Trainer guide
Examples and exercises (React)
React-8 – State

Contents

Trainer demo | The need for state

[Link to environment]

In this demo, you will illustrate why state is necessary in React. Then, you will show how to implement the useState hook to create and update state.

src/App.jsx

- Here we have a simple thermostat app.
- The current temperature is displayed on the page.
- This temperature is stored as a global variable `temperature` in our application.
- There are also two buttons: one to increase the temperature and one to decrease it.
- The "+" button has an `onClick` handler that calls the `increaseTemperature` function. This increments the temperature by 1 and logs the new value of `temperature` to the console.
- Similarly, the "-" button calls the `decreaseTemperature` function, which decrements the temperature by 1 and again logs the new value of `temperature` to the console.
- *Click on the buttons.*
- The buttons don't seem to be working properly. The temperature displayed appears to be stuck at 20 degrees.
- But, looking at the console, we can see that the `temperature` variable is logged, and we can see that it is indeed changing.
- This means it's just our UI that is stuck. Why does it not display the most up-to-date value of the `temperature` variable?
    - Let's think about what happens when we press one of the buttons.
        - The value of the `temperature` variable does change, proven by the messages in the console.
        - However, **React has no idea** that `temperature` has changed.
        - Because React doesn't know the variable has changed, it doesn't know that it's supposed to re-render the component.
        - Because the component is not being re-rendered, it is stuck displaying the original value of `temperature`, which was 20.
- So somehow, we need to let React know that we are changing the temperature, so that it can re-render the component.
- This is exactly the problem that is solved by "state".
    - State allows us to define variables inside our components and change their values over time, while also ensuring that the UI will actually react to those changes.

Solution (src/App.jsx)

- Let's look at how to use state to fix this application.
- To use state within a function component, we need to import a function called useState.
    - *`import React, { useState } from 'react'`*
- You'll often hear useState referred to as the "useState hook". A "hook" is simply a React term for a special function that allows you to access, or "hook into", React features such as state. There are various other built-in hooks that do other things, some of which we'll cover in more detail later.
- The useState hook is what we use if we want to create state in React.
- The next thing to note is that we are going to be defining state within our component, so we can get rid of the global temperature variable.
    - *Delete the line `let temperature = 20`*
- Let's write the code to create the state, and then break it down afterwards.
    - *Within the body of `App()`, write:*
      `const [temperature, setTemperature] = useState(20)`
- *Breakdown*:
    - `useState(20)` creates a new piece of state. The argument passed to the useState function is the initial value. In our case, we want the temperature to be initialised to the value 20, so we pass 20 to the function.
    - `useState()` returns an array with two values.
        - The first thing that's returned, which we've called "temperature", holds the **current value of the state.**
        - The second thing that's returned, which we've called "setTemperature", is a **state setter function** that we will use any time we want to update the state.
        - When we update the state, we never directly modify the value of `temperature`. This is why we need the `setTemperature` function – when we call this, it lets React know that we are updating the state so that it knows to re-render our component.
    - We've also used array destructuring to extract both values returned from the useState call in a single line.
    - As a side note, when we named the state setter function (`setTemperature`), we simply used the same name as the state value (`temperature`) prefixed with the word "set". This just follows the common naming convention that most React developers use.

- Now that we've set up our state, whenever we want to access the state value, we can reference `temperature`, just as we've done within the <h2> tag.
- And whenever we want to update the state, we will simply call `setTemperature`. Let's do that now.

- Since our state exists only within our App component, we will need to move the `increaseTemperature` and `decreaseTemperature` functions inside of `App`.
    - *Move the global functions inside the body of `App()`, below the `useState()` call.*
- Currently these functions both directly modify the value of `temperature`. As mentioned, when working with state we never do this, as it won't actually work. So, let's rewrite this to use the `setTemperature` function instead.
    - *Inside the body of `increaseTemperature`:*
    - *Clear the body of function.*
    - *Write*: `setTemperature(`
    - Inside `setTemperature`, we can pass in the value that we want to set the temperature to. In this case, that's the current temperature plus one.
    - `setTemperature(temperature + 1)`

- Now, if we press the "+" button, we can see that the UI updates. Because the state change is **managed through React** via the `setTemperature` function, our App component now knows it should re-render when the temperature is changed.

