JSX && Babel

Fundamentally, JSX just provides syntactic sugar for theReact.createElement(component, props, ...children) function.

With React.createElement, we sacrifice the well-known, easy to read HTML syntax. Fortunately, JavaScript Syntax Extension (**JSX**) was designed to provide us with some syntactic sugar. Unfortunately, JSX isn't a language that we can just start typing and expect things to work properly. The caveat with JSX is it's not a language that the browsers can interpret out of the box. If we tried to run JSX code right now, we would simply be blessed with a JavaScript syntax error:

*Uncaught SyntaxError: Unexpected token <*

So in order to make JSX work, we need to use a transpiler, named -- Babel.

Enter Babel

Babel is a transpiler that will turn JSX into JavaScript that browsers can understand. We can add Babel into our page by using a simple script tag.

Let's take a look at the difference between JSX and JavaScript.

Raw JavaScript (React.createElement)

*const* App = () *=>* {

    // Here we simply created a function that returns a new React element

    return React.createElement("h1", {}, "Our First React page has rendered");

}

/\*

\* SIDE NOTE:

\* Because the function above returns 1 thing, we can re-write this function in 1 line:

\*  const App = () => React.createElement("h1", {}, "Our First React page has rendered");

\*/

// Then we call the function that returns the element that we intend to render.

ReactDOM.render(App(), document.getElementById("root"));

// Though, take a mental note on how we are firing the App function just like any other function

JSX -> Babel -> Javascript

// Notice the HTML syntax here. Although this may look like regular HTML, it's actually not.

ReactDOM.render(<h1>Hello!</h1>, document.getElementById("root"));

// \*\*Here is what the Babel translation looks like\*\*

ReactDOM.render(React.createElement("h1", null, "Hello World"), document.getElementById("root"));

copy

Enter JSX

JSX is an optional syntax that is simply just syntactic sugar to make it so (a) we can bring back the regular html syntax, (b) we don't have to type React.createElement() all the time and, (c) ultimately reduce the amount of code we write. Using JSX does not mean that anything is different. We are still creating React Functional components but, with JSX, it looks nicer. Let's look but before we do, remember that mental note I mentioned in the first code snippet above about firing a function?

*const* App = () *=>* <h1>Our First React page has rendered</h1>;

// Notice The difference in syntax. Here we've created a React Functional Component, named App which looks

// like regular HTML syntax

ReactDOM.render(<App></App>, document.getElementById("root"));

// Instead of the traditional way we are familiar with firing a function - App() -

// we wrap our function name in an HTML tag instead - <App /> -

In the ReactDOM.render() method,

The way you know the difference between a normal HTML element vs a React Component is the first letter in our component name is uppercase. *Note:* React treats lowercase letters as DOM tags. I.E: if our App function name started with lowercase (i.e: const app=()=> {...}), React would assume that you are trying to use a native html tag named app but, because app is not a valid html tag, it will throw the following error:

Warning: The tag app; is unrecognized in this browser. If you meant to render a React component, start its name with an uppercase letter.

We also have the ability to self close our JSX tags. In contrast to regular HTML, self-closing elements such as hr must be explicitly closed in order for the JSX to be valid. For example, the existing HTML <input /> tag is a self closing tag:

<!--

Lets look at a normal HTML input tag that is self closed.

-->

<input .... />

<!--

With JSX, we can do the same thing. So, because our <App></App> tag doesn't wrap anything, we can self close it.

-->

<script type="text/babel">

    ReactDOM.render(<App />, document.getElementById('root'));

</script>

copy

Pretty neat, eh? You get to write HTML-like code in JavaScript, and then, in the end, Babel compiles that JSX down to React.createElement() calls.

**IMPORTANT: When we create a project via create-react-app, we are installing React. Within our React files, when we write**import React from 'react'**, we can write in JSX within that file without needing to add any extra dependencies.**