Creating a Counter App with useState

In this lesson, we create a tiny counter app with useState in addition to createContext and useContext.

WE'LL COVER THE FOLLOWING

Before you begin

If you haven't learned React basics and React Hooks basics, please visit the official site before continuing. This course assumes a basic understanding of React and React Hooks.

What we create

We are creating a so-called counter app. There are two numbers that we can increment or decrement. We don't structure code and put everything into one file in this lesson.

The code

```
import React, { createContext, useContext, useState } from 'react';
```

This imports `React` and some hooks from the library.

```
const initialState = {
  count1: 0,
  count2: 0,
```

```
};
```

This defines `initialState` which includes two numbers, `count1` and `count2`

```
const useValue = () => useState(initialState);
```

This is a handy custom hook that we can use in a provider below.

```
const Context = createContext(null);
```

This creates a context. Its value is initially `null`, but will be replaced by `useValue()`.

```
const useGlobalState = () => {
  const value = useContext(Context);
  if (value === null) throw new Error('Please add GlobalStateProvider');
  return value;
};
```

This is a custom hook to return the context value. It checks whether a provider is used as expected. If the value is `null`, it is not updated because `useValue()` never returns `null`.

```
const GlobalStateProvider = ({ children }) => (
  <Context.Provider value={useValue()}>{children}</Context.Provider>
);
```

This is a context provider. This is a component that should be put near the root.

```
const Counter = ({ name }) => {
  const [state, setState] = useGlobalState();
  const count = state[name] || 0;
  const increment = () => {
    setState({ ...state, [name]: count + 1 });
  };
  const decrement = () => {
    setState({ ...state, [name]: count - 1 });
  };
  return (
    <div>
      {count}
```

```
      <button onClick={increment}>+1</button>
      <button onClick={decrement}>-1</button>

    </div>
  );
};
```

This is a component to display a number. It receives a `name` prop to specify which number to display, `count1` or `count2`. It has two buttons to increment and decrement the number.

```
const App = () => (
  <GlobalStateProvider>
    <h1>Count1</h1>
    <Counter name="count1" />
    <Counter name="count1" />
    <h1>Count2</h1>
    <Counter name="count2" />
    <Counter name="count2" />
  </GlobalStateProvider>
);
```

This is an app component. It has `GlobalStateProvider` and several `Counter` components.

```
export default App;
```

Finally, we export the app component.

The working example

Check out the app below.

```
import React from 'react';
require('./style.css');

import ReactDOM from 'react-dom';
import App from './app.js';

ReactDOM.render(
  <App />,
  document.getElementById('root')
);
```

Exercise

Try adding a new counter, `count3`. Interestingly, you don't need to modify `initialState` if the initial value of `count3` is `0`.

Next

In the next lesson, we will learn a different pattern with `useReducer`.