AI ASSISTED CODING

Program :B.tech(CSE)

Name :Peravena Navadeep

En No. :2403A52019

Batch No. :02

LAB ASSIGNMENT-11.3

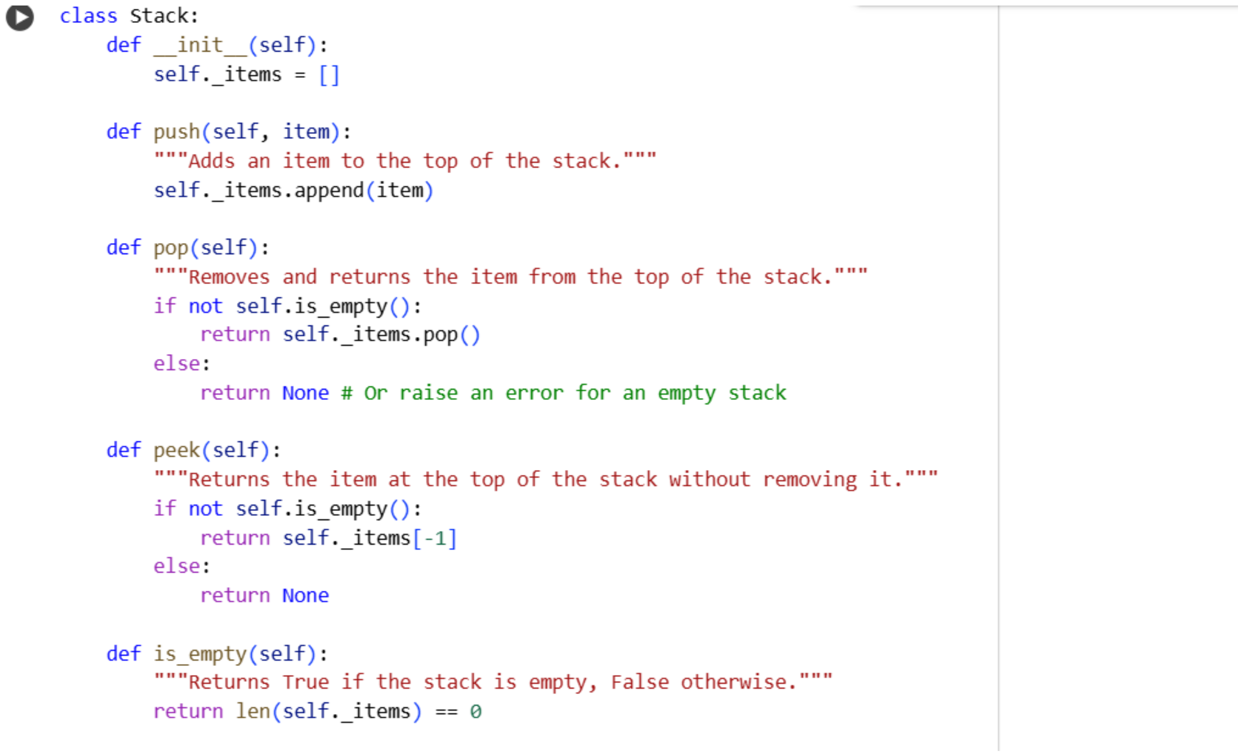
TASK DESCRIPTION-1:

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

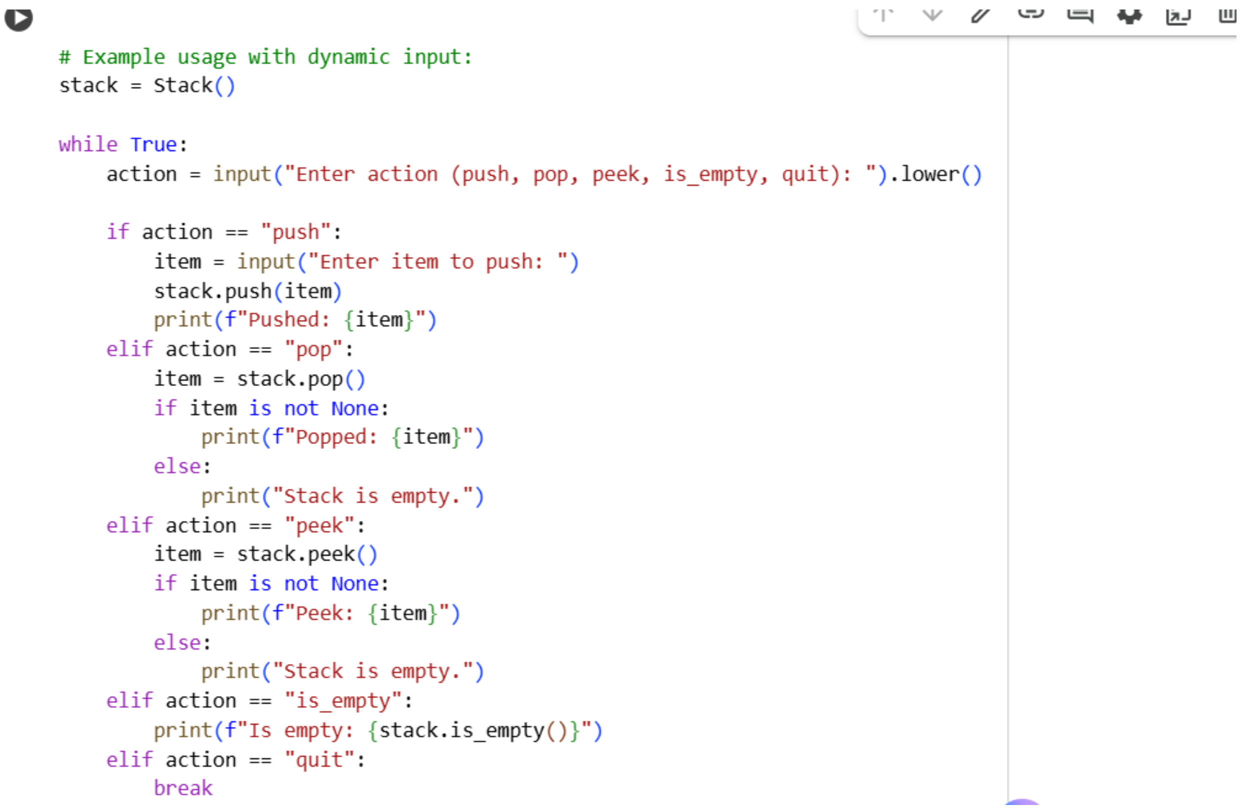
PROMPT:

