Finishing the ReadingTime component

We're going to go ahead and finish up the `ReadingTime` component in this lesson and we'll see how updating state makes the UI update.

First we're going to add a function that is going to simply count the number of words in a string of text that is passed into it.

```
countWords(text) {
  return text.split(/\s+/).length
}
```

The `countWords` function takes a string, splits it wherever there is a space and counts how many words there are.

Component lifecycles

The next thing we need to do is estimate what the reading time is going to be. We *could* execute that code in the constructor of the component, but if we did that, the component would not be properly updated when it received new props from the parent component. We're going to use React's
lifecycle hooks to run this code any time the component receives props. Our component is going to receive new props any time the `textarea` on the parent component is updated. There are quite a few
lifecycle hooks in React. For this use case, we'll be using the `componentWillReceiveProps`, and the `componentWillMount` hooks, so let's add these to our `ReadingTime` component. React convention is to put this function immediately after the constructor function and before any other code:

```
componentWillMount() {
  this.updateReadingTime(this.props);
}

componentWillReceiveProps(nextProps) {
  this.updateReadingTime(nextProps);
}

updateReadingTime(props) {
  const words = this.countWords(props.text);
```

```
  const readTime = Math.round(words / props.wordsPerMinute);

  this.setState({ readTime });
}
```

Ok so what's happening here? We have to use the `componentWillMount` hook for the initial render of the component, because the `componentWillReceiveProps` hook is *not* called on the initial render.

The `componentWillReceiveProps` lifecycle hook receives one argument, and that's an object containing the new props. We'll use these new props to calculate the new reading time and set the state.

We've put the code for updating the estimated reading time into it's own function because it is going to be used in more than one place. The first thing we do is use our `countWords` function to count the number of words in the new string, and then just divide that by the `wordsPerMinute` prop and we're done! We then just set the state and we're good to go! But we can't simply update the value of `this.state`. We need to treat the component state as immutable, and use React's internal methods to update the state. That's why we use the `setState` method to set the state. Using this method also triggers a DOM update, so React will update the Virtual DOM, perform the diff, and update the pieces of the DOM that have changed.

But wait a minute! What's this weird syntax we're using? Shouldn't there be a key *AND* a value inside of that object? This is just some more new ES6 syntax that allows for more shorthand assigning of variables! It will set the key to the name of the variable passed in, and the value to the value of that variable!

Let's update the `ReadingTime` component to display the current state of the `readTime` variable:

```
render() {
  return (
    <div>
      <p>
        Estimated read time:<br /><br />
        <span>{this.state.readTime}</span>
      </p>
    </div>
  )
}
```

Our `ReadingTime` component is now displaying 0 minutes for a read time. Unfortunately because we only have 5 words in our text box, this isn't going to change to anything higher easily. So go ahead and update the initial `text` state variable in the constructor of the `ReactReadingTime` component to something really, really long, and you should see the estimated reading time jump up when the page loads.