React 101
Codecademy
JSX

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.
- 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.

Hello World

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

Is it JS? HTML? Or something else?

This code will should not work either in JS or HTML. Then where does it belong???

The answer is …. a JS 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.

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.

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>RashaLoa</li>,
      smallForward: <li>Tayshaun Dasmoto</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”
```

Some examples:

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

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

Nested JSX

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

Example:

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

This can also be written as:

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

Whenever JSX expression takes up more than one line, we must wrap the multi-line JSX expression in parentheses. This is done to avoid JavaScript’s automatic semicolon insertion which will add semicolons to terminate statements according to rules specified, which is not something we want.

```
(
<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!

JSX Outer Elements

Rule: A JSX expression must have exactly one outermost element.

Example: This 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 won't:

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

The first opening tag and final closing tag must be same element. In case of nested JSX just simply add a as wrapper to all.

Rendering JSX

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

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


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

This code will give an output of Hello world on the browser.

ReactDOM.render()

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.

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.

The first argument being passed in ReactDOM.render() should be a JSX expression, and it will be rendered to the screen.

document.getElementById('app') ,is the second argument of *ReactDOM.render()* which tells where onscreen should that first argument appear. The first argument is appended to whatever element is selected by the second argument.

Passing a Variable to ReactDOM.render()

ReactDOM.render()'s first argument should evaluate to a JSX expression. The first argument could also be a variable, so long as that variable evaluates to a JSX expression.

```
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 if you render the exact same thing twice in a row, the second render will do nothing.

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*.