codecademy

Hooks

Function Components

In React, you can use a function as a component instead of a class. Function components receive `props` as a parameter.
In the example code, we show two equivalent components: one as a class and one as a function.

```
// The two components below are equivalent.
class GreeterAsClass extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}!</h1>;
  }
}

function GreeterAsFunction(props) {
  return <h1>Hello, {props.name}!</h1>;
}
```

Why Hooks?

Hooks are functions that let us “hook into” state and lifecycle functionality in function components.
Hooks allow us to:

- reuse stateful logic between components
- simplify and organize our code to separate concerns, rather allowing unrelated data to get tangled up together
- avoid confusion around the behavior of the `this` keyword
- avoid class constructors, binding methods, and related advanced JavaScript techniques

Rules for Using Hooks

There are two main rules to keep in mind when using Hooks:

1. Only call Hooks from React functions
2. Only call Hooks at the top level, to be sure that Hooks are called in the same order each time a component renders.

Common mistakes to avoid are calling Hooks inside of loops, conditions, or nested functions.

```
// Instead of confusing React with code like this:
if (userName !== '') {
 useEffect(() => {
   localStorage.setItem('savedUserName', userName);
 });
}

// We can accomplish the same goal, while consistently calling our Hook every time:
useEffect(() => {
 if (userName !== '') {
   localStorage.setItem('savedUserName', userName);
 }
});
```

The State Hook

The `useState()` Hook lets you add React state to function components. It should be called at the top level of a React function definition to manage its state. `initialState` is an optional value that can be used to set the value of `currentState` for the first render. The `stateSetter` function is used to update the value of `currentState` and rerender our component with the next state value.

```
const [currentState, stateSetter] = useState(initialState);
```

State Setter Callback Function

When the previous state value is used to calculate the next state value, pass a function to the state setter. This function accepts the previous value as an argument and returns an updated value.
If the previous state is not used to compute the next state, just pass the next state value as the argument for the state setter.

```
function Counter({ initialCount }) {
 const [count, setCount]
= useState(initialCount);
 return (
   <div>
     Count: {count}
     <button onClick={() =>
setCount(initialCount)}>Reset</button>
     <button onClick={() =>
setCount((prevCount) => prevCount - 1)}>-
</button>
     <button onClick={() =>
setCount((prevCount) => prevCount + 1)}>+
</button>
   </div>
 );
}
```

Multiple State Hooks

`useState()` may be called more than once in a component. This gives us the freedom to separate concerns, simplify our state setter logic, and organize our code in whatever way makes the most sense to us!

```
function App() {
 const [sport, setSport]
= useState('basketball');
 const [points, setPoints] = useState(31);
 const [hobbies, setHobbies] = useState([]);
}
```

Side Effects

The primary purpose of a React component is to return some JSX to be rendered. Often, it is helpful for a component to execute some code that performs side effects in addition to rendering JSX.
In class components, side effects are managed with lifecycle methods. In function components, we manage side effects with the Effect Hook. Some common side effects include: fetching data from a server, subscribing to a data stream, logging values to the console, interval timers, and directly interacting with the DOM.

The Effect Hook

After importing `useEffect()` from the `'react'` library, we call this Hook at the top level of a React function definition to perform a side effect. The callback function that we pass as the first argument of `useEffect()` is where we write whatever JavaScript code that we'd like React to call after each render.

```
import React, { useState, useEffect } from
'react';

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

  useEffect(() => {
    document.title = `You clicked ${count}
times`;
  });

  return <button onClick={(prev) =>
setCount(prev + 1)}>+</button>;
}
```

Effect Cleanup Functions

The cleanup function is optionally returned by the first argument of the Effect Hook.
If the effect does anything that needs to be cleaned up to prevent memory leaks, then the effect returns a cleanup function. The Effect Hook will call this cleanup function before calling the effect again as well as when the component is being unmounted from the DOM.

```
useEffect(() => {
  document.addEventListener('keydown',
handleKeydown);
  return () =>
document.removeEventListener('keydown',
handleKeydown);
});
```

Multiple Effect Hooks

`useEffect()` may be called more than once in a component. This gives us the freedom to individually configure our dependency arrays, separate concerns, and organize our code in whatever way makes the most sense to us!

```
function App(props) {
 const [title, setTitle] = useState('');
 useEffect(() => {
   document.title = title;
 }, [title]);

 const [time, setTime] = useState(0);
 useEffect(() => {
   const intervalId = setInterval(() =>
setTime((prev) => prev + 1), 1000);
   return () => clearInterval(intervalId);
 }, []);

 // ...
}
```

Effect Dependency Array

The dependency array is used to tell the `useEffect()` method when to call our effect.
By default, with no dependency array provided, our effect is called after every render. An empty dependency array signals that our effect never needs to be re-run. A non-empty dependency array signals to the Effect Hook that it only needs to call our effect again when the value of one of the listed dependencies has changed.

```
useEffect(() => {
 alert('called after every render');
});

useEffect(() => {
 alert('called after first render');
}, []);

useEffect(() => {
 alert('called when value of `endpoint` or
`id` changes');
}, [endpoint, id]);
```