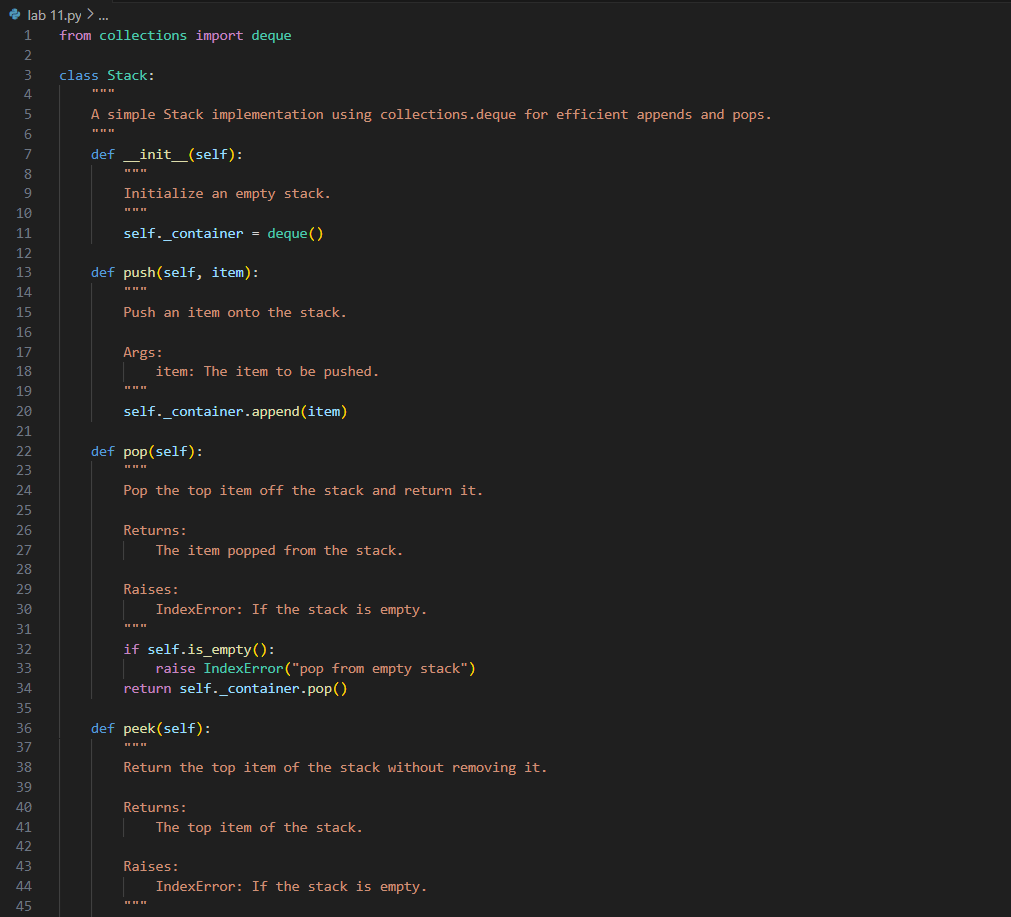
# LAB ASSIGNMENT-11

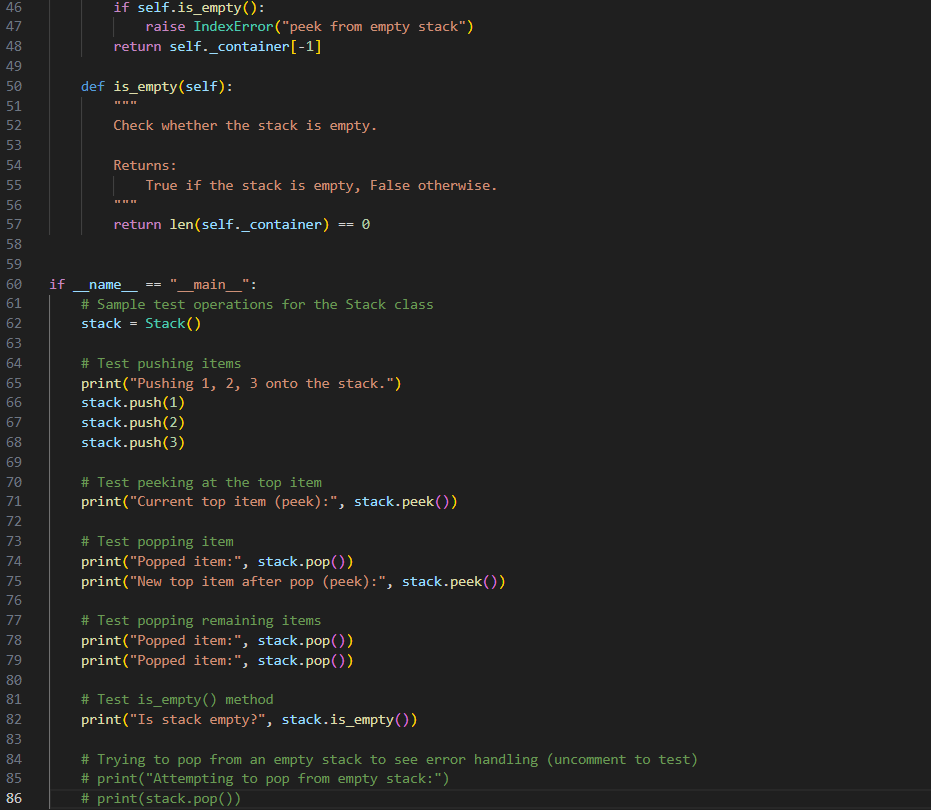
TASK-1:

Prompt:

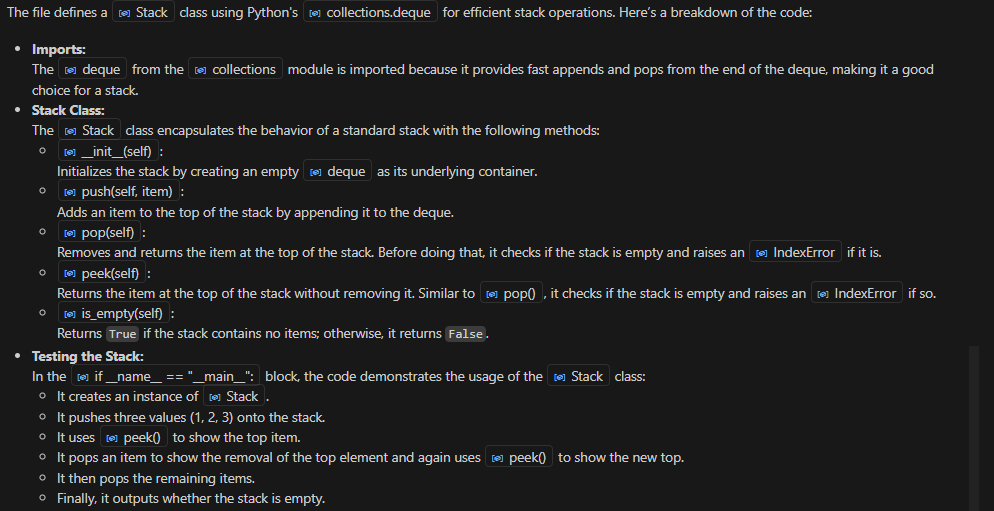
Implement a Stack class in Python with the following operations: push(), pop(), peek(), and is\_empty().Generate code skeleton with docstrings also Test stack operations using sample data. Use optimizations or alternative implementations (e.g., using collections.deque) if needed.

