Reacting to React

Getting Started

1. Create a new project with Create React App.
2. **Delete the src directory and all of its contents.**
3. Create a new src directory and a new index.js file inside of src.
4. Create a new directory named components inside of src.

You should now have a project structure that resembles the following:

node\_modules/

public/

src/

|- components/

|- index.js

.gitignore

package-lock.json

package.json

README.md

Before starting the following tasks, make sure that the **Chrome Developer Tools** are open in your "localhost:3000" tab, and that you also have the **React Developer Tools** installed. You should be checking these as much as possible to not only troubleshoot these tasks, but to confirm your understanding of what you are doing.

Simple Tasks a la React

Each time you save your files while completing these tasks, check the terminal and browser for compilation errors, and pay attention to what they are telling you. For example, if something is "undefined", why does React think that? What *should* be defined, and where?

1. React is component-based, and each component should be separated into its own file. Start this exercise by creating an "App.jsx" component in your src/components directory. Then, create a functional component named "App", and define a simple JSX element inside of that component (*Hint: start with something obvious, like a <h1> element*). Then, return that element from the App component.
2. Try to render your new App component from index.js. What happens in your terminal and browser? Fix any issues that arise.
3. Now that you have your basic App component rendered in the browser through index.js, create a property of your choosing on App in index.js. The value for that property should be a string. Pass in your *props* to App and replace your original <h1> text with the property value you just passed in to the component. Your browser should now display the property value you passed into the component.
4. Time to convert your functional component into a class-based component. Without changing any actual content, comment out your functional App component and create a new class-based App component that renders the same *props* as the functional one did.
5. Now that your App is a class-based component, you can initialize *state*. Do that, and define a new property called "text" with some string as the value.
6. Your new component should display both values from props and state. (*Hint: you have two options here - you can either concatenate these strings in the same pair of JSX curly braces, or you can place two sets of braces for each string. Your choice! Choose wisely.*)
7. Add an input element to your App. Use the component's state to define its placeholder attribute. Add an "onChange" event to the input, and then *handle* the event to set a **new property** on the App's state (pro tip: name this something appropriate considering it is an *input* event). Replace any "wrapping" divs with the <React.Fragment> tag. How does that change the ReactDOM? Why might that be important?

**NOTE: Any methods you define on the class *(i.e., between the constructor and render methods)* should have this structure:**

handleEvent(params) {

*// do something useful*

}

**HOWEVER: Any event handlers you define in JSX elements should use arrow notation (and the concise body syntax, where applicable):**

onEvent={(params) => this.handleEvent(params)}

1. Add a button beneath that input. When clicked, the button should invoke a method defined on the class to toggle a "hasLoaded" state property, which has a default value of false.
2. Before you return your elements to be rendered, check whether or not the hasLoaded property is true. If it is, return the elements that you have already defined; otherwise, return a new heading with the text "Loading..." and a button beneath it that is identical to the one you created in step 8. What happens? Which React elements are displayed first, and why?
3. If you can answer the questions at the end of the previous task, you already have an intuitive understanding of the lifecycle of a React Component. Given this, you can start thinking about invoking methods when these lifecycle events take place. So, write a new method with the name componentDidMount, which will set the state property "hasLoaded" to true. Now what happens when the application re-renders? Which React elements are displayed first, and why?

**BONUS:**

You should have three state properties at this point. Set the one that contains just a string (that was being used for your <h1> in the first few tasks) to the *props* value that you passed in to <App /> in step 3. Then, for the time being, remove the *props* parameters from the constructor and super methods. What happens?

Manually revert your state property back to a regular string. *Now what happens?* What's going on here?