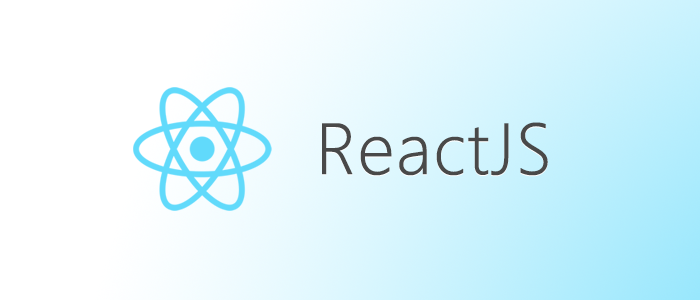
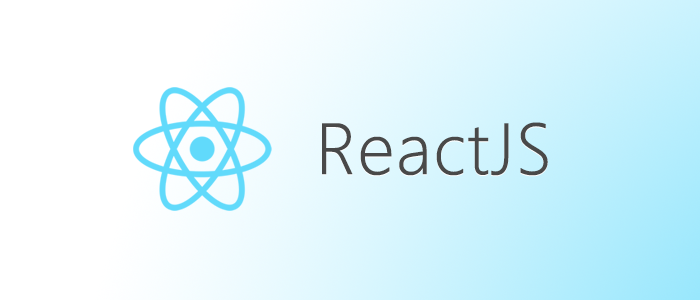
**Exercise07\_02\_01 – Step 1**

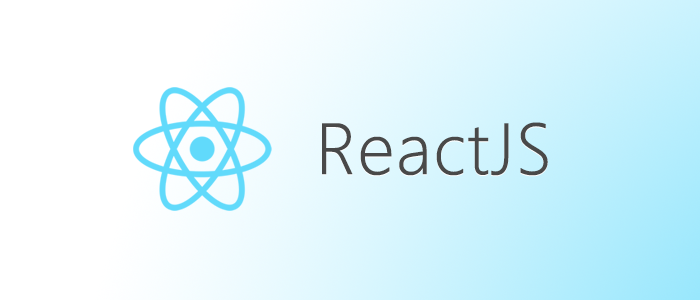
1. We can begin working with React in ***/src/index.js***. first let’s import what we need from our downloaded modules:  
   ***import React from 'react';  
   import ReactDOM from 'react-dom';***
2. Now let’s render something with a ***react-dom render()*** method. The method takes two parameters. The first ***creates*** an element, and we will leave its parameter blank for the moment. The second tells React ***where*** we want it to be rendered, which for the moment will be to an element with an ***id*** of ***root***, which we have not yet created:  
   ***ReactDOM.render(  
    React.createElement(...),  
    document.getElementById('root')  
   );***
3. We will put the element ***root*** in ***/views/index.ejs***. We will also put the existing content inside the new element:  
   <%- include('header') -%>  
    ***<div id="root"><%- content %></div>***<%- include('footer') -%>
4. In ***/src/index.js*** we can build out the ***createElement()*** method. The first argument is a string with the type of element to create. The second is the ***attributes*** for the element, which will be ***null*** for this particular case. The third argument will be the ***children*** we want the element to have. In this case, it will be element ***content***, or its ***innerHTML***:  
   ReactDOM.render(  
    ***React.createElement('h2', null, 'Hello React!'),*** document.getElementById('root')  
   );
5. Now let’s run our package.json ***dev*** script at a terminal or at IDE level as follows:  
   ***npm run dev***  
   This will run webpack in watch and development mode, which will do our translations. It will pack up our components based on the ***rules*** in ***webpack.config.js***, which right now is just our ***.js*** files. It will generate a ***bundle.js*** file and place it in ***/public*** for us. This will put it where our static files are being served by Node.js with Express.
6. We will need to get that ***bundle.js*** file into our HTML to use it. The best spot will be to put a ***<script>*** into ***footer.ejs*** before the closing ***</body>***.  
   ***<script src="/bundle.js" charset="utf-8"></script>***<body>  
   To test this we will also need our server running with an ***npm start*** from a terminal. So now we have both ***nodemon*** watching and rebooting our ***server*** when necessary, and ***webpack*** taking care of our ***JavaScript React*** resources. Run a browser, go to ***localhost:8080***, and we should see our first React element rendered.
7. Open a new tab in chrome and go to the react official site at ***Reactjs.com***. React is globally available here. Open Developer tools to Console, and we will directly enter our code as follows:  
    ***React.createElement('h2', null, 'Hello React!')***  
   The command produces an object with a bunch of properties. The most important properties are the ***type*** of the object, which is an h2, and the ***props*** property, which has a children property that holds the text that we just rendered.
8. Go back to our browser tab. Now let’s expand our Chrome Developer tools. Google ***react plugin*** and let’s install it in Chrome. It will put an icon into Chrome. Click on it and open Developer Tools. There will be a React tab to the right, click it. You will see the results of the runtime of the code. To the right is a convenient display of the ***props***.
9. We have written the code in straight JavaScript. Let’s modify it to use ***JSX*** instead of the React ***createElement()*** method. This will allow us to directly embed ***HTML*** in our JavaScript code:  
   ReactDOM.render(  
    ***<h2>Hello React with JSX!</h2>,*** document.getElementById('root')  
   );  
   Save and webpack will create a new bundle, then test with a browser regen and React tab open. It is the same and much more convenient.
10. Let’s put some JavaScript into our JSX code:  
     ***<h2>Hello React with JSX and a JS Expression: {Math.random()}</h2>,***Give this another test. Check out the ***props*** ***children*** in the React tab.
11. Let’s modify the code a bit to show some more power of mixing the React ***JavaScript*** and ***JSX***:  
    ***const color = (Math.random() > 0.5) ? 'green' : 'red';***  
      
