JavaScript learning path

Perfect 🚀 Let's go **deep into** `var` — its **history, internals, quirks, optimizations, and interview brain-benders**.

🔎 `var` in JavaScript (Deep Dive)

---

1. Background

- `var` is the **original way** to declare variables in JavaScript (since 1995).
- Still works, but **modern JS prefers** `let` **and** `const` because `var` has **quirks** that cause bugs.
- In interviews, you'll often be asked about `var` vs `let` / `const`.

---

2. Core Properties of `var`

✅ a) Function-scoped (not block-scoped)

```js
function test() {
  if (true) {
    var x = 10;
  }
  console.log(x); // ✅ 10 (var escapes block!)
}
test();
```

- Unlike `let` and `const`, which are **block-scoped**, `var` ignores `{}` blocks except function blocks.

✅ b) Hoisting

- Variables declared with `var` are **hoisted** to the **top of their scope** (function or global).
- But **only the declaration is hoisted, not the initialization.**

```js
console.log(a); // undefined (hoisted but not assigned)
var a = 5;
console.log(a); // 5
```

🔍 Internally, JS engine transforms code like this:

```js
var a;          // declaration hoisted
console.log(a);  // undefined
a = 5;          // assignment happens here
console.log(a);  // 5
```

✅ c) Can be re-declared

```js
var x = 1;
var x = 2; // ✅ No error
console.log(x); // 2
```

- Unlike `let` / `const` which throw errors on redeclaration.

✅ d) Attached to `window` (in browsers)

```js
var name = "Anoop";
console.log(window.name); // "Anoop" ✅
```

- `var` creates **properties on the global object** (`window` in browsers, `global` in Node).
- `let` and `const` do not.

✅ e) Default value is `undefined`

```js
```

```
var y;
console.log(y); // undefined
```

---

3. ⚡ Common Pitfalls (Interview Traps)

❌ a) Block leakage

```js
for (var i = 0; i < 3; i++) {
  setTimeout(() => console.log(i), 1000);
}
// Output: 3, 3, 3 (not 0, 1, 2!)
```

Why?

- `var` is **function-scoped**, so the same `i` is shared across all loop iterations.

👉 Fix with `let`:

```js
for (let i = 0; i < 3; i++) {
  setTimeout(() => console.log(i), 1000);
}
// Output: 0, 1, 2
```

❌ b) Redeclaration bugs

```js
var score = 50;
if (true) {
  var score = 100; // same variable!
}
console.log(score); // 100 (oops, overwritten)
```

❌ c) Hoisting confusion

```js
console.log(msg); // undefined, not error
var msg = "Hello";
```

👉 Developers expect an error, but `var` silently returns `undefined`.

---

4. 🧠 Interview Brain-Benders (with Solutions)

Q1:

```js
var a = 1;
function test() {
  console.log(a);
  var a = 2;
}
test();
```

Answer: `undefined`

- Inside `test()`, `var a` is **hoisted** → local `a` shadows global `a`.
- Declaration hoisted but not initialized, so first `console.log(a)` is `undefined`.

Q2:

```js
var x = 5;
(function() {
  console.log(x); // ?
  var x = 10;
})();
```

Answer: `undefined`

- Same reasoning: hoisting creates local `x`.

Q3:

```js
if (true) {
  var foo = "bar";
}
console.log(foo); // ?
```

Answer: `"bar"` ✅

- `var` ignores block scope.

---

5. 🔧 Real-World Optimizations & Best Practices

- Avoid `var` in **modern code**; use `let` / `const`.
- If you must use `var` (e.g., legacy projects):
  1. Always declare at the **top of the function** (to avoid hoisting confusion).
  2. Use **IIFE** (Immediately Invoked Function Expressions) to simulate block scope:

```js
(function() {
  var temp = "isolated";
  console.log(temp); // isolated
})();
console.log(temp); // ReferenceError
```

---

6. ⚙️ Hidden Internals

- `var` variables live in the **Variable Environment Record** of the Execution Context.
- On compilation phase:
  - JS engine scans and hoists all `var` declarations.
  - Assignments happen in runtime.

This is why **Temporal Dead Zone (TDZ)** does NOT exist for `var`, but exists for `let` / `const`.

✅ That's the **complete deep dive of** `var`.

Shall I now give you a **side-by-side comparison of** `var`, `let`, **and** `const` **with internals + performance trade-offs**, before we move to `let` individually?