RICE®

Web Development

COMP 431 / COMP 531

ReactJS

Scott E Pollack, PhD

September 22, 2016

Recap

- HTML and HTML5, Storage, Canvas
- JavaScript and Scope
- Forms, CSS, Events
- jQuery, AJAX, and fetch
- Modern JS
- MVC

~~Modern JS~~

~~MVC~~

React & Redux

Testing

Homework Assignment 4

(JavaScript Game)

Due Thursday 9/29

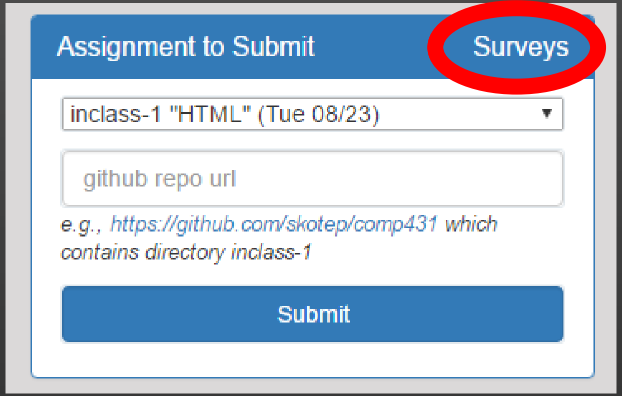
COMP 531

Draft Front-End Review

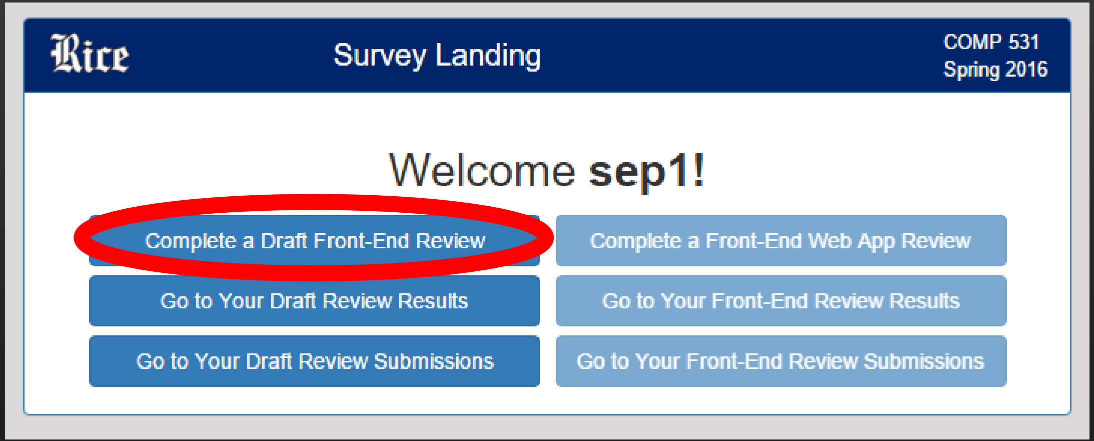
Due Tuesday 9/27

COMP 531 Draft Front-End Review

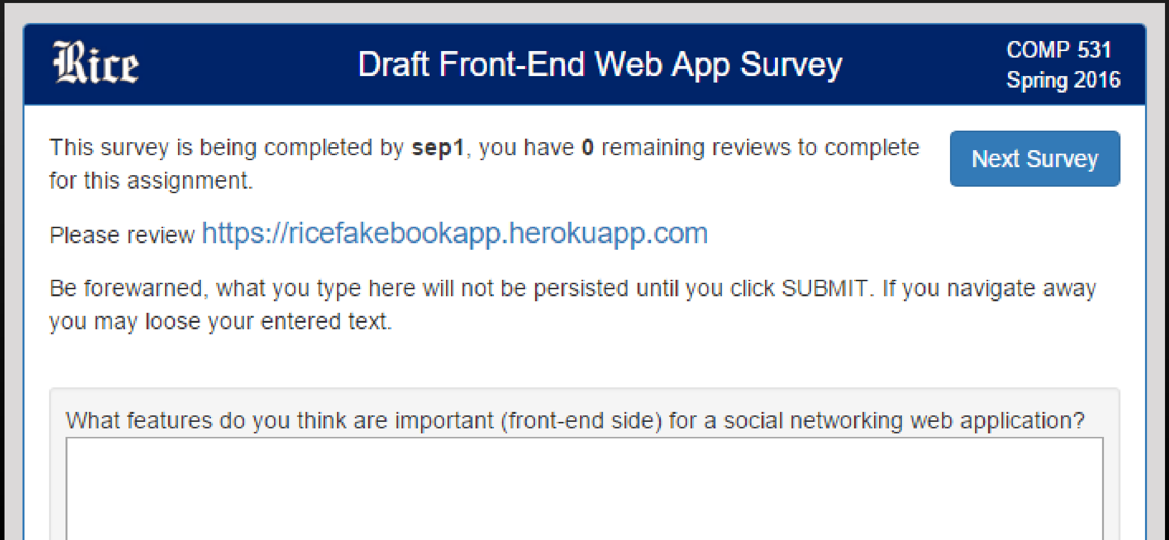
- Go to the normal assignment submission page:
- https://webdev-rice.herokuapp.com/
- Click on "Surveys"
- Click on "Complete a Draft Front-End Review"
- You will be asked to review 7 websites of your peers
- Provide useful, constructive feedback
- Provide a critical comparative evaluation of each site with yours



e.g., https://github.com/skotep/comp431 which contains directory inclass-1


Rice
Survey Landing
COMP 531
Spring 2016
Complete a Draft Front-End Review
Complete a Front-End Web App Review
Go to Your Draft Review Results
Go to Your Front-End Review Results
Go to Your Draft Review Submissions
Go to Your Front-End Review Submissions

Welcome **sep1**!



This survey is being completed by **sep1**, you have **0** remaining reviews to complete for this assignment.

Please review https://ricefakebookapp.herokuapp.com

Be forewarned, what you type here will not be persisted until you click SUBMIT. If you navigate away you may loose your entered text.

What features do you think are important (front-end side) for a social networking web application?

