JavaScript learning path

Perfect 👌 you’ve brought in the **Nullish Coalescing Operator (`??`)** topic.
This is one of the most misunderstood operators in JavaScript, so let’s do a **deep-dive with reasoning, hidden internals, pitfalls, and interview tricks**.

🔎 Nullish Coalescing Operator (`??`) in Depth

---

1. What It Really Does

- `a ?? b` means:
  - If `a` **is not** `null` **or** `undefined`, return `a`.
  - Otherwise, return `b`.

👉 Think of it as: *“Give me the first defined value.”*

```js
let user = null;
console.log(user ?? "Anonymous"); // "Anonymous"
```

Equivalent to:

```js
let result = (a !== null && a !== undefined) ? a : b;
```

---

2. Why Was It Added?

Before `??`, people used `||` (OR) to provide defaults:

```js
let height = 0;

console.log(height || 100); // 100 ❌ (oops, overwrote 0)
console.log(height ?? 100); // 0 ✅
```

- `||` treats **falsy values** (`false, 0, "", NaN`) as if they don’t exist.
- `??` treats only `null` **and** `undefined` as missing.

💡 **Use `??` when you want to keep falsy-but-valid values (like `0` or `""`).**

---

3. Typical Use Cases

✅ Defaulting user input:

```js
let username = inputName ?? "Guest";
```

✅ Chaining multiple fallbacks:

```js
let name = firstName ?? lastName ?? nickName ?? "Anonymous";
```

✅ Config parameters:

```js
function createUser({age, city} = {}) {
  let finalAge = age ?? 18;  // if age is missing, set to 18
  let finalCity = city ?? "Unknown";
  return {finalAge, finalCity};
}
```

---

4. Precedence & Parentheses

- Precedence of `??` = **3** (same as `||`).
- Lower than arithmetic `+`, `*`.
- Higher than assignment `=` and ternary `?:`.

Example:

```js
let h = null, w = null;
let area = (h ?? 100) * (w ?? 50); // ✅ 5000
```

⚠ Without parentheses:

```js
let area = h ?? 100 * w ?? 50;
// = h ?? (100 * w) ?? 50
```

---

5. Safety Restriction with `&&` and `||`

To avoid ambiguity, JS **forbids mixing `??` with `||` or `&&` without parentheses.**

```js
let x = 1 && 2 ?? 3; // ❌ SyntaxError
let y = (1 && 2) ?? 3; // ✅ 2
```

---

6. Under the Hood (Internal Behavior)

JS uses the **Abstract Operation** `IsNullish`:

```text
IsNullish(value) → value === null || value === undefined
```

- If true → return right operand.
- Else → return left operand.

So internally, `a ?? b` is just a specialized conditional.

---

7. Pitfalls & Gotchas

1. **Falsy vs Nullish**

   ```js
   console.log("" ?? "default"); // "" ✅
   console.log("" || "default"); // "default" ❌
   ```

2. **Confusing with `||` in legacy code**
   - Many older tutorials use `||`.

- Switching to `??` may change behavior for falsy-but-valid values.

3. **Forgetting parentheses**

   ```js
   let result = a + b ?? 0;
   // parsed as (a + b) ?? 0, not a + (b ?? 0)
   ```

---

8. Interview-Level Brain Benders

👉 What’s the output?

Q1:

```js
console.log(null ?? 0 ?? false ?? "JS");
```

Answer: `0`

- `null` → skip → `0` (defined) → stops.

Q2:

```js
let value = false ?? "fallback";
console.log(value);
```

Answer: `false` ✅ (because `false` is *defined*).

Q3:

```js
let val = 0 || "zero";
let val2 = 0 ?? "zero";
console.log(val, val2);
```

Answer:

- `val = "zero"` (because `0` is falsy)
- `val2 = 0` (because `0` is defined)

Q4:

```js
let obj = { count: 0 };
console.log(obj.missingProp ?? obj.count ?? 100);
```

Answer: `0` ✅

- `missingProp` → undefined → skip
- `count` → defined (0) → chosen

---

9. When to Use `??` vs `||`

- Use `||` → when you want to skip **all falsy values**.
- Use `??` → when only **null/undefined** should trigger fallback.

✅ Summary

- `??` = “first defined value” operator.
- Safe alternative to `||` when `0`, `false`, `""` are valid inputs.
- Restricted with `&&` / `||` → must use parentheses.
- Precedence is low → careful with arithmetic.
- Internally runs `IsNullish` check.

👉 Do you want me to now **compare `??` with the newer `??=` operator** (nullish assignment), which is also an important part of interview traps?