🧠 JavaScript - Scope (Master Notes with Tricks & Examples)

🔍 What is Scope?

Scope determines **where you can access a variable** in your code.

There are **three major types** of scopes in JavaScript:

1. 🌍 Global Scope
2. 🧱 Block Scope
3. 🧩 Function Scope

1️⃣ Global Scope

If a variable is declared **outside any {}**, it's in the **global scope**.

📦 Example:

js

CopyEdit

```
let a = 10;
const b = 20;
var c = 30;


console.log(a); // ✅ 10
console.log(b); // ✅ 20
console.log(c); // ✅ 30
```

🧠 Memory Trick:

"Declared outside, **global pride**. Use it anywhere, it's worldwide."

2️⃣ Block Scope

Anything declared **inside {}** — like in if, for, or any block — is block-scoped for let and const.

js

CopyEdit

```
if (true) {
  let a = 10;
```

```
  const b = 20;
  var c = 30;
}

console.log(a); // ❌ ReferenceError
console.log(b); // ❌ ReferenceError
console.log(c); // ✅ 30 -> (var ignores block scope!)
```

📊 **Keyword Comparison Table**:

Keyword	Scope Type	Accessible Outside Block?
let	Block	❌ No
const	Block	❌ No
var	Function/Global	✅ Yes (⚠️ risky!)

🔥 **Real-Life Analogy:**

- let & const → like **things in a drawer** — can't access from outside.
- var → like **stuff on a table** — visible from anywhere.

🛑 **Why var is Dangerous?**

Because it **ignores block scope**, it can accidentally **overwrite variables** or leak values, especially in loops or conditions.
✅ Always prefer let or const in modern JS.

3️⃣ Function Scope (Super Important!)

Variables defined inside a **function** are accessible **only within that function** and its inner functions.

js

CopyEdit

```
function one() {
  const username = "pranay";

  function two() {
```

```
    const website = "Youtube";
    console.log(username); // ✅ Accessible (closure)
  }

  // console.log(website); // ❌ Not accessible here
  two();
}

one();
```

🧠 **Trick**:

"Inner can access outer, but outer can't access inner."

This behavior is the **foundation of closures** in JavaScript.

💉 **Function Scope inside if:**

```
js
CopyEdit
if (true) {
  const username = "pranay";
  if (username === "pranay") {
    const website = " youtube";
    console.log(username + website); // ✅ Accessible
  }
  // console.log(website); ❌ Not accessible
}

// console.log(username); ❌ Not accessible
```

📌 **Note**: Only functions create their **own scope** (for var). Blocks create scope only for let and const.

✨ **Closure Alert! (Preview)**

Closures happen when **an inner function** accesses variables from **its outer function**, even after the outer function has finished running.

This is exactly what's happening in the above nested functions!
We'll study closures in detail later.

🚀 BONUS: Function Declaration vs Expression

js

CopyEdit

```
function addOne(num) {  // 👈 Function Declaration
  return num + 1;
}

const addTwo = function(num) {  // 👈 Function Expression
  return num + 2;
}

addOne(5); // ✅ Works even if called before declaration
addTwo(5); // ✅ Works ONLY if declared before calling
```

🧠 **Trick**:

- **Declaration** = Hoisted ⬆️ (You can call it before defining!)
- **Expression** = Not Hoisted ❌

📝 Summary Cheatsheet

Concept	Meaning
Scope	Where a variable can be accessed
Global Scope	Declared outside any block
Block Scope	Declared inside {} with let/const
Function Scope	Variables inside function, scoped locally
var	Function scoped, not block scoped
let / const	Block scoped, safer and modern

Concept	Meaning
Closure	Inner function "remembers" outer variables
Declaration	Hoisted
Expression	Not hoisted

🔥 **Pro Tip Pattern to Remember:**

📦 Scope = { Location + Keyword }

- let/const in {} = local/block
- var in {} = still global (if not inside a function!)
- Function = **isolated island** — has its own world