Virtual DOM

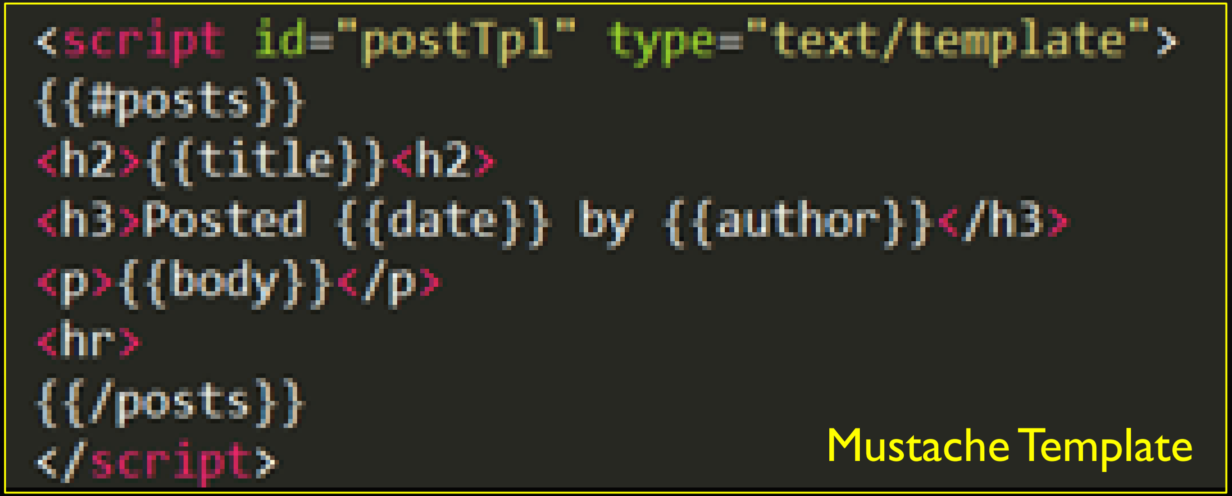
- An in-memory representation of the DOM
  - it’s an abstraction of an abstraction!

```html
<div>
    <p class="title">A list of stuff</p>
    <ul class="stuff">
        <li><span class="fancy">thing1</span></li>
        <li>thing2</li>
        <li><img src="seuss.png" alt="cat"/></li>
    </ul>
</div>
```

```javascript
h('div', null, [
    h('p', { className: 'title' }, 'A list of stuff'),
    h('ul', { className: 'stuff' }, [
        h('li', null, h('span', { className: 'fancy' }, 'thing1')),
        h('li', null, 'thing2'),
        h('li', null, h('img', { src: 'seuss.png', alt: 'cat' }))
    ])
])
```

Looking for a better seamless experience

- Virtual DOM is fast and powerful
- We want HTML but don't want the overhead of templates – we want virtual DOM
- Writing functions for html tags is weird – want seamless experience



```
<script id="postTpl" type="text/template">
{{#posts}}
<h2>{{title}}<h2>
<h3>Posted {{date}} by {{author}}</h3>
<p>{{body}}</p>
<hr>
{{/posts}}
</script>
```

Mustache Template

JSX

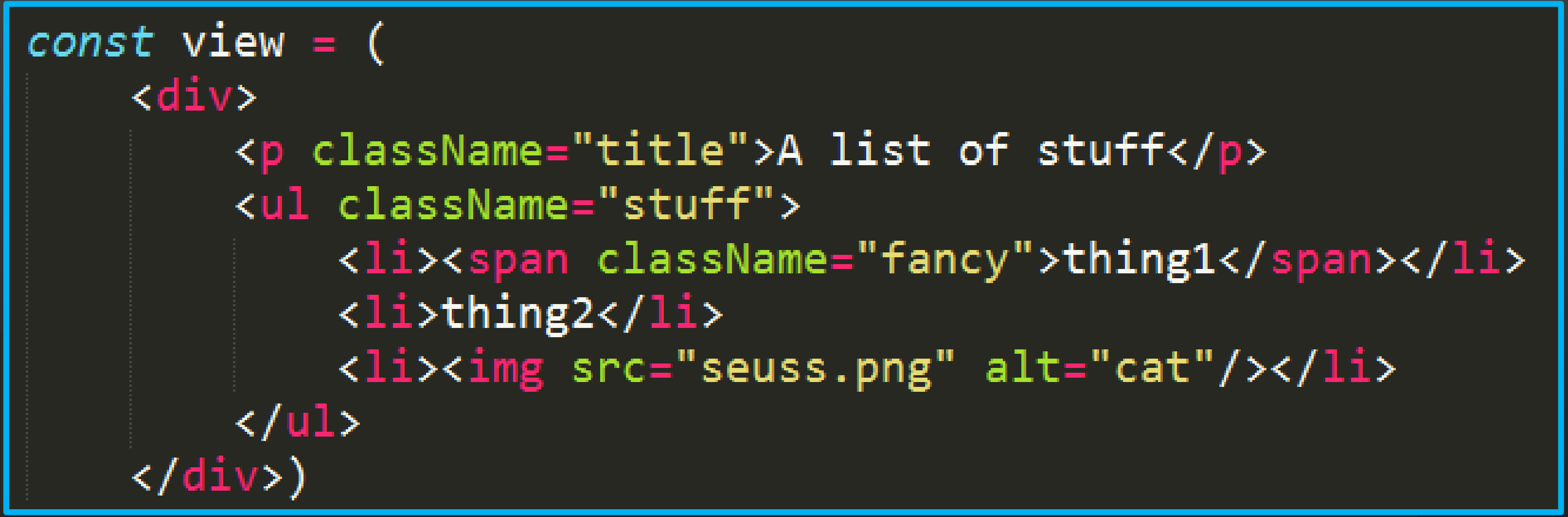
JSX is a preprocessor step that adds XML syntax to JavaScript. Just like XML, **JSX** tags have a tag name, attributes, and children. If an attribute value is enclosed in quotes, the value is a string.

https://facebook.github.io/react/docs/jsx-in-depth.html

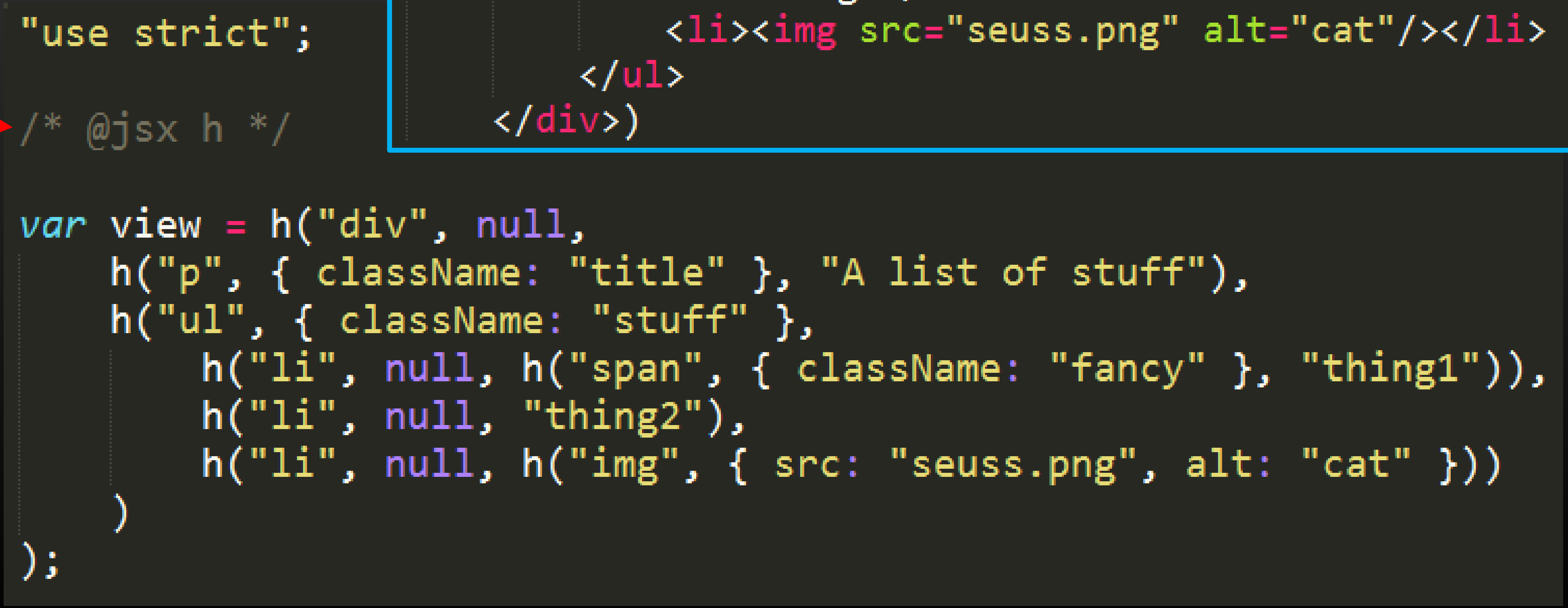
JSX Transpilation

https://babeljs.io/repl/

transpiler directive
use h() instead of React()

