AI_ASSIGNMENT-11.4

HTNO : 2403A51292

BATCH : 12

Task 1:

Implementing a Stack (LIFO)

• Task: Use AI to help implement a Stack class in Python with the following operations: push(), pop(), peek(), and is_empty().

• Instructions:

o Ask AI to generate code skeleton with docstrings.

o Test stack operations using sample data.

o Request AI to suggest optimizations or alternative implementations (e.g., using collections.deque).

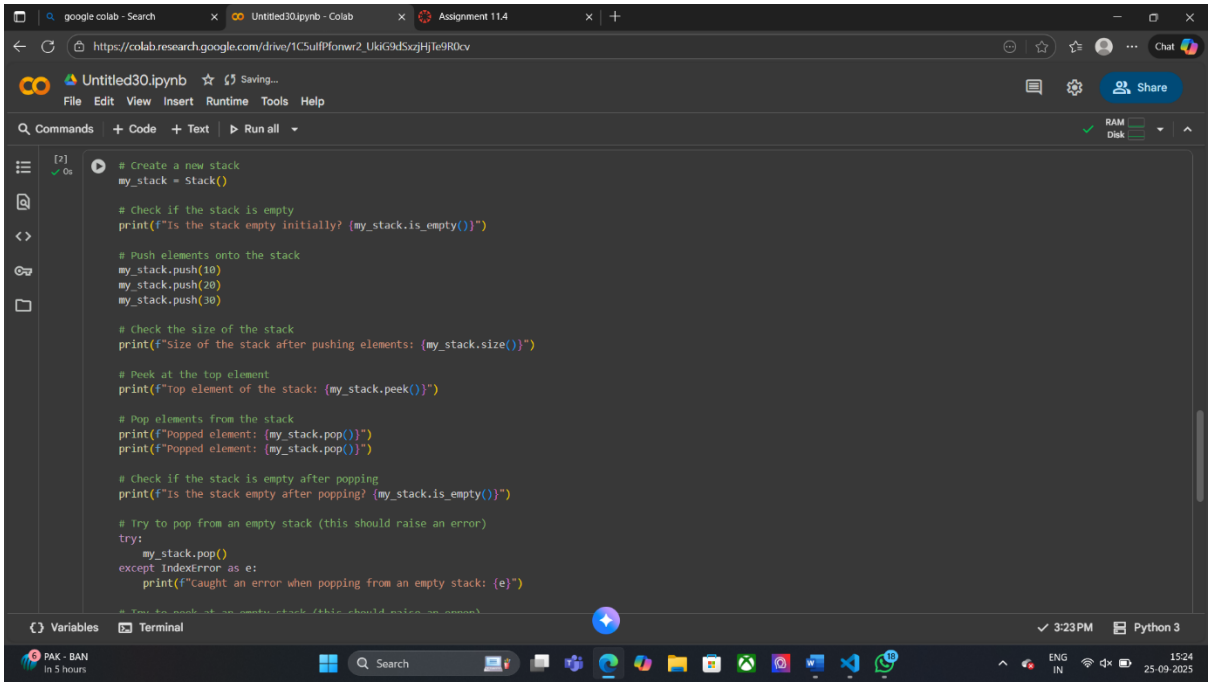
• Expected Output:

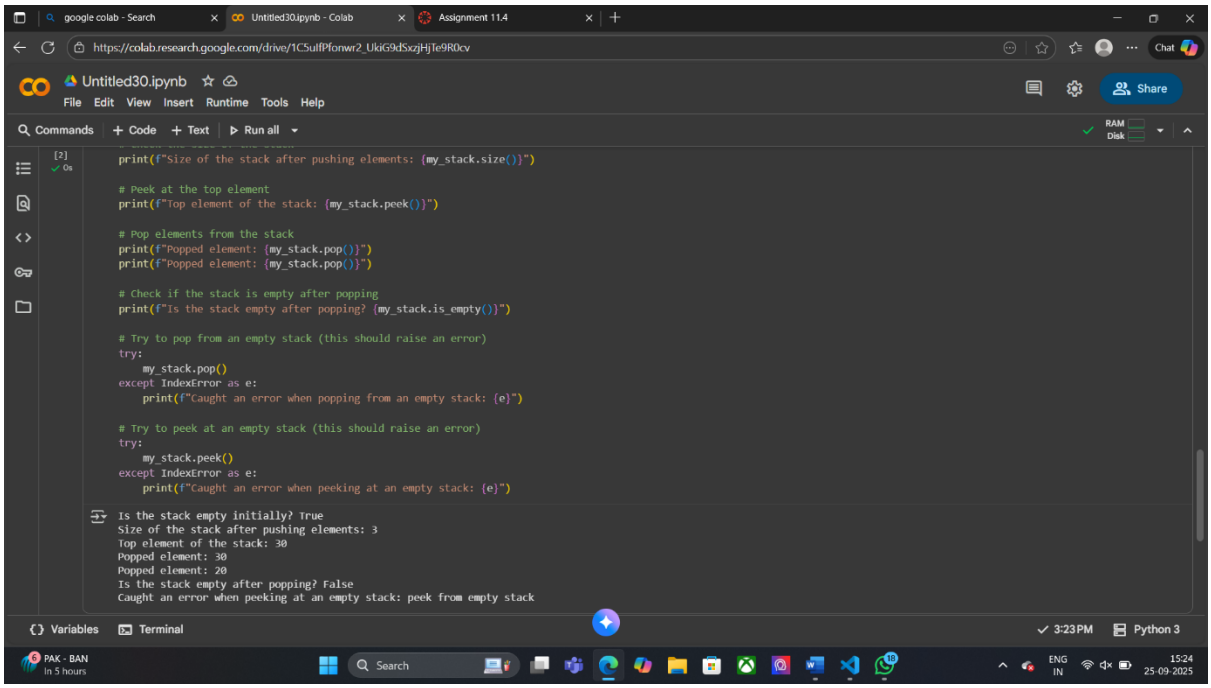
o A working Stack class with proper methods, Google-style docstrings, and inline comments for tricky parts

