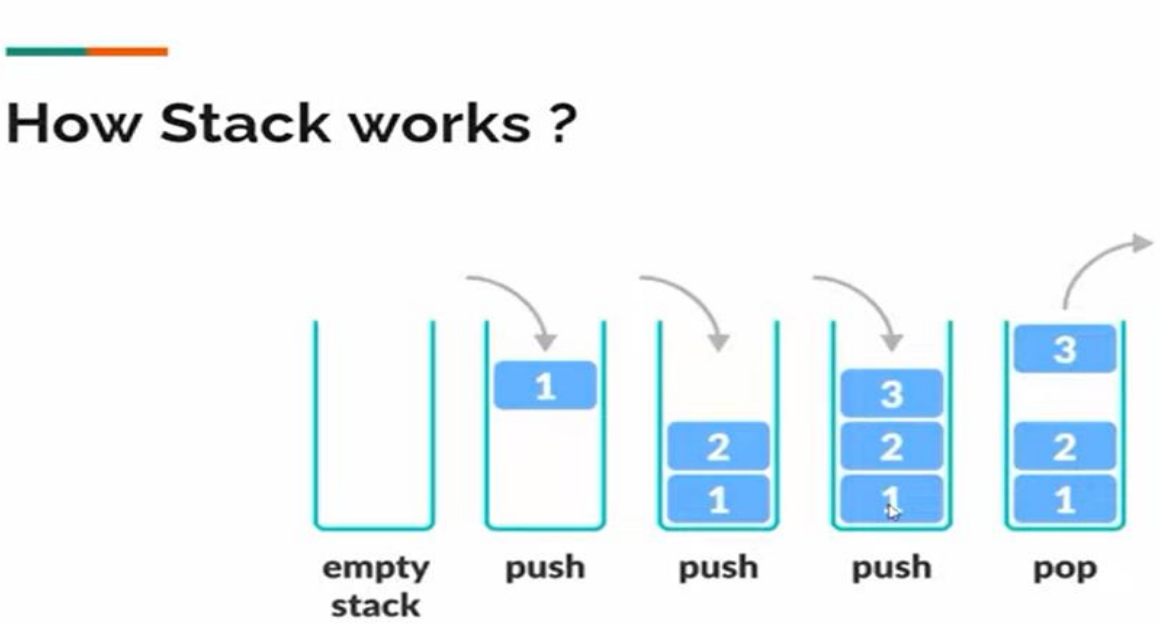
Stacks

A stack is a linear data structure that follows the Last-In-First-Out (LIFO) principle. It behaves like a stack of plates, where the last plate added is the first one to be removed.

Key Operations on Stack Data Structures

- **Push**: Adds an element to the top of the stack.
- **Pop**: Removes the top element from the stack.
- **Peek**: Returns the top element without removing it.
- **IsEmpty**: Checks if the stack is empty.
- **Size**: Finds the number of elements in the stack.



Applications of Stacks:

- **Function calls:** Stacks are used to keep track of the return addresses of function calls, allowing the program to return to the correct location after a function has finished executing.
- **Recursion:** Stacks are used to store the local variables and return addresses of recursive function calls, allowing the program to keep track of the current state of the recursion.
- **Expression evaluation:** Stacks are used to evaluate expressions in postfix notation (Reverse Polish Notation).
- **Syntax parsing:** Stacks are used to check the validity of syntax in programming languages and other formal languages.
- **Memory management:** Stacks are used to allocate and manage memory in some operating systems and programming languages.

Advantages of Stacks:

- **Simplicity:** Stacks are a simple and easy-to-understand data structure, making them suitable for a wide range of applications.
- **Efficiency:** Push and pop operations on a stack can be performed in constant time **(O(1))**, providing efficient access to data.
- **Last-in, First-out (LIFO):** Stacks follow the LIFO principle, ensuring that the last element added to the stack is the first one removed. This behavior is useful in many scenarios, such as function calls and expression evaluation.
- **Limited memory usage:** Stacks only need to store the elements that have been pushed onto them, making them memory-efficient compared to other data structures.

Disadvantages of Stacks:

- **Limited access:** Elements in a stack can only be accessed from the top, making it difficult to retrieve or modify elements in the middle of the stack.
- **Potential for overflow:** If more elements are pushed onto a stack than it can hold, an overflow error will occur, resulting in a loss of data.
- **Not suitable for random access:** Stacks do not allow for random access to elements, making them unsuitable for applications where elements need to be accessed in a specific order.
- **Limited capacity:** Stacks have a fixed capacity, which can be a limitation if the number of elements that need to be stored is unknown or highly variable.