ScopeTreasure Courses

REACT COURSES · LESSON 5.15 · CODE SCOPE SERIES

Conditional Rendering

Show this, not that: if/else, the ternary, the && shortcut and its famous 0 trap, loading–data–error screens, early returns, and picking the right tool.

Introduction

Real screens are full of "it depends": show the login button *or* the user's name, a spinner *while* loading, an error *if* it failed. **Conditional rendering** is how a component chooses what to draw from the current data — React has no special template syntax for it, just ordinary JavaScript decisions placed where they fit. *(Roman Urdu: Conditional rendering matlab — data dekh kar faisla, screen par kya dikhega aur kya nahi.)*

Key points

- **Four tools, four sizes:**
  - **if / else with early returns** — for whole-screen choices (loading screen vs content vs error). Return early, keep the main return clean.
  - **Ternary** `cond ? A : B` — for choosing between two pieces inside JSX.
  - `cond && A` — for showing something or *nothing*.
  - **A variable holding JSX** — compute `let banner = …` before the return when the choice has several branches.
- **The `&&` zero trap:** `{count && <p>…</p>}` with `count = 0` renders the number **0** on the page, because `&&` returns the falsy left side itself and React prints numbers. The fix: force a boolean — `{count > 0 && …}`. This is the video's interview-famous bug.
  (Roman Urdu: && ke saath 0 wali ghalti mashhoor hai — hamesha saaf boolean do, number nahi.)
- **The three-state screen is conditional rendering's home:** `if (loading) …`, `if (error) …`, then the data — exactly the shape of the Weather App (5.12) and every fetch screen after it.
- **Auth gates are the same idea:** `user ? <Dashboard /> : <Login />` — one ternary standing at the door of the whole app.
- **Return `null` to render nothing:** a component (or branch) that returns `null` leaves no trace on the page — how optional banners and modals disappear cleanly.
- **Keep conditions readable:** a ternary nested inside a ternary inside a ternary is a bug waiting for a reader. Past two levels, switch to early returns or a variable.

Code examples

Example 1 — The three-state screen with early returns

```
function UserProfile({ user, loading, error }) {
  if (loading) return <p>Loading profile…</p>;
  if (error) return <p className="error">Could not load the profile.</p>;
  if (!user) return <p>No user selected.</p>;

  return (
    <div className="card">
      <h3>{user.name}</h3>
      <p>{user.email}</p>
    </div>
  );
}
```

What it does: Each state gets its own early return, in the order a reader checks them: waiting, failed, empty — and only then the real content, with no nesting at all. This exact skeleton serves every data screen in the course.

What you will see: exactly one of the four outcomes at a time — the spinner text, the error, the empty note, or the profile card — never a mixture.

Example 2 — Ternary at the door: login vs dashboard

```
import { useState } from "react";

function Gate() {
  const [user, setUser] = useState(null);

  return (
    <div>
      {user ? (
        <div>
          <p>Welcome back, {user.name}!</p>
          <button onClick={() => setUser(null)}>Log out</button>
        </div>
      ) : (
        <div>
          <p>Please log in to continue.</p>
          <button onClick={() => setUser({ name: "Ayesha" })}>Log in</button>
        </div>
      )}
    </div>
  );
}
```

What it does: One ternary in the JSX chooses between the two faces of the app. Because it lives inside the return, the choice is re-evaluated on every render — log in and the whole face swaps, no page change involved.

What you will see: "Please log in to continue." with a Log in button; click it and the same spot becomes the welcome message with a Log out button.

Example 3 — The && shortcut and its 0 trap, wrong and right

```
function InboxBadges({ unread, offers }) {
  return (
    <div>
      {/* WRONG: when unread is 0, the page prints "0" */}
      {unread && <span className="badge">{unread} unread</span>}

      {/* RIGHT: a real boolean guards the badge */}
      {offers > 0 && <span className="badge">{offers} new offers</span>}
    </div>
  );
}
```

What it does: The first badge uses the raw number as the guard — `0 && …` evaluates to `0`, and React happily renders that 0 as text. The second guard (`offers > 0`) is a true boolean, so nothing — not even a stray number — appears when there are no offers.

What you will see: with `unread={0}` and `offers={2}`, the buggy badge prints a bare "0" while the correct one shows "2 new offers". Give offers 0 too and only the stray "0" remains — the bug, isolated.

Quick recap

- Choose by size: **early returns** for whole screens, the **ternary** for A-or-B inside JSX, `&&` for show-or-nothing, a **JSX variable** for many branches.
- The loading–error–data trio and the login gate are the two shapes you will write forever.
- Guard `&&` with a real **boolean** — `{0 && …}` prints 0 on the page.

Common mistakes

1. **The 0 badge** — `{items.length && <List />}` prints "0" for an empty list. The fix: `{items.length > 0 && <List />}` — always compare to a boolean with `&&`.
2. **Ternary towers** — `a ? … : b ? … : c ? …` chains nobody can read or safely edit. The fix: past two branches, move the choice into early returns or an if-chain above the return.
3. **Forgetting the third state** — screens that handle data and loading but not error (or empty) crash or hang silently. The fix: every data screen gets all the returns — loading, error, empty, content — in that order.

Practice

1. Build a `StockBadge` component receiving `stock` as a prop: above 10 show "In stock" (green), 1–10 show "Only {stock} left" (orange), 0 show "Sold out" (red). Use an if-chain with early returns, and test all three bands.
2. Build a `Notifications` panel: it receives an array of messages. Empty array → show "All caught up " (this is where the `&&` trap would print 0 — use the boolean guard); otherwise show the count line and the first message. Switch between the two by passing different arrays.

Practice quiz

1. Which conditional tool fits a whole-screen choice like loading vs content vs error?

- The ternary inside JSX
- if / else with early returns, keeping the main return clean
- The && shortcut only
- A variable holding JSX, always

Answer: if / else with early returns, keeping the main return clean

2. What does {count && ... } render when count is 0?

- Nothing, correctly
- The number 0 on the page, because && returns the falsy left side itself and React prints numbers
- An error
- The paragraph anyway

Answer: The number 0 on the page, because && returns the falsy left side itself and React prints numbers

3. How do you fix the && zero trap?

- Use a ternary instead, always
- Force a real boolean: {count > 0 && ...}
- Use || instead of &&
- It cannot be fixed

Answer: Force a real boolean: {count > 0 && ...}

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/react-conditional-rendering · More free tools and notes at https://scopetreasure.com