**this.props**

Previously, you learned one way that components can interact: a component can *render* another component.

In this lesson, you will learn another way that components can interact: a component can *pass information* to another component.

Information that gets passed from one component to another is known as "props."

Click Next to enter props-land!

**Pass `props` to a Component**

You can *pass information* to a React component.

How? By giving that component an *attribute:*

<MyComponent foo="bar" />

Let's say that you want to pass a component the message, "This is some top secret info.". Here's how you could do it:

<Example message="This is some top secret info." />

As you can see, to pass information to a component, you need a *name* for the information that you want to pass.

In the above example, we used the name message. You can use any name you want.

If you want to pass information that isn't a string, then wrap that information in curly braces. Here's how you would pass an array:

<Greeting myInfo={["top", "secret", "lol"]} />

In this next example, we pass several pieces of information to <Greeting />. The values that aren't strings are wrapped in curly braces:

<Greeting name="Frarthur" town="Flundon" age={2} haunted={false} />

# Render a Component's props

You just passed information to a component's props object!

You will often want a component to displaythe information that you pass.

Here's how to make a component display passed-in information:

1 - Find the component class that is going to receive that information.  
2 - Include this.props.name-of-information in that component class's render method's return statement.

# Pass props From Component To Component

You have learned how to pass a prop to a component:

<Greeting firstName="Esmerelda" />

You have also learned how to access and display a passed-in prop:

render() { return <h1>{this.props.firstName}</h1>; }

The most common use of props is to pass information to a component, from a different component. You haven't done that yet, but it's very similar to what you've seen already.

In this exercise, you will pass a prop from one component to another.

**A curmudgeonly clarification about grammar:**  
You may have noticed some loose usage of the words prop and props. Unfortunately, this is pretty inevitable.

props is the name of the object that stores passed-in information. this.props refers to that storage object. At the same time, each piece of passed-in information is called a prop. This means that props could refer to two pieces of passed-in information, or it could refer to the object that stores those pieces of information :(

# Render Different UI Based on props

Awesome! You passed a prop from a component to a different component, accessed that prop from the receiver component, and rendered it!

You can do more with props than just display them. You can also use props to make decisions.

In the code editor, look at the Welcomecomponent class.

You can tell from this.props.name on line 5 that Welcome expects to receive a piece of information called name. However, Welcomenever renders this piece of information! Instead, it uses the information to make a decision.

<Welcome /> instances will render the text WELCOME "2" MY WEB SITE BABYYY!!!!!, unless the user is Mozart, in which case they will render the more respectful  
hello sir it is truly great to meet you  
here on the web.

The passed-in name is not displayed in either case! The name is used to decide what will be displayed. This is a common technique.

Select **Home.js** and look at the Homecomponent class. What will <Welcome />render to the screen?

# Put an Event Handler in a Component Class

You can, and often will, pass functions as props. It is especially common to pass event handler functions.

In the next lesson, you will pass an event handler function as a prop. However, you have to define an event handler before you can pass one anywhere. In this lesson, you will define an event handler function.

How do you define an event handler in React?

You define an event handler as a method on the component class, just like the rendermethod. Almost all functions that you define in React will be defined in this way, as methods in a class.

Take a look in the code editor. On lines 4 through 8, an event handler method is defined, with similar syntax as the render method. On line 12, that event handler method is attached to an event (a click event, in this case).

# Pass an Event Handler as a prop

Good! You've defined a new method on the Talker component class. Now you're ready to pass that function to another component.

You can pass a method in the exact same way that you pass any other information. Behold, the mighty JavaScript.

# Receive an Event Handler as a prop

Great! You just passed a function from <Talker /> to <Button />.

In the code editor, select **Button.js**. Notice that Button renders a <button></button> element.

If a user clicks on this <button></button>element, then you want your passed-in talkfunction to get called.

That means that you need to attach talk to the <button></button> as an event handler.