    ReactDOM.render(  
     **<h2 style={{color: color}}>  
     Hello React with JSX!  
     </h2>,** document.getElementById('root')  
    );  
    Give this another test. Notice the style ***node*** (attribute) under ***props***
12. We have to be careful in JSX to remember to use the JavaScript conventions for ***variable*** ***names*** of HTML and CSS ***attributes***. So ***class*** becomes ***className***. Remove the ***conditional*** operator, then let’s give that a try with a code modification that uses a ***Bootstrap*** class:  
    ReactDOM.render(  
     ***<h2 style = {{color: color}} className='text-center'>***Give this another test and examine the various props.

**Exercise07\_02\_01 – Step 2**

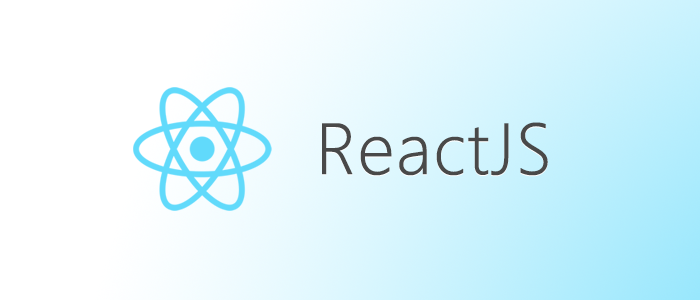
1. Make sure that we have run both our ***start*** and ***dev*** scripts to get nodemon and webpack running. Now let’s modify our existing code to build a React top level ***component*** named ***App***, as follows:  
   import ReactDOM from 'react-dom';  
     
