

Using the State Hook

Hooks are a new addition in React 16.8. They let you use state and other React features without writing a class.

The [introduction page](#) used this example to get familiar with Hooks:

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

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

We'll start learning about Hooks by comparing this code to an equivalent class example.

Equivalent Class Example

If you used classes in React before, this code should look familiar:

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }
  handleClick() {
    this.setState({ count: this.state.count + 1 });
  }
  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={this.handleClick}>Click me</button>
      </div>
    );
  }
}
```



```

    count: 0
  };
}

render() {
  return (
    <div>
      <p>You clicked {this.state.count} times</p>
      <button onClick={() => this.setState({ count: this.state.count + 1 })}>
        Click me
      </button>
    </div>
  );
}
}

```

The state starts as `{ count: 0 }`, and we increment `state.count` when the user clicks a button by calling `this.setState()`. We'll use snippets from this class throughout the page.

Note

You might be wondering why we're using a counter here instead of a more realistic example. This is to help us focus on the API while we're still making our first steps with Hooks.

Hooks and Function Components

As a reminder, function components in React look like this:

```

const Example = (props) => {
  // You can use Hooks here!
  return <div />;
}

```

or this:

```

function Example(props) {

```



```
// You can use Hooks here!  
  
return <div />;  
}
```

You might have previously known these as “stateless components”. We’re now introducing the ability to use React state from these, so we prefer the name “function components”.

Hooks **don’t** work inside classes. But you can use them instead of writing classes.

What’s a Hook?

Our new example starts by importing the `useState` Hook from React:

```
import React, { useState } from 'react';  
  
function Example() {  
  // ...  
}
```

What is a Hook? A Hook is a special function that lets you “hook into” React features. For example, `useState` is a Hook that lets you add React state to function components. We’ll learn other Hooks later.

When would I use a Hook? If you write a function component and realize you need to add some state to it, previously you had to convert it to a class. Now you can use a Hook inside the existing function component. We’re going to do that right now!

Note:

There are some special rules about where you can and can’t use Hooks within a component. We’ll learn them in [Rules of Hooks](#).



Declaring a State Variable

In a class, we initialize the `count` state to `0` by setting `this.state` to `{ count: 0 }` in the constructor:

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }
}
```

In a function component, we have no `this`, so we can't assign or read `this.state`. Instead, we call the `useState` Hook directly inside our component:

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

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);
}
```

What does calling `useState` do? It declares a “state variable”. Our variable is called `count` but we could call it anything else, like `banana`. This is a way to “preserve” some values between the function calls — `useState` is a new way to use the exact same capabilities that `this.state` provides in a class. Normally, variables “disappear” when the function exits but state variables are preserved by React.

What do we pass to `useState` as an argument? The only argument to the `useState()` Hook is the initial state. Unlike with classes, the state doesn't have to be an object. We can keep a number or a string if that's all we need. In our example, we just want a number for how many times the user clicked, so pass `0` as initial state for our variable. (If we wanted to store two different values in state, we would call `useState()` twice.)

What does `useState` return? It returns a pair of values: the current state and a function that updates it. This is why we write `const [count, setCount] = useState()`. This is similar to



updates it. This is why we write `const [count, setCount] = useState()`. This is similar to

`this.state.count` and `this.setState` in a class, except you get them in a pair. If you're not familiar with the syntax we used, we'll come back to it [at the bottom of this page](#).

Now that we know what the `useState` Hook does, our example should make more sense:

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

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);
```

We declare a state variable called `count`, and set it to `0`. React will remember its current value between re-renders, and provide the most recent one to our function. If we want to update the current `count`, we can call `setCount`.

Note

You might be wondering: why is `useState` not named `createState` instead?

"Create" wouldn't be quite accurate because the state is only created the first time our component renders. During the next renders, `useState` gives us the current state. Otherwise it wouldn't be "state" at all! There's also a reason why Hook names *always* start with `use`. We'll learn why later in the [Rules of Hooks](#).

