# **Scope & Closures in JavaScript**

## **Overview**

**Scope** refers to the context or environment in which a variable or function is declared and can be accessed. JavaScript uses function-based scope meaning variables declared inside a function are locally scoped in while variables declared outside any function have global scope.   
  
**Closures** occur in JavaScript when a function "remembers" its parent (lexical) scope even if it's executed outside that scope. In other words, a closure allows a function to access variables from its containing function and even after the containing function has finished executing.

## **1. Global vs. Local Scope**

* **Global Scope**: Variables declared outside any function are accessible anywhere in the code.
* **Local Scope**: Variables declared inside a function are only accessible inside that function.

### **Example:**

let globalVar = "I am global";

function testScope() {

let localVar = "I am local";

console.log(globalVar); // ✅ Accessible

console.log(localVar); // ✅ Accessible

}

testScope();

console.log(localVar); // ❌ ReferenceError: localVar is not defined

## **2. Function Scope vs. Block Scope**

* **Function Scope (var)**: Variables declared with var are scoped to the function.
* **Block Scope (let & const)**: Variables declared with let and const are scoped within {}.

### **Example:**

function scopeTest() {

if (true) {

var functionScoped = "I am function-scoped";

let blockScoped = "I am block-scoped";

}

console.log(functionScoped); // ✅ Works

console.log(blockScoped); // ❌ ReferenceError

}

scopeTest();

## **3. Lexical Scope & Closures**

* **Lexical Scope**: Inner functions can access variables from their outer function (parent scope).
* **Closures**: A function "remembers" variables from its outer scope even after the function has executed.

### **Example:**

function outerFunction() {

let outerVar = "Hello";

function innerFunction() {

console.log(outerVar); // ✅ Inner function remembers outerVar

}

return innerFunction;

}

const closureExample = outerFunction();

closureExample(); // Output: "Hello"

## **4. Local Scope Hoisting**

Variables are hoisted **only within the scope they are declared in**.

print();

console.log(name); // ReferenceError: name is not defined

function print() {

var name = "John"; // 'name' is hoisted but remains undefined inside the function scope

}

Here, we get a **ReferenceError: name is not defined**. Even though var is hoisted, it is only available **inside the function scope**, not outside.

Now, let’s check what happens inside the function:

print();

function print() {

console.log(name); // undefined

var name = "John";

}

Since name is declared **inside** print(), it is hoisted within that function, meaning it is **undefined** when we first try to log it.

**Best Practices**

✅ Use let and const to avoid unintentional variable leakage.  
✅ Keep global variables to a minimum for better code maintainability.   
✅ Use closures wisely for private data in JavaScript.