LAB ASSIGNMENT: 11.3

NAME: M.TANVI

ROLL NO:2403A510A4

COURSE: AI ASSISTED CODING

BATCH: 01

QUESTIONS:

Task Description #1 - Stack class implementation

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

Task Description #2 - Queue Implementation

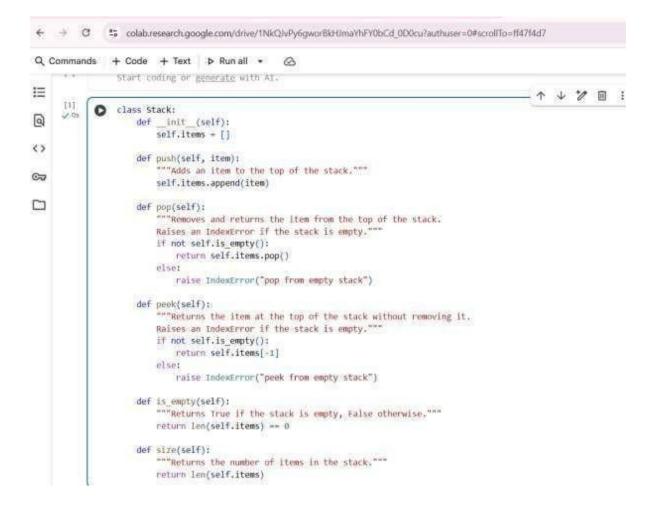
Task: Use AI to generate a Queue class with enqueue(), dequeue(), and is_empty().

Task Description #3 - Linked List Implementation

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

Task Description #4 - Binary Search Tree (BST)

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



TASK-2

```
mands + Code + Text ▶ Run all ▼
3]
        class Queue:
Os
              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.
                  Raises an IndexError if the queue is empty."""
                 if not self.is_empty():
                     return self.items.pop(0) # Removes from the beginning
                 else:
                     raise IndexError("dequeue from empty queue")
              def is empty(self):
                  """Returns True if the queue is empty, False otherwise."""
                  return len(self.items) == 0
              def size(self):
                  """Returns the number of items in the queue."""
                  return len(self.items)
```

Here's an example of how to use the Queue class:

TASK-3

```
class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None
class LinkedList:
    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 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 with the given data 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):
        """Prints the data of each node in the linked list."""
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")
```

TASK-4

```
Q Commands + Code + Text ▶ Run all -
                                                  0
                class BST:
Ξ
     ✓ 0s
                    def __init__(self):
                        self.root = None
9
                    def insert(self, key):
                        """Inserts a new node with the given key into the BST."""
3
                        if self.root is None:
                            self.root = TreeNode(key)
) T
                        else:
                            self. insert recursive(self.root, key)
                    def insert recursive(self, node, key):
                        if key < node.key:
                            if node.left is None:
                                node.left = TreeNode(key)
                            else:
                                self. insert recursive(node.left, key)
                        elif key > node.key:
                            if node.right is None:
                                node.right = TreeNode(key)
                            else:
                                self._insert_recursive(node.right, key)
                        # If key is equal, we don't insert duplicates in this simple BST
                    def inorder traversal(self):
                        """Performs an inorder traversal of the BST and prints the keys."""
                        self._inorder_traversal_recursive(self.root)
                        print() # Add a newline at the end for cleaner output
                    def _inorder_traversal_recursive(self, node):
                        if node:
                            self._inorder_traversal_recursive(node.left)
                            print(node.key, end=" ")
                            self._inorder_traversal_recursive(node.right)
```