Component constructors and state

In React, props are immutable. Meaning a component can't change it's own props, they can only change when a parent component passes down new ones. So how do we got about creating UIs that change? We use state. State is what represents what your application UI looks like at any given moment. We define the initial state of our component inside the constructor. What's a constructor? It's the "initialize" function for the class. Any time a new instance of the class is created, the constructor is called and passed in any properties that are given during initialization.

```
export default class Component extends React.Component {
  constructor(props) {
    super(props);

    this.state = props;
  }
}
```

What's going on here? Every React component is initialized with props. So when you create a component using JSX like this:

```
<Component
  foo='bar'
  bar='baz'
/>
```

The constructor of that `Component` class receives a JavaScript object literal that looks like this:

```
{
  "foo": "bar",
  "bar": "baz"
}
```

The other thing you may find strange is the `super` function call. What this does is it tells the class to call the inherited class's (in this case, the React.Component) constructor function with the arguments you give it. When extending classes it is required that the super method (if being called) be called first in the constructor. So here we're just passing off the props to React and letting React do what it normally does with them. In this case React will assign them to `this.props`. In this contrived example we're also assigning the state of the application to the value of the props.

Now let's open up our `ReadingTime` component and add the following constructor:

```
constructor(props) {
  super(props);

  this.state = {
    readTime: 0
  };
}
```

For reference, your `ReadingTime` component should now look like this:

```
import React from 'react';

export default class ReadingTime extends React.Component {
  static propTypes = {
    wordsPerMinute: React.PropTypes.number
  };

  static defaultProps = {
    wordsPerMinute: 270
  };

  constructor(props) {
    super(props);

    this.state = {
      readTime: 0
    };
  }

  render() {
    return (
      <div>Hello ReadingTime!</div>
    );
  }
}
```

Excellent! We've created another component! Let's open up our main application component and add this component to our app. Open up `example/react-reading-time.jsx` and import our component:

```
import ReadingTime from '../src/reading-time';
```

Ok now let's add this component to our view. We'll also add some classes, a `textarea` for writing an article and some default text to put into the `textarea`. Let's just rewrite the entire component:

```
constructor(props) {
```

```
  super(props);

  this.state = {
    text: 'Foo is baz and bar'
  };
}

render() {
  return (
    <div className='container' style={{ marginTop: '50px' }}>
      <div className='col-lg-8 col-lg-offset-2 form-group'>
        <textarea
          value={this.state.text}
          className='form-control'
          style={{ height: '500px', resize: 'none' }}>
        </textarea>
      </div>
      <ReadingTime text={this.state.text} className='col-lg-2 well' />
    </div>
  );
}
```

Woohoo! We've got our component set to render inside of our main application page along with a text area. You'll notice that I added some text to our compontns state. We've added this so that we can keep track of the text that is being input into the text area so that we can pass it down to the `ReadingTime` component as props. We'll be using this later to calculate the reading time.

I also added a style attribute to the element, but it doesn't look like a normal style tag that you would add to an HTML. Remember when we talked about JSX, and how it's really just JavaScript? Because of that, we have to give the style tag a regular old JavaScript object to work with, which it can then turn into the proper style tag when rendering the DOM.

Another thing to note is that by adding a `value` attribute to the `textarea` we made it what is called a `Controlled` component. In a controlled component, the value of the input will **always** be the value of the `value` attribute. User interaction will have no effect on it. The only way to change the value is through an event handler. On the other hand, if you use the JSX specific `defaultValue` attribute, React will set the initial value of the component to this value, and it will still be editable by the user.

Let's fire up the server again with `npm start` and visit `localhost:8881/example` and see what we've created so far! It's getting more and more exciting by the minute!