push(), pop(), peek() , and is_empty methods take input from the user.

```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 like IndexError

    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  # Or raise an exception

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

# Example usage with user input:
stack = Stack()
```

```python
while True:
    print("\nStack Operations:")
    print("1. Push")
    print("2. Pop")
    print("3. Peek")
    print("4. Check if Empty")
    print("5. Exit")

    choice = input("Enter your choice: ")

    if choice == '1':
        item = input("Enter item to push: ")
        stack.push(item)
        print(f"Pushed '{item}' onto the stack.")
    elif choice == '2':
        popped_item = stack.pop()
        if popped_item is not None:
            print(f"Popped '{popped_item}' from the stack.")
        else:
            print("Stack is empty. Cannot pop.")
    elif choice == '3':
        peeked_item = stack.peek()
        if peeked_item is not None:
            print(f"Peeked at '{peeked_item}' on top of the stack.")
        else:
            print("Stack is empty. Nothing to peek at.")
    elif choice == '4':
        if stack.is_empty():
            print("Stack is empty.")
```

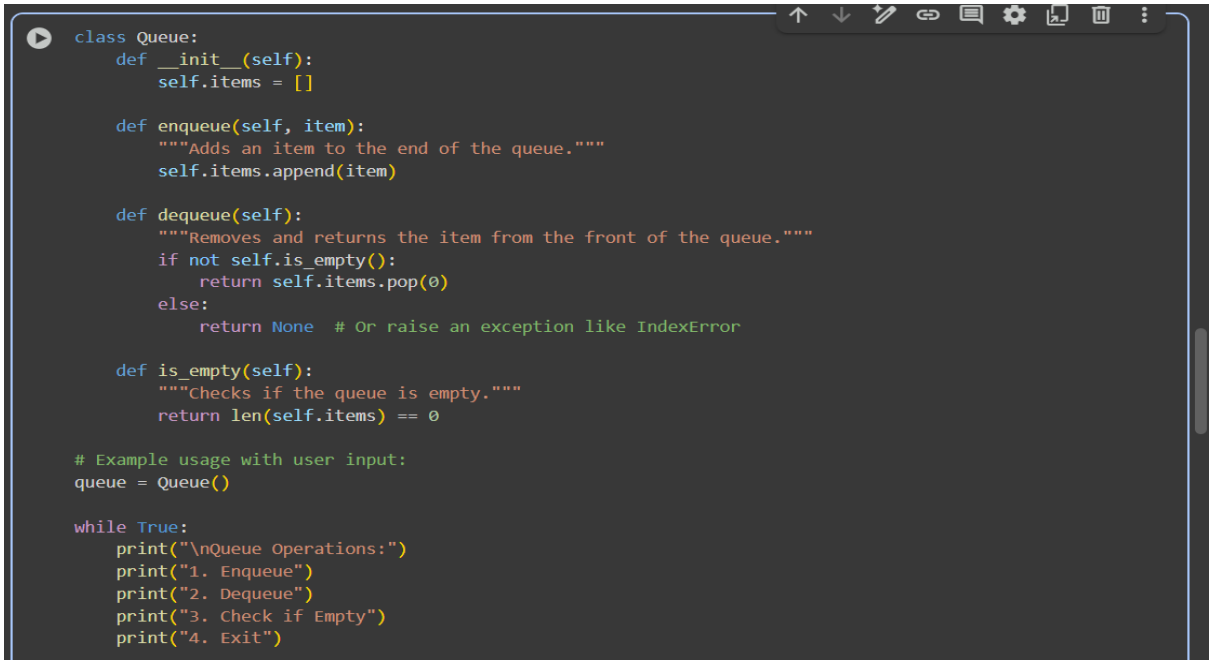
```
            print("Stack is empty.")
        else:
            print("Stack is not empty.")
    elif choice == '5':
        print("Exiting program.")
        break
    else:
        print("Invalid choice. Please try again.")
```

```
Stack Operations:
1. Push
2. Pop
3. Peek
4. Check if Empty
5. Exit
Enter your choice: 1
Enter item to push: hi
Pushed 'hi' onto the stack.

Stack Operations:
1. Push
2. Pop
3. Peek
4. Check if Empty
5. Exit
Enter your choice: 2
Popped 'hi' from the stack.
```

