What Are Effects in React, and How Does the useEffect Hook Work?

In React, an effect is anything that happens outside the component rendering process. That is, anything React does not handle directly as part of rendering the UI.

Common examples include fetching data, updating the browser tab's title, reading from or writing to the browser's local storage, getting the user's location, and much more. These operations interact with the outside world and are known as side effects.

React provides the `useEffect` hook to let you handle those side effects. `useEffect` lets you run a function after the component renders or updates.

Let's see how the `useEffect` hook works and why it's essential for modern React development.

To use the `useEffect` hook, you first need to import it:

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

Then you use it as a function, like this:

```
useEffect(() => {
  // Your side effect logic (usually a function) here
}, [dependencies]);
```

The effect function runs after the component renders, while the optional `dependencies` argument controls when the effect runs.

Note that `dependencies` can be an array of "reactive values" (state, props, functions, variables, and so on), an empty array, or omitted entirely. Here's how all of those options control how `useEffect` works:

- If `dependencies` is an array that includes one or more reactive values, the effect will run whenever they change.
- If `dependencies` is an empty array, `useEffect` runs only once when the component first renders.
- If you omit `dependencies`, the effect runs every time the component renders or updates.

For example, in this `Counter` application, we don't pass in a `dependencies` argument, so the effect runs when the component renders and every time it updates:

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

const Counter = () => {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log("Component renders");
  });

  return (
    <div
      style={{
        display: "flex",
        alignItems: "center",
        flexDirection: "column",
      }}
    >
      <h2>{count}</h2>
      <div>
        <button onClick={() => setCount(count + 1)}>Increase</button>
        <button onClick={() => setCount(count - 1)}>Decrease</button>
      </div>
    </div>
  );
};

export default Counter;
```

But if we pass in an empty array as a dependency, the effect only runs on the first render:

```
useEffect(() => {
  console.log('Component renders');
}, []);
```

If you pass in the `count` state as a dependency, the effect runs when the component first renders, and when `count` changes:

```
useEffect(() => {
  document.title = `The current count is ${count}`;
  console.log('component renders');
}, [count]);
```

Note that, if the effect you set up persists beyond the component's rendering lifecycle, you might need another function to "clean up" that function after the component renders or updates.

For example, if your effect function uses `setInterval()`, sets an event listener like `window.addEventListener()`, or connects to a server, you'll need a cleanup function to run `clearInterval()`, `window.removeEventListener()`, and disconnect from the server, respectively.

Here's the syntax for returning a cleanup function from the `useEffect` hook:

```
useEffect(() => {
  // Your side effect logic here
  return () => {
    // Cleanup logic here (optional)
  };
}, [dependencies]);
```

For instance, if you add a scroll event listener, you can clean it up by removing it in your cleanup function:

```
useEffect(() => {
  const handleScroll = () => {
    // Handle scroll logic
  };
  window.addEventListener("scroll", handleScroll);

  return () => {
    window.removeEventListener("scroll", handleScroll);
  };
}, []);
```

Questions

What is considered an effect in React?

Any update to the component's state.

Operations outside the rendering process that React doesn't manage.

Rendering the UI after each state update.

Updating the component's props.

What determines how side effects run in a React app?

Effects always run once, regardless of dependencies.

Effects run only when props change, ignoring state.

The dependency array determines when effects run.

Effects always run on every render unless explicitly disabled.

What are common examples of side effects in React?

Rendering components and updating state.

Passing props to child components.

Defining component styles and layout.

Fetching data, updating tab titles, and reading from storage.