Understanding **Shadowing** in JavaScript

What is Shadowing?

- When a **variable inside a block** or **function** has the **same name** as a variable outside, the inner one **shadows** (hides) the outer one.
- **For** `var`, shadowing can overwrite the outer variable because `var` is **function scoped**, not block scoped.
- **For** `let` **and** `const`, shadowing creates **separate, independent variables** due to **block scoping**.

Example: Shadowing with `var`, `let`, and `const`

```
var a = 100;

{
  var a = 10; // same name as global 'a'
  let b = 20;
  const c = 30;
  console.log(a); // 10
  console.log(b); // 20
  console.log(c); // 30
}

console.log(a); // 10 (global 'a' got overwritten!)
```

☑ Inside the block:

- `b` and `c` are block-scoped.
- `a` (declared with `var`) **overwrites** the global `a`!

Now with `let` and `const`

```
let b = 100;

{
  var a = 10;
  let b = 20;
  const c = 30;
  console.log(b); // 20
}

console.log(b); // 100 (no overwrite, both are separate!)
```

☑ Here, two `b` variables live independently:

- One inside the block `{}` (value 20).
- One outside globally (value 100).

The same behavior applies to `const` as well!

🚀 Shadowing Inside Functions

```
const c = 100;

function x() {
  const c = 10;
  console.log(c); // 10
}

x();
console.log(c); // 100
```

☑ Two different `c`s:

- Inside `x()`, `c = 10`.
- Globally, `c = 100`.

⚠ What is **Illegal Shadowing**?

```
let a = 20;

{
  var a = 20; // ❌ SyntaxError: Identifier 'a' has already been declared
}
```

- **We CANNOT shadow a `let` with a `var`.**
- But **we CAN shadow a `var` with a `let`.**

☑ Valid example:

```
var a = 20;

{
  let a = 30; // Valid shadowing
  console.log(a); // 30
}
```

```
console.log(a); // 20
```

⚡ Bonus: `var` with Functions

Because `var` is **function scoped**, shadowing with `var` inside a **function** is allowed:

```
let a = 20;

function x() {
  var a = 10; // ☑ Legal
}
```
