Custom hooks

React has some built-in hooks, such as the `useState` hook, or the `useRef` hook, which you learned about earlier. However, as a React developer, you can write your own hooks. So, why would you want to write a custom hook?

In essence, hooks give you a repeatable, streamlined way to deal with specific requirements in your React apps. For example, the `useState` hook gives us a reliable way to deal with state updates in React components.

A custom hook is simply a way to extract a piece of functionality that you can use again and again. Put differently, you can code a custom hook when you want to avoid duplication or when you do not want to build a piece of functionality from scratch across multiple React projects. By coding a custom hook, you can create a reliable and streamlined way to reuse a piece of functionality in your React apps.

To understand how this works, let's explore how to build a custom hook. To put this in context, let's also code a very simple React app.

The entire React app is inside the App component below:

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

function App() {
  const [count, setCount] = useState(0);

  function increment() {
    setCount(prevCount => prevCount + 1)
  }

  return (
    <div>
      <h1>Count: {count}</h1>
      <button onClick={increment}>Plus 1</button>
    </div>
  );
}

export default App;
```

coach

This is a simple app with an `h1` heading that shows the value of the count state variable and a button with an `onClick` event-handling attribute which, when triggered, invokes the `increment()` function.

The hook will be simple too. It will console log a variable's value whenever it gets updated.

Remember that the proper way to handle `console.log()` invocations is to use the `useEffect` hook.

So, this means that my custom hook will:

1. Need to use the `useEffect` hook and
2. Be a separate file that you'll then use in the App component.

How to name a custom hook

A custom hook needs to have a name that begins with use.

Because the hook in this example will be used to log values to the console, let's name the hook `useConsoleLog`.

Coding a custom hook

Now's the time to explore how to code the custom hook.

First, you'll add it as a separate file, which you can name `useConsoleLog.js`, and add it to the root of the `src` folder, in the same place where the App.js component is located.

Here's the code of the useConsoleLog.js file:

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

function useConsoleLog(varName) {
  useEffect(() => {
    console.log(varName);
  }, [varName]);
}

export default useConsoleLog;
```

coach

Using a custom hook

Now that the custom hook has been coded, you can use it in any component in your app.

Since the app in the example only has a single component, named App, you can use it to update this component.

The `useConsoleLog` hook can be imported as follows:

`import useConsoleLog from "./useConsoleLog";`

And then, to use it, under the state-setting code, I'll just add the following line of code:

`useConsoleLog(count);`

Here's the completed code of the App.js file:

```
import { useState } from "react";
import useConsoleLog from "./useConsoleLog";

function App() {
  const [count, setCount] = useState(0);
  useConsoleLog(count);

  function increment() {
    setCount(prevCount => prevCount + 1);
  }

  return (
    <div>
      <h1>Count: {count}</h1>
      <button onClick={increment}>Plus 1</button>
    </div>
  );
}

export default App;
```

This update confirms the statement made at the beginning of this reading, which is that custom hooks are a way to extract functionality that can then be reused throughout your React apps

Conclusion

You have learned how to name, build and use custom hooks in React.