PROGRAM : B.TECH/CSE

SPECIALIZATION : AIML

COURSE TITLE : AI ASSISTED CODING

COURSE CODE : 24CS101PC214

SEMESTER : 3RD

NAME OF THE STUDENT : KARNAKANTI GODADEVI

ENROLLMENT NO : 2403A52003

BATCH NO : 01

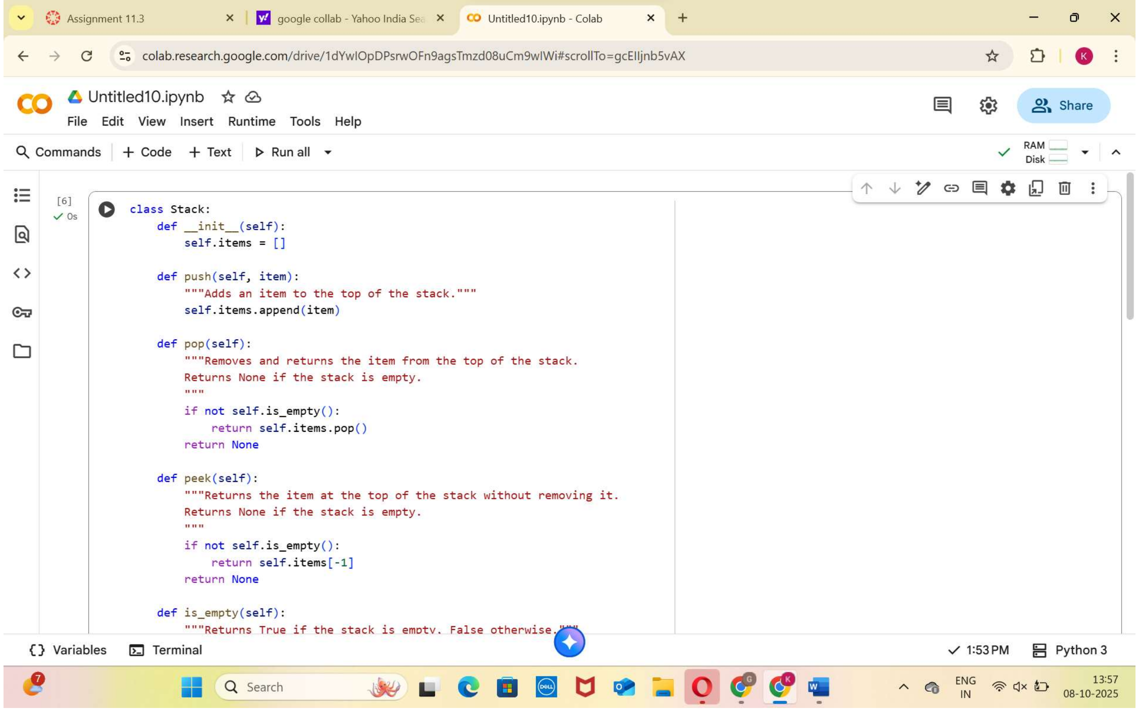
Task Description #1 – Stack class implementation

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

Prompt:

generate a python code to implement a stack class with push(), pop(), peek() and is_empty() methods.

Code &Output:



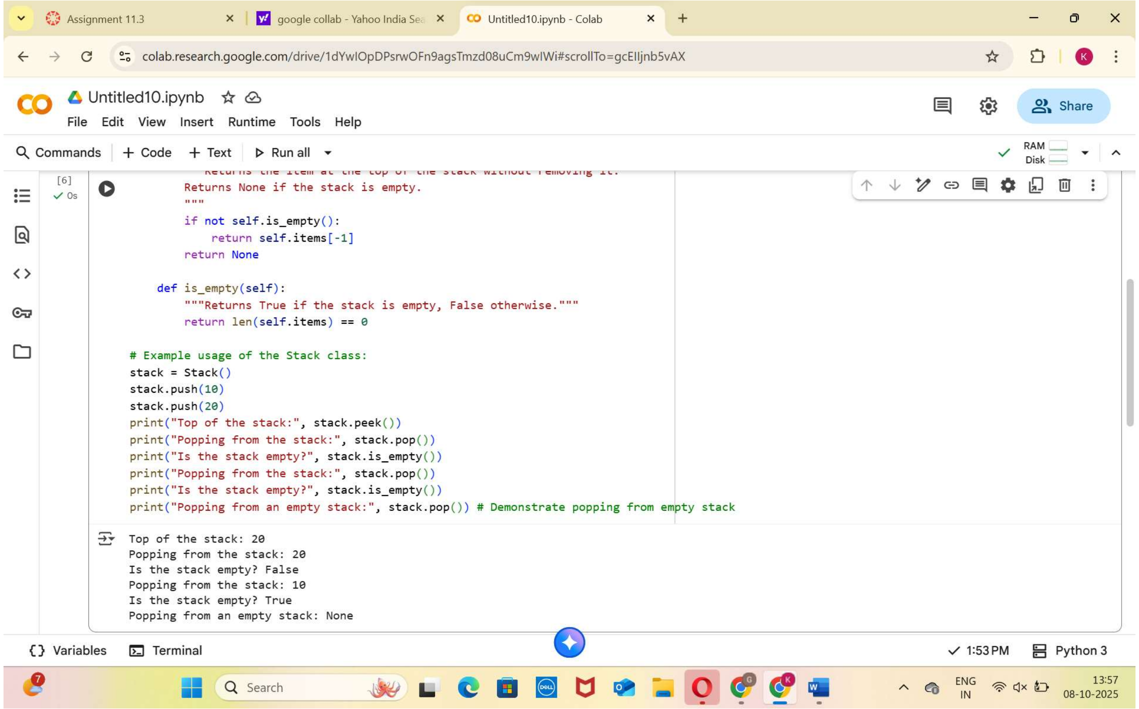
```
class Stack:
    def __init__(self):
        self.items = []

    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.
        Returns None if the stack is empty.
        """
        if not self.is_empty():
            return self.items.pop()
        return None

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

    def is_empty(self):
        """Returns True if the stack is empty, False otherwise."""
```


Assignment 11.3
google collab - Yahoo India Se
Untitled10.ipynb - Colab
colab.research.google.com/drive/1dYwlOpDPsrwOFn9agsTmzd08uCm9wIWi#scrollTo=gcElljnb5vAX
Untitled10.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM Disk
[6]
0s
Variables
Terminal
1:53 PM
Python 3
Search
ENG IN
13:57
08-10-2025

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

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

# Example usage of the Stack class:
stack = Stack()
stack.push(10)
stack.push(20)
print("Top of the stack:", stack.peek())
print("Popping from the stack:", stack.pop())
print("Is the stack empty?", stack.is_empty())
print("Popping from the stack:", stack.pop())
print("Is the stack empty?", stack.is_empty())
print("Popping from an empty stack:", stack.pop()) # Demonstrate popping from empty stack
```

