JSX

Nested JSX elements

In order for the code to compile, a JSX expression must have exactly one outermost element. In the below block of code the `<a>` tag is the outermost element.

```
const myClasses = (
  <a href="https://www.codecademy.com">
    <h1>
      Sign Up!
    </h1>
  </a>
);
```

JSX Syntax and JavaScript

JSX is a syntax extension of JavaScript. It's used to create DOM elements which are then rendered in the React DOM.

A JavaScript file containing JSX will have to be compiled before it reaches a web browser. The code block shows some example JavaScript code that will need to be compiled.

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

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

Multiline JSX Expression

A JSX expression that spans multiple lines must be wrapped in parentheses: `(` and `)`. In the example code, we see the opening parentheses on the same line as the constant declaration, before the JSX expression begins. We see the closing parentheses on the line following the end of the JSX expression.

```
const myList = (
  <ul>
    <li>item 1</li>
    <li>item 2</li>
    <li>item 3</li>
  </ul>
);
```

JSX syntax and HTML

In the block of code we see the similarities between JSX syntax and HTML: they both use the angle bracket opening and closing tags (`<h1>` and `</h1>`).

When used in a React component, JSX will be rendered as HTML in the browser.

```
const title = <h1>Welcome all!</h1>
```

JSX attributes

The syntax of JSX attributes closely resembles that of HTML attributes. In the block of code, inside of the opening tag of the `<h1>` JSX element, we see an `id` attribute with the value `"example"`.

```
const example = <h1 id="example">JSX
Attributes</h1>;
```

ReactDOM JavaScript library

The JavaScript library `react-dom`, sometimes called `ReactDOM`, renders JSX elements to the DOM by taking a JSX expression, creating a corresponding tree of DOM nodes, and adding that tree to the DOM.
The code example begins with `ReactDOM.render()`. The first argument is the JSX expression to be compiled and rendered and the second argument is the HTML element we want to append it to.

```
ReactDOM.render(
  <h1>This is an example.</h1>,
  document.getElementById('app')
);
```

Embedding JavaScript in JSX

JavaScript expressions may be embedded within JSX expressions. The embedded JavaScript expression must be wrapped in curly braces.
In the provided example, we are embedding the JavaScript expression `10 * 10` within the `<h1>` tag. When this JSX expression is rendered to the DOM, the embedded JavaScript expression is evaluated and rendered as `100` as the content of the `<h1>` tag.

```
let expr = <h1>{10 * 10}</h1>;
// above will be rendered as <h1>100</h1>
```

The Virtual Dom

React uses Virtual DOM, which can be thought of as a blueprint of the DOM. When any changes are made to React elements, the Virtual DOM is updated. The Virtual DOM finds the differences between it and the DOM and re-renders only the elements in the DOM that changed. This makes the Virtual DOM faster and more efficient than updating the entire DOM.

JSX className

In JSX, you can't use the word `class`! You have to use `className` instead. This is because JSX gets translated into JavaScript, and `class` is a reserved word in JavaScript.
When JSX is rendered, JSX `className` attributes are automatically rendered as `class` attributes.

```
// When rendered, this JSX expression...
const heading = <h1 className="large-
heading">Codecademy</h1>;

// ...will be rendered as this HTML
<h1 class="large-heading">Codecademy</h1>
```

JSX and conditional

In JSX, `&&` is commonly used to render an element based on a boolean condition. `&&` works best in conditionals that will sometimes do an action, but other times do nothing at all.

If the expression on the left of the `&&` evaluates as true, then the JSX on the right of the `&&` will be rendered. If the first expression is false, however, then the JSX to the right of the `&&` will be ignored and not rendered.

```
// All of the list items will display if
// baby is false and age is above 25
const tasty = (
  <ul>
    <li>Applesauce</li>
    { !baby && <li>Pizza</li> }
    { age > 15 && <li>Brussels
Sprouts</li> }
    { age > 20 && <li>Oysters</li> }
    { age > 25 && <li>Grappa</li> }
  </ul>
);
```

JSX conditionals

JSX does not support if/else syntax in embedded JavaScript. There are three ways to express conditionals for use with JSX elements:

1. a ternary within curly braces in JSX

2. an `if` statement outside a JSX element, or

3. the `&&` operator.

```
// Using ternary operator
const headline = (
  <h1>
    { age >= drinkingAge ? 'Buy Drink'
: 'Do Teen Stuff' }
  </h1>
);

// Using if/else outside of JSX
let text;

if (age >= drinkingAge) { text = 'Buy
Drink' }
else { text = 'Do Teen Stuff' }

const headline = <h1>{ text }</h1>

// Using && operator. Renders as empty div
if length is 0
const unreadMessages = [ 'hello?',
'remember me!'];

const update = (
  <div>
    {unreadMessages.length > 0 &&
      <h1>
        You have {unreadMessages.length}
unread messages.
      </h1>
    }
  </div>
);
```

Embedding JavaScript code in JSX

Any text between JSX tags will be read as text content, not as JavaScript. In order for the text to be read as JavaScript, the code must be embedded between curly braces `{ }`.

```
<p>{ Math.random() }</p>

// Above JSX will be rendered something
like this:
<p>0.88</p>
```

JSX element event listeners

In JSX, event listeners are specified as attributes on elements. An event listener attribute's *name* should be written in camelCase, such as `onClick` for an `onclick` event, and `onMouseOver` for an `onmouseover` event.
An event listener attribute's *value* should be a function. Event listener functions can be declared inline or as variables and they can optionally take one argument representing the event.

```
// Basic example
const handleClick = () => alert("Hello world!");

const button = <button onClick=
{handleClick}>Click here</button>;

// Example with event parameter
const handleMouseOver = (event) =>
event.target.style.color = 'purple';

const button2 = <div onMouseOver=
{handleMouseOver}>Drag here to change
color</div>;
```

Setting JSX attribute values with embedded JavaScript

When writing JSX, it's common to set attributes using embedded JavaScript variables.

```
const introClass = "introduction";
const introParagraph = <p className=
{introClass}>Hello world</p>;
```

JSX .map() method

The array method `map()` comes up often in React. It's good to get in the habit of using it alongside JSX.
If you want to create a list of JSX elements from a given array, then `map()` over each element in the array, returning a list item for each one.

```
const strings = ['Home', 'Shop', 'About
Me'];

const listItems = strings.map(string =>
<li>{string}</li>);

<ul>{listItems}</ul>
```

JSX empty elements syntax

In JSX, empty elements must explicitly be closed using a closing slash at the end of their tag: `<tagName />`.
A couple examples of empty element tags that must explicitly be closed include `<br>` and `<img>`.

```
<br />
<img src="example_url" />
```

React.createElement() Creates Virtual DOM Elements

The `React.createElement()` function is used by React to actually create virtual DOM elements from JSX. When the JSX is compiled, it is replaced by calls to `React.createElement()`.

You usually won't write this function yourself, but it's useful to know about.

```
// The following JSX...
const h1 = <h1 className="header">Hello world</h1>;

// ...will be compiled to the following:
const h1 = React.createElement(
  'h1',
  {
    className: 'header',
  },
  'Hello world'
);
```

JSX key attribute

In JSX elements in a list, the `key` attribute is used to uniquely identify individual elements. It is declared like any other attribute.

Keys can help performance because they allow React to keep track of whether individual list items should be rendered, or if the order of individual items is important.

```
<ul>
  <li key="key1">One</li>
  <li key="key2">Two</li>
  <li key="key3">Three</li>
  <li key="key4">Four</li>
</ul>
```