**Why React?**

React.js is a JavaScript library. It was developed by engineers at Facebook.

Here are just a few of the reasons why people choose to program with React:

* React is *fast*. Apps made in React can handle complex updates and still feel quick and responsive.
* React is *modular*. Instead of writing large, dense files of code, you can write many smaller, reusable files. React's modularity can be a beautiful solution to JavaScript's [maintainability problems](https://en.wikipedia.org/wiki/Spaghetti_code).
* React is *scalable*. Large programs that display a lot of changing data are where React performs best.
* React is *flexible*. You can use React for interesting projects that have nothing to do with making a web app. People are still figuring out React's potential. [There's room to explore.](https://medium.mybridge.co/22-amazing-open-source-react-projects-cb8230ec719f#.o5umedb6v)
* React is *popular*. While this reason has admittedly little to do with React's quality, the truth is that understanding React will make you more employable.

If you are new to React, then this course is for you! No prior React knowledge is expected. We will start at the very beginning and move slowly.

If you are new to JavaScript, then consider taking [our JavaScript course](https://www.codecademy.com/learn/javascript) and then returning to React.

The three Codecademy React courses are not a high-level overview. They are a deep dive. Take your time! By the end, you will be ready to program in React with a real understanding of what you're doing.

# Hello World

Take a look at the following line of code:

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

What kind of weird hybrid code is that? Is it JavaScript? HTML? Or something else?

It seems like it must be JavaScript, since it starts with const and ends with ;. If you tried to run that in an HTML file, it wouldn't work.

However, the code also contains <h1>Hello world</h1>, which looks exactly like HTML. Thatpart wouldn't work if you tried to run it in a JavaScript file.

What's going on?

# The Mystery, Revealed

Good!

Take another look at the line of code that you wrote.

Does this code belong in a JavaScript file, an HTML file, or somewhere else?

The answer is...a JavaScript file! Despite what it looks like, your code doesn't actually contain any HTML at all.

The part that looks like HTML, <h1>Hello world</h1>, is something called JSX.

Click Next to learn about JSX.

# What is JSX?

JSX is a syntax extension for JavaScript. It was written to be used with React. JSX code looks a lot like HTML.

What does "syntax extension" mean?

In this case, it means that JSX is not valid JavaScript. Web browsers can't read it!

If a JavaScript file contains JSX code, then that file will have to be compiled. That means that before the file reaches a web browser, a JSX compiler will translate any JSX into regular JavaScript.

Codecademy's servers already have a JSX compiler installed, so you don't have to worry about that for now. Eventually we'll walk through how to set up a JSX compiler on your personal computer.

# JSX Elements

A basic unit of JSX is called a JSX element.

Here's an example of a JSX element:

<h1>Hello world</h1>

This JSX element looks exactly like HTML! The only noticeable difference is that you would find it in a JavaScript file, instead of in an HTML file.

# JSX Elements And Their Surroundings

JSX elements are treated as JavaScript expressions. They can go anywhere that JavaScript expressions can go.

That means that a JSX element can be saved in a variable, passed to a function, stored in an object or array...you name it.

Here's an example of a JSX element being saved in a variable:

const navBar = <nav>I am a nav bar</nav>;

Here's an example of several JSX elements being stored in an object:

const myTeam = { center: <li>Benzo Walli</li>, powerForward: <li>Rasha Loa</li>, smallForward: <li>Tayshaun Dasmoto</li>, shootingGuard: <li>Colmar Cumberbatch</li>, pointGuard: <li>Femi Billon</li> };

# Attributes In JSX

JSX elements can have attributes, just like HTML elements can.

A JSX attribute is written using HTML-like syntax: a name, followed by an equals sign, followed by a value. The value should be wrapped in quotes, like this:

my-attribute-name="my-attribute-value"

Here are some JSX elements with attributes:

<a href="http://www.example.com">Welcome to the Web</a>; const title = <h1 id="title">Introduction to React.js: Part I</h1>;

A single JSX element can have many attributes, just like in HTML:

const panda = <img src="images/panda.jpg" alt="panda" width="500px" height="500px" />;

# Nested JSX

You can nest JSX elements inside of other JSX elements, just like in HTML.

Here's an example of a JSX <h1> element, nested inside of a JSX <a> element:

<a href="https://www.example.com"><h1>Click me!</h1></a>

To make this more readable, you can use HTML-style line breaks and indentation:

<a href="https://www.example.com"> <h1> Click me! </h1> </a>

If a JSX expression takes up more than one line, then you must wrap the multi-line JSX expression in parentheses. This looks strange at first, but you get used to it:

( <a href="https://www.example.com"> <h1> Click me! </h1> </a> )

Nested JSX expressions can be saved as variables, passed to functions, etc., just like non-nested JSX expressions can! Here's an example of a nested JSX expression being saved as a variable:

const theExample = ( <a href="https://www.example.com"> <h1> Click me! </h1> </a> );

# JSX Outer Elements

There's a rule that we haven't mentioned: a JSX expression must have exactly one outermost element.

In other words, this code will work:

const paragraphs = ( <div id="i-am-the-outermost-element"> <p>I am a paragraph.</p> <p>I, too, am a paragraph.</p> </div> );

But this code will not work:

const paragraphs = ( <p>I am a paragraph.</p> <p>I, too, am a paragraph.</p> );

The first opening tag and the final closing tag of a JSX expression must belong to the same JSX element!

It's easy to forget about this rule, and end up with errors that are tough to diagnose.

If you notice that a JSX expression has multiple outer elements, the solution is usually simple: wrap the JSX expression in a <div></div>.

# Rendering JSX

You've learned how to write JSX elements! Now it's time to learn how to render them.

To render a JSX expression means to make it appear onscreen.

# ReactDOM.render() II

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 whereon 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>

# 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](https://www.codecademy.com/articles/react-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.