Assignment:11.3

NAME:B.SRINITHI

HALL TICKET NO:2403A51413

COURSE:AI ASSISTED CODING

BRANCH:CSE-AIML

QUESTIONS:

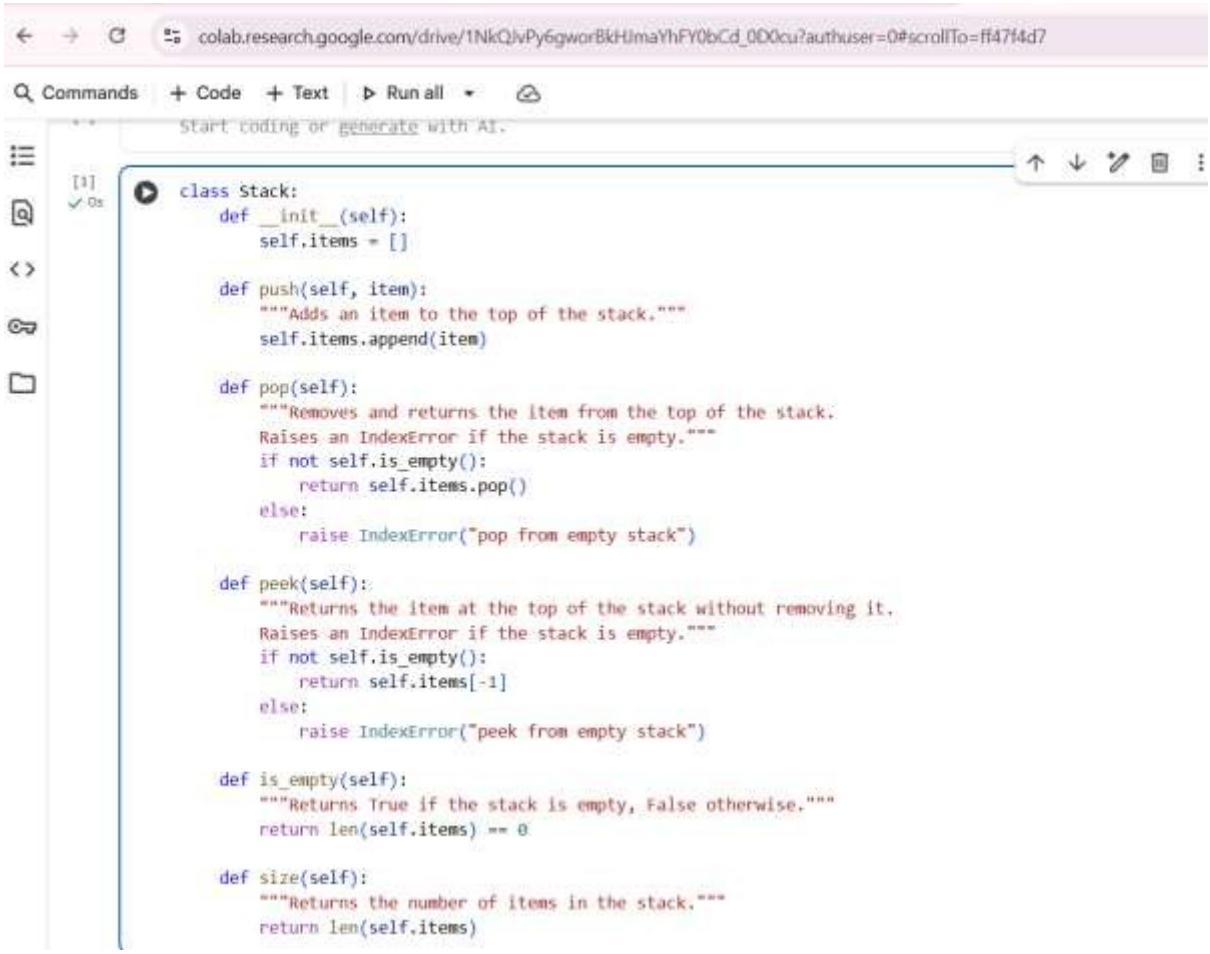
Task Description #1 – Stack class implementation
Task: Ask AI to implement a stack class with push(), pop(), peek() and is_empty() methods