Create a Python class Stack that implements a stack with methods push(item), pop(), peek(), and is_empty() with dynamic input.

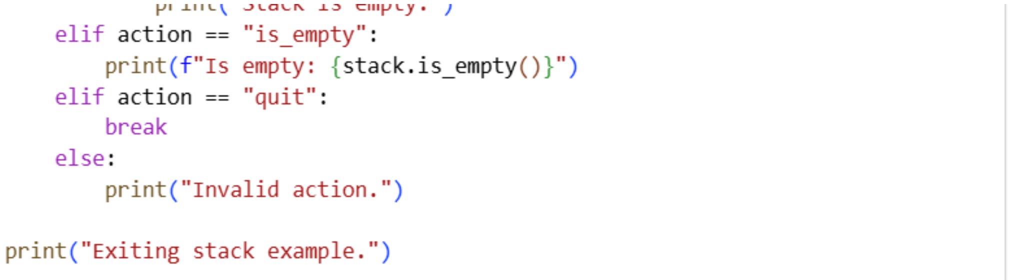
CODE:



```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 error for an empty stack

    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):
        """Returns True if the stack is empty, False otherwise."""
        return len(self._items) == 0
```

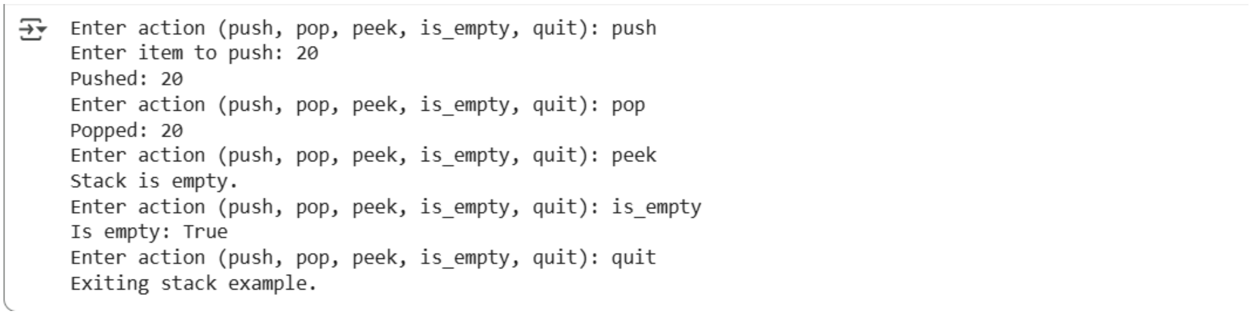


```python
# Example usage with dynamic input:
stack = Stack()

while True:
    action = input("Enter action (push, pop, peek, is_empty, quit): ").lower()

    if action == "push":
        item = input("Enter item to push: ")
        stack.push(item)
        print(f"Pushed: {item}")
    elif action == "pop":
        item = stack.pop()
        if item is not None:
            print(f"Popped: {item}")
        else:
            print("Stack is empty.")
    elif action == "peek":
        item = stack.peek()
        if item is not None:
            print(f"Peek: {item}")
        else:
            print("Stack is empty.")
    elif action == "is_empty":
        print(f"Is empty: {stack.is_empty()}")
    elif action == "quit":
        break
```



```python
            print("Stack is empty.")
    elif action == "is_empty":
        print(f"Is empty: {stack.is_empty()}")
    elif action == "quit":
        break
    else:
        print("Invalid action.")

print("Exiting stack example.")
```

