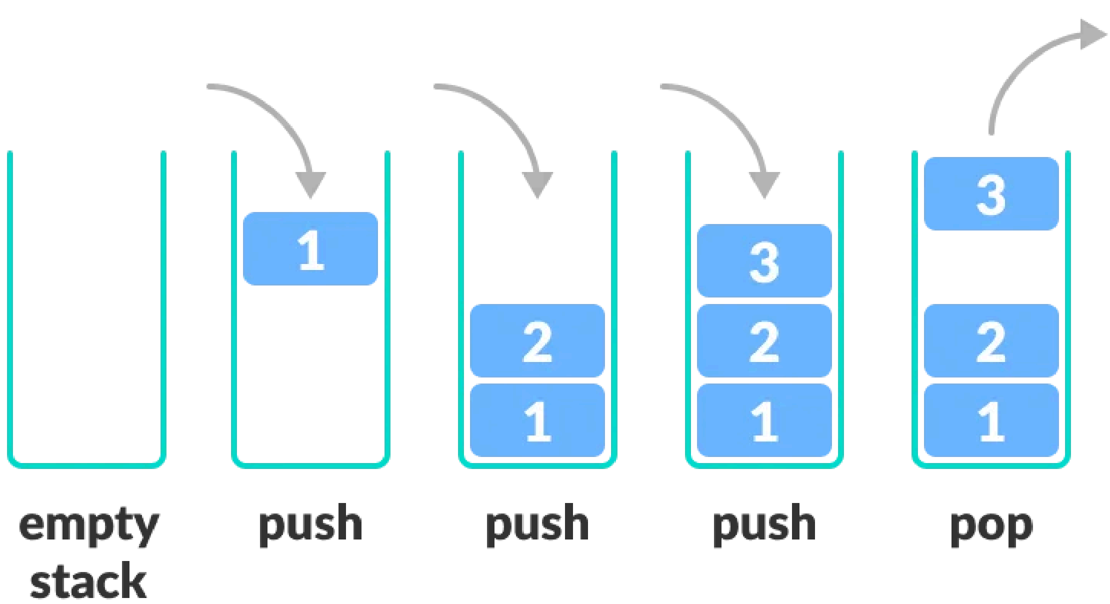
Summary of the Stack Data Structure

💡 A **stack** is a linear data structure that follows the **LIFO (Last In, First Out)** principle. This means that the **last** element added is the **first** one to be removed, just like a stack of plates.

🔷 Key Operations:

1. **Push(X)** → Adds an element **X** to the top of the stack.
2. **Pop()** → Removes and returns the **top** element of the stack.
3. **Peek()** → Returns the top element **without removing** it.
4. **IsEmpty()** → Checks if the stack is empty.
5. **Count()** → Returns the number of elements in the stack.



```
public void Push(T item)
{
    items.Add(item);
}

public T Pop()
{
    if (IsEmpty()) return default;
    T item = items[items.Count - 1];
    items.RemoveAt(items.Count - 1);
    return item;
}

public T Peek()
{
    if (IsEmpty()) return default;
    return items[items.Count - 1];
}

public int Count()
{
    return items.Count;
}

public bool IsEmpty()
{
    return items.Count == 0;
}
```

}

🔷 Real-World Examples:

- **Undo/Redo** in text editors.

- **Browser back/forward** history.
- **Function call stack** in programming languages.

🕹️ Use Case in Unity: Box Stacking Game

In our Unity project, we used a **custom stack** to manage stacking and removing boxes.

🔷 How We Used the Stack:

1. **Press "A"** → A new box **spawns** and is added to the stack (**Push**).
2. **Press "Space"** → The topmost box is **removed** (**Pop**).

🔷 Why Use a Stack Here?

- Ensures that the last box placed is the first one removed (**LIFO** behavior).
- Provides a structured way to manage stacked objects dynamically.
- Avoids complex position tracking—each new box stacks using `stack.Count()`.