```
Top of the stack: 20
Popping from the stack: 20
Is the stack empty? False
Popping from the stack: 10
Is the stack empty? True
Popping from an empty stack: None
```

Observation:

The code defines a Stack class with standard stack operations: push, pop, peek, and is_empty.

The example usage demonstrates these operations:

- Items 10 and 20 are pushed onto the stack.
- peek() shows 20 is at the top.
- pop() removes and returns 20, then 10.
- is_empty() correctly reports the stack's state after each pop.
- Attempting to pop() from an empty stack returns None.

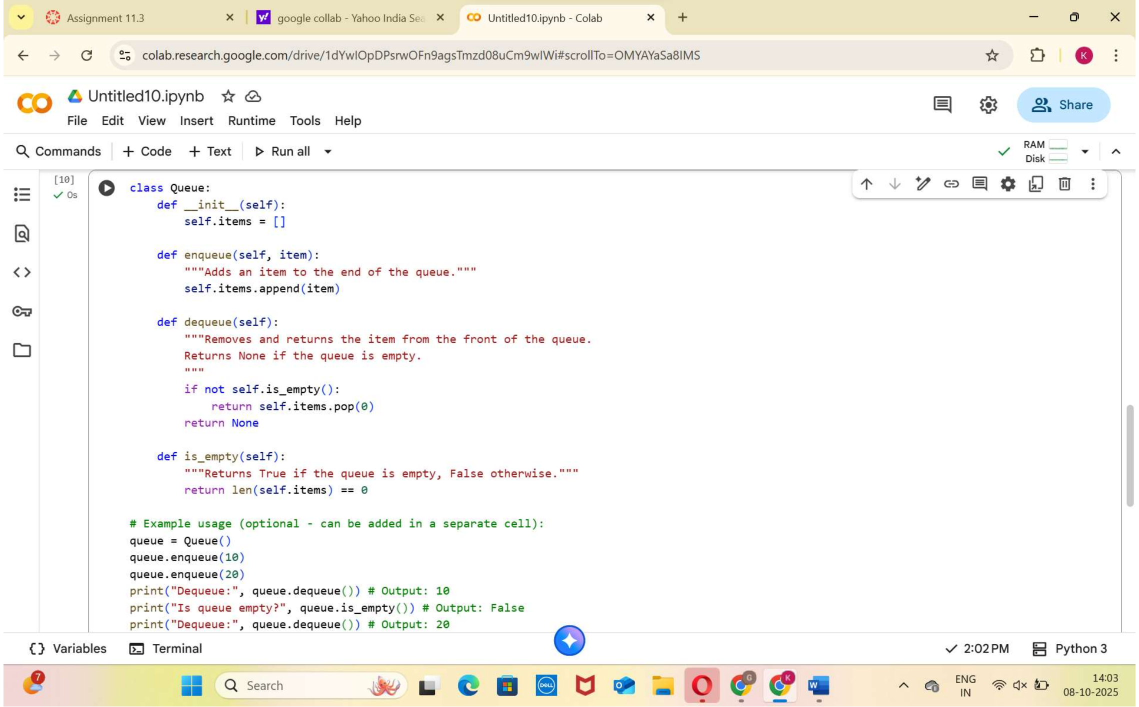
Task Description #2 – Queue Implementation

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

Prompt:

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

Code&Output:



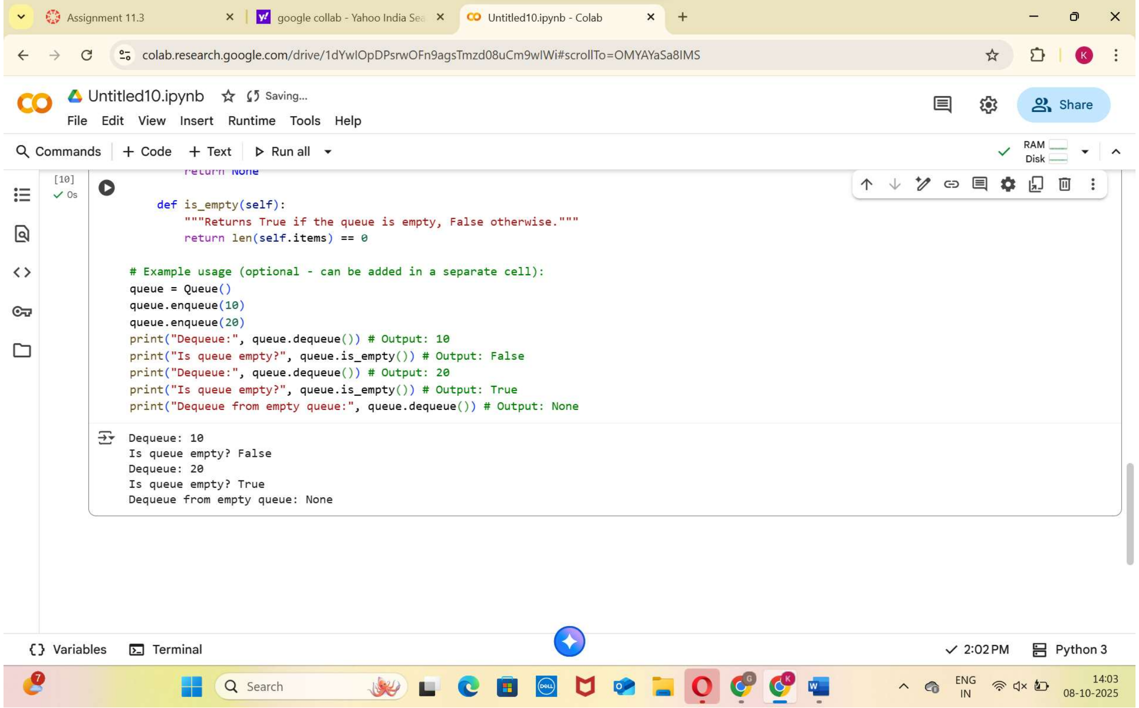
```
class Queue:
    def __init__(self):
        self.items = []

    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.
        Returns None if the queue is empty.
        """
        if not self.is_empty():
            return self.items.pop(0)
        return None

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

# Example usage (optional - can be added in a separate cell):
queue = Queue()
queue.enqueue(10)
queue.enqueue(20)
print("Dequeue:", queue.dequeue()) # Output: 10
print("Is queue empty?", queue.is_empty()) # Output: False
print("Dequeue:", queue.dequeue()) # Output: 20
```



```
        return None

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

# Example usage (optional - can be added in a separate cell):
queue = Queue()
queue.enqueue(10)
queue.enqueue(20)
print("Dequeue:", queue.dequeue()) # Output: 10
print("Is queue empty?", queue.is_empty()) # Output: False
print("Dequeue:", queue.dequeue()) # Output: 20
print("Is queue empty?", queue.is_empty()) # Output: True
print("Dequeue from empty queue:", queue.dequeue()) # Output: None
```

