JavaScript learning path

Perfect 👍 Let's now do a **deep dive into** `const` — the most commonly used way of declaring variables in modern JavaScript.

We'll again go step by step: **rules → examples → pitfalls → internals → interview traps → best practices**.

🔎 `const` in JavaScript (Deep Dive)

---

1. Background

- Introduced in **ES6 (2015)** alongside `let`.
- Designed for variables that should **not be reassigned**.
- Encourages **immutability** (but only partially — we'll uncover why).

---

2. Core Rules of `const`

✅ a) Must be initialized at declaration

```
js

const x;     // ❌ SyntaxError: Missing initializer
const y = 10; // ✅ Works
```

✅ b) Block-scoped

```
js

if (true) {
  const a = 5;
}
console.log(a); // ❌ ReferenceError
```

- Same block scope behavior as `let`.

✅ c) No Re-declaration

```js
const a = 10;
const a = 20; // ❌ SyntaxError
```

✅ d) No Reassignment

```js
const pi = 3.14;
pi = 3.14159; // ❌ TypeError
```

- Cannot change the binding (reference).

✅ e) Hoisted but in TDZ

```js
console.log(num); // ❌ ReferenceError
const num = 100;
```

- Just like `let`, exists in **TDZ** until declared.

---

3. ⚠ The Tricky Part — Const Objects & Arrays

Even though the **reference** is fixed, the **contents** can change!

✅ a) Objects

```js
const person = { name: "Anoop" };
person.name = "Rahul"; // ✅ allowed
console.log(person.name); // Rahul

person = { name: "Vikas" }; // ❌ TypeError (reassignment)
```

✅ b) Arrays

```js
const arr = [1, 2, 3];
arr.push(4);     // ✅ allowed
console.log(arr); // [1,2,3,4]

arr = [5, 6]; // ❌ TypeError
```

👉 `const` freezes the *reference*, **not the value inside**.
To make the object truly immutable, use `Object.freeze()`.

---

4. 🧠 Interview Traps

Q1: What happens?

```js
const a = 10;
{
  const a = 20;
  console.log(a);
}
console.log(a);
```

Answer:

- Inside block → `20`
- Outside block → `10`
  (Block scope shadowing still works)

Q2:

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

Answer: `2` ✅ (object mutated, reference unchanged)

Q3:

```js
const obj = { x: 1 };
obj = { y: 2 };
```

Answer: ❌ TypeError — reassignment not allowed.

---

5. ⚙️ Internals

- During **compilation**, `const` is stored in **Lexical Environment Record**.
- Behaves like `let` (block-scoped, TDZ).
- Only difference → binding is **immutable** (cannot be reassigned).

---

6. 🔧 Best Practices

- ✅ Use `const` by **default**.
- ✅ Switch to `let` only if reassignment is truly required.
- ❌ Avoid `var` in modern code.
- ✅ For objects/arrays, if true immutability is required → use libraries (Immutable.js) or `Object.freeze()`.

📌 Summary

- `const` = safer than `let` and `var`.
- Reference can't be changed, but values inside objects/arrays can.
- Default choice in modern JavaScript for variable declaration.

👉 Do you want me to now create a **side-by-side comparison (var vs let vs const)** with tricky code examples and outputs (like a cheatsheet), or should we move on to the **next JS topic** after variables?