Appendix D: `var` vs `let` in JavaScript

JavaScript has two main ways to declare variables: `var` (the traditional keyword) and `let` (introduced in ES6). While they seem similar, they behave quite differently.

✅ `let i = 3;`

Introduced in ES6 (2015)

- **Block-scoped** — confined to the nearest `{ ... }`
- **Not hoisted for use** (hoisted, but not initialized)
- **Cannot be re-declared** in the same block

```
{
  let i = 3;
  console.log(i); // 3
}
console.log(i); // ❌ ReferenceError: i is not defined
```

🔶 `var i = 3;`

Legacy style from pre-ES6 JavaScript

- **Function-scoped** — ignores block boundaries
- **Hoisted** — declaration moved to top of function or global scope
- **Can be re-declared** in the same scope

```
{
  var i = 3;
  console.log(i); // 3
}
console.log(i); // ✅ Still 3 — var is not block-scoped
```

🧠 Hoisting Behavior

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

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

- `var` declarations are hoisted and initialized as `undefined`

- `let` declarations are hoisted but not initialized — the variable is in a **Temporal Dead Zone** until its definition is evaluated

---

🧪 Re-declaration

```
var a = 1;
var a = 2; // ✅ Allowed

let b = 1;
let b = 2; // ❌ SyntaxError: Identifier 'b' has already been declared
```

📌 Summary

Feature	`var`	`let`
Scope	Function scope	Block scope
Hoisted	Yes (initialized to `undefined`)	Yes (but not initialized)
Re-declarable	Yes	No (in same block)
Temporal Dead Zone	No	Yes
Preferred?	❌ No (legacy)	✅ Yes (modern JS)

🧠 When to Use `let`

Always use `let` in modern JavaScript unless you have a reason to use `const` (for immutable bindings). Avoid `var` unless maintaining legacy code.