```
Dequeue: 10
Is queue empty? False
Dequeue: 20
Is queue empty? True
Dequeue from empty queue: None
```

Observation:

The code defines a Queue class with standard queue operations: enqueue, dequeue, and is_empty.

The example usage demonstrates these operations:

- Items 10 and 20 are enqueued into the queue.
- dequeue() removes and returns 10, then 20 from the front.
- is_empty() correctly reports the queue's state after each dequeue.
- Attempting to dequeue() from an empty queue returns None.

Task Description #3 – Linked List Implementation

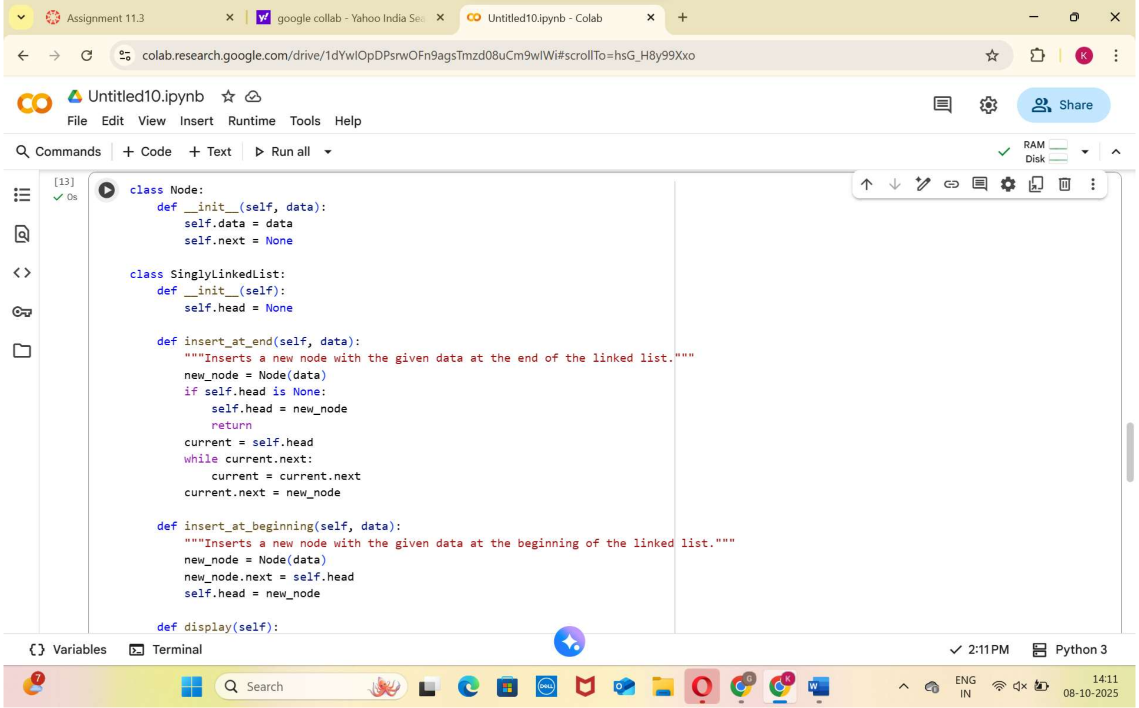
Task: Ask AI to create a singly linked list with insert_at_end(), insert_at_beginning(), and display().

Prompt:

generate a python code to create a singly linked list with insert_at_end(),

insert_at_beginning(), and display().

Code&Output:



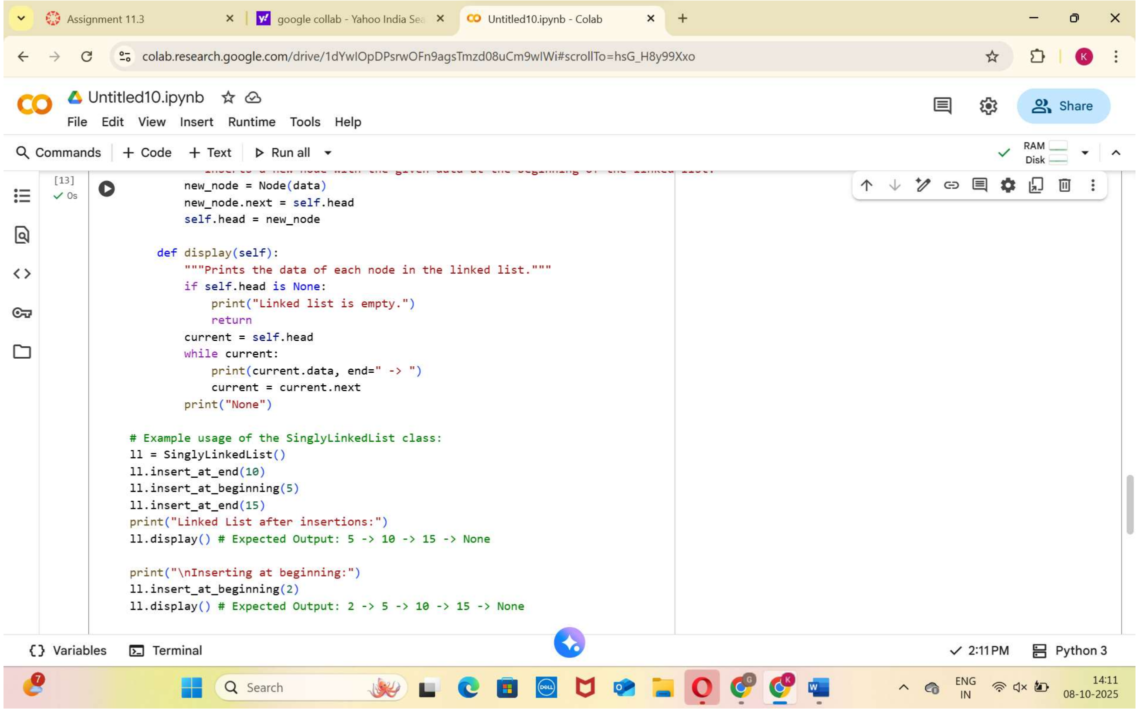
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

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 linked list."""
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node

    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 display(self):
```

