Components

React is a library that allows you to reuse code extensively and provide faster loading times even with infinitely more complex sites. React pages are created using 'components'. You can think of a component as a reusable function to create HTML / elements that will be displayed in the browser. It uses JSX, which is basically HTML combined with JavaScript.

For example, you could have a component that creates an item for a list. Without React, you would have to individually make those list items either with HTML or JavaScript. With React's components, you can create a component for the list item, and re-use that component to display a bunch of list items, which is way faster and easier than with pure JavaScript.

Functional Components

In React, there are two main ways to declare components. While there are many offshoots of these two, the big two are functional and class based components. React now mainly uses functional components instead, and so the remainder of the course will be focusing on functional components.

Functional components are literally just that: A function that returns something to render to the UI. For the longest time they were not able to do everything a class-based component could do, however they have gained a lot more popularity with the addition of React Hooks. Here's what a functional component looks like:

- OPTION 1 - Declaring and exporting in a single line with `export default function`

```
// This imports the React library so we can use it in our file
import React from "react";

// We'll talk more about props tomorrow, but you can think of it like parameters
and arguments
export default function Welcome(props) {
  // In React, the code in the return statement is what actually gets displayed on
screen
  return (
    <div>Hello!</div>
  );
}
```

- OPTION 2 - Creating the function as an arrow function (generally seen more often) and exporting on a lower line

```
import React from "react";

const Welcome = (props) => {
  return <div>Hello!</div>;
};

export default Welcome;
```

As you can see, the components are named with a capital letter at the beginning. Whether it is a class or functional component, you ALWAYS have the component be capitalized and follow Pascal Casing: `ExampleComponent`. As long as your component is returning JSX to the screen, it must be capitalized, or else you'll get an error.

Rendering with Functional Components

While not mentioned above, one of the MOST important things about rendering is that there can be only *ONE* top level HTML element in the render function / returned by the component. The following would be invalid JSX and would actually trigger an error at compile time:

```
export default function Welcome(props) {
  return (
    <div>Hello!</div>
    <div>Hello!</div>
  );
}
```

Everything needs to be in a single element. There are two ways to fix the above, the first would be to simply wrap everything in a div like this:

```
export default function Welcome(props) {
  return (
    <div>
      <div>Hello!</div>
      <div>Hello!</div>
    </div>
  );
}
```

But what if that might mess up your styling to have another parent div? React has a special element that you can use as a wrapper that doesn't break layout. It is simply an empty element (`<></>`) called a JSX fragment. You could then have the above look like:

```
export default function Welcome(props) {
  return (
    <>
      <div>Hello!</div>
      <div>Hello!</div>
    </>
  );
}
```