Code and Output:





Code Explanation:

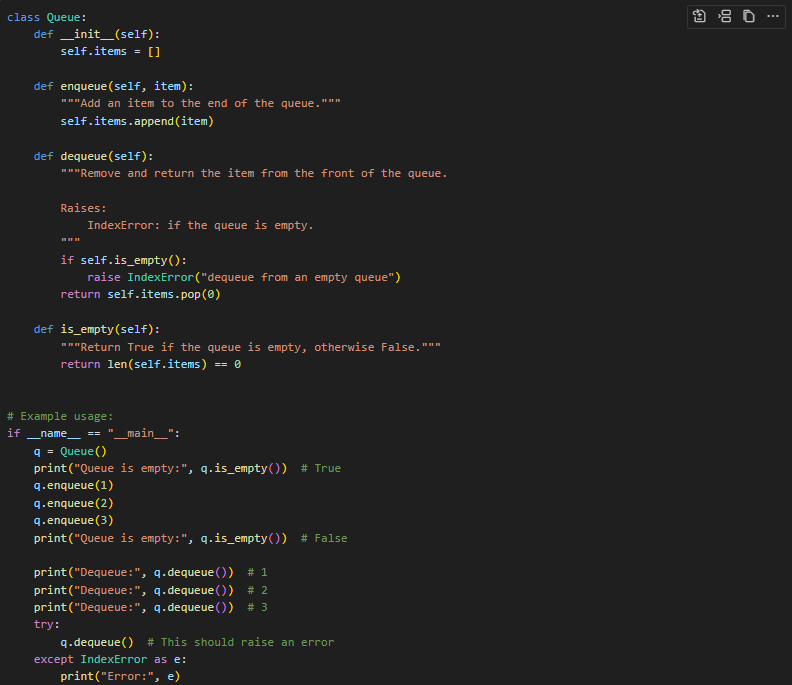


TASK-2:

Prompt:

Implement a Queue with enqueue(), dequeue(), and is\_empty() methods using Python lists.

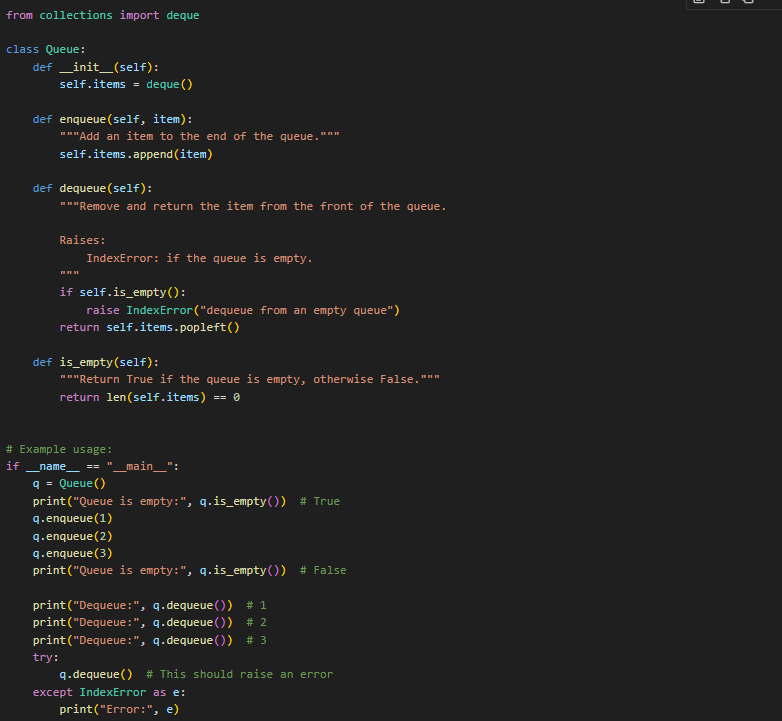
Code and Output:



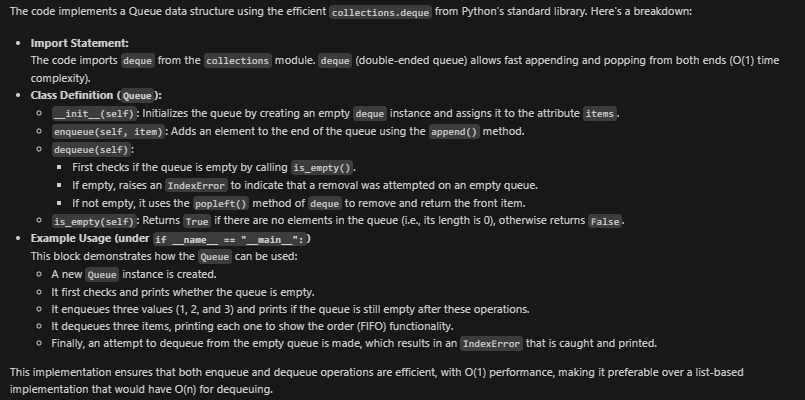
Prompt:

Review performance and suggest a more efficient implementation (using collections.deque).