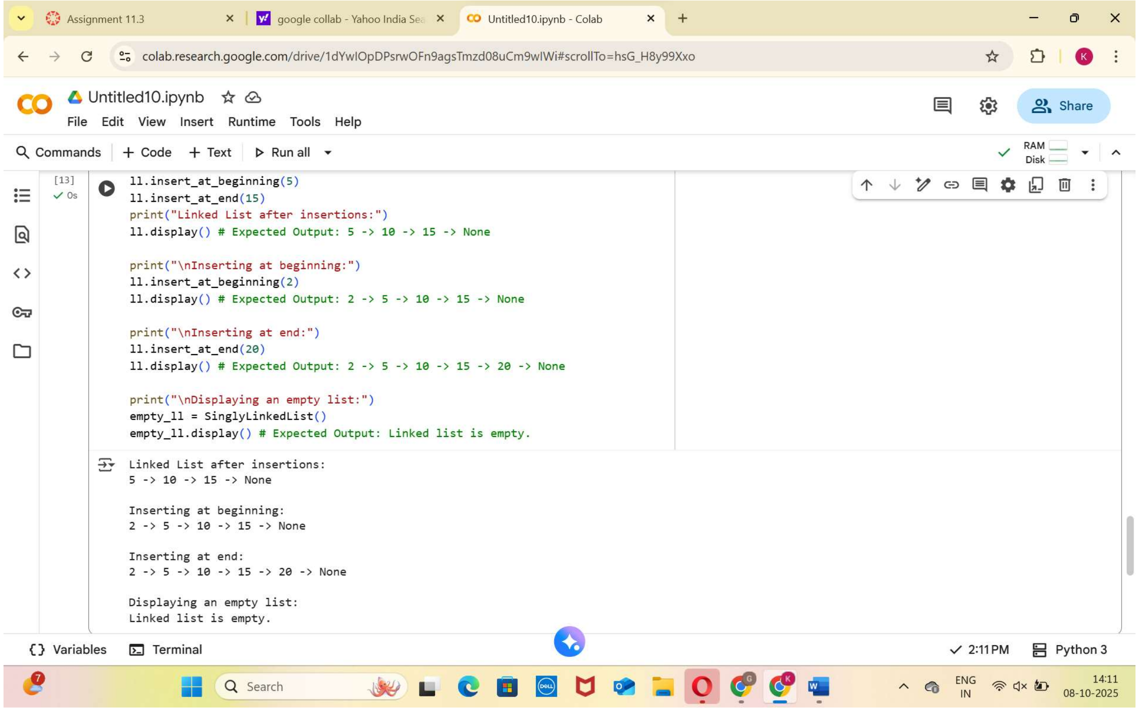

Assignment 11.3
google collab - Yahoo India Se
Untitled10.ipynb - Colab
colab.research.google.com/drive/1dYwlOpDPsrwOFn9agsTmzd08uCm9wlWi#scrollTo=hsG_H8y99Xxo
Untitled10.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM
Disk
[13]
0s
Variables
Terminal
2:11 PM
Python 3
Search
ENG
IN
14:11
08-10-2025

```
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node

    def display(self):
        """Prints the data of each node in the linked list."""
        if self.head is None:
            print("Linked list is empty.")
            return
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")

# Example usage of the SinglyLinkedList class:
ll = SinglyLinkedList()
ll.insert_at_end(10)
ll.insert_at_beginning(5)
ll.insert_at_end(15)
print("Linked List after insertions:")
ll.display() # Expected Output: 5 -> 10 -> 15 -> None

print("\nInserting at beginning:")
ll.insert_at_beginning(2)
ll.display() # Expected Output: 2 -> 5 -> 10 -> 15 -> None
```


Assignment 11.3
google collab - Yahoo India Se
Untitled10.ipynb - Colab
colab.research.google.com/drive/1dYwlOpDPsrwOFn9agsTmzd08uCm9wlWi#scrollTo=hsG_H8y99Xxo
Untitled10.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM
Disk
[13]
0s
Variables
Terminal
2:11 PM
Python 3
Search
ENG
IN
14:11
08-10-2025

```
ll.insert_at_beginning(5)
