* Make sure that jsx elements is wrapped by a parent element or else it won’t render.
* ReactDOM offers a simple method to render React elements to the DOM which looks like this:
* ReactDOM.render(componentToRender, targetNode). Example of this is: ReactDOM.render(JSX, document.getElementById(‘root’));

Define an HTML Class in JSX

* No longer able to use the word ‘class’ in jsx. Instead use className.
* The naming convention for all HTML attr and event refs in jsx become camelCase. Example: in js onclick is onClick in jsx.

Learn About Self-Closing JSX Tags

* Another key difference of jsx to html is the closing tags.
* Any jsx element can be written with a self-closing tag

Create a Stateless Functional Component

* Everything in react is a component
* Two ways to create a React component.
* #1 use a js function - this creates a stateless functional component.
* Stateless component is one that can receive data and render it, but does not manage or track changes to that data.
* Simply write a js function to return a jsx or null. Note: React requires your function name to begin with a capital letter.

Create a React Component

* #2 way to create a react component using the es6 class syntax
* class Kitten extends React.Component {
* constructor(props) {
* super(props);
* }
* render() {
* return (
* <h1>Hi</h1>
* );
* }
* }
* It uses super() to call the constructor of the parent class.
* It is best practice to call a component’s constructor with super() and pass props to both.

Create a Component with Composition

* To render a component as a child in a React component, include the component name written as a custom HTML tag in JSX. Example:
* return (
* <App>
* <Navbar />
* <Dashboard />
* <Footer />
* </App>
* )

Use React to Render Nested Components

* Break down your UI into its basic building blocks, and those pieces become the components.

Compose React Components

* Rendering ES6 style class components within other components is no different than rendering the simple components you used in the last few challenges. You can render JSX elements, stateless functional components, and ES6 class components within other components.

Rendering a Class Component to the DOM

* ReactDOM.render(componentToRender, targetNode). The first argument is the React component that you want to render. The second argument is the DOM node that you want to render that component within.
* ReactDOM.render(<ComponentToRender />, targetNode). You use this syntax for both ES6 class components and functional components.

Create a component from scratch

* class MyComponent extends React.Component {  
   constructor(props){  
   super(props);  
   }  
    
   render(){  
   return (  
   <div>  
   <h1>My First React Component!</h1>  
   </div>  
   );  
   }  
  }  
    
  ReactDOM.render(<MyComponent />, document.getElementById('challenge-node'));

Pass PROPS to a Stateless Functional Component

* You use custom HTML attr that React provides support to pass a property.
* const Welcome = (props) => <h1>Hello, {props.user}!</h1>
* when dealing with stateless functional components, you basically consider it as an argument to a function which returns JSX. You can access the value of the argument in the function body.
* const CurrentDate = (props) => {  
   return (  
   <div>  
   { /\* change code below this line \*/ }  
   <p>The current date is: {props.date}</p>  
   { /\* change code above this line \*/ }  
   </div>  
   );  
  };  
    
  class Calendar extends React.Component {  
   constructor(props) {  
   super(props);  
   }  
   render() {  
   return (  
   <div>  
   <h3>What date is it?</h3>  
   { /\* change code below this line \*/ }  
   <CurrentDate date={Date()} />  
   { /\* change code above this line \*/ }  
   </div>  
   );  
   }  
  };

Pass an Array as Props

* To pass an array to a JSX element, it must be treated as JavaScript and wrapped in curly braces.
* Array methods such as join() can be used when accessing the property.
* const ChildComponent = (props) => <p>{props.colors.join(', ')}</p>
* const List= (props) => {  
   { /\* change code below this line \*/ }  
   return <p>{props.tasks.join(', ')}</p>  
   { /\* change code above this line \*/ }  
  };  
    
  class ToDo extends React.Component {  
   constructor(props) {  
   super(props);  
   }  
   render() {  
   return (  
   <div>  
   <h1>To Do Lists</h1>  
   <h2>Today</h2>  
   { /\* change code below this line \*/ }  
   <List tasks={['walk dog', 'study']}/>  
   <h2>Tomorrow</h2>  
   <List tasks={['swim', 'dine with friends', 'do some errands']}/>  
   { /\* change code above this line \*/ }  
   </div>  
   );  
   }  
  };

