ScopeTreasure Courses

REACT COURSES · LESSON 5.26 (SHORT) · CODE SCOPE SERIES

Top 5 React Interview Questions (Short)

The Short, in notes form: the five questions almost every React interview opens with — and short, strong answers you can say out loud with confidence.

Introduction

Interviews rarely begin with the hardest question — they begin with the same five foundations, asked to see whether you truly understand the tool you claim. This closing lesson of the React series gives each question a **short, strong spoken answer**: one core idea, one example, no rambling. *(Roman Urdu: Interview ke pehle paanch sawal taqreeban har jagah wohi hote hain — yeh un ke chhote aur mazboot jawab hain.)*

Key points

- **Answer in layers:** the definition in one line, then *why it matters*, then a tiny example from a real project. Interviewers score the second and third layers — anyone can recite the first.
- **Q1 — What is the Virtual DOM?** React keeps a lightweight copy of the page in memory. On a change, it builds the new copy, diffs it against the old, and updates only the real nodes that actually changed — the minimum possible browser work.
- **Q2 — Props vs state?** Props are *given* to a component by its parent — read-only inputs. State is *owned* by the component itself and changes over time; changing state is what triggers a re-render.
- **Q3 — What are the Rules of Hooks?** Hooks run at the **top level only** (never in loops, conditions or nested functions) and only inside **React functions** (components or custom hooks) — because React tracks hooks purely by call order.
- **Q4 — Why do lists need keys?** Keys give each row a stable identity so React can tell exactly which items changed, moved or vanished — without them it matches by position and mixes up rows (the index-key trap, 5.16).
- **Q5 — Controlled vs uncontrolled inputs?** Controlled: React state holds the value, every keystroke flows through `onChange` — validation and formatting stay possible. Uncontrolled: the DOM holds the value and you read it when needed (refs) — simpler, less power.

Code examples

Example 1 — Q1 + Q2 in one tiny component

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

// Props arrive from the parent: name (read-only input)
function Counter({ name }) {
  // State is owned here: count (changes over time)
  const [count, setCount] = useState(0);

  return (
    <button onClick={() => setCount(count + 1)}>
      {name}: {count}
    </button>
  );
}
```

What it does: The say-it-with-code version of two answers at once. When the button is clicked, only the state changes; React produces a new virtual tree, diffs it, and touches exactly one real text node — the number — while the parent-given `name` prop sits untouched and untouchable.

What you will see: the count climbs per click and *only the number on the page changes* — the smallest possible DOM update, which is the Virtual DOM answer made visible.

Example 2 — Q3: the rule broken, then obeyed

```
// WRONG — hook inside a condition: call order can change
function Profile({ user }) {
  if (user) {
    const [open, setOpen] = useState(false); // breaks the rules
  }
  return <p>{user ? user.name : "Guest"}</p>;
}

// RIGHT — hooks at the top level, unconditionally
function ProfileFixed({ user }) {
  const [open, setOpen] = useState(false);
  return <p>{user ? user.name : "Guest"} {open ? "▾" : "▸"}</p>;
}
```

What it does: React pairs each hook call with its slot purely by the *order of calls* in the render. A hook behind a condition can vanish between renders, shifting every later hook into the wrong slot — state lands in the wrong variables. Top-level-only calls keep the order identical on every render, forever.

What you will see: the wrong version throws or behaves erratically when `user` flips between renders; the fixed version is boringly correct — which is the entire point of the rule.

Example 3 — Q5: both input styles, side by side

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

function BothInputs() {
  const [email, setEmail] = useState(""); // controlled
  const phoneRef = useRef(null); // uncontrolled

  function handleSubmit(e) {
    e.preventDefault();
    console.log("Email (from state):", email);
    console.log("Phone (from the DOM):", phoneRef.current.value);
  }

  return (
    <form onSubmit={handleSubmit}>
      <input value={email} onChange={(e) => setEmail(e.target.value)} placeholder="Email" />
      <input ref={phoneRef} placeholder="Phone" />
      <button>Submit</button>
    </form>
  );
}
```

What it does: The email field's value lives in state — the component can validate it per keystroke ("@ missing" as you type). The phone field's value lives in the DOM and is read once, at submit, through the ref. Quoting this trade-off — power per keystroke vs simplicity — is the strong answer.

What you will see: both values logged at submit; the difference is invisible at the end but was decisive during typing — only the controlled one could have warned you mid-type.

Quick recap

- **Virtual DOM:** diff the copy, patch only what changed. **Props:** given, read-only. **State:** owned, change triggers render.
- **Hooks:** top level, React functions only — because order is identity.
- **Keys:** row identity for the diff. **Controlled inputs:** state owns the value; uncontrolled: the DOM does.

Common mistakes

1. **Reciting without the "why"** — "Virtual DOM is a copy of the DOM" earns a nod, not a job. The fix: attach the consequence — "…so React updates only the changed nodes, which is why big UIs stay fast."

2. **Defining hooks by list instead of by rule** — naming ten hooks matters less than the two rules and their reason (call order). The fix: lead with the rules; the list is evidence, not the answer.
3. **Half the keys answer** — "keys help React identify items" without the failure story sounds memorized. The fix: add the trap — "with index keys, inserting at the top shifts identities and inputs land on the wrong rows; I have seen it break a real list."

Practice

1. Say each of the five answers out loud, timed: definition + why + example inside 30 seconds each. Record yourself on your phone; any answer that needs the notes in front of you is not ready yet.
2. Write the *sixth* answer yourself: "What is the difference between useEffect and an event handler?" — using the same shape (definition, why, one example from this course: data loading vs the search button of 5.12). Then check it against lessons 5.5 and 5.19.

Practice quiz

1. In an interview, what is the Virtual DOM answer in layers?

- Just: it is a copy of the DOM
- Definition, then why it matters, then a real example: React keeps a lightweight copy, diffs it, and updates only the real nodes that actually changed, the minimum possible browser work, which is why big UIs stay fast
- Only an example, no definition
- It is a real DOM, not a copy

Answer: Definition, then why it matters, then a real example: React keeps a lightweight copy, diffs it, and updates only the real nodes that actually changed, the minimum possible browser work, which is why big UIs stay fast

2. Props vs state, in one line each?

- They are the same
- Props are given to a component by its parent, read-only inputs; state is owned by the component itself and changes over time, and changing state triggers a re-render
- Props change, state never changes
- State is given by the parent, props are owned

Answer: Props are given to a component by its parent, read-only inputs; state is owned by the component itself and changes over time, and changing state triggers a re-render

3. Why do the Rules of Hooks exist?

- To make code look tidy
- Because React tracks hooks purely by call order, so hooks must run at the top level only and only inside React functions

- Because hooks are slow otherwise
- There is no reason, they are suggestions

Answer: Because React tracks hooks purely by call order, so hooks must run at the top level only and only inside React functions

---

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-interview-questions · More free tools and notes at https://scopetreasure.com