What is React?

React is a JavaScript library for building user interfaces. It is the view layer for web applications.

At the heart of all React applications are components. A component is a self-contained module that renders some output. We can write interface elements like a button or an input field as a React component. Components are *compostable*. A component might include one or more other components in its output.

Broadly speaking, to write React apps we write React components that correspond to various interface elements. We then organize these components inside higher-level components which define the structure of our application.

For example, take a form. A form might consist of many interface elements, like input fields, labels, or buttons. Each element inside the form can be written as a React component. We'd then write a higher-level component, the form component itself. The form component would specify the structure of the form and include each of these interface elements inside of it.

Importantly, each component in a React app abides by strict data management principles. Complex, interactive user interfaces often involve complex data and application state. The surface area of React is limited and aimed at giving us the tools to be able to anticipate how our application will look with a given set of circumstances. We dig into these principles later in the course.

How do we use it?

React is a JavaScript framework. Using the framework is as simple as including a JavaScript file in our HTML and using the react exports in our application's JavaScript.

For instance, the *HelloWorld* example of a React website can be as simple as:

```
<html>
<head>
 <meta charset="utf-8">
 <title>HelloWorld</title>
 <!-- Script tags including React -->
 <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.3.1/react.min.js"></script>
 <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.3.1/react-dom.min.js"></script>
 <script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
</head>
<body>
 <div id="app"></div>
 <script type="text/babel">
  ReactDOM.render(
   <h1>HelloWorld</h1>,
   document.querySelector('#app')
  );
 </script>
</body>
</html>

```

Although it might look a little scary, the JavaScript code is a single line that dynamically adds *HelloWorld* to the page. Note that we only needed to include a handful of JavaScript files to get everything working.

How does it work?

Unlike many of its predecessors, React operates not directly on the browser's Document Object Model (DOM) immediately, but on a **virtual DOM**. That is, rather than manipulating the document in a browser after changes to our data (which can be quite slow) it resolves changes on a DOM built and run entirely in memory. After the virtual DOM has been updated, React intelligently determines what changes to make to the actual browser's DOM.

The React Virtual DOM exists entirely in-memory and is a representation of the web browser's DOM. Because of this, when we write a React component, we're not writing directly to the DOM, but we're writing a virtual component that React will turn into the DOM.

In the next article, we'll look at what this means for us as we build our React components and jump into JSX and writing our first real components.