Use default props

* const ShoppingCart = (props) => {  
   return (  
   <div>  
   <h1>Shopping Cart Component</h1>  
   </div>  
   )  
  };  
  // change code below this line  
    
  ShoppingCart.defaultProps = { items: 0};
* React assigns default props if props are undefined, but if you pass null as the value for a prop, it will remain null.

Override Default Props

* The way to override the default props is to explicitly set the prop values for a component.
* Remember that the syntax to add a prop to a component looks similar to how you add HTML attributes. However, since the value for quantity is an integer, it won't go in quotes but it should be wrapped in curly braces. For example, {100}. This syntax tells JSX to interpret the value within the braces directly as JavaScript.
* const Items = (props) => {  
   return <h1>Current Quantity of Items in Cart: {props.quantity}</h1>  
  }   
    
  Items.defaultProps = {  
   quantity: 0  
  }  
    
  class ShoppingCart extends React.Component {  
   constructor(props) {  
   super(props);  
   }  
   render() {  
   { /\* change code below this line \*/ }  
   return <Items quantity = {10} />  
   { /\* change code above this line \*/ }  
   }  
  };

Use PropTypes to Define the Props You Expect

* It's considered a best practice to set propTypes when you know the type of a prop ahead of time. You can define a propTypes property for a component in the same way you defined defaultProps.
* Here's an example to require the type function for a prop called handleClick:  
    
  MyComponent.propTypes = { handleClick: PropTypes.func.isRequired }
* Note: As of React v15.5.0, PropTypes is imported independently from React, like this:  
    
  import React, { PropTypes } from 'react';

Access Props Using this.props

* Used in es6 class component
* To access props within a class component, you preface the code that you use to access it with this. For example, if an ES6 class component has a prop called data, you write {this.props.data} in JSX.

Review Using Props with Stateless Functional Components

* A stateless functional component is any function you write which accepts props and returns JSX. A stateless component, on the other hand, is a class that extends React.Component, but does not use internal state (covered in the next challenge). Finally, a stateful component is any component that does maintain its own internal state. You may see stateful components referred to simply as components or React components.
* A common pattern is to try to minimize statefulness and to create stateless functional components wherever possible. This helps contain your state management to a specific area of your application. In turn, this improves development and maintenance of your app by making it easier to follow how changes to state affect its behavior.
* class CampSite extends React.Component {  
   constructor(props) {  
   super(props);  
   }  
   render() {  
   return (  
   <div>  
   <Camper name={'Christian'} />  
   </div>  
   );  
   }  
  };  
  // change code below this line  
    
  const Camper = (props) => {  
   return (  
   <div>  
   <h1> Review on Props and Stateless Components </h1>  
   <p>{props.name}</p>  
   </div>  
   )  
  };  
    
  Camper.defaultProps = {  
   name: 'CamperBot'  
  }  
    
  Camper.propTypes = {  
   name: PropTypes.string.isRequired  
  }

Create a Stateful Component

* State consists of any data an application needs to know about, that can change over time.
* class StatefulComponent extends React.Component {
* constructor(props) {
* super(props);
* // initialize state here
* this.state = {
* name: 'Christian'
* }
* }
* render() {
* return (
* <div>
* <h1>{this.state.name}</h1>
* </div>
* );
* }
* };

