What Are Hooks?

What a hook is, why hooks exist, the Rules of Hooks, and their relationship to function components -- with simple examples.

Intermediate 10 min

Last updated: August 18, 2026
Read online: https://www.learnforgex.com/en/topics/what-are-hooks

---

What Are Hooks?

In State & Events, we used `useState` many times, but never stopped to ask what a hook actually is. This lesson explains the concept of a hook from the ground up -- and from here on, the React course gets a bit more advanced.

What Is a Hook?

A **hook** is a function whose name starts with `use` that lets you add React features (state, side effects, DOM references, and more) to function components:

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

// useState is a HOOK that React provides out of the box -- a function that
// starts with "use" and adds React features like state to function
// components.
function WhatIsAHookExample() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>useState is a hook: {count}</p>
      <button onClick={() => setCount(count + 1)}>+1</button>
    </div>
  );
}
```

`useState` is actually a hook React gives you out of the box. The `useEffect`, `useRef`, `useMemo`, and `useCallback` we'll see in this category are hooks too -- they all follow the same `use...` naming pattern.

Why Hooks?

Before Hooks, state and other React features could only be used in "class components," written with a different syntax. Hooks brought these features to regular function components -- there's no longer any need to write class components at all. In this course we've only used function components from the start; hooks are the mechanism that makes that possible.

Rules of Hooks

There are two basic rules you need to follow when using hooks:

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

function RulesOfHooksExample({ showExtra }) {
  // CORRECT: Hooks are ALWAYS called at the TOP level of the component.
  const [count, setCount] = useState(0);

  // WRONG: Calling a hook conditionally (never do this).
  // if (showExtra) {
  //   const [extra, setExtra] = useState(0); // Breaks the rules!
  // }
  //
  // Why? React tracks each hook by assuming they're called in the SAME
  // ORDER on EVERY RENDER. A conditional hook breaks this order by being
  // called on some renders and not others -- React gets confused about
  // which state belongs to which hook.

  return <p>Count: {count}</p>;
}
```

First: hooks are always called at the TOP LEVEL of a component -- they are NEVER placed inside an `if`, a `for` loop, or another function. Second: hooks are only called from inside function components, or from inside another hook -- never from a regular function. React matches state to the right component by assuming hooks are called in the same order on every render -- breaking these rules confuses React.

Function Components and Hooks

Hooks only work inside function components -- calling `useState` inside a regular JavaScript function (that isn't a component) causes an error. That's why every component we've seen in this course (`App`, `Card`, `ProfileInfo`, etc.) is a function component, and all of them are eligible to use hooks.

Summary and Glossary

A hook is a function starting with `use` that adds React features to function components. Hooks are only called at the top level of a component, and only from inside function components (or other hooks) -- they can't be called conditionally or inside a loop.

Glossary

Hook — A function starting with `use` that adds React features to function components.

Rules of Hooks — The two rules stating that hooks can only be called at the top level of a component, and only from function components or other hooks.

Function Component — A React component written as a JavaScript function; hooks only work inside these.