Prompt:

Use AI to generate a Python Stack class with push(), pop(), peek(), and is_empty() methods.

- Include Google-style docstrings for each method.
- Add inline comments for tricky logic.
- Test stack operations with sample data.
- Ask AI to suggest optimizations or alternative implementations (e.g., using collections.deque).





Task 2 :

Queue Implementation with Performance Review

• Task: Implement a Queue with enqueue(), dequeue(), and is_empty() methods.

• Instructions:

o First, implement using Python lists.

o Then, ask AI to review performance and suggest a more efficient implementation (using collections.deque).

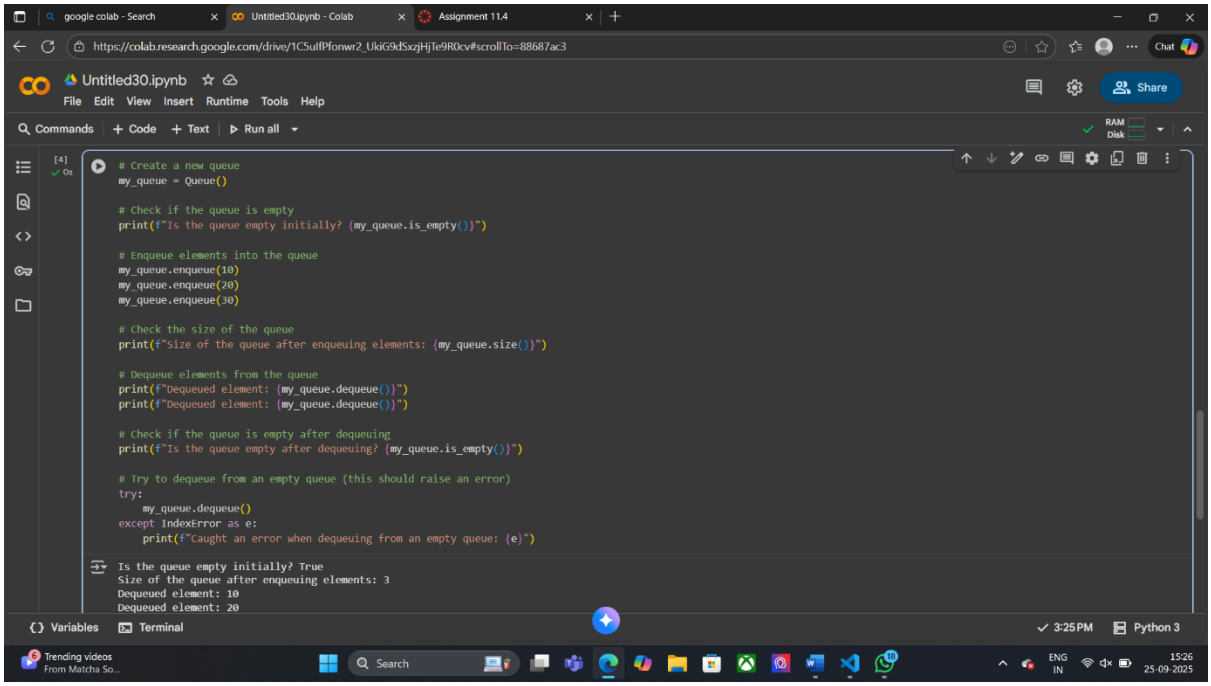
• Expected Output:

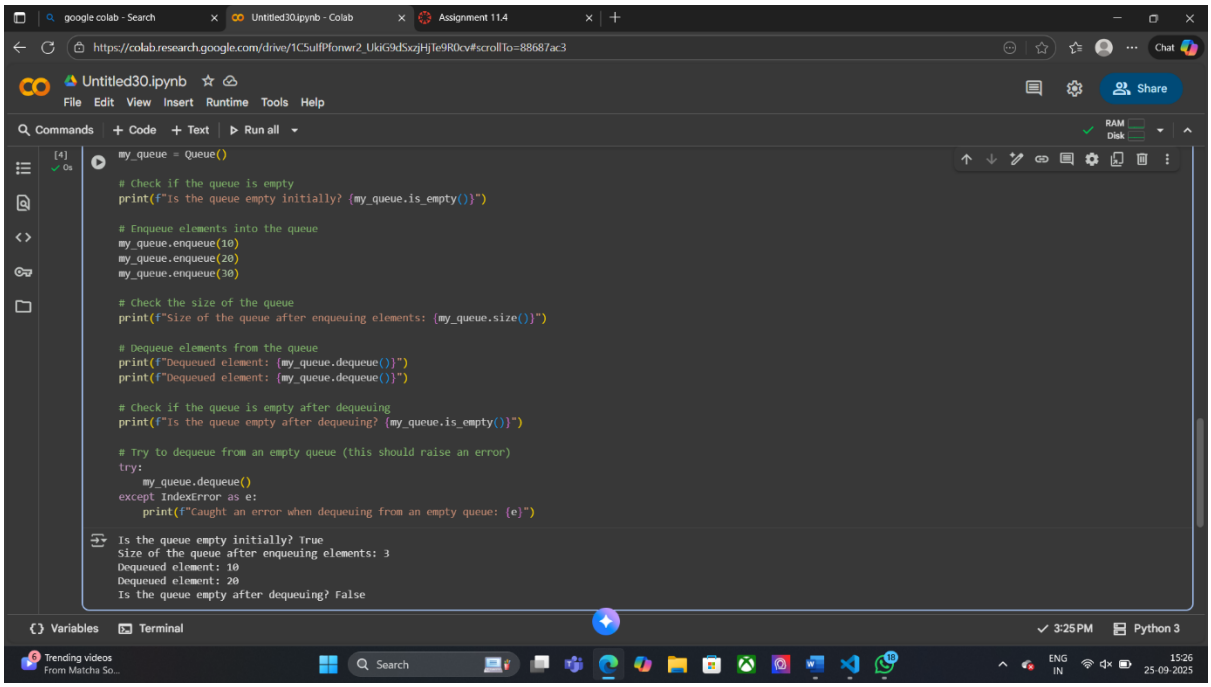
o Two versions of a queue: one with lists and one optimized with deque, plus an AI-generated performance comparison

Prompt:

Implement a Python Queue class with enqueue(), dequeue(), and is_empty() methods using lists.

- Ask AI to review the performance and suggest a more efficient implementation using collections.deque.
- Provide both versions and an AI-generated performance comparison.





Task 3 :

