JavaScript: `var` vs `let` / `const`

1. `var` in Global Scope

- Declared with `var` outside of any function → becomes a property of the **global object** (`window` in browsers).
- Can overwrite existing globals and cause conflicts between scripts.

```
var myVar = 42;
console.log(window.myVar); // 42
```

---

2. `var` is Function-Scoped, not Block-Scoped

- `var` ignores block boundaries like `if` or `for` loops.
- Variables can leak outside of the block where they are defined.

```
if (true) {
  var x = 'hello';
}
console.log(x); // 'hello'
```

---

3. `let` / `const` in Global Scope

- Variables declared with `let` or `const` are **not properties of `window`**.
- Reduces risk of accidental overwrites.

```
let myLet = 42;
console.log(window.myLet); // undefined
```

---

4. Hoisting Differences

- `var` is hoisted **and initialized to** `undefined` → can cause hidden bugs.
- `let` / `const` are hoisted but **uninitialized** → accessing them before declaration causes a `ReferenceError`.

```
console.log(a); // undefined
var a = 10;

console.log(b); // ReferenceError
let b = 10;
```

---

5. Temporal Dead Zone (TDZ)

- The TDZ is the time between **entering scope** and the **variable declaration** where the variable cannot be accessed.
- Applies to `let` and `const`.
- Accessing the variable during TDZ causes a `ReferenceError`.

```
console.log(a); // ReferenceError
let a = 10;
console.log(a); // 10
```

Visual Timeline:

```
Scope Start ■■■ [ TDZ ] ■■■ Declaration ■■■ Initialization ■■■ Scope End
↑ Access here → ReferenceError
```

Summary Table

Feature	`var`	`let`	`const`
Scope	Function-scoped	Block-scoped	Block-scoped
Global	Adds to `window` (global)	Not added to window	Not added to window
Hoisting	Hoisted & initialized to `undefined`	Hoisted, but in TDZ	Hoisted, but in TDZ
Init	Optional	Optional	Required