How Can You Create Custom Hooks in React?

freecodecamp.org/learn/front-end-development-libraries-v9/lecture-understanding-effects-and-referencing-values-in-react/how-can-you-create-custom-hooks-in-react

React provides many built-in hooks that let you implement different features in your projects. These include `useState`, `useEffect`, `useContext`, and others.

But sometimes, you'll need to add a feature that none of the built-in hooks can help with. Fortunately, you can create your own custom hooks in React.

Custom hooks are not as complicated as they might seem. They're just reusable functions that let you share logic across multiple components. That means reusability is another reason why you would want to build your own hook.

With a custom hook, you can extract logic away from any components that use them, like data fetching, state management, toggling, side effects like checking for the online or offline status of users, and so on.

You can then import the hook to use in any component, so you can focus on rendering and presentation within those components. That means fewer repetitions and less duplication, which means fewer places to make changes when you want to make any updates.

Now, let's take a look at how you can make your own custom hook.

In React, all built-in hooks start with the word `use`, so your custom hook should follow the same convention. Your custom hook's name should also clearly communicate what it does.

So, if your custom hook…

- fetches data, you can call it `useFetch`
- toggles something on and off, you can call it `useToggle`
- or if it implements debouncing, `useDebounce` is a good name

Let's say you want to build a custom hook to add debouncing to your app.

Debouncing is a programming technique that limits how often a function runs. It works by waiting until a user stops performing an action for a specified period of time before executing the function. For example, in a search box, instead of making an API call for every keystroke, debouncing waits until the user pauses typing for, say, 500 milliseconds.

To create a debouncing custom hook, you first need to create a `useDebounce.jsx` or `useDebounce.js` file. Conventionally, files for any custom hooks you create are saved to a `hooks` folder.

You can use some built-in hooks within your own custom hook. For debouncing, you need the `useState` and `useEffect` hooks, so import them at the top of your file:

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

Next, create a `useDebounce` function that takes `value` and `delay` as parameters. `value` is the resource you want to wait for, and `delay` is the period of time you want to wait for. Since you want to wait for some period of time, the `setTimeout` and `clearTimeout` functions would be useful:

```
function useDebounce(value, delay) {
  const [debouncedValue, setDebouncedValue] = useState(value);

  useEffect(() => {
    const handler = setTimeout(() => {
      setDebouncedValue(value);
    }, delay);

    return () => {
      clearTimeout(handler);
    };
  }, [value, delay]);

  return debouncedValue;
}

export { useDebounce };
```

The `debouncedValue` state holds and returns the delayed value, which only updates after the specified timeout period.

`useEffect` is where the magic really happens. If you recall from the previous lesson, anything that exists outside the React rendering cycle, like setting and clearing a timer, is a side effect, and you should use the `useEffect` hook to handle them.

Within the `useEffect` hook here, you use `setTimeout` to set the `debouncedValue`. You then return a cleanup function that uses `clearTimeout` to clear the previous timeout whenever `value` or `delay` changes, or the component unmounts.

To use this hook, we've prepared a `footballers` array to filter through with a simple search bar:

```
const footballers = [
  'Lionel Messi', 'Cristiano Ronaldo', 'Neymar Jr',
  'Kylian Mbappe', 'Mohamed Salah', 'Sadio Mane',
  'Kevin De Bruyne', 'Robert Lewandowski', 'Harry Kane',
  'Sergio Ramos', 'Virgil van Dijk', 'Alisson Becker',
  'Joshua Kimmich', 'Manuel Neuer', 'Karim Benzema',
  'Thibaut Courtois', 'Eden Hazard', 'Raheem Sterling',
  'Bruno Fernandes', 'Trent Alexander-Arnold', 'Son Heung-min',
  'Pierre-Emerick Aubameyang','Sergio Aguero', 'Luis Suarez',
  'Luka Modric', 'Casemiro', 'Frenkie de Jong', 'Gerard Pique',
  'Marc-Andre ter Stegen', 'Keylor Navas', 'Angel Di Maria',
  "N'Golo Kante", 'Kai Havertz', 'Timo Werner', 'Hakim Ziyech',
  'Christian Pulisic', 'Mason Mount', 'Olivier Giroud', 'Tammy Abraham',
  'Kepa Arrizabalaga', 'Ben Chilwell', 'Thiago Silva', 'Kurt Zouma',
  'John Terry', 'Didier Drogba', 'Frank Lampard', 'Ashley Cole', 'Petr Cech',
];

export default footballers;
```

And here's a `FootballerSearch` component that uses the `useDebounce` hook to delay searching for 1 second after the user stops typing:

```
import { useState, useEffect } from "react";
import { useDebounce } from "./hooks/useDebounce";
import footballers from "./footballers";

const FootballerSearch = () => {
  const [query, setQuery] = useState("");
  const debouncedQuery = useDebounce(query, 1000); // Start searching 1 second after
the user stops typing

  useEffect(() => {
    if (debouncedQuery) {
      const results = footballers.filter((footballer) =>
        footballer.toLowerCase().includes(debouncedQuery.toLowerCase()),
      );
      console.log("Search results:", results);
    } else {
      console.log("Search results: []");
    }
  }, [debouncedQuery]);

  return (
    <>
      <h1 style={{ textAlign: "center" }}>Footballer Search App</h1>
      <div style={{ textAlign: "center" }}>
        <input
          style={{ padding: "0.5rem", width: "30%" }}
          type="text"
          value={query}
          onChange={(e) => setQuery(e.target.value)}
          placeholder="Search for a footballer..."
        />
      </div>
    </>
  );
};

export default FootballerSearch;
```

As you can see, the `debouncedQuery` variable is what initializes the `useDebounce` hook with the query state (what the user types), and the delay for 1,000 milliseconds, or 1 second. The search itself is handled inside the `useEffect` hook, and search results are logged to the console.

Questions

What is a notable benefit of creating a custom hook in React?

It makes components render faster.

It allows you to reuse logic across multiple components.

It replaces the need for built-in hooks.

It forces components to share the same state.

What is the correct naming convention for a custom hook in React?

It should start with "`use`".

It can have any name.

It should end with "`Hook`".

It must match a built-in hook name.

What would you call a custom hook that toggles a value on and off?

`useSwitch`

`useToggle`

`toggleHook`

`useBoolean`