## **STATE AND PROPS**

### **Introduction**

The last lesson was a lot of theory. There is just a little bit more you have to understand to write your first React application. If you’ve done the previous lesson, you should now have a great foundation for the next concepts that you will learn. Make sure you pay good attention to this lesson because it covers one of the most important concepts to get you started with React.

Let’s dive right into it. Feel free to code along with this lesson. Typing the code out will help you remember it significantly better.

### **Props**

In the previous lesson, you learned a lot about components and how to structure your application component-based. But how do we pass values or even functionality between those components? Yes, you guessed it right, the answer is props.

Let’s take the following example:

// MyComponent.js import React, { Component } from "react"; class MyComponent extends Component { constructor(props) { super(props); } render() { return ( <div> <h1>{this.props.title}</h1> </div> ); } } export default MyComponent;

// App.js import React, { Component } from "react"; import MyComponent from "./MyComponent"; class App extends Component { constructor(props) { super(props); } render() { return ( <div> <MyComponent title="React" /> </div> ); } } export default App;

Let’s take a look at what is happening here. Above, there are two components, MyComponent and App. As you can see, MyComponent is a child component of App, which passes down a property called title. In this specific example, the title is set to “React”. In MyComponent, we can access the title (React) that has been passed down with the syntax this.props.title. The curly brackets around it are always needed when you write JavaScript in JSX. *IMPORTANT*: Make sure you pass props to the constructor of the child component (MyComponent) as well as the super() method, otherwise you will not be able to access this.props.title in MyComponent.

Now you might be wondering how props work with functions. Believe it or not, they work the same way!

// MyComponent.js import React, { Component } from "react"; class MyComponent extends Component { constructor(props) { super(props); } render() { return ( <div> <h1>{this.props.title}</h1> <button onClick={this.props.onButtonClicked}>Click Me!</button> </div> ); } } export default MyComponent;

// App.js import React, { Component } from "react"; import MyComponent from "./MyComponent"; class App extends Component { constructor(props) { super(props); this.onClickBtn = this.onClickBtn.bind(this); } onClickBtn() { console.log("Button has been clicked!"); } render() { return ( <div> <MyComponent title="React" onButtonClicked={this.onClickBtn} /> </div> ); } } export default App;

Ok, there is a little bit more going on here, but in the end, it works exactly as in the example before. First, there is the MyComponent, which is essentially the same. The only thing we added is the { this.props.onButtonClicked } as an onClick event to the component. You should already be familiar with this syntax. Now to the App component. First, we defined the function onClickBtn above the render method. After that, we passed this function down to our MyComponent as a property, which we named onButtonClicked (Of course, you could also name it onClickBtn, and then use the function in MyComponent.js by the name of onClickBtn, but we wanted to emphasize that you can rename the functions when passing them around as props). We do that the same way we passed the title value previously. Now the only thing we have to do is bind the method to this; we do that in the constructor method at the top of our component but below the super() call. The reason we have to bind the this keyword when passing a function to another component is that it needs to stay in the same context. So remember, you have to bind this for all functions in all class components when passing them to other components.

As you can see when you are passing many properties or functions to a component, it can get quite exhausting to always refer to them with this.props.someProperty. Destructuring to the rescue! We can alternatively write the above as follows:

// MyComponent.js import React, { Component } from "react"; class MyComponent extends Component { constructor(props) { super(props); } render() { const { title, onButtonClicked } = this.props; return ( <div> <h1>{title}</h1> <button onClick={onButtonClicked}>Click Me!</button> </div> ); } } export default MyComponent;

Here, we are destructuring title and onButtonClicked from this.props, which lets us refer to them with just their names. Make sure to destructure within the render method when using class components. In functional components (more on those later!), you would destructure outside of the return statement.

Great! Hopefully, you have a better understanding of the purpose and usage of props in React. Isn’t that difficult is it?!

### **State**

Another very important concept in React is “state”. In short, all values that can change over time are handled through state.