ll.insert_at_end(15)
print("Linked List after insertions:")
ll.display() # Expected Output: 5 -> 10 -> 15 -> None

print("\nInserting at beginning:")
ll.insert_at_beginning(2)
ll.display() # Expected Output: 2 -> 5 -> 10 -> 15 -> None

print("\nInserting at end:")
ll.insert_at_end(20)
ll.display() # Expected Output: 2 -> 5 -> 10 -> 15 -> 20 -> None

print("\nDisplaying an empty list:")
empty_ll = SinglyLinkedList()
empty_ll.display() # Expected Output: Linked list is empty.
```

```
Linked List after insertions:
5 -> 10 -> 15 -> None

Inserting at beginning:
2 -> 5 -> 10 -> 15 -> None

Inserting at end:
2 -> 5 -> 10 -> 15 -> 20 -> None

Displaying an empty list:
Linked list is empty.
```

Observation:

The code defines a SinglyLinkedList class with methods to insert nodes at the beginning (insert_at_beginning), insert at the end (insert_at_end), and display the list (display).

The example usage demonstrates these operations:

- Nodes with data 10, 5, and 15 are inserted, resulting in the list 5 -> 10 -> 15.

- Inserting 2 at the beginning changes the list to 2 -> 5 -> 10 -> 15.
- Inserting 20 at the end changes the list to 2 -> 5 -> 10 -> 15 -> 20.
- Displaying an empty list correctly outputs "Linked list is empty."

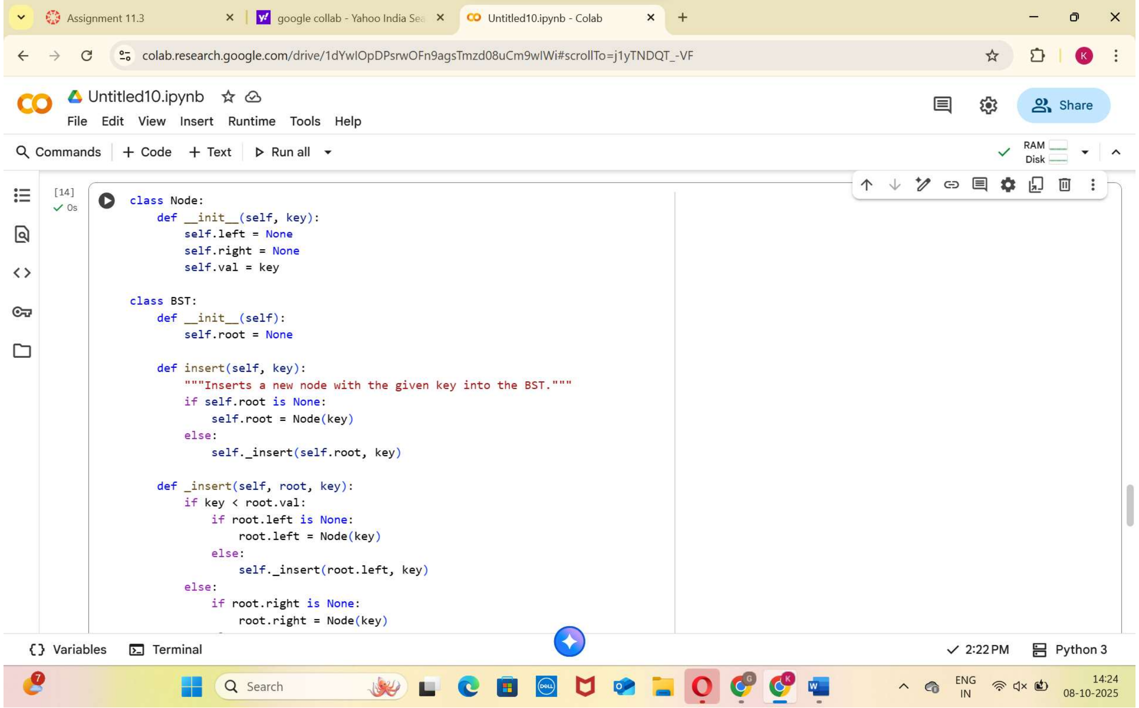
Task Description #4 – Binary Search Tree (BST)