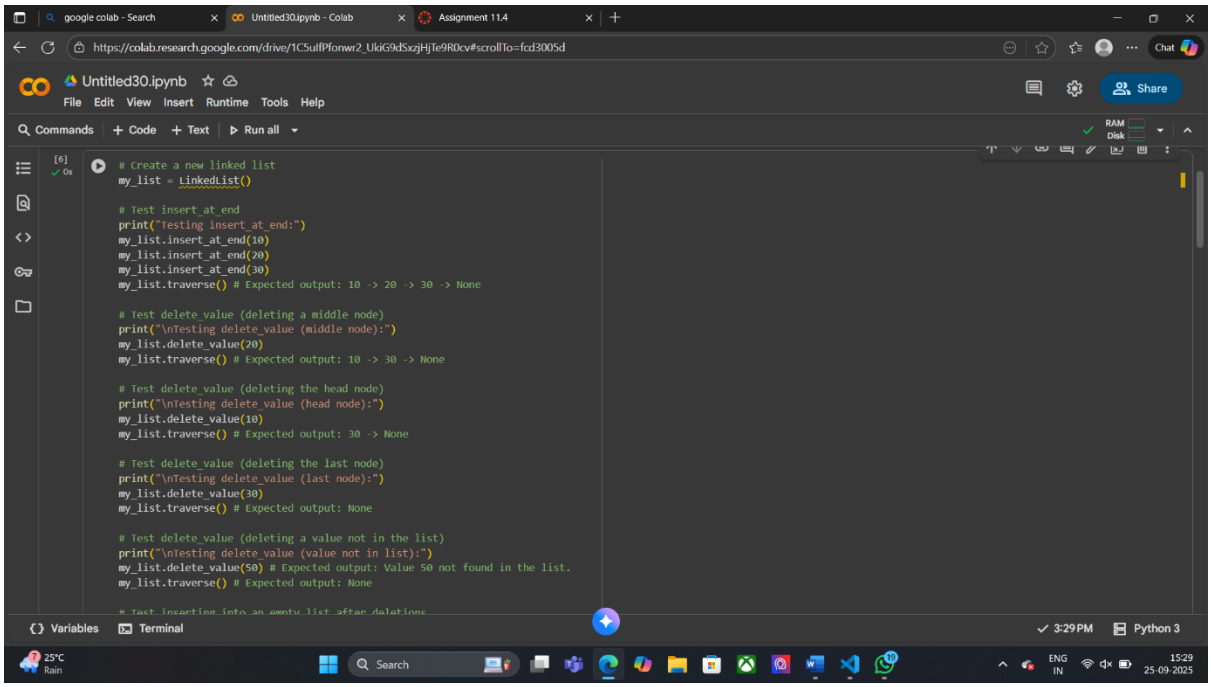
Singly Linked List with Traversal

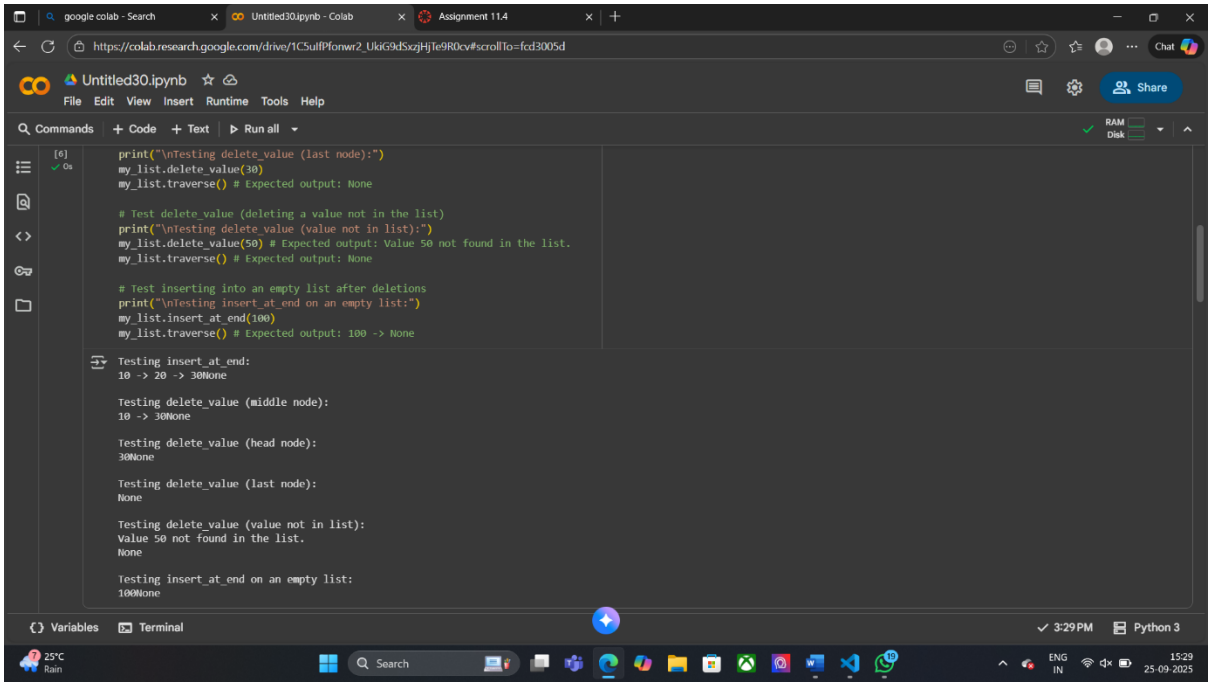
• Task: Implement a Singly Linked List with operations:
insert_at_end(), delete_value(), and traverse().
• Instructions:
o Start with a simple class-based implementation (Node, LinkedList).
o Use AI to generate inline comments explaining pointer updates (which are non-trivial).
o Ask AI to suggest test cases to validate all operations.
• Expected Output:
o A functional linked list implementation with clear comments explaining the logic of insertions and deletions.

Prompt:

Implement a singly linked list in Python with insert_at_end(), delete_value(), and traverse() methods.

- Use AI to generate inline comments explaining pointer updates.
- Ask AI to suggest test cases to validate all operations.





Task 4 :

Binary Search Tree (BST)

• Task: Implement a Binary Search Tree with methods for insert(), search(), and inorder_traversal().

• Instructions:

o Provide AI with a partially written Node and BST class.

o Ask AI to complete missing methods and add docstrings.

o Test with a list of integers and compare outputs of search() for present vs absent elements.