   ***const App = () => {  
    return (  
    <h2 className="text-center">  
    Hello React Components!  
    </h2>  
    );  
   };***
2. We can now use the new component as if it was an HTML element, with the HTML tag closing ***slash***:  
   ***ReactDOM.render(  
    <App />,  
    document.getElementById('root')  
   );***Give this a browser test with localhost:8080 and inspect with the React tool. Notice the top level component ***<App>*** is an empty object. Open the node tree and click the child <h2> element. Notice its ***props***.
3. Make sure ESLint is turned on in Brackets preferences. To make the component more reusable, we can create a ***parameter*** to the function called ***props***, which we can use as a JavaScript ***expression*** inside the component function:  
   const App = (***props***) => {  
    return (  
    <h2 className="text-center">  
    ***{props.headerMessage}*** </h2>  
    );  
   };
4. We pass the argument to props as an attribu***t***e of the component tag in the ***render()*** method:  
   ReactDOM.render(  
   ReactDOM.render( <App ***headerMessage="Hello props!"*** />, document.getElementById('root')  
   );  
   Give this a browser test. Notice there is an ESLint indicator.
5. We have an ***ESLint*** error message about ***props validation***. The application is appears to be working fine and is rendering, but we should take care of the error. This error is telling us that any prop we use has to be part of a validation, which validates the ***data types*** of the props. First we need to get a module from React to take care of the problem. At a terminal or at IDE level, install it as follows:  
   ***npm install prop-types -save***  
   Check the installation in package.json.
6. First we need to get the module available in our code:  
   import ReactDOM from 'react-dom';  
   ***import PropTypes from 'prop-types';***
7. We do this just above the ***render()*** method with a ***propTypes*** property. It gives the ***data type***, and also sets whether a prop is ***required*** at all times:  
   ***App.propTypes = {  
    headerMessage: PropTypes.string.isRequired  
   };***ReactDOM.render(  
   Save this, check ESLint, and now we have prop validation and our error should now be gone.
8. We can make this even better by setting a ***default*** prop in case we use the component and do not pass a ***required*** prop:  
   ***App.defaultProps = {  
    headerMessage: 'Default prop'  
   };***Remove the headerMessage prop from the ***<App />*** component attribute and give this a test. If it works, put back the original attribute. Now, if desired, set the default prop to the ***empty string***, which will give us good protection if we use the component without a particular prop.

**Exercise07\_02\_01 – Step 3**

1. Make sure that both the ***start*** and ***dev*** scripts are running. Now let’s add some more JSX code to our component by adding another ***<div>*** below the ***<h2>*** with some placeholder content:  
   const App = (props) => {  
    return (  
    <h2 className="text-center">  
    {props.headerMessage}  
    </h2>  
    ***<div>  
    ...  
    </div>***  
    );  
   };  
   Save this, and we notice that webpack gives us an error message about wrapping elements in an ***enclosing*** HTML tag. This is a ***JSX parsing*** error, because it now looks like two React ***createElement()*** calls when parsed.
2. We fix this by ***wrapping*** the two elements into a single ***enclosing*** ***top-level*** element. We will use a ***<div>***:  
    ***<div>*** <h2 className="text-center">  
    {props.headerMessage}  
    </h2>  
    <div>  
    ...  
    </div>  
    ***</div>***Save this, webpack regenerates the bundle without error. Test in the browser.
3. Let’s define our new ***Header*** component:  
   ***const Header = () => {  
    return (  
    <h2 className="text-center">  
    props.headerMessage  
    </h2>  
    );  
   };***This gives us some ESLint notifications, because ***props*** is not defined as a parameter.
4. We can set the parameter by using a ***destructure*** from a JSON object, which contains whatever ***props*** are passed in. Then we can use that for the content of the ***<h2>*** :  
   ***const Header = ({ message }) => {  
    return (  
    <h2 className="text-center">  
    {message}  
    </h2>  
    );  
   };***
5. Let’s employ *composability* and use our new Header component it in our top level ***App*** component. We have no props in this element, so we can remove the ***props*** parameter. We will give it a ***message*** prop:  
   const App = () => {  
    return (  
    <div>  
    ***<Header message="Naming Contests" />*** <div>  
    ...  
    </div>  
    </div>  
    );  
   };  
   if we save and refresh the app, it is working but we still have one remaining ESLint error about ***props*** ***validation***.
6. Let’s cut our ***App.defaultProps*** object, it is not needed, we have no props in the ***<App />*** component.  
   Testing shows us to be still good, and we still have our remaining ESLint error.
7. We will move our ***App.propTypes*** object up to just below the ***Header*** component definition. We will ***rename*** and modify its ***prop*** name to fit with the new component structure:  
   ***Header.***propTypes = {  
    ***message:*** PropTypes.string  
   };  
   Let’s give this a browser test.
8. We should give the top-level enclosing element a class, so we can hook it with CSS later. Because JSX is JavaScript, we need to use the JavaScript form ***className***:  
   const App = () => {  
    return (  
    <div ***className="App"***>  
   Test this with the React tab open. The class name should reflect in our node tree..