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

Prompt:

generate a python code of simple BST with insert() and inorder_traversal().

Code&Output:

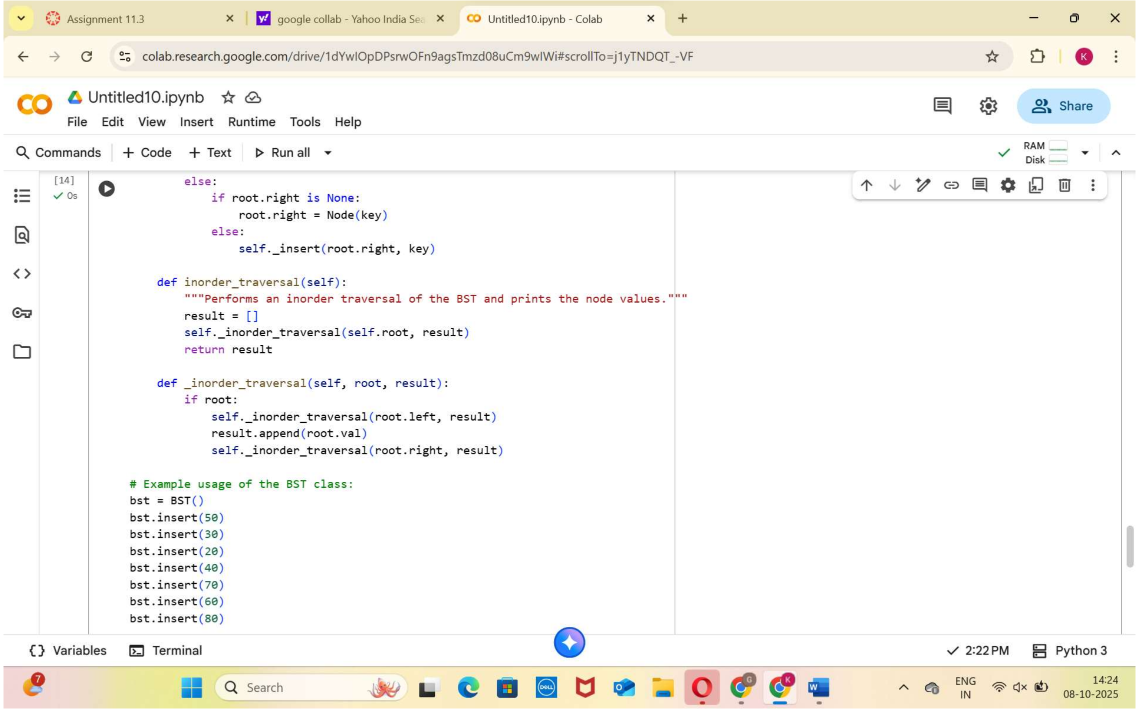


```
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key

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 = Node(key)
        else:
            self._insert(self.root, key)

    def _insert(self, root, key):
        if key < root.val:
            if root.left is None:
                root.left = Node(key)
            else:
                self._insert(root.left, key)
        else:
            if root.right is None:
                root.right = Node(key)
```

