# Day 10 notes

| | |
|---|---|
| Key Takeaway | Mastering Scope and Scope chain |
| Learning Date | @July 5, 2025 |
| Module | **Module 2: Deep Dive into Functions & Objects** |
| Status | In progress |
| Topic | ✅ **Day 10: Closures Explained in Depth** (Practical examples) |
| Video Link | https://www.youtube.com/watch?v=14H2TsrjcLo&t=9s |

## *Scope in JavaScript :*

There are four types of scope in JS

- Global Scope
- Functional Scope
- Block Scope
- Module Scope

### Global scope: Variables declared outside of a function or scope are global in Scope.

### let, const are block scope, that's why they can't attach to global Scope(window object. Only var is available in the global scope as a window object.

### ✅
### **Note:** If a variable is declared inside a function in JavaScript, its lifetime is limited to that function only — it is not accessible from

## outside the function.

## Var is a function scope, and let and const are Block scope.

✅ **Note:**

- Variables declared in the **global scope** with `var`, `let`, or `const` are accessible everywhere in the code.
- Variables declared inside a **function scope** with `var`, `let`, or `const` are **not accessible outside** that function.
- Variables declared inside a **block scope** (like `{}` in `if`, `for`, `while`) with `let` or `const` are **not accessible outside** the block, but `var` **is accessible** outside the block.

✅ **Block Scope Note:**

- In JavaScript, a **block** is everything inside `{}` (like in `if`, `for`, `while`).
- Variables declared with `*let**` or `*const**` inside a block are **only accessible inside that block**.
- Variables declared with `*var**` inside a block are **not block-scoped** — they get hoisted to the function scope (or global scope if not inside a function).

```
// Scope chain

let globalVar = "I am a global variable";

function outer() {
  let outerVar = "I am an outer variable";

  function inner() {
    let innerVar = "I am an inner variable";
    console.log(innerVar);
    console.log(outerVar);
```

```
    console.log(globalVar);
  }
  inner();
}
outer();
```

📌 **Note on Variable Shadowing:**

Variable shadowing happens when a variable declared within a certain scope (like inside a function or block) has the same name as a variable in an outer scope. The inner variable **shadows** (overrides) the outer one **within its own scope**.

👉 **Priority follows the scope chain:**

- Inner (local) scope variables have higher priority than outer (parent/global) scope variables with the same name.
- When the inner scope ends, the outer variable is visible again.

```
// Variable shadowing

let message = "I am doing great";
function situation() {
  let message = "I am not doing great";
  console.log(message);
}
situation();
console.log(message);
```

## The Scope Table

### Comparison Table: `var` vs `let` vs `const`

| Feature | `var` | `let` | `const` |
|---|---|---|---|
| **Scope** | Function scope | Block scope `{}` | Block scope `{}` |
| **Hoisting** | Hoisted & initialized as `undefined` | Hoisted but in **Temporal Dead Zone (TDZ)** | Hoisted but in **Temporal Dead Zone (TDZ)** |
| **Attached to `window`?** | ✅ Yes | ❌ No | ❌ No |
| **Can be Re-declared?** | ✅ Yes | ❌ No | ❌ No |
| **Can be Reassigned?** | ✅ Yes | ✅ Yes | ❌ No |
| **Initial Value Required?** | ❌ No | ❌ No | ✅ Yes (Must be initialized) |
| **Mutability** | Mutable | Mutable | Immutable (Can't be reassigned but mutable if it's an object or array) |
| **Use in Loops** | Allowed but not recommended (function scope issues) | ✅ Recommended | ❌ Not recommended for changing values |