**Task 1**

class Node:

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

self.value = value

self.neighbors = []

def add\_neighbor(self, neighbor):

self.neighbors.append(neighbor)

def dfs(start\_node):

visited = set()

stack = [start\_node]

while stack:

current\_node = stack.pop()

if current\_node not in visited:

print(current\_node.value)

visited.add(current\_node)

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

if neighbor not in visited:

stack.append(neighbor)

node1 = Node(1)

node2 = Node(2)

node3 = Node(3)

node4 = Node(4)

node5 = Node(5)

node1.add\_neighbor(node2)

node1.add\_neighbor(node3)

node2.add\_neighbor(node4)

node3.add\_neighbor(node5)

dfs(node1)

**Task 2**

**class Nod**e:

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

self.value = value

self.left = None

self.right = None

def preorder(root):

if root:

print(root.value, end=" ")

preorder(root.left)

preorder(root.right)

def inorder(root):

if root:

inorder(root.left)

print(root.value, end=" ")

inorder(root.right)

def postorder(root):

if root:

postorder(root.left)

postorder(root.right)

print(root.value, end=" ")

root = Node(1)

root.left = Node(2)

root.right = Node(3)

root.left.left = Node(4)

root.left.right = Node(5)

root.right.right = Node(6)

print("Preorder Traversal:")

preorder(root)

print("\nInorder Traversal:")

inorder(root)

print("\nPostorder Traversal:")

postorder(root)