**Exercise07\_02\_01 – Step 4**

1. React is ***opinionated***; it is considered a Best Practice to place components into individual ***modules***. Let’s build a new folder under ***/src*** named **components**. Create a file in it named ***Header.js***. We will cut our ***Header*** component definition, along with its ***propTypes*** object, out of index.js and paste it into the file. We will also need to ***export*** it, as well as ***import*** the React modules and ***prop-types***:  
   ***import React from 'react';  
   import PropTypes from 'prop-types';  
     
   const Header = ({ message }) => {  
    return (  
    <h2 className="text-center">  
    {message}  
    </h2>  
    );   
   };  
     
   Header.propTypes = {  
    message: PropTypes.string.isRequired  
   };  
     
   export default Header;***
2. Now we need to ***import*** the Header code into ***/src/index.js***:  
   import PropTypes from 'prop-types' and we can get rid of the prop-types import, we are not using it.:  
   ***import Header from './components/Header'***Save this and test in the browser.
3. Now let’s get the ***App*** component, along with the ***Header*** import, into another component file called ***App.js***. We will also need a copy of just the ***React*** import. Be sure to modify the ***path*** in the Header import. We will need to ***export*** the component as well:  
   ***import React from 'react';  
   import ReactDOM from 'react-dom';  
   import Header from './Header'  
     
   const App = () => {  
    return (  
    <div className="App">  
    <Header message="Naming Contests" />  
    <div>  
    ...  
    </div>  
    </div>  
    );  
   };  
     
   export default App;***
4. Now we need to ***import*** the App code into ***/src/index.js***:  
   import ReactDOM from 'react-dom';  
   ***import App from './components/app';***Save this and test in the browser. Everything should remain as it was.

**Exercise07\_02\_01 – Step 5**

1. Our components so far are ***stateless***, they are presentational only. We will need our App.js ***App*** component to be a ***stateful*** component and use React state. We will use class syntax with inheritance. The component will inherit from the React.Component class. Let’s go to and convert that into a class-based component:  
   import React from 'react';  
   import Header from './Header';  
    ***class App extends React.Component {  
    render() {  
    return (   
    <div className="App">  
    <Header message="Naming Contests" />  
    <div>  
    ...  
    </div>  
    </div>  
    );  
    }  
   }***export default App;  
   Save this and check for any syntax or ESLint problems.
2. One way of making this stateful is to use a constructor on the class to introduce state. We can set a property and then use it when the object is instantiated:  
   class App extends React.Component {  
    ***constructor(props) {  
    super(props);  
    this.state = { test: 42 };  
    }*** render() {  
    return (   
    <div className="App">  
    <Header message="Naming Contests" />  
    <div>  
    ***{this.state.test}*** </div>   
    </div>  
    );  
    }  
   }Save this and test in the browser; we should have our state variable value displayed.
3. Because we are using at least Stage 2 ES6, we can do this without a constructor. We can make use of a ***class*** ***property*** (static):  
   class App extends React.Component {  
    ***state = { test: 37 };***  
    render() {  
    return (   
    <div className="App">  
    <Header message="Naming Contests" />  
    <div>  
    {this.state.test}  
    </div>   
    </div>  
    );  
    }  
   }  
   Save this and test in the browser; we should have our new state variable value displayed.
4. Now we can employ the state for some useful work. We will put the Header message onto the state:  
   class App extends React.Component {  
    ***state = {   
    pageHeader: 'Naming Contests'   
    };***  
    render() {  
    return (   
    <div className="App">  
    <Header message="Naming Contests" />  
    <div>  
    {this.state.***pageHeader***}  
    </div>   
    </div>  
    );  
    }  
   }  
   Save this and test in the browser; we should have our new state variable value displayed. Take a look at the ***<App />*** element in React tools; we now have ***State*** showing on the right.
5. Let’s dig deeper into the Developer Tools. We can see that the ***<App />*** component has been set equal to **$r**. It is accessible in the Console tab; go to the Console and enter ***$r***. We can read a lot about the element here. Open it up by clicking on any of the objects here: we can see the state. We can also do this by entering ***$r.state***.  
     
   Now let’s set the state from the Console as follows:  
   ***$r.setState({ pageHeader: ‘Test . . .’ })***Once we hit enter, React will detect the change of state and re-render the DOM for us. We will use this in code later to change contest names.