**useState Hook**

The state of your application is bound to change at some point. This could be the value of a variable, an object, or whatever type of data exists in your component.

To make it possible to have the changes reflected in the DOM, we have to use a React hook called useState. It looks like this:

import { useState } from "react";

function App() {

const [name, setName] = useState("Ihechikara");

const changeName = () => {

setName("Chikara");

};

return (

<div>

<p>My name is {name}</p>

<button onClick={changeName}> Click me </button>

</div>

);

}

export default App;

Let's look a bit more closely at what's going on in the code above.

import { useState } from "react";

To be able to use this hook, you have to import the **useState**hook from React. We are using a functional component called app.

const [name, setName] = useState("Ihechikara");

After that, you have to create your state and give it an initial value (or initial state) which is "Ihechikara". The state variable is called name, and setName is the function for updating its value.

Having a good understanding of some of the ES6 features will help you grasp the basic functionalities of React. Above, we used the destructuring assignment to assign an initial name value to the state in useState("Ihechikara").

return (

<div>

<p>My name is {name}</p>

<button onClick={changeName}> Click me </button>

</div>

);

}

Next, the DOM has a paragraph containing the name variable and a button which fires a function when clicked. The changeName() function calls the setName() function which then changes the value of the name variable to the value passed in to the setName() function.

The values of your state must not be hard coded. In the next section, you will see how to use the useState hook in forms.

For React beginners, note that you create your functions and variables before the return statement.

**How to Use the useState Hook in Forms**

This section will help you understand how to create state values for your forms and update them when you need to do so. The process is not so different from what we saw in the previous section.

As always, import the useState hook:

import { useState } from "react";

We will create the initial state like we did before. But in this case it is going to be an empty string since we are dealing with the value of an input element. Hard coding the value means the input will have that value whenever the page is reloaded. That is:

const [name, setName] = useState("");

Now that we've created the state, let's create the input element in the DOM and assign the name variable as its initial value. It looks like this:

return (

<div>

<form>

<input

type="text"

value={name}

onChange={(e) => setName(e.target.value)}

placeholder="Your Name"

/>

<p>{name}</p>

</form>

</div>

);

You will notice that we didn't create a function above the return statement to update the value of the state – but it is still okay if you decide to use that method.

Here, we use the onChange event listener which waits for any value change in the input field. Whenever there is a change, an anonymous function (which takes in the event object as a parameter) is fired which in turn calls the setName() function to update the name variable with the current value of the input field.

Here's what the final code looks like:

import { useState } from "react";

function App() {

const [name, setName] = useState("");

return (

<div>

<form>

<input

type="text"

value={name}

onChange={(e) => setName(e.target.value)}

placeholder="Your Name"

/>

<p>{name}</p>

</form>

</div>

);

}

export default App;

**What is the useEffect Hook?**

The Effect Hook, just like the name implies, carries out an effect each time there is a state change. By default, it runs after the first render and every time the state is updated.

In the example below, we create a state variable count with an initial value of zero. A button in the DOM will increase the value of this variable by one every time it is clicked. The useEffect hook will run every time the count variable changes and then log out some information to the console.

import { useState, useEffect } from "react";

function App() {

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

useEffect(() => {

console.log(`You have clicked the button ${count} times`)

});

return (

<div>

<button onClick={() => setCount(count + 1)}>Click me</button>

</div>

);

}

export default App;

The first line of code where you import the required hook(s) is always important if you are going to "hook" into this React feature. We imported the two hooks we used above:

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

Note that you can use the useEffect hook to achieve various effects like fetching data from an external API (which you will see in another section of this article), changing the DOM in your component, and so on.

**useEffect Dependencies**

But what happens if you want your effect to run only after the first render, or if you have multiple states and only want an after effect attached to one of the states?

We can do this by using a dependency array which is passed in as a second argument in the useEffect hook.

**How to run an effect once**

For the first example, we'll pass in an array that allows the useEffect hook to run only once. Here's an example of how that works:

import { useState, useEffect } from "react";

function App() {

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

useEffect(() => {

console.log(`You have clicked the button ${count} times`)

}, []);

return (

<div>

<button onClick={() => setCount(count + 1)}>Click me</button>

</div>

);

}

export default App;

The code above is the same as in the previous section, except that the useEffect hook accepts an empty array [] as a second argument. When we leave the array empty, the effect will only run once irrespective of the changes to the state it is attached to.

**How to attach an effect to a particular state**

import { useState, useEffect } from "react";

function App() {

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

useEffect(() => {

console.log(`You have clicked the first button ${count} times`);

}, [count]);

const [count2, setCount2] = useState(0);

useEffect(() => {

console.log(`You have clicked the second button ${count2} times`)

}, [count2]);

return (

<div>

<button onClick={() => setCount(count + 1)}>Click me</button>

<button onClick={() => setCount2(count2 + 1)}>Click me</button>

</div>

);

}

export default App;

In the code above, we created two states and two useEffect hooks. Each state has an after effect attached to it by passing the name of the state [count] and [count2] to the corresponding useEffect array dependency.

So when the state of count changes, the useEffect hook responsible for watching these changes will carry out any after effect assigned to it. Same applies to count2.

**How to Create Your Own Hooks**