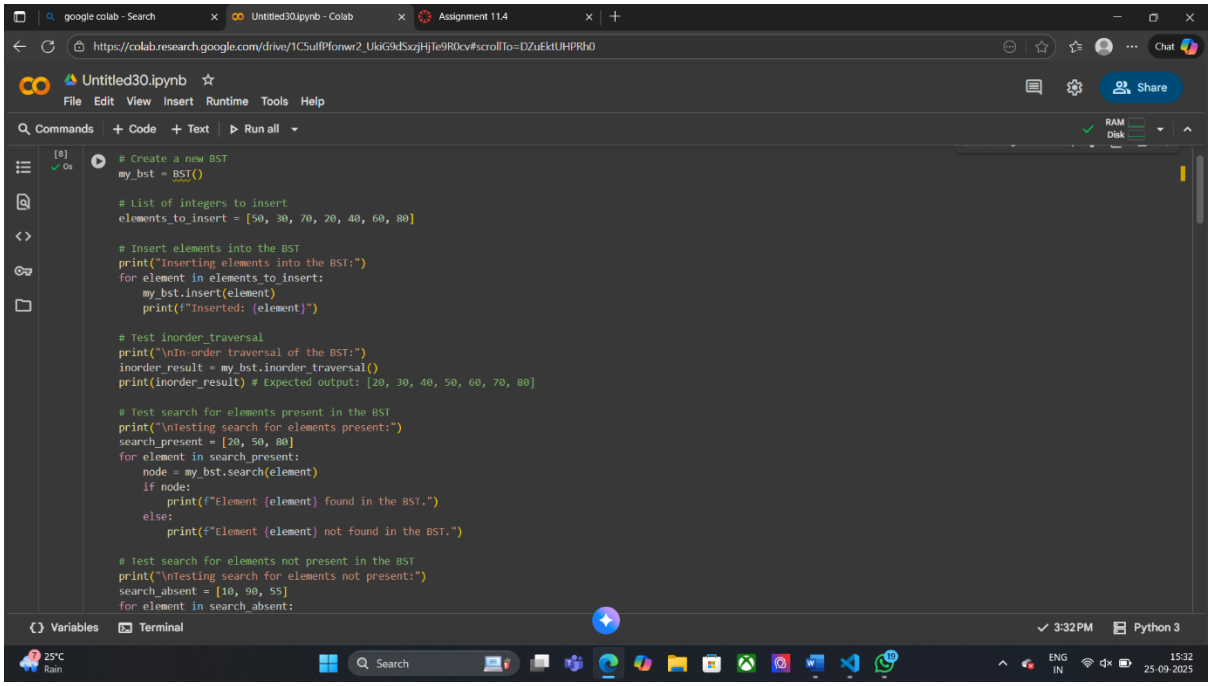
• Expected Output:

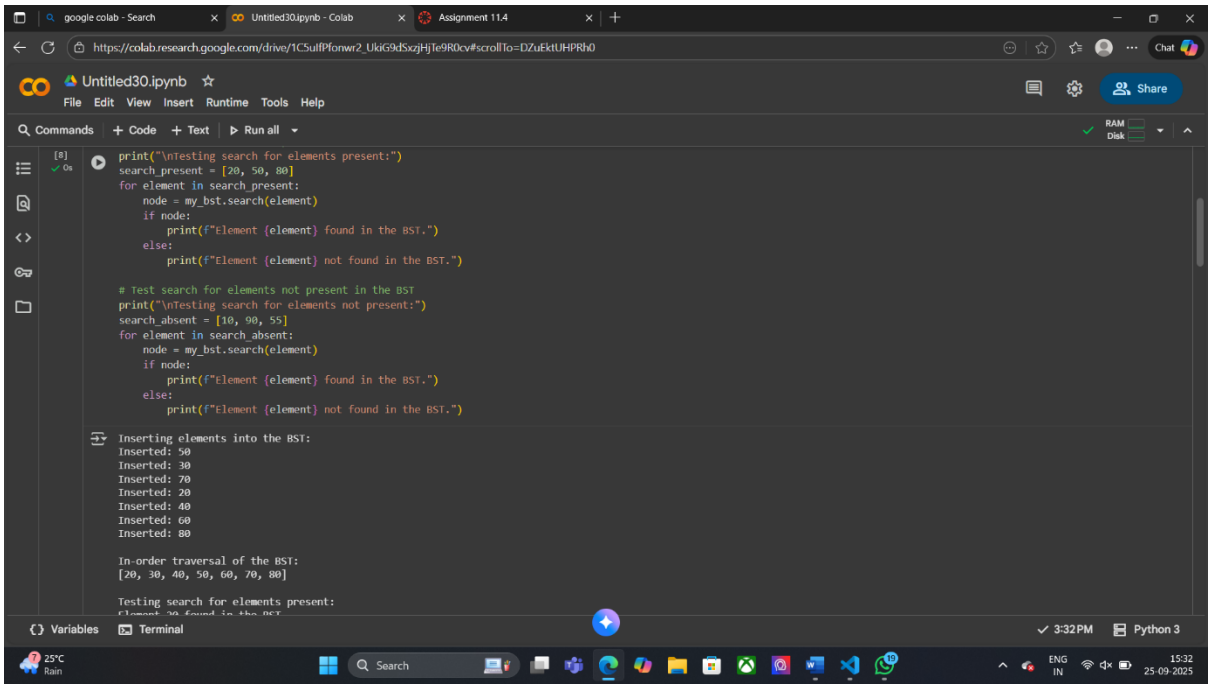
o A BST class with clean implementation, meaningful docstrings, and correct traversal output.