Code and Output:



Code Explanation:

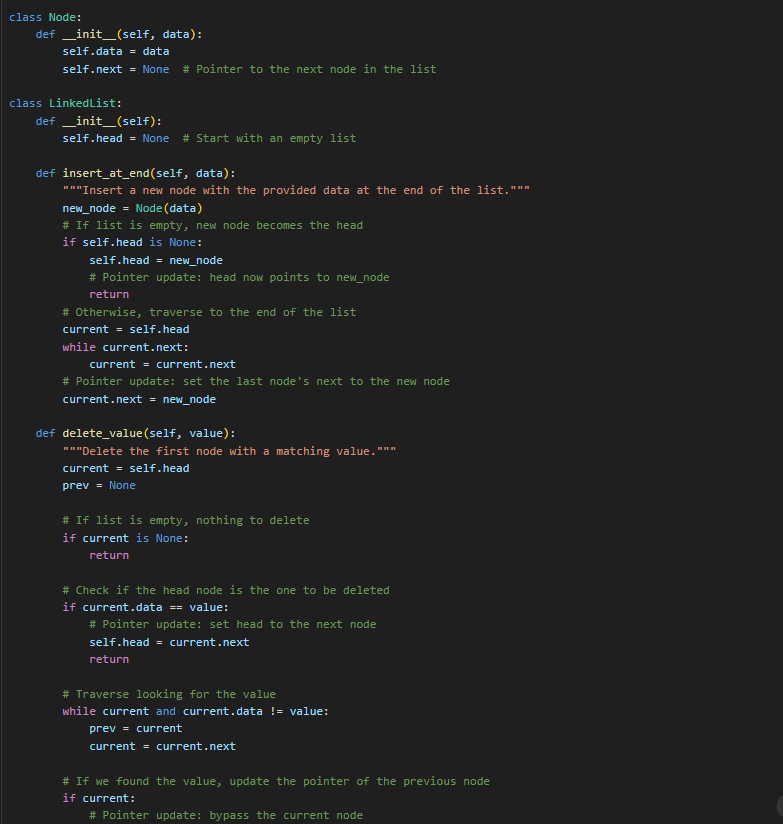


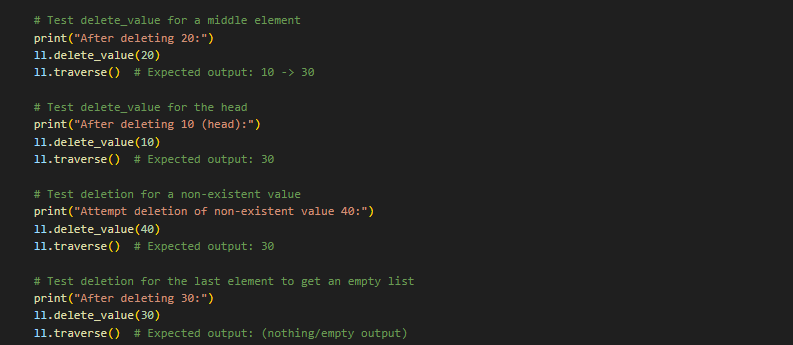
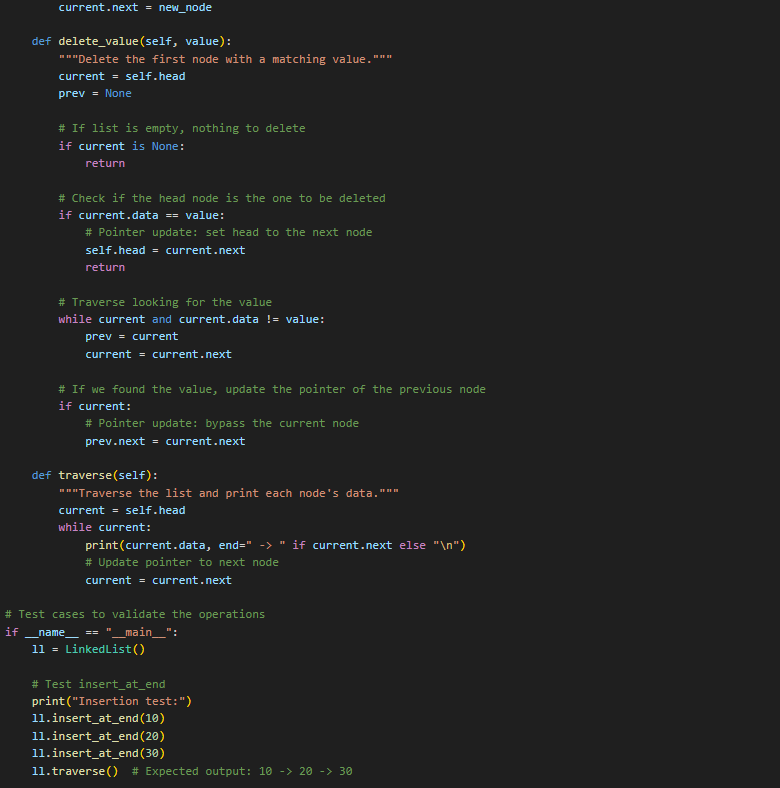
TASK-3:

Prompt:

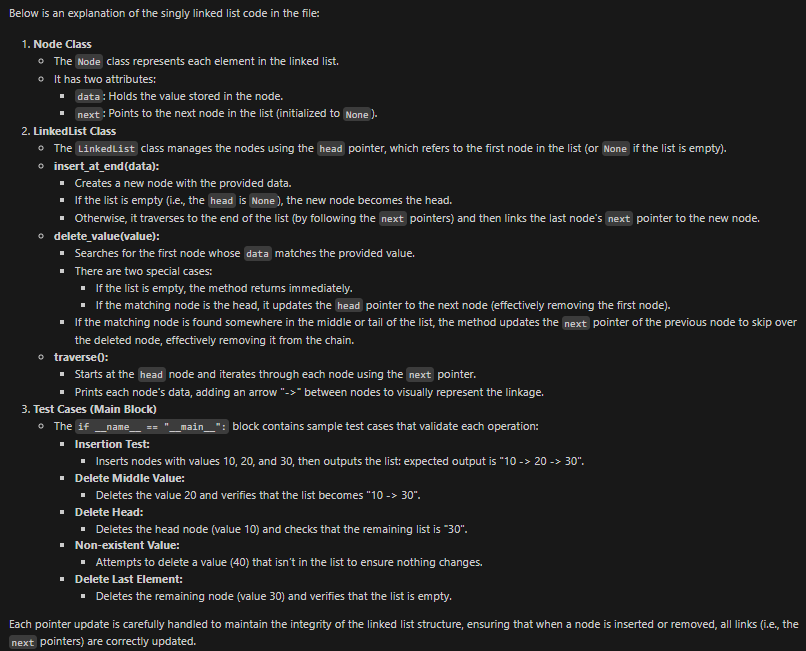
Implement a Singly Linked List with operations: insert\_at\_end(), delete\_value(), and traverse(). Start with a simple class-based implementation (Node, LinkedList). Generate inline comments explaining pointer updates (which are non-trivial) also suggest test cases to validate all operations.

Code and Output:





Code Explanation:



TASK-4:

