**But what is a Hook?**

Hooks are functions that let you “hook into” React state and lifecycle features from function components. **Hooks don’t work inside classes** — they let you use React without classes. *Hooks* are a new addition in React 16.8.

React provides a few built-in Hooks like useState. You can also create your own Hooks to reuse stateful behavior between different components. We’ll look at the built-in Hooks first.

useState is a Hook. We call it inside a function component to add some local (local to that function) state to it. React will preserve this state between re-renders.

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>

);

}

In the example above, it is 0 because our counter starts from zero. Note that unlike this.state, the state here doesn’t have to be an object — although it can be if you want. The initial state argument is only used during the first render.

**What does calling useState do?**

It declares a “state variable” (count in example give).

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. (If we wanted to store two different values in state, we would call useState() twice.)

**What does useState return?**

useState returns a pair  (an array with two items):

1. the current state value
2. a function that lets you update 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.

You can call this function (setCount ) from an event handler or somewhere else. It’s similar to this.setState in a class, except it doesn’t merge the old and new state together. (We’ll show an example comparing useState to this.state in [Using the State Hook](https://reactjs.org/docs/hooks-state.html).)

**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.

**Where to use Hook?**

const Example = (props) => {

**// You can use Hooks here!**

return <div />;

}

function Example(props) {

**// You can use Hooks here!**

return <div />;

}

## **Reading State**

In a function, we can use count directly:

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

## **Updating State**

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>

### **Using Multiple State Variables**

function ExampleWithManyStates() {

**// Declare multiple state variables!**

const [age, setAge] = useState(42);

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

const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);

function handleOrangeClick() {

**// Similar to this.setState({ fruit: 'orange' })**

setFruit('orange');}

# **Effect Hook**

The Effect Hook lets you perform side effects in function components:

**Tip:** If you’re familiar with React class lifecycle methods, you can think of useEffect Hook as componentDidMount, componentDidUpdate, and componentWillUnmount combined.

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

function Example() {

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

// Similar to componentDidMount and componentDidUpdate:

useEffect(() => {

// Update the document title using the browser API

document.title = `You clicked ${count} times`;

});

return (

<div>

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

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

Click me

</button>

</div>

);

}

Example2:

useEffect(() => {

function handleStatusChange(status) {

setIsOnline(status.isOnline);

}

ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);

return () => {

ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);

};

});

**Rules of Hooks**

### Only Call Hooks at the Top Level

**Don’t call Hooks inside loops, conditions, or nested functions.** Instead, always use Hooks at the top level of your React function. By following this rule, you ensure that Hooks are called in the same order each time a component renders. That’s what allows React to correctly preserve the state of Hooks between multiple useState and useEffect calls.

### Only Call Hooks from React Functions

**Don’t call Hooks from regular JavaScript functions.** Instead, you can:

1. ✅ Call Hooks from React function components.
2. ✅ Call Hooks from custom Hooks

By following this rule, you ensure that all stateful logic in a component is clearly visible from its source code.

**ESLint plugin**

We released an ESLint plugin called [eslint-plugin-react-hooks](https://www.npmjs.com/package/eslint-plugin-react-hooks) that enforces these two rules. You can add this plugin to your project if you’d like to try it:

npm install eslint-plugin-react-hooks --save-dev