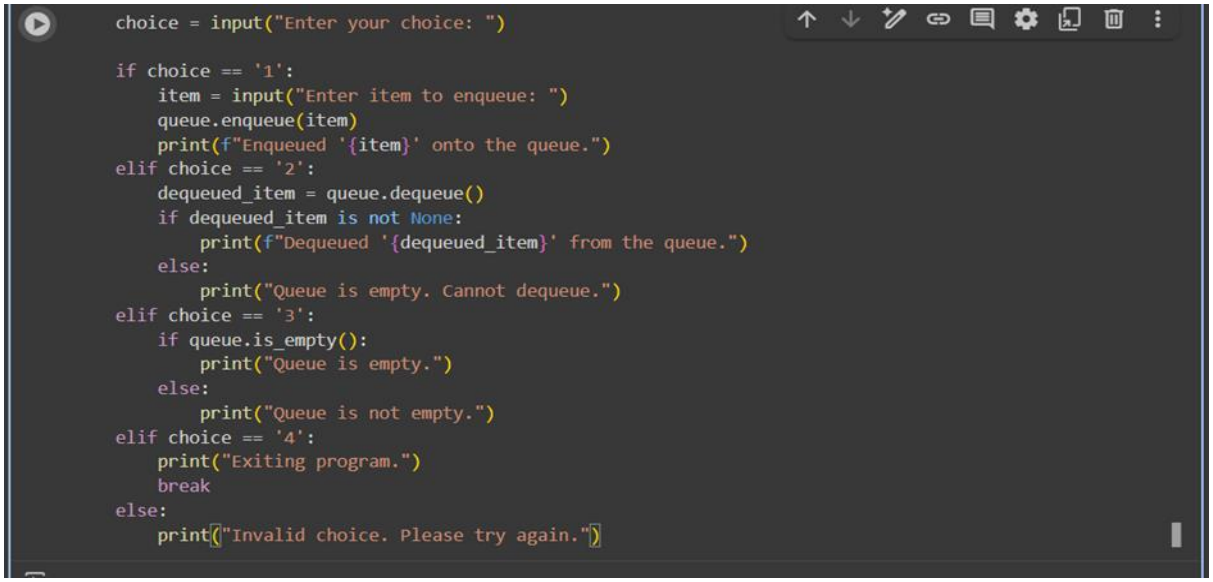
```
Stack Operations:
1. Push
2. Pop
3. Peek
4. Check if Empty
5. Exit
Enter your choice: 1
Enter item to push: hi
Pushed 'hi' onto the stack.

Stack Operations:
1. Push
2. Pop
3. Peek
4. Check if Empty
5. Exit
Enter your choice: 2
Popped 'hi' from the stack.

Stack Operations:
1. Push
2. Pop
3. Peek
4. Check if Empty
5. Exit
Enter your choice: 5
Exiting program.
```

Task Description 2 – Queue Implementation
Task: Use AI to generate a Queue class with enqueue(), dequeue(), and is_empty().