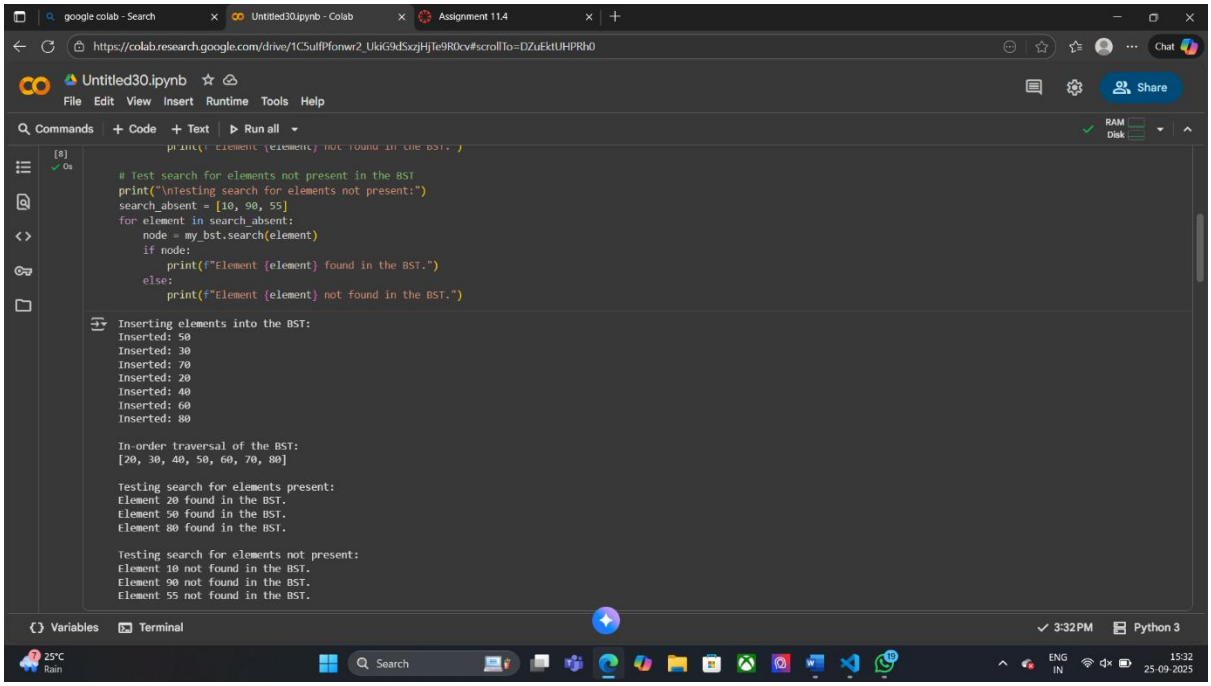
Prompt:

Given a partially written Node and BST class, use AI to help complete the implementation with insert(), search(), and inorder_traversal() methods.

- Ensure code is clear and well-documented.


google colab - Search
Untitled30.ipynb - Colab
Assignment 11.4
https://colab.research.google.com/drive/1C5ulfPfonwr2_UkiG9dSxzjHjTe9R0cv#scrollTo=DZuEktUHPRh0
Untitled30.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM Disk
[8] 0s
# Create a new BST
my_bst = BST()
# List of integers to insert
elements_to_insert = [50, 30, 70, 20, 40, 60, 80]
# Insert elements into the BST
print("Inserting elements into the BST:")
for element in elements_to_insert:
my_bst.insert(element)
print(f"Inserted: {element}")
# Test inorder_traversal
print("\nIn-order traversal of the BST:")
inorder_result = my_bst.inorder_traversal()
print(inorder_result) # Expected output: [20, 30, 40, 50, 60, 70, 80]
# Test search for elements present in the BST
print("\nTesting search for elements present:")
search_present = [20, 50, 80]
for element in search_present:
node = my_bst.search(element)
if node:
print(f"Element {element} found in the BST.")
else:
print(f"Element {element} not found in the BST.")
# Test search for elements not present in the BST
print("\nTesting search for elements not present:")
search_absent = [10, 90, 55]
for element in search_absent:
Variables Terminal
3:32PM Python 3
25°C Rain
Search
ENG IN
15:32 25-09-2025


