Learn to use JSX, the basic syntax of React.js

INTRO TO JSX
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.
- 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.
- 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 and then returning to React.

The 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. *That* part wouldn't work if you tried to run it in a JavaScript file.

What's going on?

Instructions

1.

Copy the following line into **app.js**:

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

Click Run when you're finished.

Checkpoint 2 Passed

Hint

Copy the following code:

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

Then, paste it into **app.js**.

app.js

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

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.

app.js

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

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.

Instructions

Click Next to continue.

app.js

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

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.

Instructions

1.

In **app.js**, write a JSX `<p></p>` element containing the text, `Hello world`. Use the example code above as a guide.

Checkpoint 2 Passed

Hint

If you wanted to create a `<p></p>` element that contained "Goodbye, Earth!", you would do:

```
<p>Goodbye, Earth!</p>
```

Your code will look similar.

app.js

```
<P>Hello world</p>
```

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>
};
```

Instructions

1.

Create a JSX `<article></article>` element. Save it in a variable named `myArticle`.

Checkpoint 2 Passed

Hint

To save a JSX `<button></button>` element into a variable called `myButton`, you'd do something like this:

```
const myButton = <button></button>;
```

Your solution will look similar, but will replace `myButton` with `myArticle` and replace `button` with `article`.

app.js

```
const myArticle = <article>Hello World</article>
```

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' />;
```

Instructions

1.

Declare a constant named `p1`.

Set `p1` equal to a JSX `<p></p>` element. Write the word `foo` in between the `<p></p>` tags.

Checkpoint 2 Passed

Hint

To set `button1` to a JSX `<button></button>` element that has `Hello` in between the tags, you would write:

```
const button1 = <button>Hello</button>;
```

Your solution will look similar to this.

2.

On the next line, declare a constant named `p2`.

Set `p2` equal to another JSX `<p></p>` element. Write the word `bar` in between the `<p></p>` tags.

Checkpoint 3 Passed

Hint

This will have the same format as the previous solution, but will use `p2` instead of `p1` and `bar` instead of `foo`.

3.

Give the first `<p></p>` an `id` attribute of `'large'`.

Give the second `<p></p>` an `id` attribute of `'small'`.

Checkpoint 4 Passed

Hint

To give a `<button></button>` element an `id` attribute of `mybutton`, you would do something like this:

```
<button id='mybutton'></button>
```

Your solution will look similar but will use your `<p></p>` elements instead.

app.js

```
const p1 = <p id='large'>foo</p>;
const p2 = <p id='small'>bar</p>;
```

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>
);
```

Instructions

1.

Declare a new variable named `myDiv`. Set `myDiv` equal to a JSX `<div></div>` element.

Wrap the `<div></div>` in parentheses, and use indentation and line breaks like in the examples. In between the `<div></div>` tags, *nest* an `<h1></h1>` containing the text `Hello world`.

Checkpoint 2 Passed

Hint

If you wanted to declare a new variable, `mySpan`, and nest an `<h2></h2>` element inside, you might do something like this:

```
const mySpan = (
  <span>
    <h2>Hello                                                  world</h2>
  </span>
);
```

Your solution will look similar.

app.js

```
const myDiv = (
```

```
  <div>
    <h1>Hello world</h1>
  </div>
);
```

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>`.

Instructions

1.

Your friend's blog is down! He's asked you to fix it.

You immediately diagnose the problem: a JSX expression with multiple outer elements.

Repair your friend’s broken code by wrapping their JSX in a `<div></div>`.

Checkpoint 2 Passed

Hint

JSX expressions can only have one top-level element, but this example has several. Put `<div>` at the start of the expression and `</div>` at the end to fix your friend’s problem.

blog.js

```
const blog = (
  <div>
    <img src="pics/192940u73.jpg" />
    <h1>
      Welcome to Dan's Blog!
    </h1>
    <article>
      Wow I had the tastiest sandwich today.  I <strong>literally</strong> almost freaked out.
    </article>
  </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.

Instructions

1.

The following code will render a JSX expression:

```
ReactDOM.render(<h1>Hello                    world</h1>, document.getElementById('app'));
```

Starting on line 5, carefully copy the code into the code editor. We’ll go over how it works in the next exercise.

