What is State, and How Does the useState Hook Work?

freecodecamp.org/learn/front-end-development-libraries-v9/lecture-working-with-state-and-responding-to-events-in-react/what-is-state-and-how-does-the-usestate-hook-work

State is one of the most important fundamentals of React and other frontend frameworks. It's like the brain of a component, meaning it holds information that can change over time and controls how the components behave and look.

Let's look into what state is and how the `useState` hook lets you work with it.

State represents the dynamic data in your React component, like the value from a user input, data fetched from an API, or an item in a to-do list.

Whenever the state changes, React re-renders the component without reloading the page to reflect those changes in the user interface. This reactivity makes your app interactive.

The `useState` hook is a function that lets you declare state variables in functional components.

Before hooks, you could only use state in class components. But with the introduction of hooks since React 16.8, you can use state in functional components by using the `useState` hook.

To use the `useState` hook, you need to import it from React:

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

You can also import React itself and get access to the `useState` hook as a property:

```
import React from "react";
```

Here's how you can declare a state variable when you import `useState`:

```
const [stateVariable, setStateFunction] = useState(initialValue);
```

And here's how you can declare a state variable when you import React:

```
const [stateVariable, setStateFunction] = React.useState(initialValue);
```

In the state variable you have the following:

- `stateVariable` holds the current state value
- `setStateFunction` (the setter function) updates the state variable
- `initialValue` sets the initial state

Note that the state in a React component is private, and is isolated to each component instance. This means that, if you render the same component twice, the state component of one does not affect the other. This also means that, if you'd like to share state between components, then you'd need to lift the state up to a common parent and pass it down as props.

Another thing is that hooks must be called at the top level of a component, just before the `return` keyword, to keep the state and effects consistent across renders. This means you can't use state inside loops, conditions, or nested functions.

Here's an example of managing state with the `useState` hook in a `Counter` component:

```
// Importing the useState hook
import { useState } from "react";

function Counter() {
  const initialValue = 0;

  // The state variable and setter function
  const [count, setCount] = useState(initialValue);

  return (
    <div>
      {/* Display current state value */}
      <h2>{count}</h2>

      <button onClick={() => setCount(count - 1)}>Decrement</button>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}

export default Counter;
```

In the code above, we have the `useState` hook imported from React. In the `Counter` component, the `count` represents the current state while `setCount` is the set function responsible for updating state. The current state value is `0`. The `return` statement contains the count and two buttons to decrement and increment the count by `1`.

You can manage multiple pieces of state by calling the `useState` hook multiple times. This is especially important when you have unrelated state variables:

```
function UserProfile() {
  const [isOnline, setIsOnline] = useState(false);
  const [notifications, setNotifications] = useState(0);

  // The rest of the component logic
}
```

You can also call the `useState` hook multiple times when managing multiple states that update separately, like form fields:

```
function SignUpForm() {
  const [name, setName] = useState("");
  const [username, setUsername] = useState("");
  const [email, setEmail] = useState("");

  // The rest of the component logic
}
```

But in this case, it's best to combine the states since they're all part of the same form:

```
function SignUpForm() {
  const [formData, setFormData] = useState({
    name: "",
    username: "",
    email: "",
  });

  // The rest of the component logic
}
```

That's what state is and how you can use the `useState` hook.

Questions

In which version of React were hooks introduced, allowing state to be used in functional components?

React 15.6

React 16.8

React 17.0

React 18.2

What does state represent in a React component?

Dynamic data that triggers re-renders when changed.

Only static data that doesn’t change.

The component's styling information.

Fixed values set at the beginning of the app.

Which of the following is the correct way to work with the `useState` hook?

```
const [formData, setFormData] =
useState(<
  name: "",
  username: "",
  email: "",
>);
```

```
const ref =
useState({
  name: "",
  username: "",
  email: "",
});
```

```
const <formData, setFormData> =
useState({
  name: "",
  username: "",
  email: "",
});
```

```
const [formData, setFormData] =
useState({
  name: "",
  username: "",
  email: "",
});
```