## ✨ 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>Rasha Loa</li>,
        smallForward: <li>Tayshaun Dasmoto</li>,
        shootingGuard: <li>Colmar Cumberbatch</li>,
       pointGuard: <li>Femi Billon</li>
    };
```

🎖 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"
```

Here are some JSX elements with *attributes:*

➡️ <a href='http://www.example.com'>Welcome to the Web</a>;

➡️ const title = <h1 id='title'>Introduction to React.js: Part I</h1>;

A single JSX element can have many attributes, just like in HTML: 🐼

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

🪀 You can nest  JSX elements inside of other JSX elements, just like in HTML

Here's an example of a JSX <h1> element, *nested* inside of a JSX <a> element:

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

To make this more readable, you can use HTML-style line breaks and indentation:

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

If a JSX expression takes up more than one line, then you must wrap the multi-line JSX expression in parentheses. This looks strange at first, but you get used to it:

```
  (
      <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! Here's an example of a *nested* JSX expression being saved as a variable:

```
    const theExample = (
        <a href="https://www.example.com">
            <h1>Click me!</h1>
        </a>
      );
```

## 🤖 JSX Outer Elemenets

There's a rule that we haven't mentioned: a JSX expression must have exactly *one* outermost element.

In other words, this code 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 code will not work:

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

The *first opening tag* and the *final closing tag* of a JSX expression must belong to the same JSX element!

It's easy to forget about this rule, and end up with errors that are tough to diagnose.

If you notice that a JSX expression has multiple outer elements, the solution is usually simple: wrap the JSX expression in a .

## 🧳 Rendering JSX

You've learned how to write JSX elements! Now it's time to learn how to *render* them.

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