Render State in the User Interface

* If a component is stateful, it will always have access to the data in state in its render() method. You can access the data with this.state.
* If you want to access a state value within the return of the render method, you have to enclose the value in curly braces.
* Note that if you make a component stateful, no other components are aware of its state. Its state is completely encapsulated, or local to that component, unless you pass state data to a child component as props. This notion of encapsulated state is very important because it allows you to write certain logic, then have that logic contained and isolated in one place in your code.
* class MyComponent extends React.Component {
* constructor(props) {
* super(props);
* this.state = {
* name: 'freeCodeCamp'
* }
* }
* render() {
* return (
* <div>
* { /\* change code below this line \*/ }
* <h1>{this.state.name}</h1>
* { /\* change code above this line \*/ }
* </div>
* );
* }
* };

Render State in the User Interface Another Way

* In the render() method, before the return statement, you can write JavaScript directly. For example, you could declare functions, access data from state or props, perform computations on this data, and so on. Then, you can assign any data to variables, which you have access to in the return statement.
* Because you can write JavaScript directly in this part of the code (before the return statement), you don't have to enclose this reference in curly braces.
* class MyComponent extends React.Component {
* constructor(props) {
* super(props);
* this.state = {
* name: 'freeCodeCamp'
* }
* }
* render() {
* // change code below this line
* const name = this.state.name;
* // change code above this line
* return (
* <div>
* { /\* change code below this line \*/ }
* <h1>{name}</h1>
* { /\* change code above this line \*/ }
* </div>
* );
* }
* };

Set State with this.setState

* React provides a method for updating component state called setState. You call the setState method within your component class like so: this.setState(), passing in an object with key-value pairs. The keys are your state properties and the values are the updated state data.
* this.setState({
* username: 'Lewis'
* });
* always use this.setState() when state changes occur. Also, you should note that React may batch multiple state updates in order to improve performance. What this means is that state updates through the setState method can be asynchronous.
* class MyComponent extends React.Component {
* constructor(props) {
* super(props);
* this.state = {
* name: 'Initial State'
* };
* this.handleClick = this.handleClick.bind(this);
* }
* handleClick() {
* // change code below this line
* this.setState({
* name: `React Rocks!`
* });
* // change code above this line
* }
* render() {
* return (
* <div>
* <button onClick={this.handleClick}>Click Me</button>
* <h1>{this.state.name}</h1>
* </div>
* );
* }
* };

Bind ‘this’ to a Class Method

* Class method typically needs the ‘this’ keyword so it can access properties on the class (both state and props) inside the scope of the method.
* One way is to bind this in the constructor so this becomes bound to the class methods when the component is initialized. this.handleClick = this.handleClick.bind(this)
* when you call a function like this.setState() within your class method, this refers to the class and will not be undefined.
* class MyComponent extends React.Component {
* constructor(props) {
* super(props);
* this.state = {
* itemCount: 0
* };
* // change code below this line
* this.addItem = this.addItem.bind(this);
* // change code above this line
* }
* addItem() {
* this.setState({
* itemCount: this.state.itemCount + 1
* });
* }
* render() {
* return (
* <div>
* { /\* change code below this line \*/ }
* <button onClick={this.addItem}>Click Me</button>
* { /\* change code above this line \*/ }
* <h1>Current Item Count: {this.state.itemCount}</h1>
* </div>
* );
* }
* };

Use State to Toggle an Element

* class MyComponent extends React.Component {
* constructor(props) {
* super(props);
* this.state = {
* visibility: false
* };
* // change code below this line
* this.toggleVisibility = this.toggleVisibility.bind(this);
* // change code above this line
* }
* // change code below this line
* toggleVisibility(){
* if(!this.state.visibility) {
* this.setState({
* visibility: true
* })
* } else {
* this.setState({
* visibility: false
* })
* }
* }
* // change code above this line
* render() {
* if (this.state.visibility) {
* return (
* <div>
* <button onClick={this.toggleVisibility}>Click Me</button>
* <h1>Now you see me!</h1>
* </div>
* );
* } else {
* return (
* <div>
* <button onClick={this.toggleVisibility}>Click Me</button>
* </div>
* );
* }
* }
* };