OUTPUT:



```
Enter action (push, pop, peek, is_empty, quit): push
Enter item to push: 20
Pushed: 20
Enter action (push, pop, peek, is_empty, quit): pop
Popped: 20
Enter action (push, pop, peek, is_empty, quit): peek
Stack is empty.
Enter action (push, pop, peek, is_empty, quit): is_empty
Is empty: True
Enter action (push, pop, peek, is_empty, quit): quit
Exiting stack example.
```

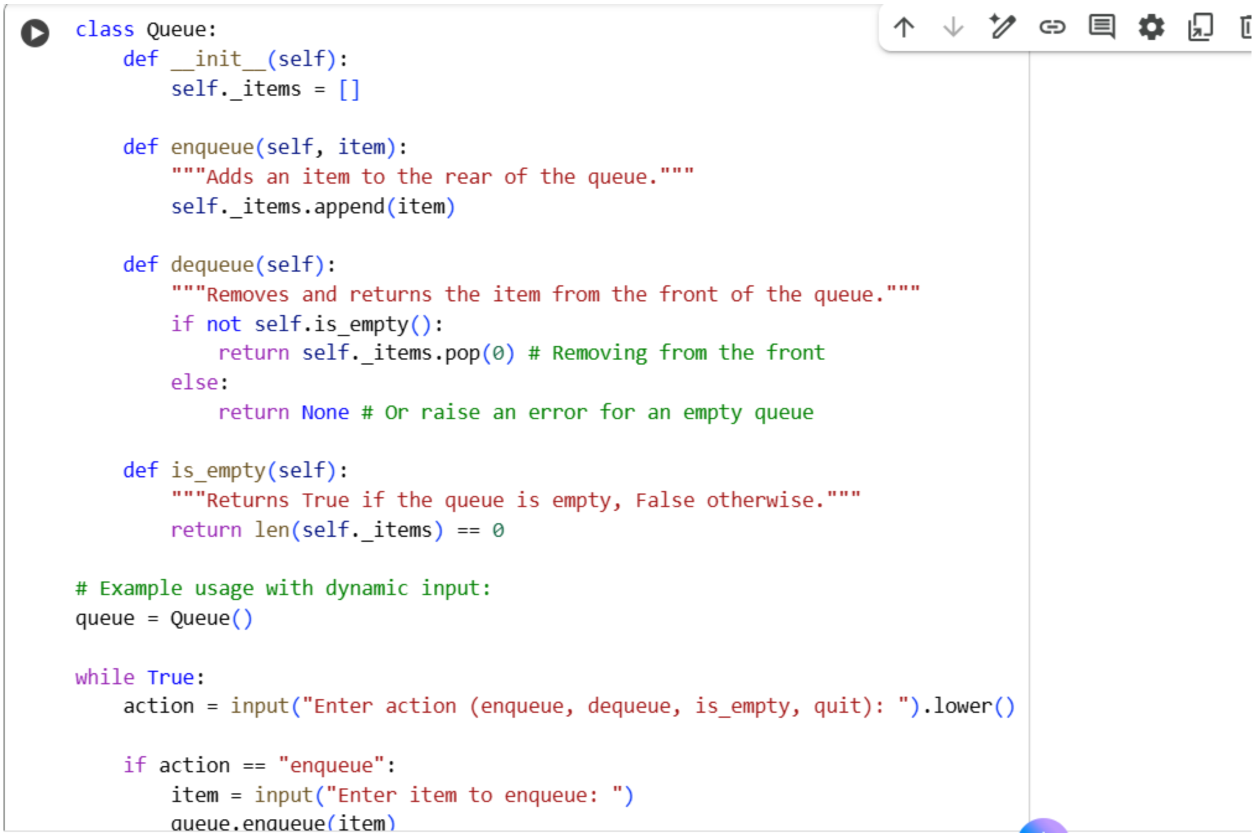
TASK DESCRIPTION-2:

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

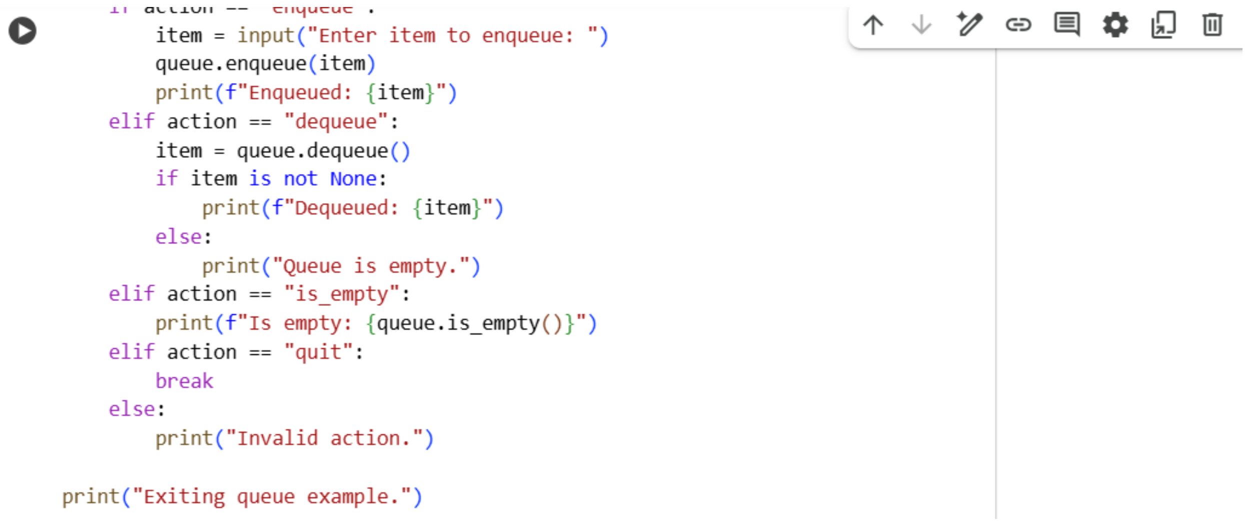
PROMPT:

Create a Python class Queue that implements a queue with methods enqueue(item), dequeue(), and is_empty() with user input.

CODE:

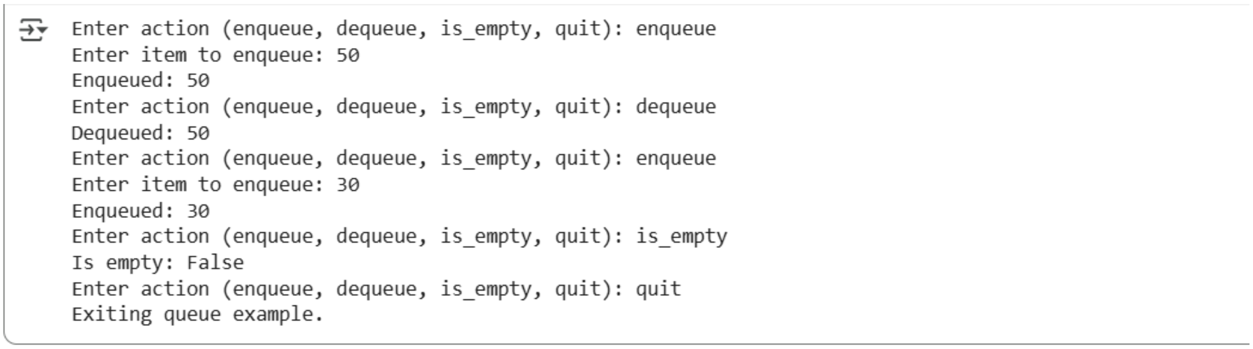


```
class Queue:
    def __init__(self):
        self._items = []

    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) # Removing from the front
        else:
            return None # Or raise an error for an empty queue

    def is_empty(self):
        """Returns True if the queue is empty, False otherwise."""
        return len(self._items) == 0

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

while True:
    action = input("Enter action (enqueue, dequeue, is_empty, quit): ").lower()

    if action == "enqueue":
        item = input("Enter item to enqueue: ")
        queue.enqueue(item)
```



```
        item = input("Enter item to enqueue: ")
        queue.enqueue(item)
        print(f"Enqueued: {item}")
    elif action == "dequeue":
        item = queue.dequeue()
        if item is not None:
            print(f"Dequeued: {item}")
        else:
            print("Queue is empty.")
    elif action == "is_empty":
        print(f"Is empty: {queue.is_empty()}")
    elif action == "quit":
        break
    else:
        print("Invalid action.")

print("Exiting queue example.")
```

OUTPUT:



```
Enter action (enqueue, dequeue, is_empty, quit): enqueue
Enter item to enqueue: 50
Enqueued: 50
Enter action (enqueue, dequeue, is_empty, quit): dequeue
Dequeued: 50
Enter action (enqueue, dequeue, is_empty, quit): enqueue
Enter item to enqueue: 30
Enqueued: 30
Enter action (enqueue, dequeue, is_empty, quit): is_empty
Is empty: False
Enter action (enqueue, dequeue, is_empty, quit): quit
Exiting queue example.
```