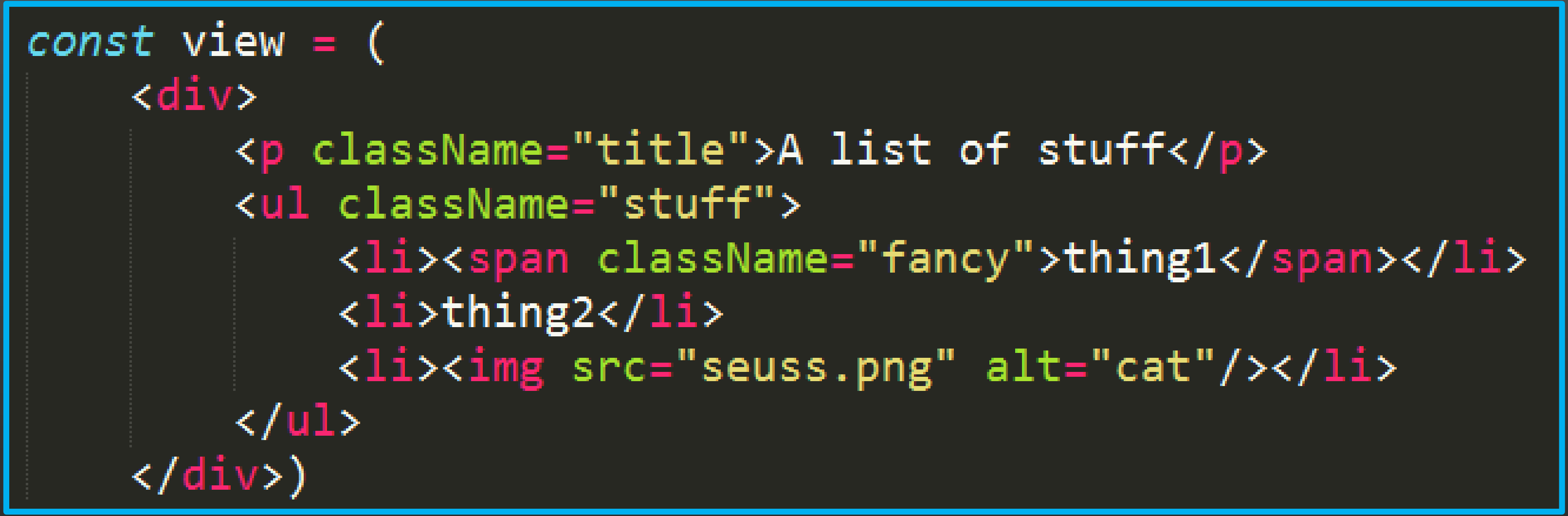


```
const view = (
    <div>
        <p className="title">A list of stuff</p>
        <ul className="stuff">
            <li><span className="fancy">thing1</span></li>
            <li>thing2</li>
            <li><img src="seuss.png" alt="cat"/></li>
        </ul>
    </div>)
```



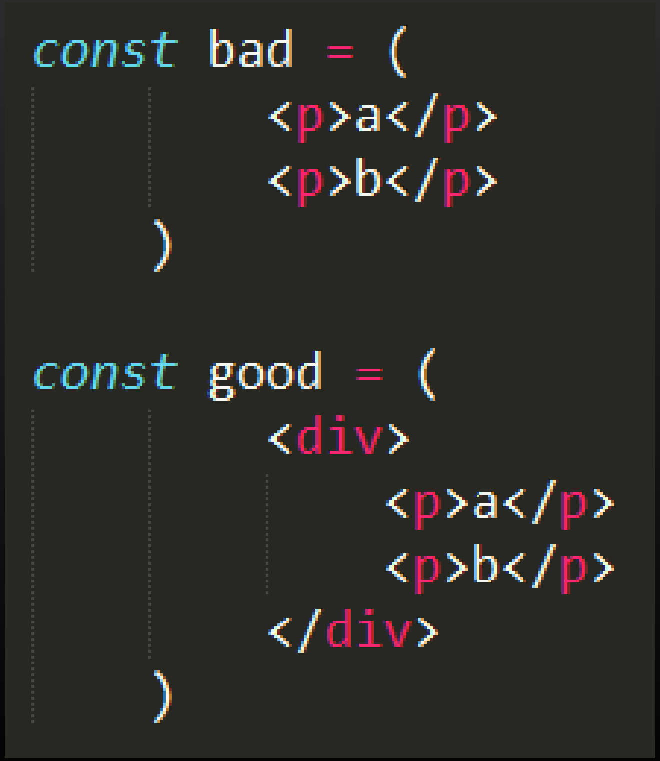
```
"use strict";

/* @jsx h */

var view = h("div", null,
    h("p", { className: "title" }, "A list of stuff"),
    h("ul", { className: "stuff" },
        h("li", null, h("span", { className: "fancy" }, "thing1")),
        h("li", null, "thing2"),
        h("li", null, h("img", { src: "seuss.png", alt: "cat" }))
    )
);
```

JSX



```
const view = (
    <div>
        <p className="title">A list of stuff</p>
        <ul className="stuff">
            <li><span className="fancy">thing1</span></li>
            <li>thing2</li>
            <li><img src="seuss.png" alt="cat"/></li>
        </ul>
    </div>)
```

- Must always have a single top level element
- Parenthesis are required for multiline statements
  - Good to use in general
- Imbed javascript with { }



