Introducing JSX

Consider this variable declaration:

```
const element = <h1>Hello, world!</h1>;
```

This funny tag syntax is neither a string nor HTML.

It is called JSX, and it is a syntax extension to JavaScript. We recommend using it with React to describe what the UI should look like. JSX may remind you of a template language, but it comes with the full power of JavaScript.

JSX produces React "elements". We will explore rendering them to the DOM in the next section. Below, you can find the basics of JSX necessary to get you started.

Why JSX?

React embraces the fact that rendering logic is inherently coupled with other UI logic: how events are handled, how the state changes over time, and how the data is prepared for display.

Instead of artificially separating *technologies* by putting markup and logic in separate files, React separates concerns with loosely coupled units called "components" that contain both. We will come back to components in a further section, but if you're not yet comfortable putting markup in JS, this talk might convince you otherwise.

React doesn't require using JSX, but most people find it helpful as a visual aid when working with UI inside the JavaScript code. It also allows React to show more useful error and warning messages.

With that out of the way, let's get started!

Embedding Expressions in JSX

In the example below, we declare a variable called `name` and then use it inside JSX by wrapping it in curly braces:

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

You can put any valid JavaScript expression inside the curly braces in JSX. For example, `2 + 2`, `user.firstName`, or `formatName(user)` are all valid JavaScript expressions.

In the example below, we embed the result of calling a JavaScript function, `formatName(user)`, into an `<h1>` element.

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);
```

```
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Try it on CodePen

We split JSX over multiple lines for readability. While it isn't required, when doing this, we also recommend wrapping it in parentheses to avoid the pitfalls of automatic semicolon insertion.

JSX is an Expression Too

After compilation, JSX expressions become regular JavaScript function calls and evaluate to JavaScript objects.

This means that you can use JSX inside of `if` statements and `for` loops, assign it to variables, accept it as arguments, and return it from functions:

```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)}!</h1>;
  }
  return <h1>Hello, Stranger.</h1>;
}
```

Specifying Attributes with JSX

You may use quotes to specify string literals as attributes:

```
const element = <div tabIndex="0"></div>;
```

You may also use curly braces to embed a JavaScript expression in an attribute:

```
const element = <img src={user.avatarUrl}></img>;
```

Don't put quotes around curly braces when embedding a JavaScript expression in an attribute. You should either use quotes (for string values) or curly braces (for expressions), but not both in the same attribute.

> **Warning:**
> Since JSX is closer to JavaScript than to HTML, React DOM uses `camelCase` property naming convention instead of HTML attribute names.
>
> For example, `class` becomes `className` in JSX, and `tabindex` becomes `tabIndex`.

Specifying Children with JSX

If a tag is empty, you may close it immediately with `/>`, like XML:

```
const element = <img src={user.avatarUrl} />;
```

JSX tags may contain children:

```
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

JSX Prevents Injection Attacks

It is safe to embed user input in JSX:

```
const title = response.potentiallyMaliciousInput;
// This is safe:
const element = <h1>{title}</h1>;
```

By default, React DOM escapes any values embedded in JSX before rendering them. Thus it ensures that you can never inject anything that's not explicitly written in your application. Everything is converted to a string before being rendered. This helps prevent XSS (cross-site-scripting) attacks.

JSX Represents Objects

Babel compiles JSX down to `React.createElement()` calls.

These two examples are identical:

```
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);
```

```
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

`React.createElement()` performs a few checks to help you write bug-free code but essentially it creates an object like this:

```
// Note: this structure is simplified
const element = {
  type: 'h1',
  props: {
    className: 'greeting',
    children: 'Hello, world!'
  }
};
```

These objects are called "React elements". You can think of them as descriptions of what you want to see on the screen. React reads these objects and uses them to construct the DOM and keep it up to date.

We will explore rendering React elements to the DOM in the next section.

> **Tip:**
> We recommend using the "Babel" language definition for your editor of choice so that both ES6 and JSX code is properly highlighted.