JavaScript is case-sensitive, so make sure to capitalize ReactDOM correctly!

Checkpoint 2 Passed

Hint

Copy the following into the code editor on line 5:

```
ReactDOM.render(<h1>Hello                    world</h1>,
document.getElementById('app'));
```

This will render a JSX element to the screen.

app.js

```
import React from 'react';
import ReactDOM from 'react-dom';

// Copy code here:
ReactDOM.render(<h1>Hello world</h1>), document.getElementBy
Id('app'));
```

ReactDOM.render() I

Let's examine the code that you just wrote. Start in **previous.js**, on line 5, all the way to the left.

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!

Instructions

1.
Select **app.js**.

Starting on line 5, call `ReactDOM.render()`.

Pass in this expression as a first argument:

```
<h1>Render me!</h1>
```

Pass in this expression as a second argument:

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

```
Checkpoint 2 Passed
```

Hint
Make sure you've selected **app.js**.

On line 5, call `ReactDOM.render()` with two arguments:

1. The first argument should be `<h1>Render me!</h1>`.
2. The second should be `document.getElementById('app')`.

previous.js

```
import React from 'react';
import ReactDOM from 'react-dom';

// This is just an example, switch to app.js for the exercise.
ReactDOM.render(<h1>Hello world</h1>, document.getElementById('app'));
```

app.js

```
import React from 'react';
import ReactDOM from 'react-dom';

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

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 *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>
```

Instructions

1.

In **index.html**, replace this:

```
<main id="app"></main>
```

with this span:

```
<span id="container"></span>
```

Checkpoint 2 Passed

Hint

Open up **index.html**. Find the code that looks like this:

```
<main id="app"></main>
```

Delete it, and replace it with this:

```
<span id="container"></span>
```

2.

Select **app.js**.

You want `<h1>Render me!</h1>` to be appended to `<span id="container"></span>`.

On line 5, make that happen by changing the string passed to `document.getElementById()`.

Checkpoint 3 Passed

Hint

Open up **app.js**.

Find the code that looks like `document.getElementById('app')`. It's currently targeting an element with the ID `'app'`, but we just changed it to an element with ID `'container'`. What should `'app'` change to?

app.js

```
import React from 'react';
import ReactDOM from 'react-dom';

// Write code here:
ReactDOM.render(<h1>Render me!</h1>, document.getElementById
('container'));
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
```

```
  <link rel="stylesheet" href="/styles.css">
  <title>Learn ReactJS</title>
</head>

<body>
  <span id="container"></span>
  <script src="https://content.codecademy.com/courses/React/
react-course-bundle.min.js"></script>
  <script src="/app.compiled.js"></script>
</body>

</html>
```

Passing a Variable to ReactDOM.render()

`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')
);
```

Instructions

1.

On line 5, declare a variable named `myList`. Set `myList` equal to a JSX `<ul></ul>` element. Wrap your `<ul></ul>` in parentheses.

Add several `<li></li>` elements in between your `<ul></ul>` tags. Put some text in each `<li></li>`. Use line breaks and indentation similar to the above example.

Checkpoint 2 Passed

Hint

Here's an example of a list called `myList2` that will look similar to your solution:

```
const myList2 = (
  <ul>
    <li>Hello, world!</li>
    <li>Goodbye, Earth!</li>
    <li>Ahoy, planet!</li>
  </ul>
);
```

2.

At the bottom of the file, call `ReactDOM.render()`.

For `ReactDOM.render()`'s first argument, pass in the variable `myList`.

For `ReactDOM.render()`'s second argument, select an HTML element with an id of `app`.

Feel free to use the example code as a guide.

Checkpoint 3 Passed

Hint

You'll need to call `ReactDOM.render()` with two arguments. You've done something like this in previous exercises. Refer to the example code as a guide.

app.js

```
import React from 'react';
```

```
import ReactDOM from 'react-dom';

// Write code here:
const myList = (
  <ul>
    <li>Learn React</li>
    <li>Become a Developer</li>
  </ul>
);

ReactDOM.render(
  myList,
  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.