The following example shows how to define state in React:

import React, { Component } from "react"; class App extends Component { constructor() { super(); this.state = { count: 0, }; this.countUp = this.countUp.bind(this); } countUp() { this.setState({ count: this.state.count + 1, }); } render() { return ( <div> <button onClick={this.countUp}>Click Me!</button> <p>{this.state.count}</p> </div> ); } }

In the above component, we declared a piece of state. You always declare state in the constructor of a class component. Keep in mind that this will work differently when we cover functional components later. The setState method we call sets the state to a new value. In React, state is immutable. This means you should never change state directly because it can lead to unexpected behavior or bugs. What exactly does that mean? Never do something like this: this.state.count = 3, instead always use the [setState](https://reactjs.org/docs/react-component.html#setstate) method React provides built-in to modify the state. Keep this in mind, it can save you a lot of debugging when you are getting started with React. Take a look at [this article](https://lorenstewart.me/2017/01/22/javascript-array-methods-mutating-vs-non-mutating/), which does a great job in analyzing many popular JavaScript methods concerning mutability.

Further, it’s important to note that with class-based components, class methods such as our countUp() method need to be bound in our constructor (using bind), so they know what context to operate in. This is a result of how this works in JavaScript, see [this article](https://www.freecodecamp.org/news/this-is-why-we-need-to-bind-event-handlers-in-class-components-in-react-f7ea1a6f93eb/) for a great explanation on why *this* is the case.

As you have also seen in the render method, we access the current state through this.state.count. This syntax should look familiar to you by now because it is the same way we accessed props. And yes, you can also destructure the state.

### **State and props in functional components**

As you have learned, React provides the ability to create functional components. They work a little bit differently than the class components we’ve discussed thus far. The concept of props works mostly the same, with the only difference being that you don’t pass props as an argument to the constructor, instead you pass it as an argument to the component itself. Another major difference between functional and class components concerning props is the way you call the props. You learned that in a class component, the props that have been passed down from the parent component will be called like so: this.props.someFunction, however in functional components, we don’t need to call this, so we access props like so: props.someFunction. That’s the main difference with props between class and functional components. Let me give you a quick example to emphasize this.

// MyComponent.js import React from "react"; const MyComponent = (props) => { return <div>{props.title}</div>; }; export default MyComponent;

// App.js import React from "react"; import MyComponent from "./MyComponent"; const App = () => { return ( <div> <MyComponent title="Hello World" /> </div> ); }; export default App;

Of course, we can also destructure from props here. By adding this line const {title} = props above the return statement in MyComponent.js we can just refer to the title using {title}.

For the state of functional components, it is a little bit different. Before the end of 2018, developers were not able to access state in functional components at all. Functional components were therefore just used for returning JSX logic. However, with the introduction of React Hooks, this has changed. Now we can set and access state in functional components, and for newer code, they are often preferred over class components. You will be exposed to both kinds of components on the job, requiring us to be very familiar with both. The way React hooks work is the topic of one of the following lectures, so don’t worry about it too much right now. We are setting you up to say “Hey! I remember that” when it’s introduced.

### **Learning Outcomes**

* How to pass functions, state or values between components
* How to handle state and props in a functional component
* How to handle state and props in a class component
* Understanding setState()

### **Assignment**

1. Go to the React homepage and go through the [props section](https://reactjs.org/docs/components-and-props.html) again. Also, check out the next article, which talks about State and Lifecycle Methods. Don’t worry about the Lifecycle methods for now, we will discuss it in the upcoming lectures. However, if you are excited and want to learn more, feel free to explore it on your own.
2. Go through [these lessons](https://www.geeksforgeeks.org/reactjs-props-set-1/). There are a couple of them linked at the bottom of the page. Go through all of the ones relating to state and props to build a good foundation for the upcoming lectures. Of course, you’re welcome to explore more on your own, if you wish.

### **Additional Resources**

* [This course](https://scrimba.com/g/glearnreact) is a great way to get more familiar with all basic concepts of React in a very short time.
* An [article](https://reactjs.org/docs/thinking-in-react.html) by the React team on the thought process that should go into the developer’s minds while building a React based apllication.