`let` vs `var` vs `const` in JavaScript: In-Depth Comparison

JavaScript provides three ways to declare variables: `var`, `let`, and `const`. While they may look similar, they behave differently when it comes to **scope**, **hoisting**, **reassignment**, and **redeclaration**.

1. Scope

Keyword	Scope Type	Example
`var`	Function scope (NOT block)	Visible inside the function, or globally if declared outside a function.
`let`	Block scope	Only accessible inside the nearest enclosing curly braces `{}`.
`const`	Block scope	Only accessible inside the nearest enclosing curly braces `{}`.

- **Block scope**: Defined by `{ }` (like inside `if`, `for`, or block).
- **Function scope**: Defined by `function { }` only.

Example:

```
if (true) {
    var a = 1;
    let b = 2;
    const c = 3;
}
console.log(a); // 1      -> var is function/global scoped
console.log(b); // Error  -> let is block scoped
console.log(c); // Error  -> const is block scoped
```

2. Hoisting

Hoisting refers to JS moving variable/function declarations to the top of the scope before code execution.

- `var` declarations are hoisted and initialized as `undefined`.
- `let` and `const` declarations are hoisted but **NOT initialized**; they live in a "temporal dead zone" (TDZ) until the line they're declared.

Example:

```
console.log(x); // undefined
var x = 5;

console.log(y); // ReferenceError: Cannot access 'y' before initialization
let y = 10;

console.log(z); // ReferenceError: Cannot access 'z' before initialization
const z = 15;
```

- Using `let` or `const` before declaration causes an error.
- Using `var` before declaration gives `undefined`.

3. Redeclaration

Keyword	Redeclaration Allowed in Same Scope?
`var`	Yes
`let`	No
`const`	No

Example:

```
var a = 1;
var a = 2;    // No error

let b = 1;
let b = 2;    // SyntaxError

const c = 1;
const c = 2;  // SyntaxError
```

4. Reassignment

Keyword	Can be Reassigned?
`var`	Yes
`let`	Yes
`const`	No (value is fixed)

But remember, for `const`, only the *variable binding* is constant. If the constant holds an object or array, properties/items can still be changed.

Example:

```
let a = 1;
a = 2;     // OK
```

```
const b = 1;
b = 2;     // Error: Assignment to constant variable

const arr = [1, 2];
arr.push(3); // OK: array contents can be changed
```

5. Global, Function (Local), and Block Scope

- **Global scope:** Declared outside functions or blocks; accessible anywhere.
- **Function (local) scope:** Declared within a function; accessible only inside that function.
- **Block scope:** Only for `let` and `const`; accessible within the block `{}`.

```
function example() {
    if (true) {
        var x = 1;  // function-scoped
        let y = 2;  // block-scoped
        const z = 3; // block-scoped
    }
    console.log(x); // 1
    console.log(y); // Error
    console.log(z); // Error
}
```

Summary Table

Keyword	Scope Type	Hoisted?	Initialized before code runs?	Redeclaration Allowed?	Reassignment Allowed?
`var`	Function/Global	✔	✔ (as undefined)	✔	✔
`let`	Block	✔	✖ (TDZ until declaration)	✖	✔
`const`	Block	✔	✖ (TDZ until declaration)	✖	✖

Key Differences in Practice

- **Prefer `let` and `const`:** Always use `let` and `const` in modern JavaScript for safer, more predictable code.
- **Use `const` by default:** Opt for `const` for any variable that does not need reassignment; use `let` only when necessary.
- **Avoid `var`:** Its function scoping and hoisting can cause confusing bugs.

In summary:

- `var`: function-scoped, hoisted as undefined, can be redeclared and reassigned; avoid in modern JS.
- `let`: block-scoped, hoisted but not initialized (TDZ), cannot be redeclared, can be reassigned.
- `const`: block-scoped, hoisted but not initialized (TDZ), cannot be redeclared, cannot be reassigned (but object/array contents can change).