Prompt:

class Node:

def \_\_init\_\_(self, key):

self.key = key

self.left = None

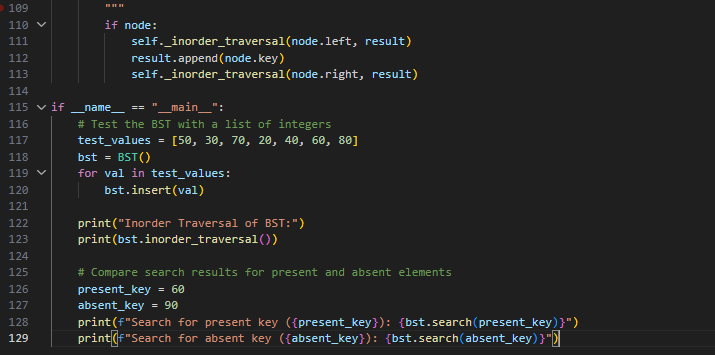
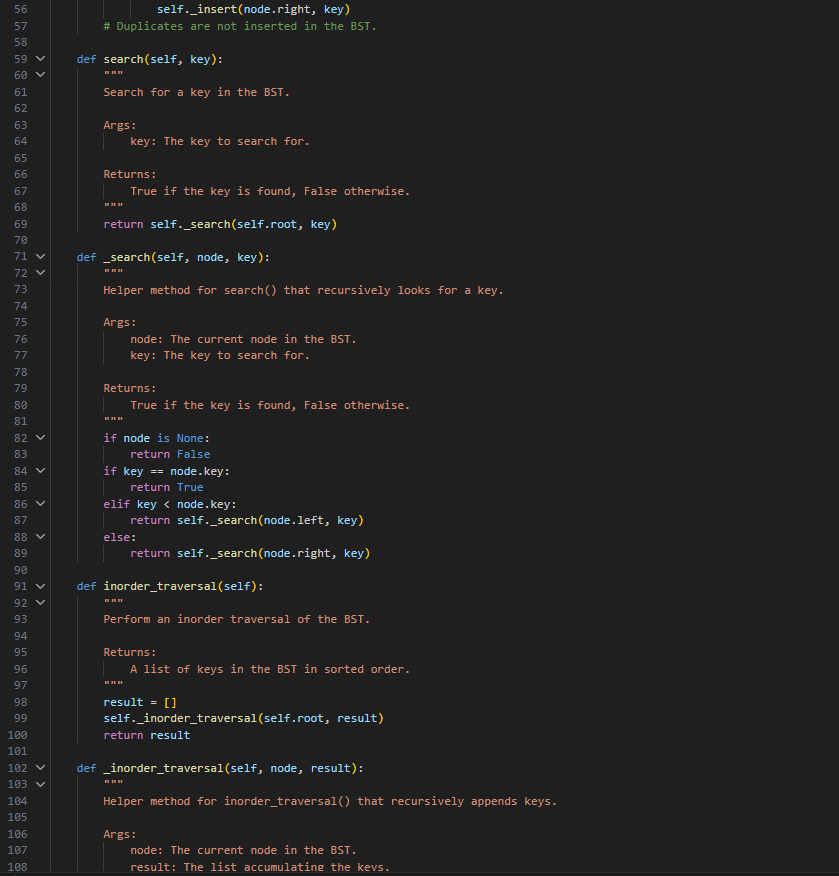
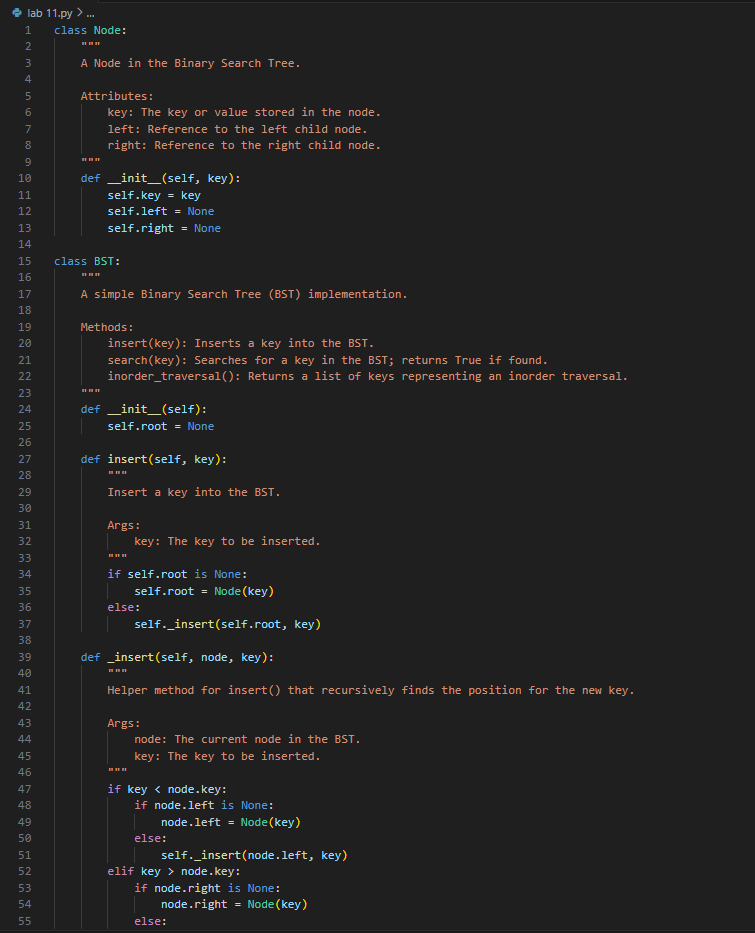
self.right = None

