**Abdull Haseeb**

**SU92-BSAIM-F24-050**

**BSAI-3A**

**dfs\_with\_stack (Depth First Search using Stack)**

**Function Name:**

dfs\_with\_stack(graph, start)

**Purpose:**

This function implements the **Depth First Search (DFS)** algorithm using an **explicit stack**.  
It traverses a graph and returns all nodes visited in the **DFS order**.

**Parameters:**

1. **graph (dict):**
   * The graph is stored as an adjacency list.
   * Each node (key) has a list (value) of its neighbors.

Example:

{

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

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

'C': ['F'],

'D': [],

'E': ['F'],

'F': []

}

1. **start (any type, usually string):**
   * The node from which the DFS traversal begins.

**Return Value:**

* **list:**  
  A list containing all the nodes visited during DFS, in the order they were visited.

**Working / Steps:**

1. **Visited Set:**
   * A set is used to keep track of nodes that have already been visited.
2. **Stack Initialization:**
   * A stack is initialized with the starting node.
   * DFS uses a stack because it follows the **LIFO (Last In, First Out)** principle.
3. **Traversal:**
   * While the stack is not empty:
     + Pop the top node from the stack.
     + If the node has not been visited:
       - Mark it as visited.
       - Add it to the result list.
4. **Neighbor Handling:**
   * Push all neighbors of the current node onto the stack **in reversed order**,  
     so that the leftmost neighbor is processed first.
   * Only unvisited neighbors are pushed.

**Example Execution:**

Graph:

A → [B, C]

B → [D, E]

C → [F]

D → []

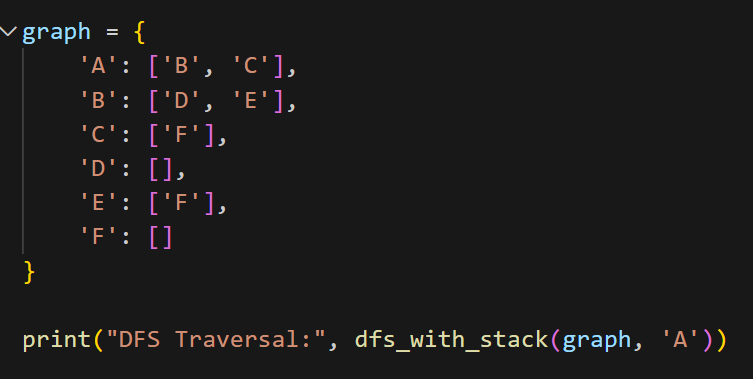
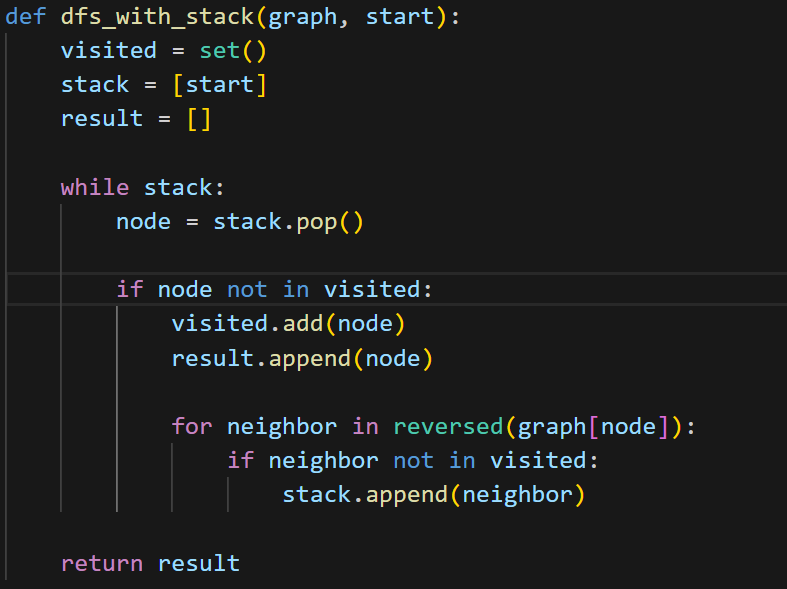
E → [F]

F → []

Start = 'A'

DFS Traversal Output:

DFS Traversal: ['A', 'B', 'D', 'E', 'F', 'C']



**Binary Tree Traversals (Preorder, Inorder, Postorder)**

**Class: Node**

* Represents a single node in a binary tree.
* **Attributes:**
  + value: Stores the data of the node.
  + left: Points to the left child node.
  + right: Points to the right child node.

**Traversal Functions:**

1. **Preorder Traversal (Root → Left → Right)**
   * Visit the root node first.
   * Traverse the left subtree recursively.
   * Traverse the right subtree recursively.
   * **Order Example:** A B D E C F
2. **Inorder Traversal (Left → Root → Right)**
   * Traverse the left subtree recursively.
   * Visit the root node.
   * Traverse the right subtree recursively.
   * **Order Example:** D B E A C F
3. **Postorder Traversal (Left → Right → Root)**
   * Traverse the left subtree recursively.
   * Traverse the right subtree recursively.
   * Visit the root node last.
   * **Order Example:** D E B F C A

**Tree Structure Used in Example:**

A

/ \

B C

/ \ \

D E F

**Example 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