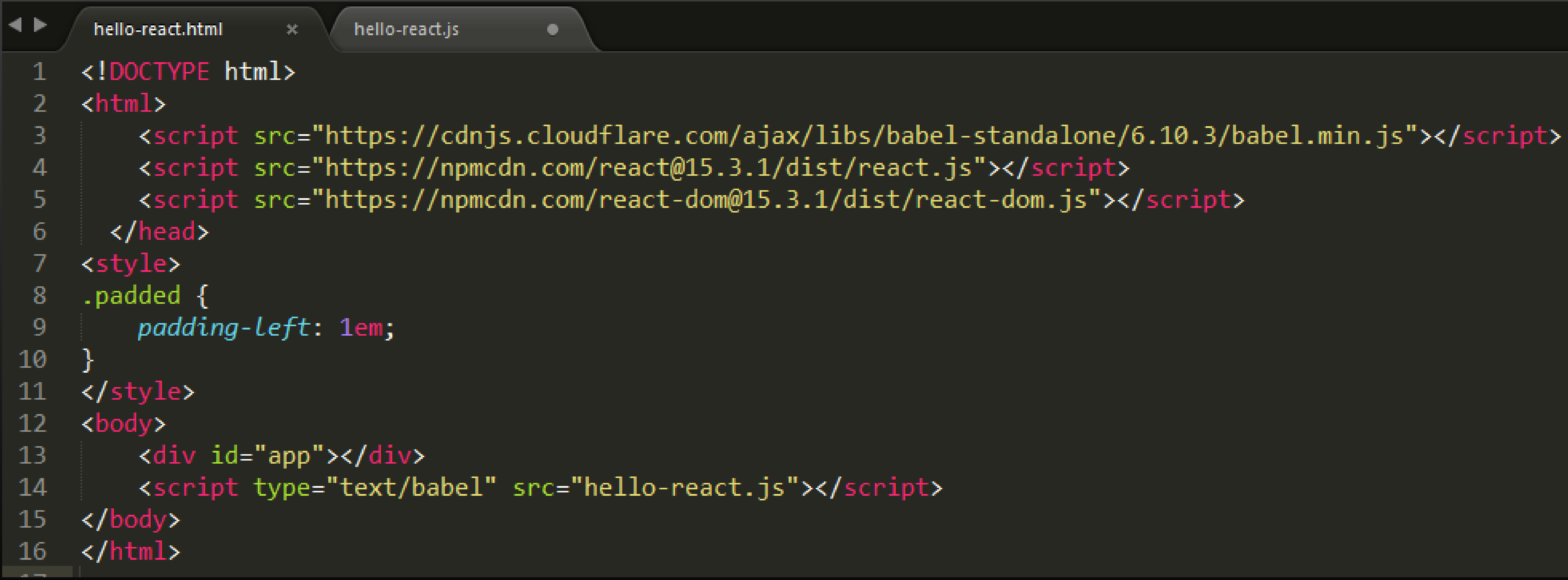
```
const bad = (
        <p>a</p>
        <p>b</p>
    )

const good = (
        <div>
            <p>a</p>
            <p>b</p>
        </div>
    )
```

What is React?

React is a JavaScript library for creating user interfaces by Facebook and Instagram. Many people choose to think of React as the V in MVC. Facebook built React to solve one problem: building large applications with data that changes over time.

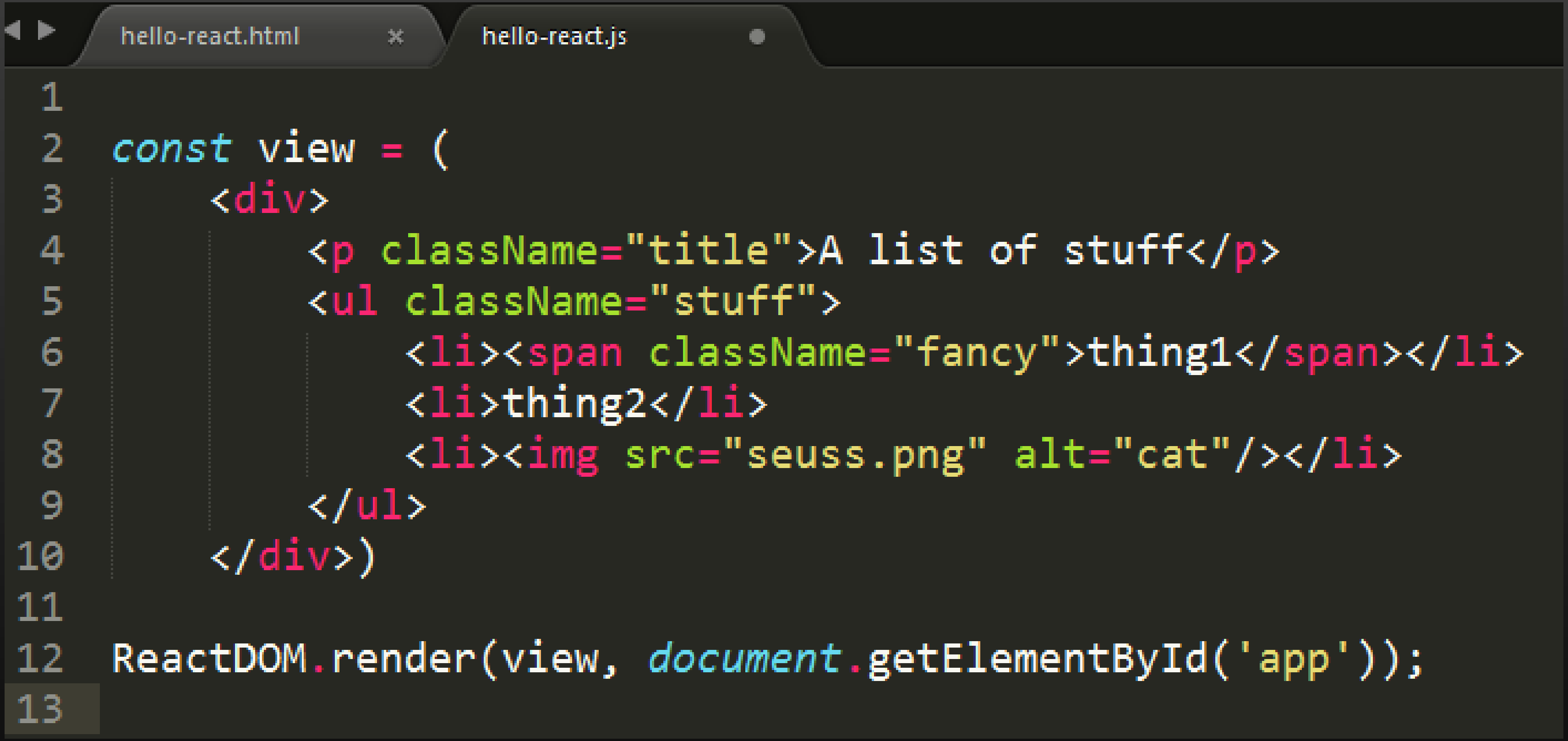
https://facebook.github.io/react/docs/why-react.html

React in Action

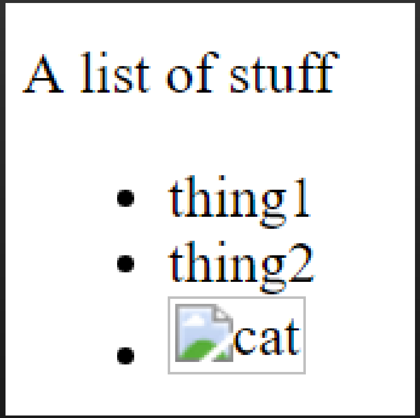


```
<!DOCTYPE html>
<html>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-standalone/6.10.3/babel.min.js"></script>
    <script src="https://npmcdn.com/react@15.3.1/dist/react.js"></script>
    <script src="https://npmcdn.com/react-dom@15.3.1/dist/react-dom.js"></script>
  </head>
<style>
.padded {
    padding-left: 1em;
}
</style>
<body>
    <div id="app"></div>
    <script type="text/babel" src="hello-react.js"></script>
</body>
</html>
```