- Let's do the same with the `decreaseTemperature` function:
    - *Inside the body of `decreaseTemperature`, write:* `setTemperature(temperature - 1)`

- Now, thanks to state, our app works as intended.
- In the next exercise, you'll get some practice doing this yourself.

Solution reference

React-8c | Exercise: Powers of two app

[Link to environment]

This exercise gives learners practice on creating and updating state in a React application.

src/App.jsx

- This is a simple app that displays the sequence of powers of two.
- The number 1 is displayed on the screen. Then, every time the "x2" button is clicked, the number displayed should double in value – to, 2, 4, 8, 16, 32, etc.
- Currently, the button does not do anything.

Exercise

- Your task is to use state to make this app function correctly.

Solve the exercise live before moving on.

Solution

- *Import* ***`useState`***

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

- *Create state within the body of* ***`App()`***

```
const [count, setCount] = useState(1)
```

- *Create a function called* ***`double()`*** *within the body of* ***`App()`***

```
function double() {
    setCount(count * 2)
}
```

- *Add an* ***`onClick`*** *handler to the button that calls the* ***`double`*** *function.*

```
<button onClick={double}>x2</button>
```

- *In the <h2> tag, access the value of* ***`count`***. *Use* ***`toLocaleString()`*** *in order to achieve number formatting with commas.*

```
<h2>{count.toLocaleString()}</h2>
```

[Solution reference]

Trainer demo | Changing state based on previous state

Link to environment

In this demo, you will show an alternative way of updating state in React, namely using a callback function to change state based on the previous state value.

App.jsx

- Here is a new version of the thermostat app.
- It now features a "Reset" button, which resets the temperature back to its initial value of 20 degrees.
- In the last version of this app, we saw that we can update the temperature by calling `setTemperature` with the new value that we want.
    - For example, in the `resetTemperature` function, we call `setTemperature` with the value 20.
- Similarly, in the `increaseTemperature` function, we set the temperature to the current value plus one.
    - One thing to note is that the new value of `temperature` depends on the previous value of `temperature`.
    - This solution is not necessarily wrong in this case - it works perfectly fine here. However, when the new state value depends on the previous state value in this way, the strategy we're currently using to update state can lead to unexpected behaviour in some circumstances.
- There is another way to update state in React.
- Instead of passing the new value directly to the `setTemperature` function, we could also pass a callback function.
    - *Change the body of `increaseTemperature` such that an arrow function is passed to `setTemperature`:*

```
function increaseTemperature() {
    setTemperature(() => {

    })
  }
```

    - One argument is passed to this callback function. That argument contains the previous state value, which we will call "previous".

```
function increaseTemperature() {
    setTemperature((previous) => {

    })
  }
```

    - Now that we have access to the previous value, we can manipulate it in whatever way we want.
    - Whatever is returned from this callback function will become the new state value. In this case, we want to increase the temperature by 1. So we will take the previous state value and return that plus one.

```
function increaseTemperature() {
    setTemperature((previous) => {
      return previous + 1
    })
  }
```

- Now the "+" button works in the same way as before.
- Let's do the same with the `decreaseTemperature` function.

```
function decreaseTemperature() {
    setTemperature((previous) => {
      return previous - 1
    })
  }
```

- Everything works as expected.
- Now we have seen an alternative way of updating state via callback functions.
- A good rule of thumb is this: **whenever the new value of state depends on the previous value of state, it's a good idea to use a callback function to update state**. This will help you to avoid unexpected behaviour in some scenarios.

Solution reference