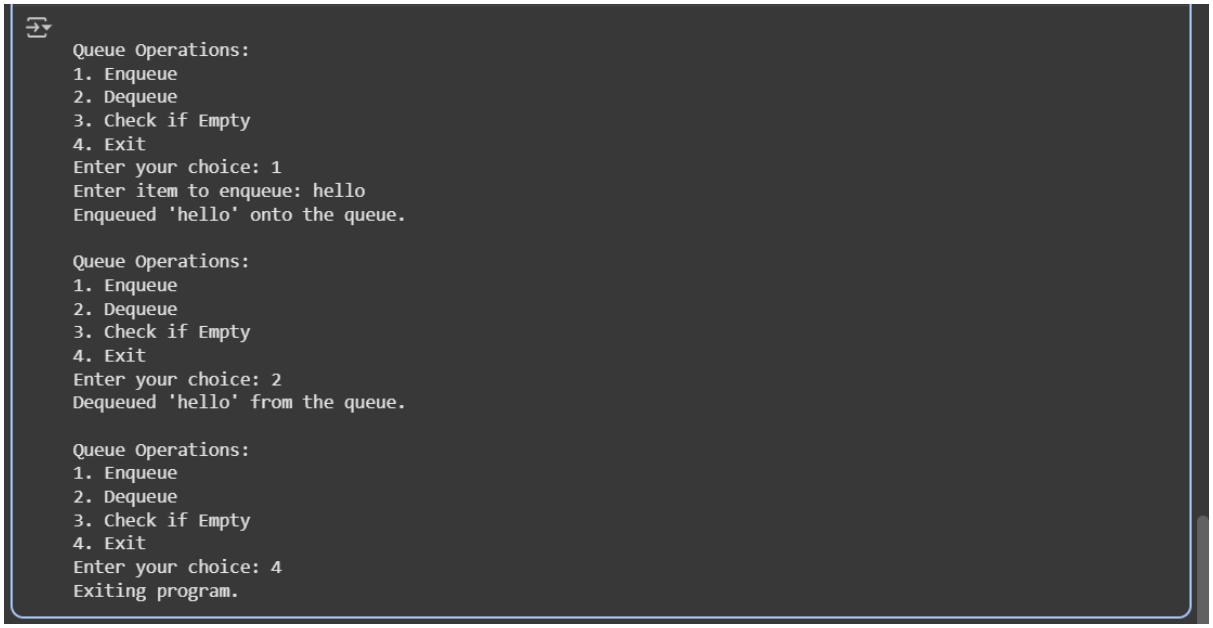
Used Prompt: Generate a queue class with enqueue(), dequeue(), is_empty() in python. take input from the user



```python
class Queue:
    def __init__(self):
        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."""
        if not self.is_empty():
            return self.items.pop(0)
        else:
            return None  # Or raise an exception like IndexError

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

# Example usage with user input:
queue = Queue()

while True:
    print("\nQueue Operations:")
    print("1. Enqueue")
    print("2. Dequeue")
    print("3. Check if Empty")
    print("4. Exit")
```



```python
    choice = input("Enter your choice: ")

    if choice == '1':
        item = input("Enter item to enqueue: ")
        queue.enqueue(item)
        print(f"Enqueued '{item}' onto the queue.")
    elif choice == '2':
        dequeued_item = queue.dequeue()
        if dequeued_item is not None:
            print(f"Dequeued '{dequeued_item}' from the queue.")
        else:
            print("Queue is empty. Cannot dequeue.")
    elif choice == '3':
        if queue.is_empty():
            print("Queue is empty.")
        else:
            print("Queue is not empty.")
    elif choice == '4':
        print("Exiting program.")
        break
    else:
        print("Invalid choice. Please try again.")
```



Task Description 3 – Linked List Implementation
Task: Ask AI to create a singly linked list with insert_at_end(), insert_at_beginning(), and display().

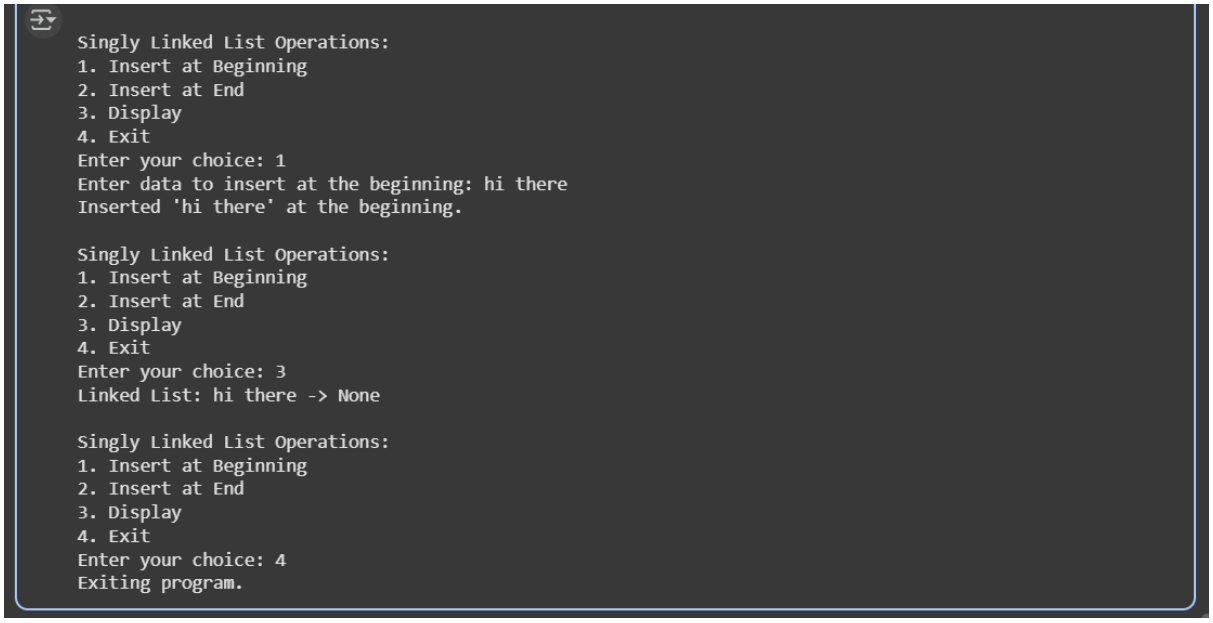
Used Prompt:

Create a singly linked list with insert_at_end(), insert_at_beginning(), and display().take input from the user

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

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

    def insert_at_beginning(self, data):
        """Inserts a new node with the given data 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):
        """Inserts a new node with the given data at the end of the list."""
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            print(f"Inserted '{data}' at the end.")
            return
        last_node = self.head
        while last_node.next:
            last_node = last_node.next
```

```python
        last_node.next = new_node
        print(f"Inserted '{data}' at the end.")

    def display(self):
        """Displays the elements of the linked list."""
        if self.head is None:
            print("Linked list is empty.")
            return
        current_node = self.head
        print("Linked List:", end=" ")
        while current_node:
            print(current_node.data, end=" -> ")
            current_node = current_node.next
        print("None")

# Example usage with user input:
linked_list = SinglyLinkedList()

while True:
    print("\nSingly Linked List Operations:")
    print("1. Insert at Beginning")
    print("2. Insert at End")
    print("3. Display")
    print("4. Exit")

    choice = input("Enter your choice: ")
```

```python
    if choice == '1':
        data = input("Enter data to insert at the beginning: ")
        linked_list.insert_at_beginning(data)
    elif choice == '2':
        data = input("Enter data to insert at the end: ")
        linked_list.insert_at_end(data)
    elif choice == '3':
        linked_list.display()
    elif choice == '4':
        print("Exiting program.")
        break
    else:
        print("Invalid choice. Please try again.")
```



Task Description 4 – Binary Search Tree (BST)
Task: Ask AI to generate a simple BST with insert() and inorder_traversal().

Used Prompt:

Generate a simple BST with insert() and inorder_traversal(). in python and take input from the user.

```python
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):
        """Inserts a new node with the given key into the BST."""
        self._insert_rec(self.root, key)
        print(f"Inserted {key} into the BST.")

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

```python
    def inorder_traversal(self):
        """Performs an in-order traversal of the BST and prints the elements."""
        print("In-order Traversal:", end=" ")
        self._inorder_traversal_rec(self.root)
        print()

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

# Example usage with user input:
bst = BST()

while True:
    print("\nBST Operations:")
    print("1. Insert")
    print("2. In-order Traversal")
    print("3. Exit")

    choice = input("Enter your choice: ")

    if choice == '1':
        try:
            key = int(input("Enter key to insert: "))
            bst.root = bst.insert(key) # Update the root after insertion
        except ValueError:
            print("Invalid input. Please enter an integer.")
```

```
            print("Invalid input. Please enter an integer.")
    elif choice == '2':
        bst.inorder_traversal()
    elif choice == '3':
        print("Exiting program.")
        break
    else:
        print("Invalid choice. Please try again.")
```

```
BST Operations:
1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 1
Enter key to insert: 124539
Inserted 124539 into the BST.

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

BST Operations:
1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 3
Exiting program.
```