React in Action, *continued…*


hello-react.html
hello-react.js

```jsx

const view = (
    <div>
        <p className="title">A list of stuff</p>
        <ul className="stuff">
            <li><span className="fancy">thing1</span></li>
            <li>thing2</li>
            <li><img src="seuss.png" alt="cat"/></li>
        </ul>
    </div>)

ReactDOM.render(view, document.getElementById('app'));

```

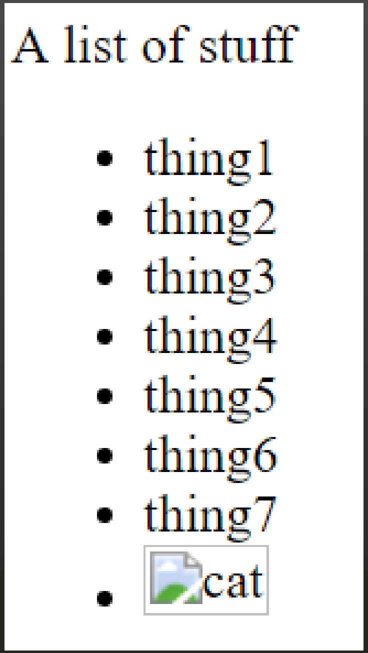

A list of stuff

- thing1
- thing2
- cat

React Concepts

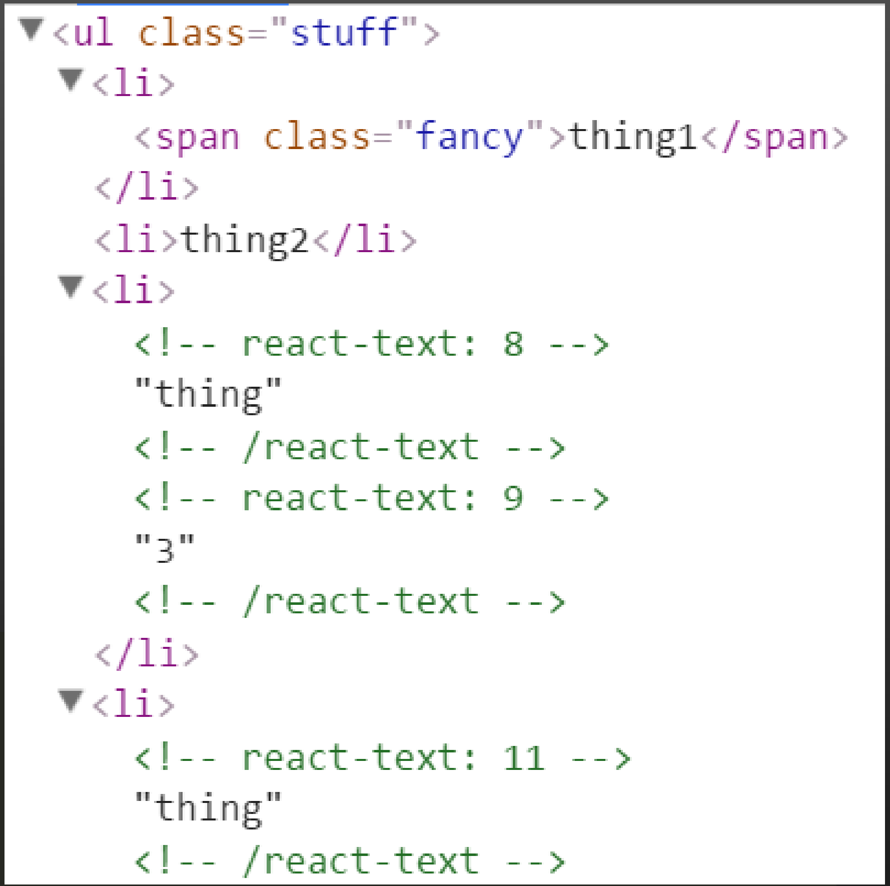
- React uses a VDOM for fast and efficient rendering updates
- Divide the page into Components
  - Divide and conquer
  - Simplify and Reduce complexity
  - Separation of model, view, and controller
- Components can be simple or complex
  - A simple Component has no explicit functionality
  - Complex Components may contain state
- Components have props (attributes) and state (data)

Repeating elements



A list of stuff

- thing1
- thing2
- thing3
- thing4
- thing5
- thing6
- thing7
- cat



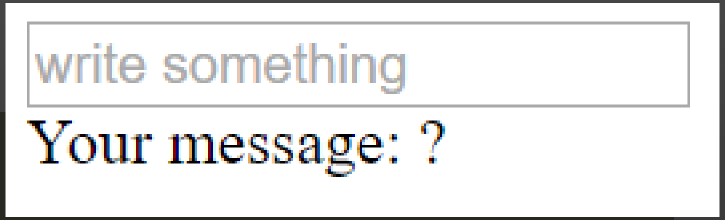
```
▼<ul class="stuff">
  ▼<li>
      <span class="fancy">thing1</span>
    </li>
    <li>thing2</li>
  ▼<li>
      <!-- react-text: 8 -->
      "thing"
      <!-- /react-text -->
      <!-- react-text: 9 -->
      "3"
      <!-- /react-text -->
    </li>
  ▼<li>
      <!-- react-text: 11 -->
      "thing"
      <!-- /react-text -->
```

```
const viewRepeat = (
        <div>
            <p className="title">A list of stuff</p>
            <ul className="stuff">
                <li><span className="fancy">thing1</span></li>
                <li>thing2</li>
                { Array(5).fill(1).map((x, i) => <li key={i}>thing{3 + i}</li>) }
                <li><img src="seuss.png" alt="cat"/></li>
            </ul>
        </div>
    )
```

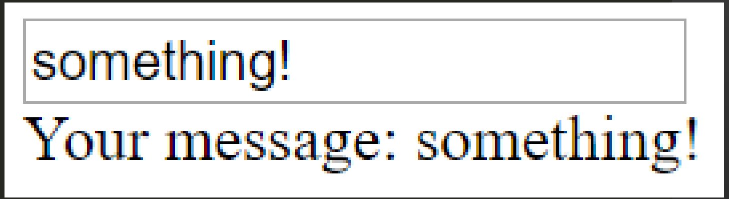
React uses the **key** to identify separate repeated elements.
Think back to the TODO assignment from last time.

React Component (ES2015+)

```
class BoundText extends React.Component {

    constructor(props) {
        super(props)
        this.state = { message: '?' }
    }

    render() {
        return (
            <div>
                <input placeholder="write something"
                    onChange={(e) => this.setState({ message: e.target.value }) }
                /><br/>
                <span>Your message: { this.state.message }</span>
            </div>
    )}

}

ReactDOM.render(<BoundText/>, document.getElementById('app'));
```


write something

Your message: ?


something!

Your message: something!