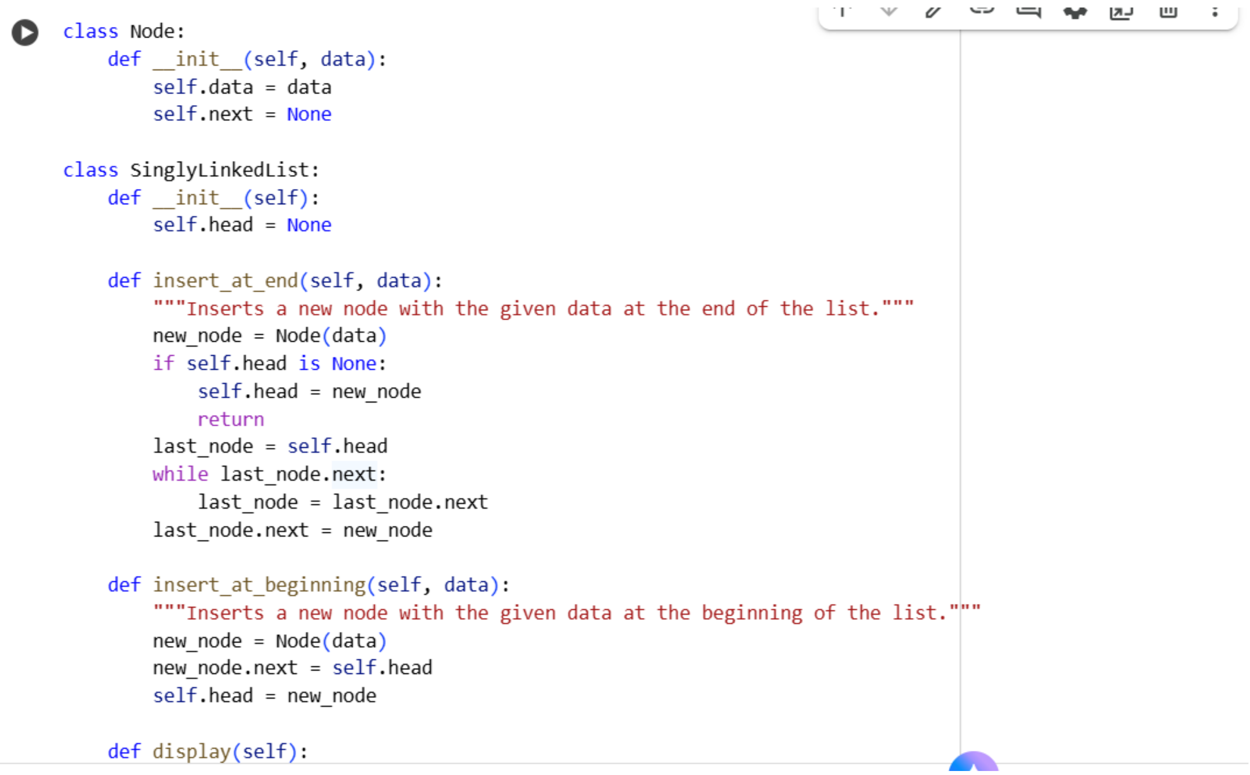
TASK DESCRIPTION-3:

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

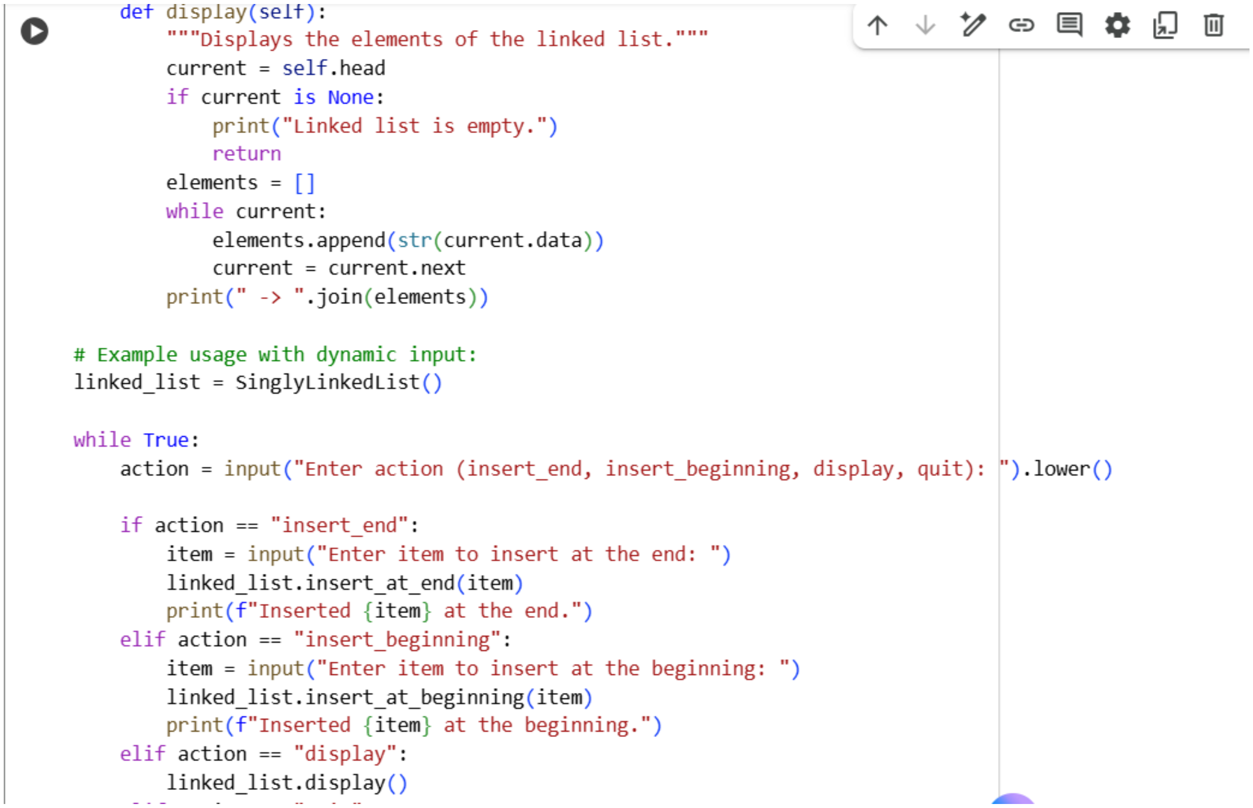
PROMPT:

Create a Python class for a singly linked list with methods insert_at_end(), insert_at_beginning(), and display() with dynamic input.