How do you do that? The same way that you attach any event handler to a JSX element: you give that JSX element a special attribute. The attribute's name should be something like onClick or onHover. The attribute's valueshould be the event handler that you want to attach.

# handleEvent, onEvent, and this.props.onEvent

Let's talk about naming things.

When you pass an event handler as a prop, as you just did, there are two names that you have to choose.

Both naming choices occur in the parentcomponent class - that is, in the component class that defines the event handler and passes it.

The first name that you have to choose is the name of the event handler itself.

Look at **Talker.js**, lines 6 through 12. This is our event handler. We chose to name it talk.

The second name that you have to choose is the name of the prop that you will use to pass the event handler. This is the same thing as your attribute name.

For our prop name, we also chose talk, as shown on line 15:

return <Button talk={this.talk} />;

These two names can be whatever you want. However, there is a naming convention that they often follow. You don't have to follow this convention, but you should understand it when you see it.

Here's how the naming convention works: first, think about what type of event you are listening for. In our example, the event type was "click."

If you are listening for a "click" event, then you name your event handler handleClick. If you are listening for a "keyPress" event, then you name your event handler handleKeyPress:

class MyClass extends React.Component { handleHover() { alert('I am an event handler.'); alert('I will be called in response to "hover" events.'); } }

Your prop name should be the word on, plus your event type. If you are listening for a "click" event, then you name your prop onClick. If you are listening for a "keyPress" event, then you name your prop onKeyPress:

class MyClass extends React.Component { handleHover() { alert('I am an event handler.'); alert('I will listen for a "hover" event.'); } render() { return <Child onHover={this.handleHover} />; } }

# this.props.children

Every component's props object has a property named children.

this.props.children will return everything in between a component's opening and closing JSX tags.

So far, all of the components that you've seen have been self-closing tags, such as <MyComponentClass />. They don't have to be! You could write <MyComponentClass></MyComponentClass>, and it would still work.

this.props.children would return everything in between <MyComponentClass> and </MyComponentClass>.

Look at **BigButton.js**. In Example 1,<BigButton>'s this.props.children would equal the text, "I am a child of BigButton."

In Example 2, <BigButton>'s this.props.children would equal a <LilButton /> component.

In Example 3, <BigButton>'s this.props.children would equal undefined.

If a component has more than one child between its JSX tags, then this.props.childrenwill return those children in an array. However, if a component has only one child, then this.props.children will return the single child, not wrapped in an array.

# defaultProps

Take a look at the Button component class.

Notice that on line 8, Button expects to receive a prop named text. The received text will be displayed inside of a <button></button>element.

What if nobody passes any text to Button?

If nobody passes any text to Button, then Button's display will be blank. It would be better if Button could display a default message instead.

You can make this happen by giving your component class a property named defaultProps:

class Example extends React.Component { render() { return <h1>{this.props.text}</h1>; } } Example.defaultProps;

The defaultProps property should be equal to an object:

class Example extends React.Component { render() { return <h1>{this.props.text}</h1>; } } // Set defaultProps equal to an object: Example.defaultProps = {};

Inside of this object, write properties for any default props that you'd like to set:

class Example extends React.Component { render() { return <h1>{this.props.text}</h1>; } } Example.defaultProps = { text: 'yo' };

If an <Example /> doesn't get passed any text, then it will display "yo."

If an <Example /> does get passed some text, then it will display that passed-in text.

# this.props Recap

That completes our lesson on props!

props is quite possibly the longest and most difficult lesson in all of our React courses. Congratulations on getting this far!

Here are some of the skills that you have learned:

* Passing a prop by giving an attribute to a component instance
* Accessing a passed-in prop via this.props.prop-name
* Displaying a prop
* Using a prop to make decisions about what to display
* Defining an event handler in a component class
* Passing an event handler as a prop
* Receiving a prop event handler and attaching it to an event listener
* Naming event handlers and event handler attributes according to convention
* this.props.children
* getDefaultProps

That's a lot! Don't worry if it's all a bit of a blur. Soon you'll get plenty of practice!