# Lab Task 5

## Task 1:

def dfs\_stack(graph, start):  
 visited = set()   
 stack = [start]   
  
 while stack:   
 vertex = stack.pop()   
 if vertex not in visited:  
 print(vertex, end=" ")   
 visited.add(vertex)  
  
 for neighbor in reversed(graph[vertex]):  
 if neighbor not in visited:  
 stack.append(neighbor)  
  
  
graph = {  
 'A': ['B', 'C'],  
 'B': ['D', 'E'],  
 'C': ['F'],  
 'D': [],  
 'E': ['F'],  
 'F': []  
}  
  
print("DFS using Stack:")  
dfs\_stack(graph, 'A')

Line-by-line explanation:

001. def dfs\_stack(graph, start):

→ Defines function 'dfs\_stack' with parameters (graph, start).

002. visited = set()

→ Assigns the value/expression on the right to the variable(s) on the left: 'visited'.

003. stack = [start]

→ Assigns the value/expression on the right to the variable(s) on the left: 'stack'.

004. (blank)

005. while stack:

→ Starts a while-loop that repeats until a condition becomes false.

006. vertex = stack.pop()

→ Assigns the value/expression on the right to the variable(s) on the left: 'vertex'.

007. if vertex not in visited:

→ Starts a conditional block that runs code only if the condition is true.

008. print(vertex, end=" ")

→ Prints output to the console / notebook output.

009. visited.add(vertex)

→ Executes: visited.add(vertex)

010. (blank)

011. for neighbor in reversed(graph[vertex]):

→ Starts a for-loop to iterate over a sequence or iterator.

012. if neighbor not in visited:

→ Starts a conditional block that runs code only if the condition is true.

013. stack.append(neighbor)

→ Executes: stack.append(neighbor)

014. (blank)

015. (blank)

016. graph = {

→ Assigns the value/expression on the right to the variable(s) on the left: 'graph'.

017. 'A': ['B', 'C'],

→ Executes: 'A': ['B', 'C'],

018. 'B': ['D', 'E'],

→ Executes: 'B': ['D', 'E'],

019. 'C': ['F'],

→ Executes: 'C': ['F'],

020. 'D': [],

→ Executes: 'D': [],

021. 'E': ['F'],

→ Executes: 'E': ['F'],

022. 'F': []

→ Executes: 'F': []

023. }

→ Executes: }

024. (blank)

025. print("DFS using Stack:")

→ Prints output to the console / notebook output.

026. dfs\_stack(graph, 'A')

→ Executes: dfs\_stack(graph, 'A')

—

## Task 2:

tree = {  
 'A': ['B', 'C'],  
 'B': ['D', 'E'],  
 'C': [None, 'F'],  
 'D': [None, None],  
 'E': [None, None],  
 'F': [None,None ]  
}  
  
def preorder(tree, root):  
 if root is None:  
 return  
 print(root, end=" ")  
 preorder(tree, tree[root][0])   
 preorder(tree, tree[root][1])   
  
def inorder(tree, root):  
 if root is None:  
 return  
 inorder(tree, tree[root][0])   
 print(root, end=" ")  
 inorder(tree, tree[root][1])   
  
def postorder(tree, root):  
 if root is None:  
 return  
 postorder(tree, tree[root][0])   
 postorder(tree, tree[root][1])   
 print(root, end=" ")  
  
print("Preorder Traversal:")  
preorder(tree, 'A')  
print("\nInorder Traversal:")  
inorder(tree, 'A')  
print("\nPostorder Traversal:")  
postorder(tree, 'A')

**Line-by-line explanation:**

001. tree = {

→ Assigns the value/expression on the right to the variable(s) on the left: 'tree'.

002. 'A': ['B', 'C'],

→ Executes: 'A': ['B', 'C'],

003. 'B': ['D', 'E'],

→ Executes: 'B': ['D', 'E'],

004. 'C': [None, 'F'],

→ Executes: 'C': [None, 'F'],

005. 'D': [None, None],

→ Executes: 'D': [None, None],

006. 'E': [None, None],

→ Executes: 'E': [None, None],

007. 'F': [None,None ]

→ Executes: 'F': [None,None ]

008. }

→ Executes: }

009. (blank)

010. def preorder(tree, root):

→ Defines function 'preorder' with parameters (tree, root).

011. if root is None:

→ Starts a conditional block that runs code only if the condition is true.

012. return

→ Executes: return

013. print(root, end=" ")

→ Prints output to the console / notebook output.

014. preorder(tree, tree[root][0])

→ Executes: preorder(tree, tree[root][0])

015. preorder(tree, tree[root][1])

→ Executes: preorder(tree, tree[root][1])

016. (blank)

017. def inorder(tree, root):

→ Defines function 'inorder' with parameters (tree, root).

018. if root is None:

→ Starts a conditional block that runs code only if the condition is true.

019. return

→ Executes: return

020. inorder(tree, tree[root][0])

→ Executes: inorder(tree, tree[root][0])

021. print(root, end=" ")

→ Prints output to the console / notebook output.

022. inorder(tree, tree[root][1])

→ Executes: inorder(tree, tree[root][1])

023. (blank)

024. def postorder(tree, root):

→ Defines function 'postorder' with parameters (tree, root).

025. if root is None:

→ Starts a conditional block that runs code only if the condition is true.

026. return

→ Executes: return

027. postorder(tree, tree[root][0])

→ Executes: postorder(tree, tree[root][0])

028. postorder(tree, tree[root][1])

→ Executes: postorder(tree, tree[root][1])

029. print(root, end=" ")

→ Prints output to the console / notebook output.

030. (blank)

031. print("Preorder Traversal:")

→ Prints output to the console / notebook output.

032. preorder(tree, 'A')

→ Executes: preorder(tree, 'A')

033. print("\nInorder Traversal:")

→ Prints output to the console / notebook output.

034. inorder(tree, 'A')

→ Executes: inorder(tree, 'A')

035. print("\nPostorder Traversal:")

→ Prints output to the console / notebook output.

036. postorder(tree, 'A')

→ Executes: postorder(tree, 'A')