google colab - Search
Untitled30.ipynb - Colab
Assignment 11.4
https://colab.research.google.com/drive/1C5ulfPfonwr2_UkiG9dSxzjHjTe9R0cv#scrollTo=DZuEktUHPRh0
Untitled30.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM Disk
[8] 0s
print("\nTesting search for elements present:")
search_present = [20, 50, 80]
for element in search_present:
node = my_bst.search(element)
if node:
print(f"Element {element} found in the BST.")
else:
print(f"Element {element} not found in the BST.")
# Test search for elements not present in the BST
print("\nTesting search for elements not present:")
search_absent = [10, 90, 55]
for element in search_absent:
node = my_bst.search(element)
if node:
print(f"Element {element} found in the BST.")
else:
print(f"Element {element} not found in the BST.")
Inserting elements into the BST:
Inserted: 50
Inserted: 30
Inserted: 70
Inserted: 20
Inserted: 40
Inserted: 60
Inserted: 80
In-order traversal of the BST:
[20, 30, 40, 50, 60, 70, 80]
Testing search for elements present:
Variables Terminal
3:32PM Python 3
25°C Rain
Search
ENG IN
15:32 25-09-2025



Task 5 :

Graph Representation and BFS/DFS Traversal

• Task: Implement a Graph using an adjacency list, with traversal methods BFS() and DFS().

• Instructions:

o Start with an adjacency list dictionary.

o Ask AI to generate BFS and DFS implementations with inline comments.

o Compare recursive vs iterative DFS if suggested by AI.

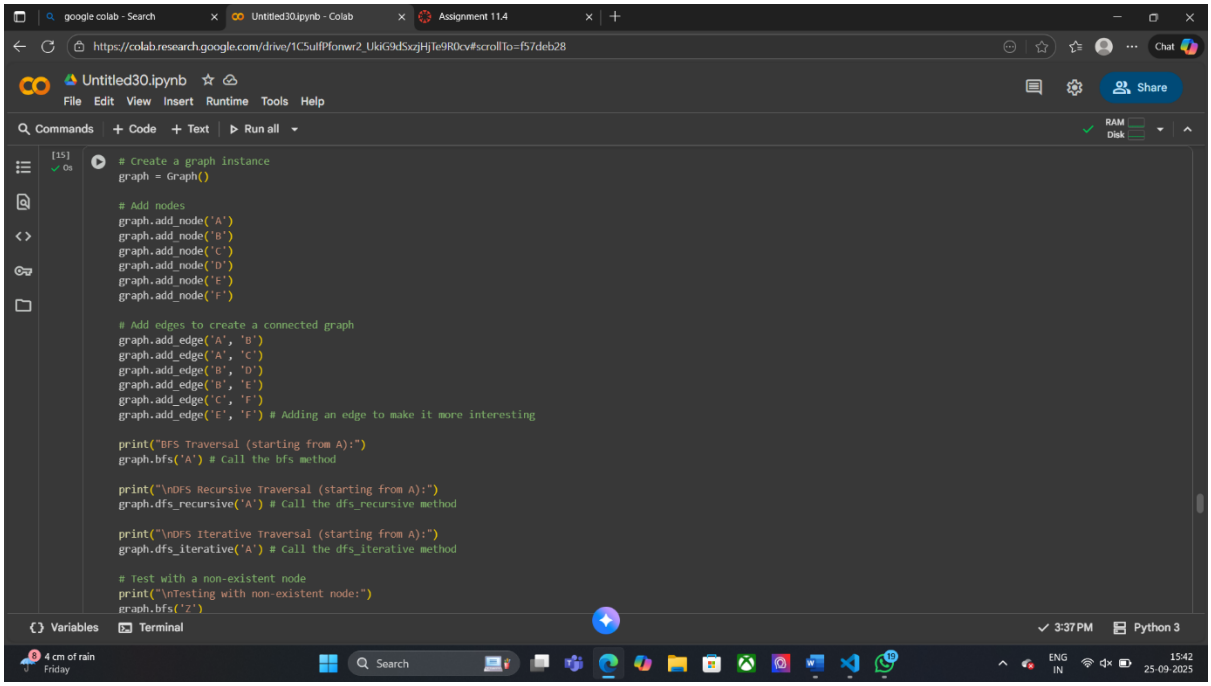
• Expected Output:

