React JSX

What is JSX?

JSX stands for JavaScript XML.
JSX allows us to write HTML in React.
JSX makes it easier to write and add HTML in React.

Coding JSX

JSX allows us to write HTML elements in JavaScript and place them in the DOM without any `createElement()` and/or `appendChild()` methods.
JSX converts HTML tags into react elements.
You are not required to use JSX, but JSX makes it easier to write React applications.
Here are two examples. The first uses JSX and the second does not:

Example 1

JSX:

```
const myelement = <h1>I Love JSX!</h1>;

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

Example 2

Without JSX:

```
const myelement = React.createElement('h1', {}, 'I do not use JSX!');

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

As you can see in the first example, JSX allows us to write HTML directly within the JavaScript code.
JSX is an extension of the JavaScript language based on ES6, and is translated into regular JavaScript at runtime.

Expressions in JSX

With JSX you can write expressions inside curly braces `{ }`.
The expression can be a React variable, or property, or any other valid JavaScript expression. JSX will execute the expression and return the result:

Example

Execute the expression `5 + 5`:

```
const myelement = <h1>React is {5 + 5} times better with JSX</h1>;
```

Inserting a Large Block of HTML

To write HTML on multiple lines, put the HTML inside parentheses:

Example

Create a list with three list items:

```
const myelement = (
  <ul>
    <li>Apples</li>
    <li>Bananas</li>
    <li>Cherries</li>
  </ul>
);
```

One Top Level Element

The HTML code must be wrapped in *ONE* top level element.
So if you like to write *two* paragraphs, you must put them inside a parent element, like a `div` element.

Example

Wrap two paragraphs inside one DIV element:

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

JSX will throw an error if the HTML is not correct, or if the HTML misses a parent element.
Alternatively, you can use a "fragment" to wrap multiple lines. This will prevent unnecessarily adding extra nodes to the DOM.
A fragment looks like an empty HTML tag: `<></>`.

Example

Wrap two paragraphs inside a fragment:

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

Elements Must be Closed

JSX follows XML rules, and therefore HTML elements must be properly closed.

Example

Close empty elements with `/>`

```
const myelement = <input type="text" />;
```

JSX will throw an error if the HTML is not properly closed.

Attribute class = className

The `class` attribute is a much used attribute in HTML, but since JSX is rendered as JavaScript, and the `class` keyword is a reserved word in JavaScript, you are not allowed to use it in JSX.
Use attribute `className` instead.

JSX solved this by using `className` instead. When JSX is rendered, it translates `className` attributes into `class` attributes.

Example

Use attribute `className` instead of `class` in JSX:

```
const myelement = <h1 className="myclass">Hello World</h1>;
```

Conditions - if statements

React supports `if` statements, but not *inside* JSX.
To be able to use conditional statements in JSX, you should put the `if` statements outside of the JSX, or you could use a ternary expression instead:

Option 1:

Write `if` statements outside of the JSX code:

Example

Write "Hello" if `x` is less than 10, otherwise "Goodbye":

```
const x = 5;
let text = "Goodbye";
if (x < 10) {
  text = "Hello";
}

const myelement = <h1>{text}</h1>;
```

Option 2:

Use ternary expressions instead:

Example

Write "Hello" if `x` is less than 10, otherwise "Goodbye":

```
const x = 5;

const myelement = <h1>{(x) < 10 ? "Hello" : "Goodbye"}</h1>;
```

Note that in order to embed a JavaScript expression inside JSX, the JavaScript must be wrapped with curly braces, `{}`.