Reading State

When we want to display the current count in a class, we read `this.state.count`:

```
<p>You clicked {this.state.count} times</p>
```

In a function, we can use `count` directly:



```
<p>You clicked {count} times</p>
```

Updating State

In a class, we need to call `this.setState()` to update the `count` state:

```
<button onClick={() => this.setState({ count: this.state.count + 1 })}>  
  Click me  
</button>
```

In a function, we already have `setCount` and `count` as variables so we don't need this:

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

Recap

Let's now **recap what we learned line by line** and check our understanding.

```
1: import React, { useState } from 'react';  
2:  
3: function Example() {  
4:   const [count, setCount] = useState(0);  
5:  
6:   return (  
7:     <div>  
8:       <p>You clicked {count} times</p>  
9:       <button onClick={() => setCount(count + 1)}>  
10:        Click me
```



```
11:     </button>
12:   </div>
13: );
14: }
```

- **Line 1:** We import the `useState` Hook from React. It lets us keep local state in a function component.
- **Line 4:** Inside the `Example` component, we declare a new state variable by calling the `useState` Hook. It returns a pair of values, to which we give names. We're calling our variable `count` because it holds the number of button clicks. We initialize it to zero by passing `0` as the only `useState` argument. The second returned item is itself a function. It lets us update the `count` so we'll name it `setCount`.
- **Line 9:** When the user clicks, we call `setCount` with a new value. React will then re-render the `Example` component, passing the new `count` value to it.

This might seem like a lot to take in at first. Don't rush it! If you're lost in the explanation, look at the code above again and try to read it from top to bottom. We promise that once you try to "forget" how state works in classes, and look at this code with fresh eyes, it will make sense.

Tip: What Do Square Brackets Mean?

You might have noticed the square brackets when we declare a state variable:

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

The names on the left aren't a part of the React API. You can name your own state variables:

```
const [fruit, setFruit] = useState('banana');
```

This JavaScript syntax is called "array destructuring". It means that we're making two new variables `fruit` and `setFruit`, where `fruit` is set to the first value returned by `useState`, and `setFruit` is the second. It is equivalent to this code:

```
var fruitStateVariable = useState('banana'); // Returns a pair
```



```
var fruit = fruitStateVariable[0]; // First item in a pair
var setFruit = fruitStateVariable[1]; // Second item in a pair
```

When we declare a state variable with `useState`, it returns a pair — an array with two items. The first item is the current value, and the second is a function that lets us update it. Using `[0]` and `[1]` to access them is a bit confusing because they have a specific meaning. This is why we use array destructuring instead.

Note

You might be curious how React knows which component `useState` corresponds to since we're not passing anything like `this` back to React. We'll answer [this question](#) and many others in the FAQ section.

Tip: Using Multiple State Variables

Declaring state variables as a pair of `[something, setSomething]` is also handy because it lets us give *different* names to different state variables if we want to use more than one:

```
function ExampleWithManyStates() {
  // Declare multiple state variables!
  const [age, setAge] = useState(42);
  const [fruit, setFruit] = useState('banana');
  const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);
}
```

In the above component, we have `age`, `fruit`, and `todos` as local variables, and we can update them individually:

```
function handleOrangeClick() {
  // Similar to this.setState({ fruit: 'orange' })
  setFruit('orange');
}
```

You **don't have to** use many state variables. State variables can hold objects and arrays just fine, so you can still group related data together. However, unlike `this.setState` in a class,



time, so you can still group related data together. However, unlike `this.setState` in a class, updating a state variable always *replaces* it instead of merging it.



We provide more recommendations on splitting independent state variables [in the FAQ](#).

Next Steps

On this page we've learned about one of the Hooks provided by React, called `useState`. We're also sometimes going to refer to it as the "State Hook". It lets us add local state to React function components — which we did for the first time ever!

We also learned a little bit more about what Hooks are. Hooks are functions that let you "hook into" React features from function components. Their names always start with `use`, and there are more Hooks we haven't seen yet.

Now let's continue by [learning the next Hook: `useEffect`](#). It lets you perform side effects in components, and is similar to lifecycle methods in classes.

Is this page useful?  

[Edit this page](#)

[Previous article](#)

[Hooks at a Glance](#)

[Next article](#)

[Using the Effect Hook](#)

