# DFS using Stack - Explanation

## How it Works:

1. Graph Representation:  
- The graph is stored as a dictionary where each key is a node and its value is a list of neighbors.  
- Example: 'A': ['B', 'C'] means node A is connected to B and C.  
  
2. DFS using Stack:  
- Start node is pushed into the stack.  
- While the stack is not empty:  
 • Pop the last element from the stack.  
 • If it has not been visited, print it and mark as visited.  
 • Push its neighbors (in reversed order) into the stack.  
  
3. Visiting Order:  
- Stack works on LIFO (Last In, First Out).  
- This ensures DFS explores deeper nodes first.

## Why this Approach is Used:

- Stack-based DFS is an alternative to recursive DFS.  
- It avoids recursion depth limit issues.  
- Gives more control over traversal order.

## Output:

DFS using Stack:

A B D E F C

# 2. Research about 'Inorder, Preorder, Postorder' and Implementation in DFS

## How they Work:

1. Inorder Traversal (Left → Root → Right):  
- Visits the left subtree first, then the root node, and finally the right subtree.  
- Commonly used in Binary Search Trees to get data in sorted order.  
  
2. Preorder Traversal (Root → Left → Right):  
- Visits the root first, then explores the left subtree, and then the right subtree.  
- Useful for creating a copy of the tree or generating prefix expressions.  
  
3. Postorder Traversal (Left → Right → Root):  
- Visits the left subtree first, then the right subtree, and finally the root.  
- Commonly used for deleting trees or evaluating postfix expressions.

## Why this Approach is Used:

- These three traversals are different methods of implementing DFS in tree structures.  
- Each serves a unique purpose: Inorder for sorted data, Preorder for prefix notation, and Postorder for cleanup or postfix evaluation.

## Python Implementation:

class Node:  
 def \_\_init\_\_(self, value):  
 self.value = value  
 self.left = None  
 self.right = None  
def inorder(root):  
 if root:  
 inorder(root.left)  
 print(root.value, end=' ')  
 inorder(root.right)  
def preorder(root):  
 if root:  
 print(root.value, end=' ')  
 preorder(root.left)  
 preorder(root.right)  
def postorder(root):  
 if root:  
 postorder(root.left)  
 postorder(root.right)  
 print(root.value, end=' ')

Example Tree

        A

       / \

      B   C

     / \   \

    D   E   F

root = Node('A')  
root.left = Node('B')  
root.right = Node('C')  
root.left.left = Node('D')  
root.left.right = Node('E')  
root.right.left = Node('F')

print('Inorder Traversal:')  
inorder(root)  
print('\nPreorder Traversal:')  
preorder(root)  
print('\nPostorder Traversal:')  
postorder(root)

## Output :

Preorder Traversal:

A B D E C F

Inorder Traversal:

D B E A C F

Postorder Traversal:

D E B F C A