JavaScript learning path

Perfect 🔥 Let's create a **side-by-side comparison of `var`, `let`, and `const`.**
We'll look at **rules, internals, behaviors, corner cases, and tricky interview questions** in one place (like a cheatsheet + deep explanations).

⚔️ `var` vs `let` vs `const` in JavaScript

---

1. Scope

Feature	`var`	`let`	`const`
Scope type	**Function-scoped**	**Block-scoped**	**Block-scoped**

Example

```js
if (true) {
  var x = 10;
}
console.log(x); // 10 ✅
```

|

```js
if (true) {
  let y = 20;
}
console.log(y); // ❌ ReferenceError
```

|

```js
if (true) {
  const z = 30;
}
console.log(z); // ❌ ReferenceError
```

---

2. Hoisting

Feature	`var`	`let`	`const`
Hoisted?	✅ Yes	✅ Yes	✅ Yes
Initialized before declaration?	✅ `undefined`	❌ No (TDZ)	❌ No (TDZ)

Example

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

|

```js
console.log(b); // ❌ ReferenceError (TDZ)
let b = 5;
```

|

```js
console.log(c); // ❌ ReferenceError (TDZ)
const c = 5;
```

---

3. Redeclaration & Reassignment

Feature	`var`	`let`	`const`
Redeclare in same scope?	✅ Allowed	❌ Error	❌ Error

Feature	`var`	`let`	`const`
Reassign value?	✅ Allowed	✅ Allowed	❌ Not allowed

Example

```js
var x = 1;
var x = 2; // ✅ works
```

|

```js
let y = 1;
let y = 2; // ❌ SyntaxError
```

|

```js
const z = 1;
z = 2;    // ❌ TypeError
```

---

4. Global Object Binding

Feature	`var`	`let`	`const`
Added to `window` (in browser)?	✅ Yes	❌ No	❌ No

Example

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

|

```js
```

```js
let name = "JS";
console.log(window.name); // ❌ undefined
```

|

```js
const name = "JS";
console.log(window.name); // ❌ undefined
```

---

5. Loop Behavior

Feature	`var`	`let`	`const`
Same variable reused in loop	✅ Yes (buggy)	❌ No	❌ No

Example

```js
for (var i = 0; i < 3; i++) {
  setTimeout(() => console.log(i), 100);
}
// 3, 3, 3
```

|

```js
for (let i = 0; i < 3; i++) {
  setTimeout(() => console.log(i), 100);
}
// 0, 1, 2
```

|

```js
for (const i = 0; i < 3; i++) {
  // ❌ TypeError: i can't be reassigned
}
```

---

6. Objects & Arrays

Feature	`var`	`let`	`const`
Object mutation	✅ Allowed	✅ Allowed	✅ Allowed
Reassignment of object/array	✅ Allowed	✅ Allowed	❌ Not allowed

Example

```js
const obj = {a:1};
obj.a = 2; // ✅ allowed
obj = {b:3}; // ❌ TypeError
```

---

7. Interview Brain-Benders

Q1: What’s the output?

```js
function test() {
  console.log(a);
  var a = 10;
}
test();
```

✅ **Output:** `undefined`

Reason: `var` hoisted + initialized as `undefined`.

Q2:

```js
```

```js
function test() {
  console.log(b);
  let b = 20;
}
test();
```

❌ **Output:** ReferenceError (TDZ).

Q3:

```js
const obj = { x: 1 };
obj.x = 2;
console.log(obj.x);
```

✅ **Output:** `2` (mutation allowed).

Q4:

```js
for (var i = 0; i < 2; i++) {
  setTimeout(() => console.log(i), 0);
}
```

✅ **Output:** `2, 2`

Reason: Same `i` reused.

Q5:

```js
for (let i = 0; i < 2; i++) {
  setTimeout(() => console.log(i), 0);
}
```

✅ **Output:** `0, 1`

Reason: New block scope per iteration.

---

8. Best Practices

- ✅ Use `const` by default.
- ✅ Use `let` if variable must change.
- ❌ Avoid `var` unless maintaining legacy code.
- ✅ For true immutability: use `Object.freeze(obj)` or immutability libraries.

📌 Final Rule of Thumb

- `var` = old, buggy, avoid.
- `let` = for mutable values.
- `const` = for default use.

Would you like me to now create a **visual execution flow diagram** (showing how `var`, `let`, and `const` behave during hoisting + TDZ step by step in memory) so you can see internals more clearly?