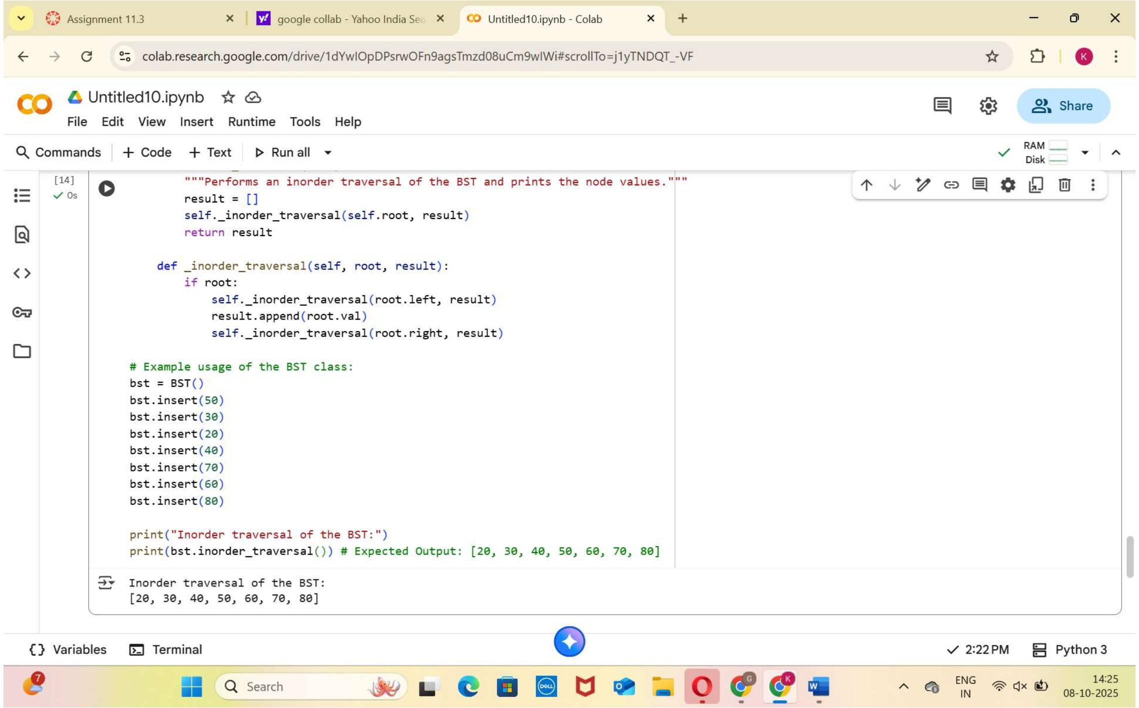

Assignment 11.3
google collab - Yahoo India Se
Untitled10.ipynb - Colab
colab.research.google.com/drive/1dYwIOpDPsrwOFn9agsTmzd08uCm9wIWi#scrollTo=j1yTNDQT_-VF
Untitled10.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM
Disk
[14]
0s
Variables
Terminal
2:22 PM
Python 3
Search
ENG
IN
14:24
08-10-2025

```
        else:
            if root.right is None:
                root.right = Node(key)
            else:
                self._insert(root.right, key)

    def inorder_traversal(self):
        """Performs an inorder traversal of the BST and prints the node values."""
        result = []
        self._inorder_traversal(self.root, result)
        return result

    def _inorder_traversal(self, root, result):
        if root:
            self._inorder_traversal(root.left, result)
            result.append(root.val)
            self._inorder_traversal(root.right, result)

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


Assignment 11.3
google collab - Yahoo India Se
Untitled10.ipynb - Colab
colab.research.google.com/drive/1dYwIOpDPsrwOFn9agsTmzd08uCm9wIWi#scrollTo=j1yTNDQT_-VF
Untitled10.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM
Disk
[14]
0s
Variables
Terminal
2:22 PM
Python 3
Search
ENG
IN
14:25
08-10-2025

```
        """Performs an inorder traversal of the BST and prints the node values."""
        result = []
        self._inorder_traversal(self.root, result)
        return result

    def _inorder_traversal(self, root, result):
        if root:
            self._inorder_traversal(root.left, result)
            result.append(root.val)
            self._inorder_traversal(root.right, result)

# Example usage of the BST class:
bst = BST()
bst.insert(50)
bst.insert(30)
bst.insert(20)
bst.insert(40)
bst.insert(70)
bst.insert(60)
bst.insert(80)

print("Inorder traversal of the BST:")
print(bst.inorder_traversal()) # Expected Output: [20, 30, 40, 50, 60, 70, 80]
```

```
Inorder traversal of the BST:
[20, 30, 40, 50, 60, 70, 80]
```

Observation:

The code defines a BST class with methods to insert nodes (insert) and perform an inorder traversal (inorder_traversal).

The example usage demonstrates inserting several values into the BST.

The inorder_traversal() method visits the nodes in ascending order, resulting in the output [20, 30, 40, 50, 60, 70, 80], which confirms the property of a Binary Search Tree where an inorder traversal yields a sorted sequence.