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

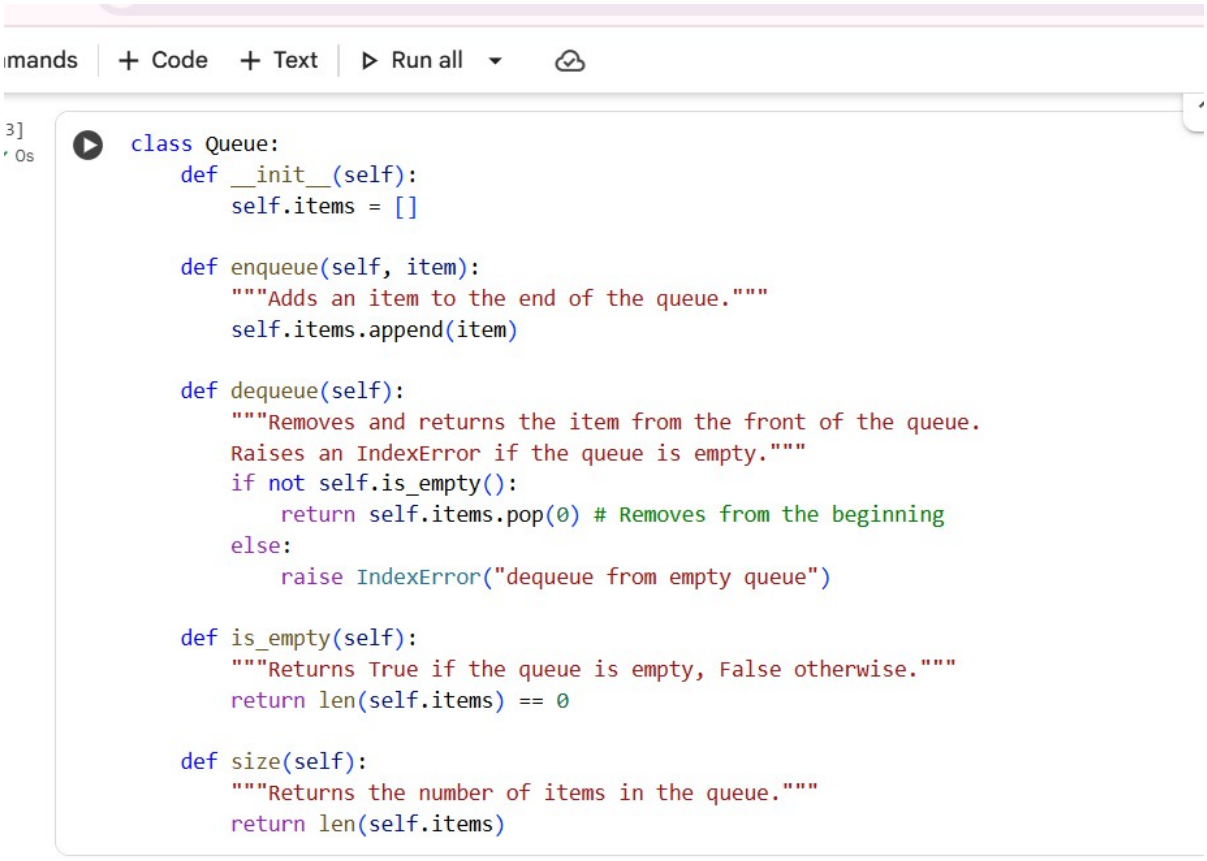
Task Description #3 – Linked List Implementation
Task: Ask AI to create a singly linked list with insert_at_end(), insert_at_beginning(), and display().

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

TASK-1


colab.research.google.com/drive/1NkQJvPy6gwor8kHJmaYhFY0bCd_0D0cu?authuser=0#scrollTo=ff47f4d7
Commands
+ Code
+ Text
Run all
Start coding or generate with AI.
[1]
0s
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.
Raises an IndexError if the stack is empty."""
if not self.is_empty():
return self.items.pop()
else:
raise IndexError("pop from empty stack")
def peek(self):
"""Returns the item at the top of the stack without removing it.
Raises an IndexError if the stack is empty."""
if not self.is_empty():
return self.items[-1]
else:
raise IndexError("peek from empty stack")
def is_empty(self):
"""Returns True if the stack is empty, False otherwise."""
return len(self.items) == 0
def size(self):
"""Returns the number of items in the stack."""
return len(self.items)

TASK-2


mands
+ Code
+ Text
Run all
3]
Os
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.
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

```python
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

Commands | + Code | + Text | Run all

[7] ✓ 0s

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

    def insert(self, key):
        """Inserts a new node with the given key into the BST."""
        if self.root is None:
            self.root = TreeNode(key)
        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)
```