# **Execution Context & Call Stack in JavaScript**

## **What is an Execution Context?**

An **execution context** is an environment in which JavaScript code is evaluated and executed. There are two main types:

1. **Global Execution Context (GEC)**
   * Created when JavaScript starts execution.
   * The window object (in browsers) or global object (in Node.js) is created.
   * Variables and functions defined globally are stored here.
2. **Function Execution Context (FEC)**
   * Created whenever a function is invoked.
   * Each function call gets its own execution context.
   * A function’s execution context is removed once the function finishes execution.

## **Phases of Execution Context**

When JavaScript executes, it follows these steps:

### **1️⃣ Creation Phase (Memory Allocation)**

* The Global Execution Context is created.
* Variables declared with var are hoisted and assigned undefined.
* Function declarations are hoisted with their full definition.
* let and const variables are hoisted but remain in the **Temporal Dead Zone (TDZ)**.

### **2️⃣ Execution Phase**

* JavaScript starts executing line by line.
* Function calls create new **Function Execution Contexts**.
* Each function’s execution context is pushed onto the **Call Stack**.

## **Understanding the Call Stack**

The **Call Stack** keeps track of function calls. It follows the **LIFO (Last In, First Out)** principle.

### **Example 1: Basic Call Stack Flow**

function first() {

console.log("First function");

second();

}

function second() {

console.log("Second function");

third();

}

function third() {

console.log("Third function");

}

// Start execution

first();

### **Call Stack Process:**

1. first() is called → Added to Call Stack.
2. Inside first(), second() is called → Added to Call Stack.
3. Inside second(), third() is called → Added to Call Stack.
4. third() executes → Removed from Call Stack.
5. second() finishes execution → Removed from Call Stack.
6. first() finishes execution → Removed from Call Stack.

#### **Call Stack Visualization**

Stack Empty → first() → second() → third()

third() done → first() → second()

second() done → first()

first() done → Stack Empty

## **Example 2: Execution Context with Hoisting**

console.log(a); // undefined (hoisted)

var a = 10;

greet(); // "Hello!" (function is hoisted)

function greet() {

console.log("Hello!");

}

**Explanation:**

1. var a is hoisted with undefined.
2. greet() is hoisted with its full function definition.
3. Execution phase runs console.log(a) (prints undefined), then greet().

## **Best Practices**

✔️ Avoid deeply nested function calls to prevent **stack overflow**.  
✔️ Keep functions small and modular for easier debugging.