ReactDOM

Let's examine the code that you just wrote in the last exercise.

```
ReactDOM.render(<h1>Render me!</h1>, document.getElementById('app'));
```

You can see something called ReactDOM. What's that?

ReactDOM is the name of a JavaScript library. This library contains several React-specific methods, all of which deal with the DOM in some way or another.

We'll talk more later about how ReactDOM got into your file. For now, just understand that it's yours to use.

Move slightly to the right, and you can see one of ReactDOM's methods: ReactDOM.render().

ReactDOM.render() is the most common way to *render* JSX. It takes a JSX expression, creates a corresponding tree of DOM nodes, and adds that tree to the DOM. That is the way to make a JSX expression appear onscreen.

Move to the right a little more, and you come to this expression:

```
<h1>Hello world</h1>
```

This is the first *argument* being passed to ReactDOM.render(). ReactDOM.render()'s first argument should be a JSX expression, and it will be rendered to the screen.

We'll discuss the second argument in the next exercise!

🪂 Further readings

Move to the right a little more, and you will see this expression:

```
document.getElementById('app')
```

You just learned that ReactDOM.render() makes its *first* argument appear onscreen. But *where* on the screen should that first argument appear?

The first argument is *appended* to whatever element is selected by the *second* argument.

In the code editor, select index.html. See if you can find an element that would be selected by document.getElementById('app').

That element acted as a *container* for ReactDOM.render()'s first argument! At the end of the previous exercise, this appeared on the screen:

```
<main id="app">
  <h1>Render me!</h1>
</main>
```

🛸 Passing a variable to DOM

ReactDOM.render()'s first argument should *evaluate* to a JSX expression, it doesn't have to literally *be* a JSX expression.

The first argument could also be a variable, so long as that variable evaluates to a JSX expression.

In this example, we save a JSX expression as a *variable* named toDoList. We then pass toDoList as the first argument to ReactDOM.render():

```
const toDoList = (
        <ol>
          <li>Learn React</li>
          <li>Become a Developer</li>
        </ol>
    );
    ReactDOM.render(  toDoList,   document.getElementById('app'));
```

🍣 The Virtual DOM

One special thing about ReactDOM.render() is that it *only updates DOM elements that have changed*.

That means that if you render the exact same thing twice in a row, the second render will do nothing:

```
const hello = <h1>Hello world</h1>;

// This will add "Hello world" to the screen:
ReactDOM.render(hello, document.getElementById('app'));

// This won't do anything at all:
ReactDOM.render(hello, document.getElementById('app'));
```

This is significant! Only updating the necessary DOM elements is a large part of what makes React so successful.

React accomplishes this thanks to something called *the virtual DOM.* Before moving on to the end of the lesson, read this article about the Virtual DOM.

🎈 JSX Recap

Congratulations! You've learned to create and render JSX elements! This is the first step towards becoming fluent in React.

In the next lesson, you'll learn some powerful things that you can do with JSX, as well as some common JSX issues and how to avoid them.