class BST:

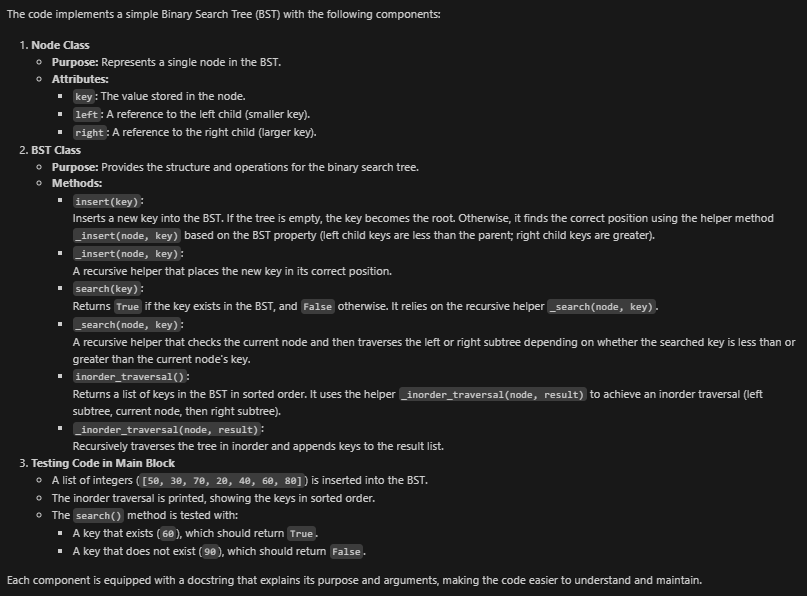
def \_\_init\_\_(self):

self.root = None

Here, is a partially written Node and BST class. Implement a Binary Search Tree with methods for insert(), search(), and inorder\_traversal() and complete missing methods and add docstrings. Test with a list of integers and compare outputs of search() for present vs absent elements.



Code Explanation:

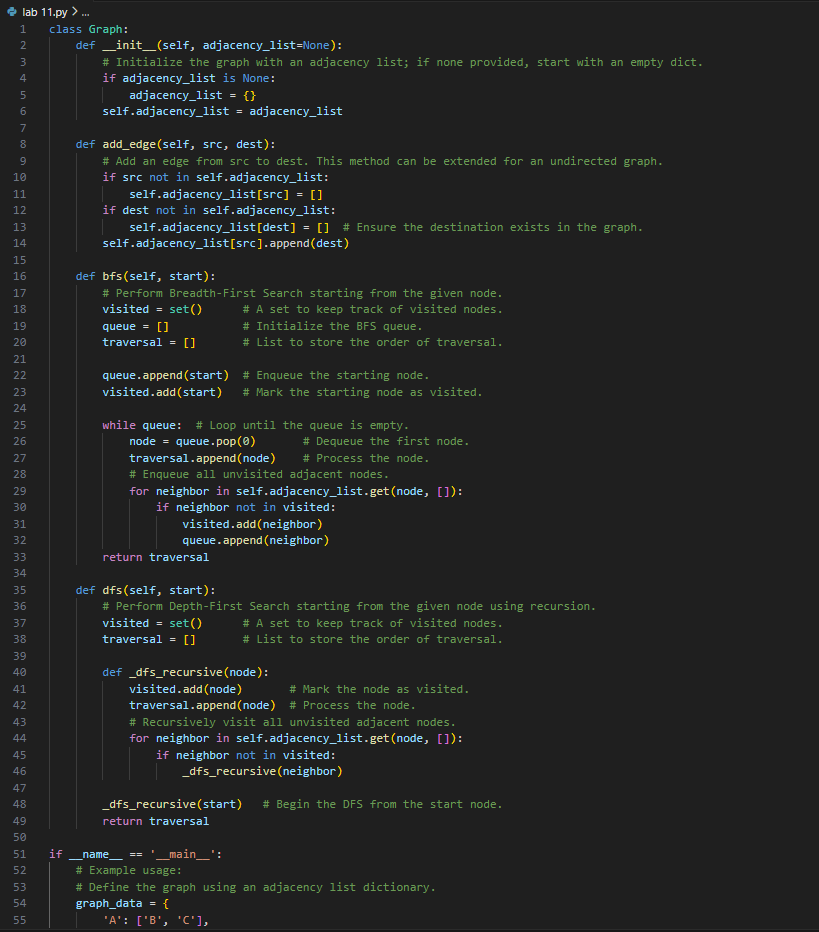


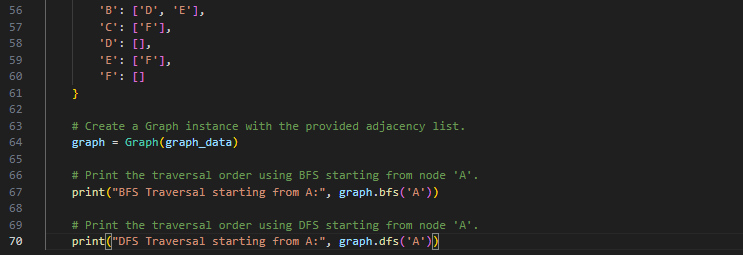
TASK-5:

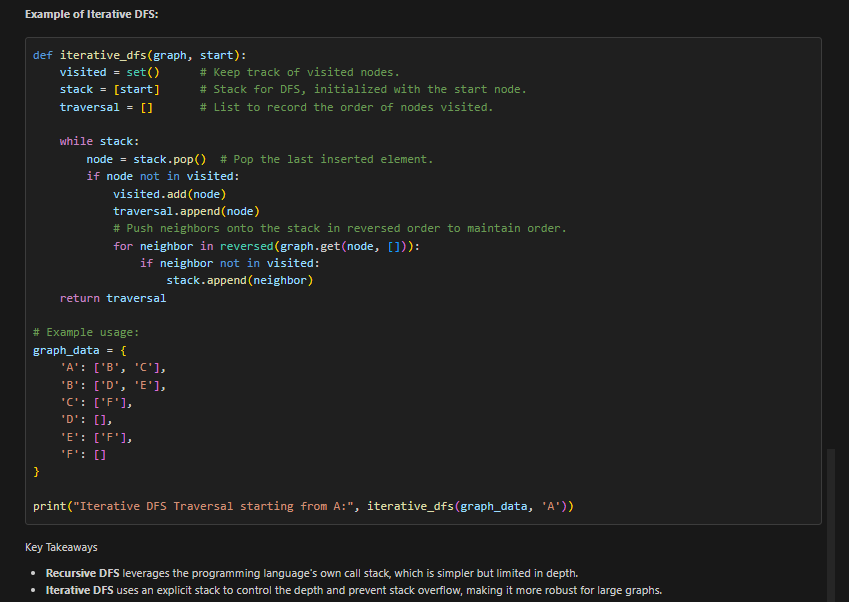
Prompt:

Implement a Graph using an adjacency list, with traversal methods BFS() and DFS(). Start with an adjacency list dictionary. Generate BFS and DFS implementations with inline comments.

Code and Output:







Code Explanation:

