1. What is a Stack?

- A stack is a linear data structure following the **LIFO (Last In, First Out)** principle.
  - **LIFO** means the last element added to the stack is the first one to be removed.
- Think of it like a stack of plates; you add to the top and remove from the top.

2. Why Use a Stack?

- Provides a systematic way of managing data.
- Helps in scenarios requiring:
  1. Reversal (e.g., strings, numbers).
  2. Function calls (e.g., recursion).
  3. Matching problems (e.g., balanced parentheses).
- Optimized for specific operations like **push** and **pop**, ensuring constant-time complexity (**O(1)**).

3. How Does a Stack Work?

Key Operations

1. **Push**: Add an element to the top of the stack.
2. **Pop**: Remove the topmost element from the stack.
3. **Peek (or Top)**: Access the top element without removing it.
4. **isEmpty**: Check if the stack is empty.

4. Implementation of Stack

1. **Using Arrays**:
   - Fixed size; straightforward to implement.
   - Limitation: Size cannot be changed once defined.
2. **Using Linked List**:
   - Dynamic size; nodes are added/removed dynamically.
   - More flexible but requires additional memory for pointers.

5. Applications of Stack

1. **Expression Evaluation and Conversion**:
   - Solve postfix or prefix expressions.
2. **Backtracking**:
   - Used in maze-solving and puzzles.
3. **Recursion**:

    - Function call stack maintains order of calls.
4. **Undo/Redo**:
    - Common in text editors or graphical applications.

---

6. Benefits of Using Stack

- Provides a clean and efficient way to manage data where order matters.
- Simplifies solving problems that are recursive or hierarchical in nature.

---

7. Example: Reverse a String

Approach:

1. Push all characters of the string onto the stack.
2. Pop each character one by one to form the reversed string.