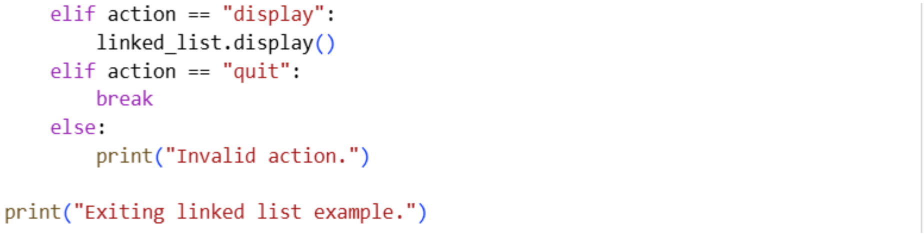
CODE:



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

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

    def insert_at_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
            return
        last_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node

    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

    def display(self):
```

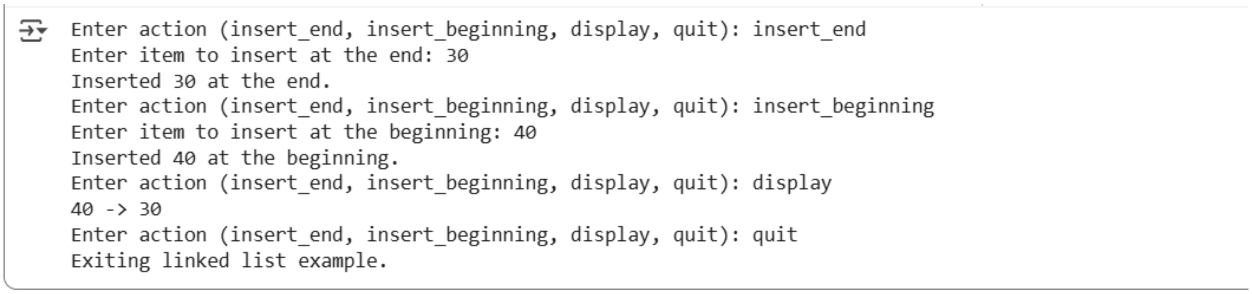


```
    def display(self):
        """Displays the elements of the linked list."""
        current = self.head
        if current is None:
            print("Linked list is empty.")
            return
        elements = []
        while current:
            elements.append(str(current.data))
            current = current.next
        print(" -> ".join(elements))

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

while True:
    action = input("Enter action (insert_end, insert_beginning, display, quit): ").lower()

    if action == "insert_end":
        item = input("Enter item to insert at the end: ")
        linked_list.insert_at_end(item)
        print(f"Inserted {item} at the end.")
    elif action == "insert_beginning":
        item = input("Enter item to insert at the beginning: ")
        linked_list.insert_at_beginning(item)
        print(f"Inserted {item} at the beginning.")
    elif action == "display":
        linked_list.display()
```



```
    elif action == "display":
        linked_list.display()
    elif action == "quit":
        break
    else:
        print("Invalid action.")

print("Exiting linked list example.")
```

OUTPUT:



```
Enter action (insert_end, insert_beginning, display, quit): insert_end
Enter item to insert at the end: 30
Inserted 30 at the end.
Enter action (insert_end, insert_beginning, display, quit): insert_beginning
Enter item to insert at the beginning: 40
Inserted 40 at the beginning.
Enter action (insert_end, insert_beginning, display, quit): display
40 -> 30
Enter action (insert_end, insert_beginning, display, quit): quit
Exiting linked list example.
```

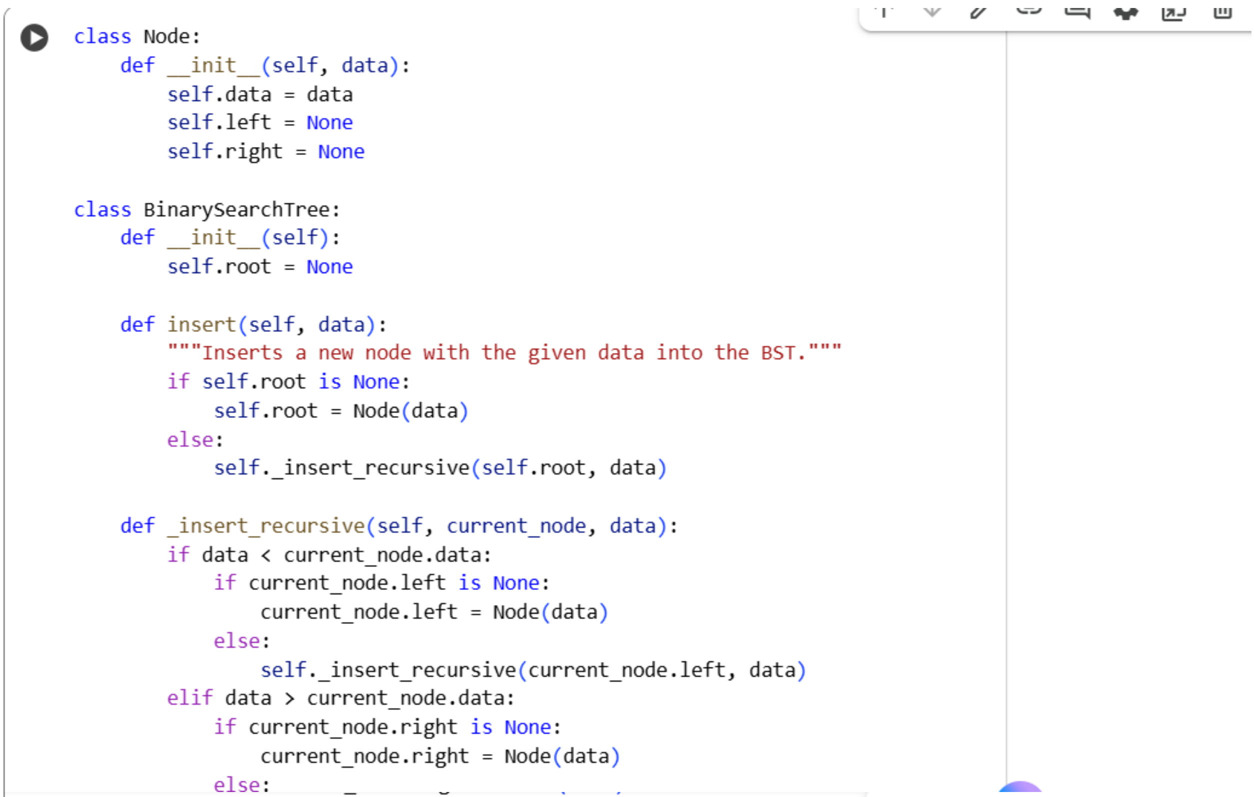
TASK DISCRIPTION-4:

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

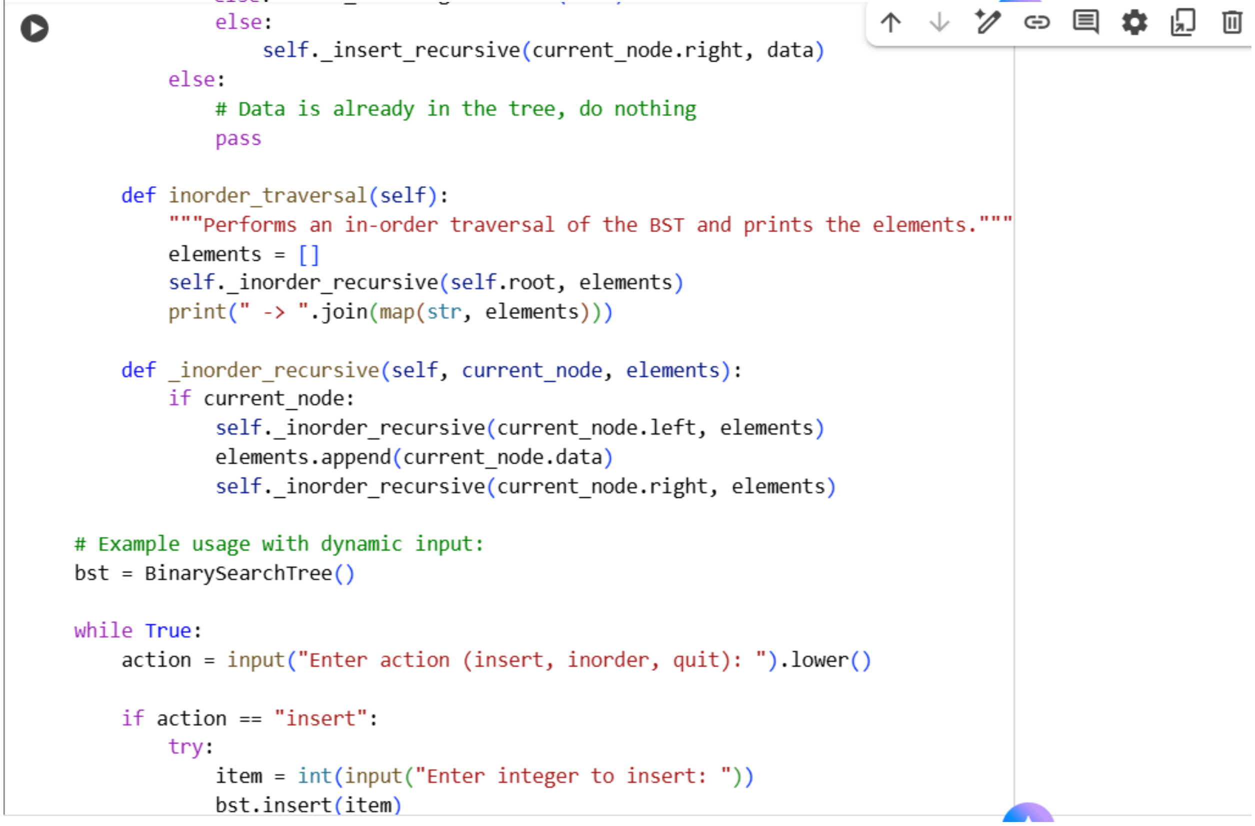
PROMPT:

Create a Python class for a Binary Search Tree (BST) with methods insert(data) and inorder_traversal() with dynamic input.

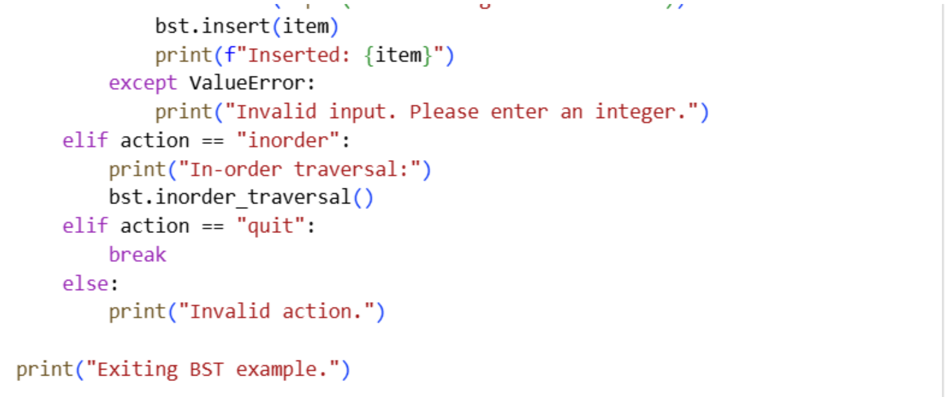
CODE:



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

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

    def insert(self, data):
        """Inserts a new node with the given data into the BST."""
        if self.root is None:
            self.root = Node(data)
        else:
            self._insert_recursive(self.root, data)

    def _insert_recursive(self, current_node, data):
        if data < current_node.data:
            if current_node.left is None:
                current_node.left = Node(data)
            else:
                self._insert_recursive(current_node.left, data)
        elif data > current_node.data:
            if current_node.right is None:
                current_node.right = Node(data)
            else:
```

