**📘 Stack – Data Structure & Algorithm**

**🔹 What is a Stack?**

A **stack** is a linear data structure that follows the **LIFO** principle:

**LIFO** → Last In, First Out  
The last element added to the stack is the first one to be removed.

**🔹 Real-Life Examples:**

* A stack of plates
* Browser back button (history)
* Undo feature in editors (Ctrl + Z)
* Recursion call stack

**🔹 Basic Stack Operations:**

| **Operation** | **Description** |
| --- | --- |
| push() | Add an item to the top of the stack |
| pop() | Remove the top item from the stack |
| peek() / top() | View the top item without removing it |
| is\_empty() | Check if the stack is empty |
| size() | Returns the number of elements |

**🔹 Stack Representation in Memory**

* Implemented using **List** or **Linked List**
* Items are added and removed **only from the top**

**🔹 Stack Using Python List (Built-in)**

stack = []

# Push elements

stack.append(10)

stack.append(20)

# Pop element

stack.pop() # Removes 20

# Peek

top = stack[-1]

# Check if empty

is\_empty = len(stack) == 0

**🔹 Custom Stack Class in Python**

class Stack:

def \_\_init\_\_(self):

self.stack = []

def push(self, data):

self.stack.append(data)

def pop(self):

if not self.is\_empty():

return self.stack.pop()

return "Stack Underflow"

def peek(self):

if not self.is\_empty():

return self.stack[-1]

return "Stack is empty"

def is\_empty(self):

return len(self.stack) == 0

def size(self):

return len(self.stack)

def display(self):

print("Stack (top → bottom):", list(reversed(self.stack)))

**🔹 Example Usage:**

s = Stack()

s.push(10)

s.push(20)

s.push(30)

s.display() # Output: [30, 20, 10]

print(s.pop()) # Output: 30

print(s.peek()) # Output: 20

print(s.size()) # Output: 2

**🔹 Time Complexity**

| **Operation** | **Time Complexity** |
| --- | --- |
| Push | O(1) |
| Pop | O(1) |
| Peek | O(1) |
| is\_empty | O(1) |

**🔹 Applications of Stack**

✅ Used in:

* **Expression Evaluation** (Infix → Postfix)
* **Function Call/Recursion** tracking
* **Undo operations** in editors
* **Backtracking** problems (mazes, puzzles)
* **Balanced Parentheses Check**
* **Depth-First Search (DFS)** in Graphs

Let me know if you want extended notes on:

* Stack using Linked List
* Expression evaluation using Stack
* Real coding interview questions on Stacks