🔍 Understanding **Scope** and **Lexical Environment** in JavaScript

---

☑️ Scope

- Scope is the area in the code where a variable is **declared** and can be **accessed**.
- JavaScript uses **lexical (static) scoping**, which means scope is determined **at the time of writing code**, not during execution.

☑️ Lexical Environment

A Lexical Environment is:

- **Local Memory** (where variables and functions of the current scope are stored)
- **Reference to the parent Lexical Environment**

This structure forms a **chain of environments**—known as the **Scope Chain**.

🔎 Examples and Explanation

🔹 Case 1

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

📌 `a()` is defined in the **global scope**, so it has access to `b` defined globally.

🔹 Case 2

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

📌 Even the **nested function** `c()` can access `b`, because it is **lexically** inside `a`, which is inside the **global scope** where `b` exists.

🔸 Case 3

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

📌 Here, `b` inside `c()` **shadows** the global `b`. It uses the **local `b = 100`**, so the output is `100`.

🔸 Case 4

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

📌 Function `c()` can access `b` from its **parent function** `a`.
But `b` is not available **globally**, hence the error.

🧠 Lexical Environment in Action

```
function outer() {
    var x = 10;
    function inner() {
        console.log(x); // ✅ Has access to x
    }
    inner();
}
outer();
```

🔁 This is because:

- `inner()` is **lexically inside** `outer()`

- So it has access to everything inside `outer()`'s lexical environment

---

🪜 Scope Chain

Whenever a variable is accessed:

1. JavaScript first looks in the **current function's memory**
2. If not found, it looks up to the **parent's memory**
3. This continues **until the global scope is reached**

🗃️ Summary

- **Lexical Environment = Local Memory + Parent Reference**
- **Scope Chain** is the path JavaScript follows when looking for variables.
- **Inner functions** have access to variables in **outer functions**
- The reverse is not true—outer functions **cannot** access inner function variables.
