🎬 Episode 7: The Scope Chain, Scope & Lexical Environment 🔗

🌟 Scope & Lexical Environment in JavaScript

- 👉 In JavaScript, **Scope** is **directly related** to the **Lexical Environment**.
- Let's explore some examples to **understand it better**! 🚀

🧪 Case Studies:

⚡ Case 1:

```
function a() {
    console.log(b); // 10
}
var b = 10;
a();
```

- 🧠 Here, `function a` is **able to access** variable `b` from the **Global Scope**.

⚡ Case 2:

```
function a() {
    c();
    function c() {
        console.log(b); // 10
    }
}
var b = 10;
a();
```

- 🔥 Even inside a **nested function**, the **global variable** `b` is **accessible**!

⚡ Case 3:

```
function a() {
    c();
    function c() {
        var b = 100;
        console.log(b); // 100
    }
}
var b = 10;
a();
```

- 🎯 A **local variable** (`b = 100`) **overrides** the **global** `b = 10` inside the nested function `c()`.

⚡ Case 4:

```
function a() {
    var b = 10;
    c();
    function c() {
        console.log(b); // 10
    }
}
a();
console.log(b); // ❌ Error: b is not defined
```

- ⚡ **Functions** can access **outer variables**, but the **Global Execution Context ❌ can't access local variables** inside functions.

🧩 Quick Recap of Outputs:

Case	Output	Why?
1	10	`a()` accesses `b` from Global Scope
2	10	Nested function `c()` still accesses global `b`
3	100	Local `b` in `c()` overshadows global `b`
4	10 inside `c()`, ❌ Error outside	Global can't access local variables

---

🧠 Call Stack Visual

- `c()`
- `a()`
- `Global Execution Context (EC)`

🗂️ Memory Structure

```
Global Memory:
  - b: 10
  - a: {...}

a's Memory:
  - c: {...}

c's Memory:
  - b: 100 (only inside c if redeclared)
```

🔥 Important Concepts

- **Lexical Environment (LE)** = **Local Memory** + **Lexical Environment of Parent**. 🧠➕👨‍👩‍👧‍👦
- **Lexical** = 📜 "in hierarchy" / "in order" based on **physical placement** in code.
- Every time an **Execution Context (EC)** is created, a **Lexical Environment (LE)** is created too! 🔄

🔗 The Scope Chain (aka Lexical Environment Chain)

- When accessing a variable:
  1. Check local memory 🧠.
  2. If not found, check parent's memory 🔍.
  3. Repeat until found or reach the global level 🌍.

```javascript
function a() {
    function c() {
        // logic here
    }
    c(); // c is lexically inside a
}
// a is lexically inside Global EC
```

🌍 Lexical / Static Scope

- Variables and functions are **accessible** based on their **physical location** in the **source code**.
- Example:

  ```
  Global {
    Outer {
      Inner
    }
  }
  ```

- 🏡 `Inner` is **surrounded by** the **lexical scope** of `Outer`.

🎯 TL;DR

- ➡️ **Inner functions** can **access variables** from **outer functions** 💧.
- ➡️ **Functions cannot access variables** that are **not in their scope** ❌.