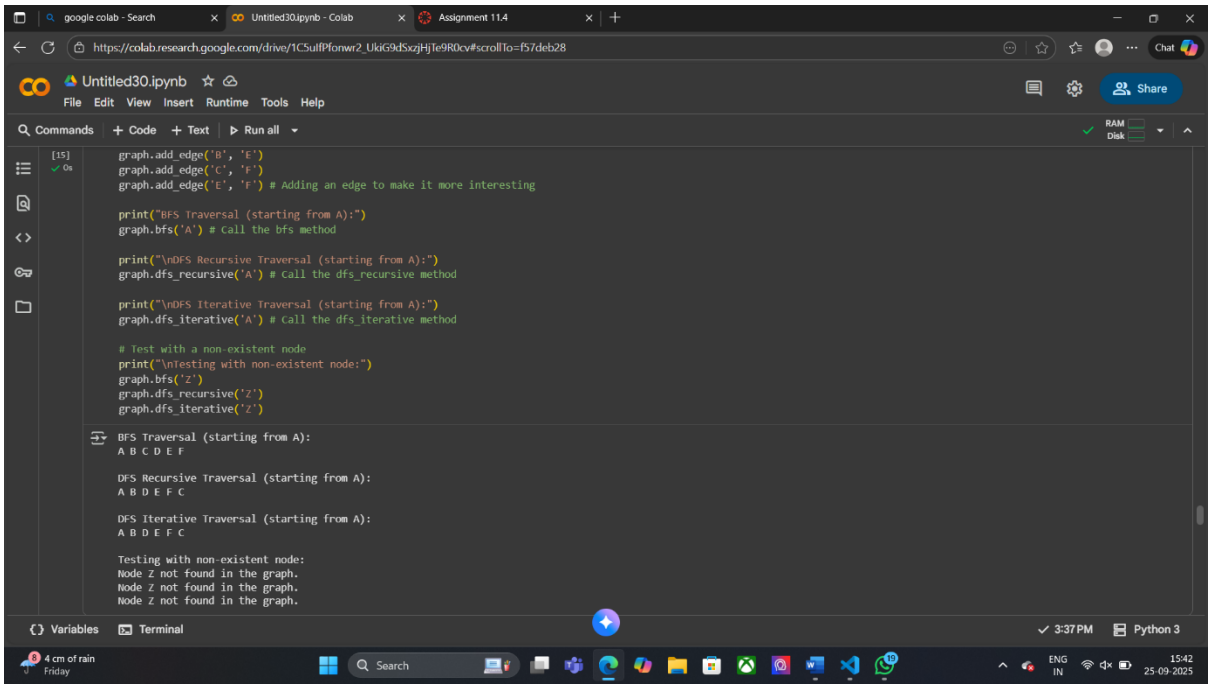
o A graph implementation with BFS and DFS traversal methods, with AI-generated comments explaining traversal steps.

Prompt:

Implement a Python Graph class using an adjacency list (dictionary).

- Add methods for BFS(start) and DFS(start) traversal.
- Ask AI to generate both BFS and DFS implementations with inline comments explaining each traversal step.
- If suggested by AI, compare recursive and iterative DFS approaches.
- Ensure the code is clear, with comments that explain the logic of each traversal


google colab - Search
Untitled30.ipynb - Colab
Assignment 11.4
https://colab.research.google.com/drive/1C5ulfPfonwr2_UkiG9dSxzjHjTe9R0cv#scrollTo=f57deb28
Untitled30.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM Disk
[15]
0s
# Create a graph instance
graph = Graph()
# Add nodes
graph.add_node('A')
graph.add_node('B')
graph.add_node('C')
graph.add_node('D')
graph.add_node('E')
graph.add_node('F')
# Add edges to create a connected graph
graph.add_edge('A', 'B')
graph.add_edge('A', 'C')
graph.add_edge('B', 'D')
graph.add_edge('B', 'E')
graph.add_edge('C', 'F')
graph.add_edge('E', 'F') # Adding an edge to make it more interesting
print("BFS Traversal (starting from A):")
graph.bfs('A') # Call the bfs method
print("\nDFS Recursive Traversal (starting from A):")
graph.dfs_recursive('A') # Call the dfs_recursive method
print("\nDFS Iterative Traversal (starting from A):")
graph.dfs_iterative('A') # Call the dfs_iterative method
# Test with a non-existent node
print("\nTesting with non-existent node:")
graph.bfs('Z')
Variables Terminal
3:37 PM Python 3
4 cm of rain Friday
Search
ENG IN
15:42 25-09-2025


google colab - Search
Untitled30.ipynb - Colab
Assignment 11.4
https://colab.research.google.com/drive/1C5ulfPfonwr2_UkiG9dSxzjHjTe9R0cv#scrollTo=f57deb28
Untitled30.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM Disk
[15]
0s
graph.add_edge('B', 'E')
graph.add_edge('C', 'F')
graph.add_edge('E', 'F') # Adding an edge to make it more interesting
print("BFS Traversal (starting from A):")
graph.bfs('A') # Call the bfs method
print("\nDFS Recursive Traversal (starting from A):")
graph.dfs_recursive('A') # Call the dfs_recursive method
print("\nDFS Iterative Traversal (starting from A):")
graph.dfs_iterative('A') # Call the dfs_iterative method
# Test with a non-existent node
print("\nTesting with non-existent node:")
graph.bfs('Z')
graph.dfs_recursive('Z')
graph.dfs_iterative('Z')
BFS Traversal (starting from A):
A B C D E F
DFS Recursive Traversal (starting from A):
A B D E F C
DFS Iterative Traversal (starting from A):
A B D E F C
Testing with non-existent node:
Node Z not found in the graph.
Node Z not found in the graph.
Node Z not found in the graph.
Variables Terminal
3:37 PM Python 3
4 cm of rain Friday
Search
ENG IN
15:42 25-09-2025