JavaScript learning path

Perfect 👍 Let's **move to the next part of destructuring** —
👉 **Special Use Cases & Gotchas** that often confuse even experienced developers.

🎯 Special Use Cases of Destructuring

1. Swapping Variables

Classic trick with array destructuring:

```js
let a = 1, b = 2;
[a, b] = [b, a];
console.log(a, b); // 2, 1
```

✅ Internally: engine just makes a temporary array and reassigns.

---

2. Destructuring Strings

Strings are iterable → destructuring works:

```js
const [x, y, z] = "hey";
console.log(x, y, z); // h e y
```

Also works with object style (string has properties):

```js
const { length } = "hello";
console.log(length); // 5
```

---

3. Destructuring in Loops

Useful for iterating over objects/arrays:

```js
const users = [
  { id: 1, name: "Alice" },
  { id: 2, name: "Bob" }
];

for (const { id, name } of users) {
  console.log(id, name);
}
```

✅ Cleaner than `user.id`, `user.name` inside loop.

---

4. Setting Defaults in Function Params

```js
function connect({ host = "localhost", port = 80 } = {}) {
  console.log(host, port);
}

connect(); // localhost 80
connect({ host: "127.0.0.1" }); // 127.0.0.1 80
```

⚠ Without the `= {}` default, calling `connect()` would crash (`Cannot destructure undefined`).

---

5. Ignoring Values with Commas

```js
const [a, , b] = [1, 2, 3];
console.log(a, b); // 1 3
```

Or with objects:

```js
const { x, ...rest } = { x: 1, y: 2, z: 3 };
console.log(rest); // { y: 2, z: 3 }
```

6. Destructuring with Dynamic Keys

```js
const key = "age";
const { [key]: userAge } = { name: "Sam", age: 25 };
console.log(userAge); // 25
```

✅ Handy when property names are not known until runtime.

---

7. Nested Default Values

```js
const { profile: { city = "Unknown" } = {} } = {};
console.log(city); // "Unknown"
```

✅ Avoids `Cannot read property of undefined` error.

⚠ Gotchas

1. **Primitive Destructuring**

```js
const { toString } = 123;
console.log(toString); // function from Number.prototype
```

✅ Works because primitives are auto-boxed (wrapped in objects).

---

2. **Invalid Left-Hand Side**

```js
({ a } = { a: 10 }); // ✅ ok
{ a } = { a: 10 };   // ❌ SyntaxError
```

⚠ Why? → Without `()`, JS thinks `{}` starts a block, not destructuring.

3. Duplicate Variables

```js
const { a, a } = { a: 1 };
// ❌ SyntaxError: Duplicate declaration
```

✅ So, destructuring is **very flexible**, but pitfalls appear when mixing with defaults, nested structures, or unusual left-hand assignments.

👉 Do you want me to now go **deeper into "Performance Pitfalls & Best Practices"** of destructuring (where it can hurt speed or readability), or move on to **next ES6 feature (like Spread & Rest operators)**?