DATA STRUCTURE -BIT-EXERCISE NO:4

Project 30

Stack Questions:

Q3. Challenge: Show stack trace for ["X", "Y", "Z", "W"] with 2 pops

For stack follow LIFO, this means the last item pushed in the stack ("W") will be the first popped out.

Algorithm Design:

Step 1: Initialize an empty stack

```
stack = []  # We use a Python list to simulate a stack (LIFO)
```

Step 2: Push items onto the stack in the given order: X ➜ Y ➜ Z ➜ W

```
stack.append("X")  # Stack becomes: ['X']
stack.append("Y")  # Stack becomes: ['X', 'Y']
stack.append("Z")  # Stack becomes: ['X', 'Y', 'Z']
stack.append("W")  # Stack becomes: ['X', 'Y', 'Z', 'W']
```

Step 3: Show Initial Stack

```
print("\n📌 Stack after 1st pop:")
print(stack)  # Output: ['X', 'Y', 'Z']
```

Step 4: Perform First Pop Operation

```
stack.pop()  # Removes 'W'
```

Step 5: Perform Second Pop Operation

```
stack.pop()  # Removes 'Z'
```

Final Output

```
📌 Initial Stack:
['X', 'Y', 'Z', 'W']

📌 Stack after 1st pop:
['X', 'Y', 'Z']

📌 Stack after 2nd pop:
['X', 'Y']
```

DATA STRUCTURE -BIT-EXERCISE NO:4

Reflection: Why stack supports temporary storage of steps?

- A stack works on the **Last In, First Out (LIFO)** principle, meaning the most recent step is accessed first.
- This makes it ideal for situations where **recent actions need to be reversed or revisited** before earlier ones.
- It helps **temporarily store steps** in a way that preserves their order but allows accessing them in reverse when needed.
- Stacks are especially useful for **backtracking**, such as undoing actions, managing function calls, or navigating nested tasks.
- Since only the top item is accessible, stacks **simplify management of temporary data** by focusing on the most recent context.
- This ensures **controlled, reliable, and step-by-step processing** of actions, making stacks essential for both computing and logical workflows.