JavaScript Variables and Scopes – Complete Guide for Beginners & Interviews

---

📘 1. What Are Variables in JavaScript?

A **variable** is a container used to store data values. Think of it like a box with a name — you can put something inside (a value), change it, or read it later.

Example:

```
let name = "Alice";
let age = 25;
console.log(name);  // Output: Alice
console.log(age);   // Output: 25
```

Variables allow us to reuse and manipulate data easily in programs.

🧰 2. Types of Variables in JavaScript

JavaScript provides **three ways** to declare variables:

1. **var** – The old way (function-scoped)
2. **let** – The modern way (block-scoped)
3. **const** – For constants (block-scoped and cannot be reassigned)

Let's understand each in detail.

⚙️ 3. `var` Keyword

Definition:

- Introduced in **ES5 (before 2015)**.
- Has **function scope** or **global scope** (not block scope).
- Can be **redeclared** and **updated**.
- Gets **hoisted** (moved to the top of scope) with an initial value of `undefined`.

Example:

```
var x = 10;
var x = 20; // ✅ Redeclaration allowed
```

```
x = 30;     // ✅ Update allowed
console.log(x); // Output: 30
```

Function Scope Example:

```
function test() {
  var a = 10;
  if (true) {
    var a = 20; // Same variable (no block scope)
    console.log(a); // Output: 20
  }
  console.log(a); // Output: 20
}
test();
```

⚙️4. `let` Keyword

Definition:

- Introduced in **ES6 (2015)**.
- Has **block scope** (inside `{}`).
- Can be **updated**, but **not redeclared** in the same scope.
- Also **hoisted**, but **not initialized** (Temporal Dead Zone).

Example:

```
let y = 10;
y = 15;      // ✅ Allowed
// let y = 20; ❌ Error: redeclaration not allowed
console.log(y);
```

Block Scope Example:

```
function testLet() {
  let b = 10;
  if (true) {
    let b = 20; // Different variable (block-scoped)
    console.log(b); // Output: 20
  }
  console.log(b); // Output: 10
}
testLet();
```

⚙️5. `const` Keyword

Definition:

- Also **block-scoped**.
- Cannot be **redeclared** or **updated**.
- Must be **initialized** when declared.

Example:

```
const PI = 3.14159;
// PI = 3.14; ❌ Error: reassignment not allowed
console.log(PI); // Output: 3.14159
```

> ⚠️**Note:** For objects and arrays declared with `const`, you can **modify their contents**, but not reassign the variable itself.

Example:

```
const user = { name: "Alice" };
user.name = "Bob"; // ✅ Allowed
// user = { name: "Charlie" }; ❌ Error
console.log(user.name); // Output: Bob
```

📦6. Variable Hoisting

Hoisting means moving declarations to the top of the current scope during compilation.

Example with `var`:

```
console.log(a); // Output: undefined
var a = 5;
```

Internally, JavaScript does this:

```
var a;
console.log(a);
a = 5;
```

Example with `let` and `const`:

```
console.log(b); // ❌ ReferenceError (Temporal Dead Zone)
let b = 10;
```

🕐7. JavaScript Scopes

Scope defines **where** a variable can be accessed.

Types of Scope:

1. **Global Scope**
2. **Function (Local) Scope**
3. **Block Scope**

🧹7.1 Global Scope

A variable declared **outside any function or block** is globally scoped.

```
var globalVar = "I am global";

function showGlobal() {
  console.log(globalVar); // Accessible inside function
}

showGlobal();
console.log(globalVar); // Accessible globally
```

⚙️7.2 Function Scope

Variables declared **inside a function** are accessible **only within that function**.

```
function testFunc() {
  var localVar = "I am local";
  console.log(localVar); // Accessible here
}

// console.log(localVar); ❌ Error: localVar is not defined
```

📦7.3 Block Scope

Block scope means a variable declared inside `{}` can only be accessed inside that block.

```
if (true) {
  let blockVar = "Inside block";
  console.log(blockVar); // Works
```

```
}
// console.log(blockVar); ❌ Error
```

`let` and `const` respect block scope, but `var` ignores it.

🕐8. Variable Scopes and Behavior Comparison

Feature	var	let	const
Scope	Function	Block	Block
Redeclaration	✅Allowed	❌Not allowed	❌Not allowed
Reassignment	✅Allowed	✅Allowed	❌Not allowed
Hoisting	✅Yes (undefined)	✅(TDZ)	✅(TDZ)
Initialization Required	❌No	❌No	✅Yes

---

9. Common Interview Questions and Answers

Q1: What is the difference between `var`, `let`, and `const`?

A: - `var`: Function-scoped, can be redeclared and reassigned. - `let`: Block-scoped, can be reassigned but not redeclared. - `const`: Block-scoped, cannot be redeclared or reassigned.

Q2: What is hoisting in JavaScript?

A: Hoisting moves variable declarations to the top of their scope. However, `let` and `const` remain in the **Temporal Dead Zone (TDZ)** until initialized.

Q3: What is a block scope?

A: Variables declared inside `{}` using `let` or `const` are accessible only within that block.

Q4: Can you change a `const` object's properties?

A: Yes, but you cannot reassign the variable.

```
const person = { name: 'John' };
person.name = 'Doe'; // ✅ Allowed
// person = { name: 'Jane' }; ❌ Error
```

Q5: What will be the output?

```
var a = 10;
if (true) {
  var a = 20;
}
console.log(a);
```

✅Output: `20` (since `var` is function-scoped, not block-scoped)

Q6: What will be the output?

```
let b = 10;
if (true) {
  let b = 20;
}
console.log(b);
```

✅Output: `10` (different `b` inside block)

Q7: What happens if you use a variable before declaring it?

```
console.log(x);
let x = 5;
```

❌Output: ReferenceError (because of TDZ)

Q8: Which keyword should you use mostly?

A: Prefer `const` by default. Use `let` when you need to reassign. Avoid `var` in modern code.

🧰10. Practice Exercises

Exercise 1

Predict the output:

```
var x = 5;
function test() {
  var x = 10;
  if (true) {
    var x = 20;
    console.log(x);
  }
  console.log(x);
}
```

```
test();
console.log(x);
```

Exercise 2

Predict the output:

```
let x = 5;
function test() {
  let x = 10;
  if (true) {
    let x = 20;
    console.log(x);
  }
  console.log(x);
}
test();
console.log(x);
```

🔗 Summary

- Use `let` and `const` instead of `var`.
- Understand **scopes**: global, function, block.
- Learn **hoisting** and **TDZ**.
- Practice examples to understand behavior in different scopes.

⬜ **Tip:** Always use `const` unless you know you'll need to reassign the variable.

Would you like me to include **real-world examples** (like variable usage in loops, functions, DOM manipulation, and APIs) and **advanced interview questions** next?