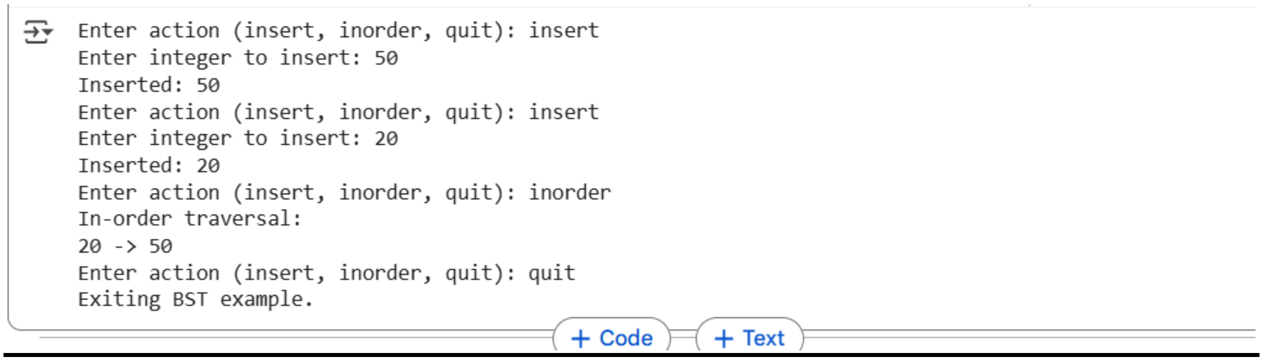


```
            else:
                self._insert_recursive(current_node.right, data)
        else:
            # Data is already in the tree, do nothing
            pass

    def inorder_traversal(self):
        """Performs an in-order traversal of the BST and prints the elements."""
        elements = []
        self._inorder_recursive(self.root, elements)
        print(" -> ".join(map(str, elements)))

    def _inorder_recursive(self, current_node, elements):
        if current_node:
            self._inorder_recursive(current_node.left, elements)
            elements.append(current_node.data)
            self._inorder_recursive(current_node.right, elements)

# Example usage with dynamic input:
bst = BinarySearchTree()

while True:
    action = input("Enter action (insert, inorder, quit): ").lower()

    if action == "insert":
        try:
            item = int(input("Enter integer to insert: "))
            bst.insert(item)
```



```
            bst.insert(item)
            print(f"Inserted: {item}")
        except ValueError:
            print("Invalid input. Please enter an integer.")
    elif action == "inorder":
        print("In-order traversal:")
        bst.inorder_traversal()
    elif action == "quit":
        break
    else:
        print("Invalid action.")

print("Exiting BST example.")
```

OUTPUT:



```
Enter action (insert, inorder, quit): insert
Enter integer to insert: 50
Inserted: 50
Enter action (insert, inorder, quit): insert
Enter integer to insert: 20
Inserted: 20
Enter action (insert, inorder, quit): inorder
In-order traversal:
20 -> 50
Enter action (insert, inorder, quit): quit
Exiting BST example.
```

---END---