ScopeTreasure Courses

REACT COURSES · LESSON 5.1 · CODE SCOPE SERIES

What is React?

The library that runs the modern web: the problem it solves, components thinking, the virtual DOM in plain words, and where React fits — Vite, Next.js, React Native.

Introduction

React is a JavaScript **library** for building user interfaces — the screens people see and touch. It was made by Meta (Facebook) and is now used by countless big apps, because it solves one painful problem: on a busy page, keeping the screen and the data in sync by hand is slow and full of bugs. React's answer is simple: **you describe what the screen should look like for the current data, and React updates the page for you.** *(Roman Urdu: React aik library hai — aap sirf bata dein ke screen kaisi dikhni chahiye, page ko update karna React ka kaam hai.)*

Key points

- **A library, not a framework:** React does one job — the view (the interface). You add other pieces (routing, data fetching) as you need them. Frameworks like Angular hand you the whole kitchen; React hands you the best knife.
- **Components are the unit:** a React screen is built from small, reusable pieces called **components** — a button, a card, a navbar. Build a piece once, use it everywhere.
- **Declarative, not imperative:** in plain JavaScript you write every step ("find this div, change its text, hide that one"). In React you declare the result ("if the user is logged in, show the Dashboard") and React works out the steps.

  (Roman Urdu: Purana tareeqa har step batata tha, React sirf final result batata hai.)
- **The virtual DOM in plain words:** React keeps a light copy of the page in memory. When data changes, it compares the new copy with the old one and touches **only the parts of the real page that actually changed** — not the whole page.
- **One-way data flow:** data moves down from parent components to children. This makes apps predictable: when something looks wrong, you know exactly where the data came from.
- **React is a family: Vite** is the tool that creates and runs React projects while you learn; **Next.js** is a framework built on React for big production sites; **React Native** uses the same ideas to build real Android and iOS apps.
- **Honest cons:** React needs a build setup (Vite handles it), the ecosystem is huge so choices can confuse a beginner, and for a tiny static page plain HTML is honestly enough.

Code examples

Example 1 — The old way vs the React way

```
<!-- Plain JavaScript: you give every step -->
<p id="msg">Hello</p>
<button onclick="document.getElementById('msg').textContent = 'Welcome!'">
  Change
</button>
```

```
// React: you describe the result, React does the steps
function App() {
  return <h1>Welcome!</h1>;
}
```

What it does: The plain-JavaScript version orders the browser around step by step — find the element, change its text. The React version only declares what the screen should show. As apps grow, the step-by-step style turns into a spider web; the describing style stays readable.

What you will see: both versions can show the same greeting — but only the React one stays simple when the page has a hundred moving parts.

Example 2 — Your first React component

```
function WelcomeCard() {
  const name = "Ali";
  return (
    <div className="card">
      <h2>Welcome, {name}</h2>
      <p>This whole card is one component.</p>
    </div>
  );
}
```

What it does: A component is a JavaScript function that returns JSX — the HTML-like code React understands. The `{name}` inside is a real JavaScript value placed into the markup, and whatever the function returns becomes a piece of the screen.

What you will see: a card on the page with the heading "Welcome, Ali" — the name came from the variable, not from typing it twice.

Example 3 — A page is components inside components

```
function Navbar() {
  return <nav className="navbar">Code Scope</nav>;
}

function CourseCard() {
  return <div className="card">React Course</div>;
}

function App() {
  return (
    <div>
      <Navbar />
      <CourseCard />
      <CourseCard />
      <CourseCard />
    </div>
  );
}
```

What it does: The `App` component is the whole page, built by placing other components inside it — one `Navbar` and the same `CourseCard` used three times. This is the core React habit: small pieces, composed into big screens, reused without copy-paste.

What you will see: a navbar on top and three identical course cards under it — written once, used three times.

Quick recap

- React is Meta's JavaScript **library** for building interfaces — you describe the screen for the current data, React updates the page.
- Screens are built from **components**, data flows one way (down), and the **virtual DOM** lets React change only what actually changed.
- The family: **Vite** to build and learn, **Next.js** for production websites, **React Native** for mobile apps.

Common mistakes

1. **Calling React a framework and waiting for it to do everything** — React is the view layer only; routing and data tools are added separately (React Router, React Query — later lessons). The fix: learn React as the interface engine, and meet its partner tools one by one.
2. **Trying to update the page by hand inside React** — reaching for `document.getElementById` in a React app fights the library and causes bugs. The fix: change the **data** (state, next lessons) and let React re-render the screen.

3. **Learning React before JavaScript is comfortable** — components are functions, props are arguments, lists are `map()`. The fix: finish the JavaScript series first; React will feel like a natural next step, not a new language.

Practice

1. Look at any app you use daily (a food app, a banking app). List five components you can spot on its home screen — pieces that repeat or could be reused. Write the list as component names in capital letters, like `StoryCircle`, `PostCard`.
2. In your own words, write two sentences: one that explains the virtual DOM to a friend without any technical words, and one that says why describing the result (declarative) beats giving every step (imperative) on a big page.

Practice quiz

1. Is React a framework or a library, and what job does it do?

- A framework that does everything, routing and data included
- A library that does one job, the view (the interface); routing and data tools are added separately
- A language that replaces JavaScript
- A database

Answer: A library that does one job, the view (the interface); routing and data tools are added separately

2. What does declarative mean in React?

- You write every step: find this div, change its text
- You declare the result for the current data, and React works out the steps
- You declare variables only
- You write no code at all

Answer: You declare the result for the current data, and React works out the steps

3. What does the virtual DOM do when data changes?

- It rebuilds the whole real page
- It compares the new light copy with the old one and touches only the parts of the real page that actually changed
- It deletes the page and starts over
- It does nothing, the browser does it all

Answer: It compares the new light copy with the old one and touches only the parts of the real page that actually changed

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-what-is-react · More free tools and notes at https://scopetreasure.com