class Node:

    def \_\_init\_\_(self, value):

        self.value = value

        self.neighbors = []

    def add\_neighbor(self, node):

        self.neighbors.append(node)

def dfs\_with\_stack(start\_node):

    visited = set()

    stack = [start\_node]

    while stack:

        current\_node = stack.pop()

        if current\_node not in visited:

            print(current\_node.value, end=" ")

            visited.add(current\_node)

            for neighbor in reversed(current\_node.neighbors):

                if neighbor not in visited:

                    stack.append(neighbor)

a = Node('A')

b = Node('B')

c = Node('C')

d = Node('D')

e = Node('E')

a.add\_neighbor(b)

a.add\_neighbor(c)

b.add\_neighbor(d)

c.add\_neighbor(e)

dfs\_with\_stack(a)

class TreeNode:

    def \_\_init\_\_(self, value):

        self.value = value

        self.left = None

        self.right = None

def inorder\_traversal(node):

    if node:

        inorder\_traversal(node.left)

        print(node.value, end=" ")

        inorder\_traversal(node.right)

def preorder\_traversal(node):

    if node:

        print(node.value, end=" ")

        preorder\_traversal(node.left)

        preorder\_traversal(node.right)

def postorder\_traversal(node):

    if node:

        postorder\_traversal(node.left)

        postorder\_traversal(node.right)

        print(node.value, end=" ")

root = TreeNode(1)

root.left = TreeNode(2)

root.right = TreeNode(3)

root.left.left = TreeNode(4)

root.left.right = TreeNode(5)

print("Inorder Traversal: ", end="")

inorder\_traversal(root)

print("\nPreorder Traversal: ", end="")

preorder\_traversal(root)

print("\nPostorder Traversal: ", end="")

postorder\_traversal(root)