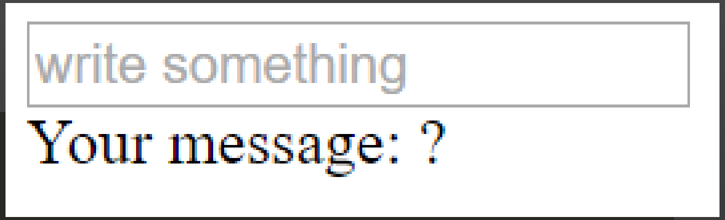
React Component (ES2015+)

```
class BoundText extends React.Component {

    constructor(props) {
        super(props)
        this.state = { message: '?' }
    }

    render() {
        return (
            <div>
                <input placeholder="write something"
                    onChange={(e) => this.setState({ message: e.target.value }) }
                /><br/>
                <span>Your message: { this.state.message }</span>
            </div>
    )}

}

ReactDOM.render(<BoundText/>, document.getElementById('app'));
```

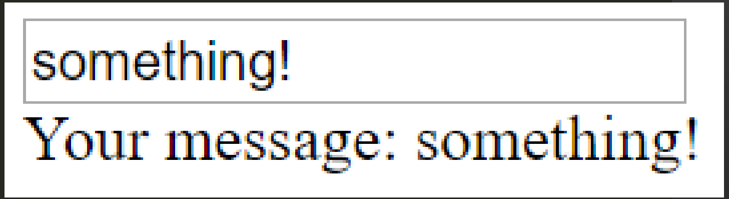
Initial value of state

Handlebars to access "JavaScript" from JSX

Update state



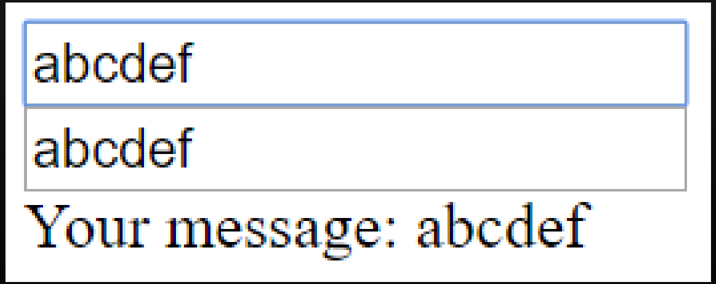
Your message: ?



Your message: something!

Two-Way Data Binding: View ⇔ Model

```
<input placeholder="write something"
    value={ this.state.message }
    onChange={(e) => this.setState({ message: e.target.value }) }
/><br/>
<input placeholder="write something"
    value={ this.state.message }
    onChange={(e) => this.setState({ message: e.target.value }) }
/><br/>
<span>Your message: { this.state.message }</span>
```



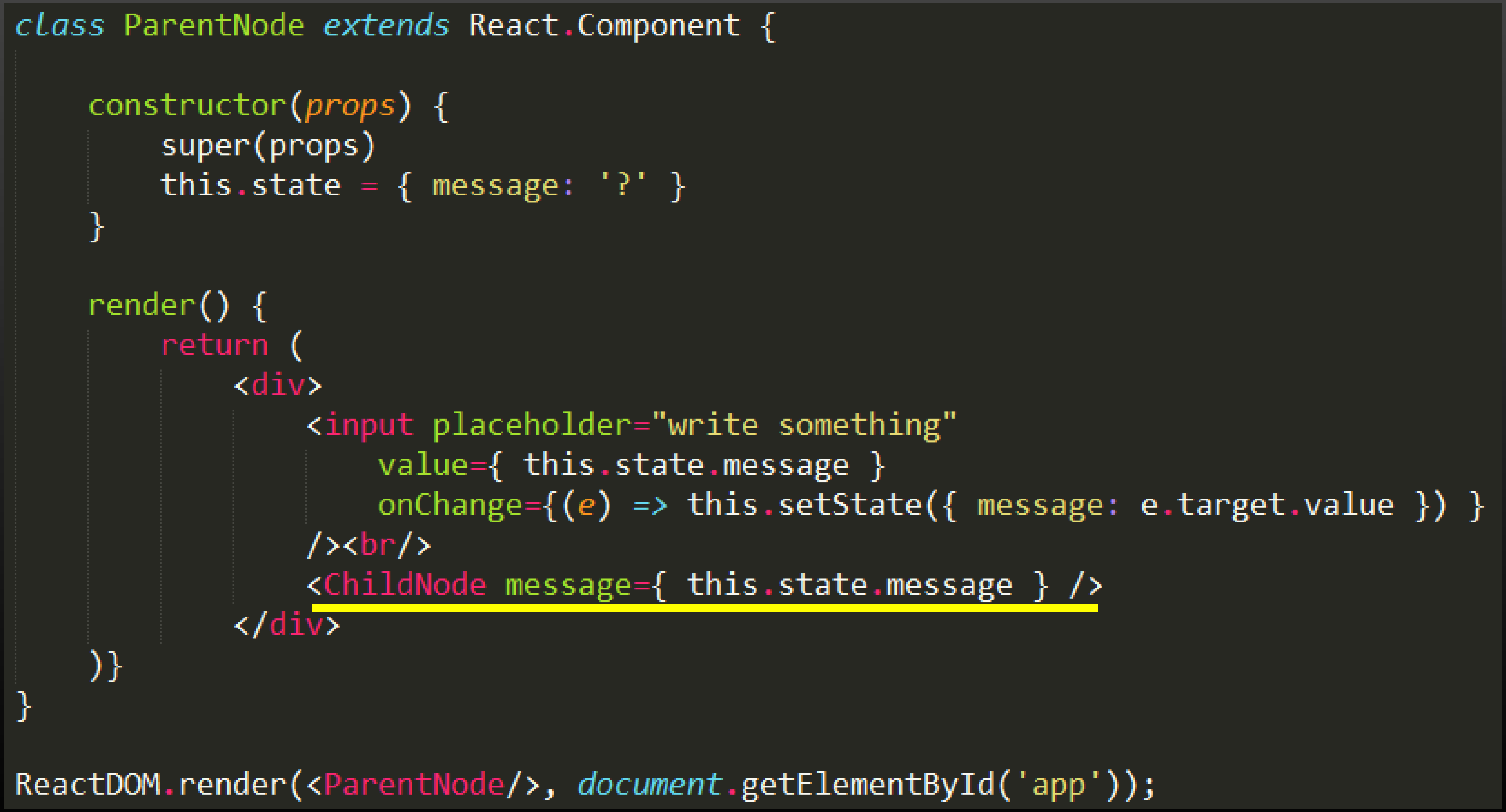
```
abcdef
abcdef
Your message: abcdef
```

Component Lifecycle Methods

- componentWillMount
  - Invoked once, on both client & server, before rendering occurs.
- componentDidMount
  - Invoked once, only on the client, after rendering occurs.
- componentWillReceiveProps
  - Invoked each time props change
- shouldComponentUpdate
  - Return value determines whether component should update.
- componentWillUpdate
  - Used for preprocessing before render is called
- componentDidUpdate
  - Postprocessing after render is called
- componentWillUnmount
  - Invoked prior to unmounting component.

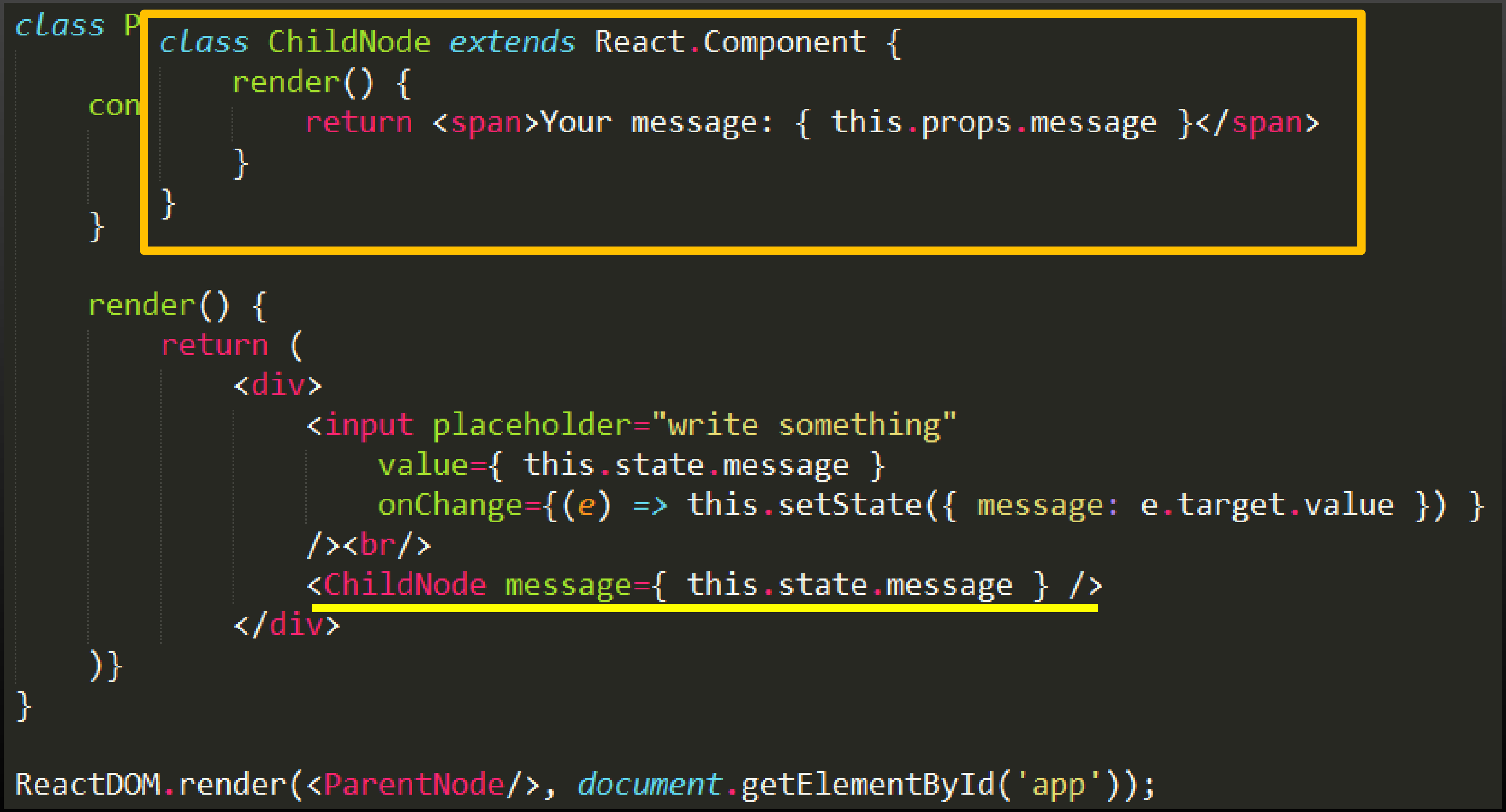
https://scotch.io/tutorials/learning-react-getting-started-and-concepts

Composing Containers and props



```
class ParentNode extends React.Component {

    constructor(props) {
        super(props)
        this.state = { message: '?' }
    }

    render() {
        return (
            <div>
                <input placeholder="write something"
                    value={ this.state.message }
                    onChange={(e) => this.setState({ message: e.target.value }) }
                /><br/>
                <ChildNode message={ this.state.message } />
            </div>
    )}
}

ReactDOM.render(<ParentNode/>, document.getElementById('app'));
```

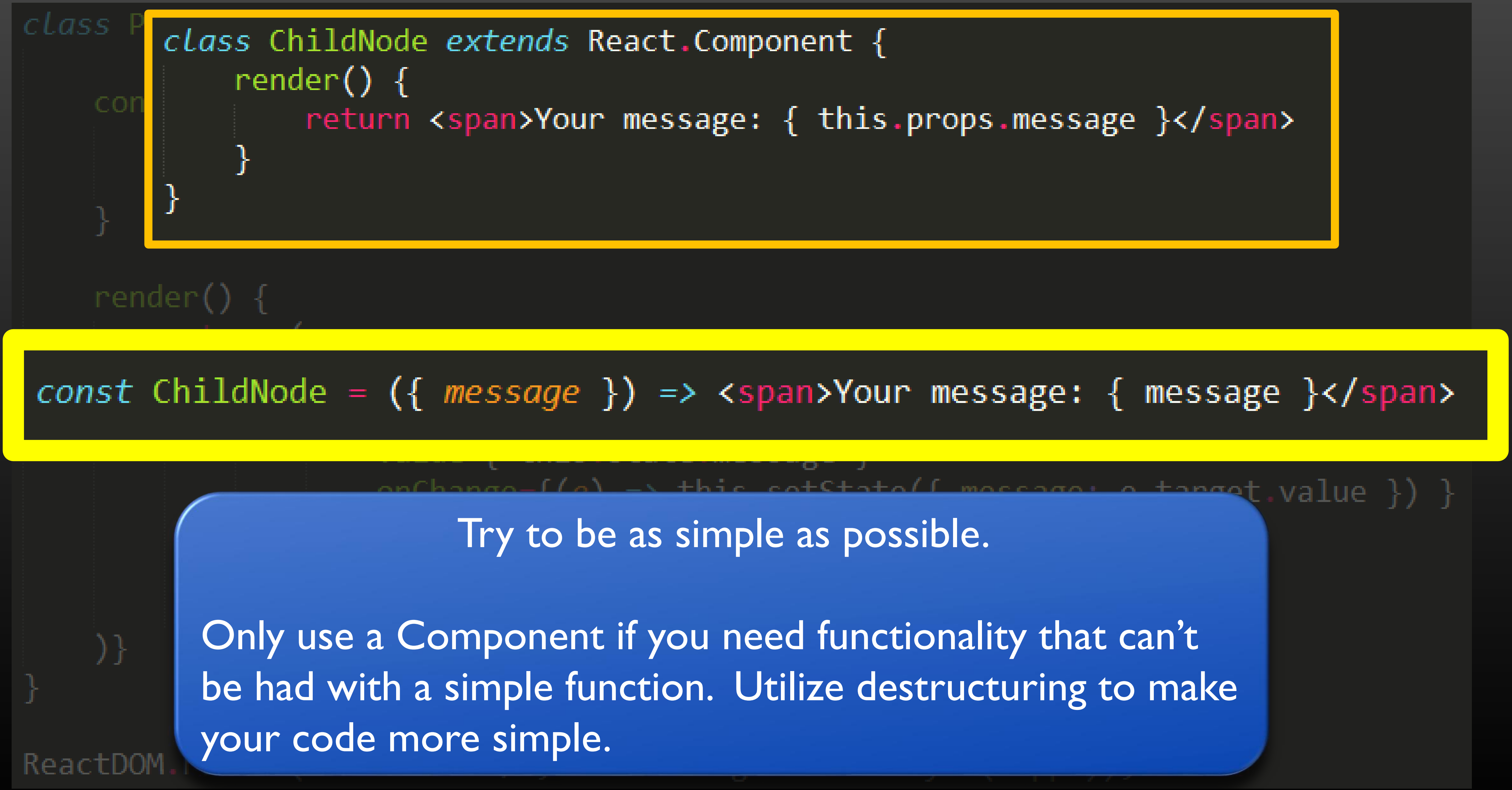
Composing Containers and props



```
class ChildNode extends React.Component {
    render() {
        return <span>Your message: { this.props.message }</span>
    }
}
```

```
class P
    
    con

    }

    render() {
        return (
            <div>
                <input placeholder="write something"
                    value={ this.state.message }
                    onChange={(e) => this.setState({ message: e.target.value }) }
                /><br/>
                <ChildNode message={ this.state.message } />
            </div>
    )}
}

ReactDOM.render(<ParentNode/>, document.getElementById('app'));
```

Composing Containers and props



```
class ChildNode extends React.Component {
    render() {
        return <span>Your message: { this.props.message }</span>
    }
}
```

```
const ChildNode = ({ message }) => <span>Your message: { message }</span>
```

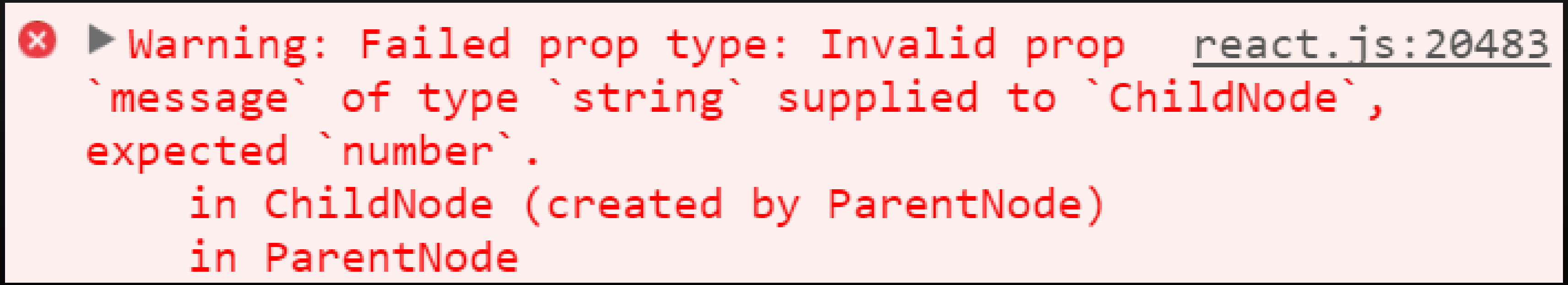
Try to be as simple as possible.

Only use a Component if you need functionality that can't be had with a simple function. Utilize destructuring to make your code more simple.

PropTypes

React will perform type checking of properties
helps during development and debugging by providing console messages

```
const ChildNode = ({ message }) => <span>Your message: { message }</span>

ChildNode.propTypes = {
    message: React.PropTypes.number.isRequired
}
```



```
Warning: Failed prop type: Invalid prop    react.js:20483
`message` of type `string` supplied to `ChildNode`,
expected `number`.
    in ChildNode (created by ParentNode)
    in ParentNode
```

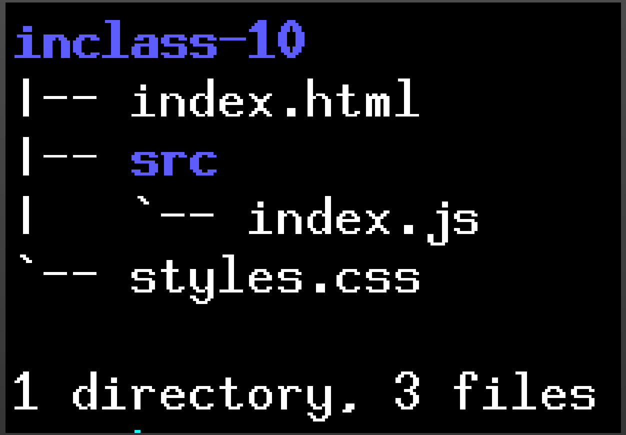
Resources

- Read this: https://facebook.github.io/react/docs/thinking-in-react.html
- React Docs https://facebook.github.io/react/docs/getting-started.html

VIDEO HOMEWORK

- Watch this series over the weekend ~3 hrs or so https://egghead.io/courses/getting-started-with-redux

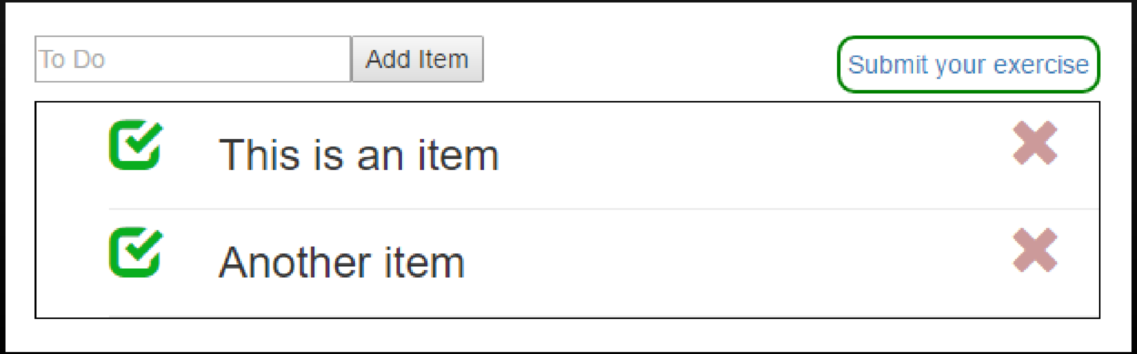
In-Class Exercise: Hello React



```
inclass-10
|-- index.html
|-- src
|   `-- index.js
`-- styles.css

1 directory, 3 files
```

/inclass-10/...

- Download and unzip https://www.clear.rice.edu/comp431/sample/helloReact.zip
- We are going to reimplement the TODO app in React
- Your task is to implement the *two* **render** functions and `ToDos.addTodo()` in `src/index.js`
- When completed the page should load like the image below
  - The check box should be functional (strike through)
  - The X should remove the task
  - "Add Item" adds new items.


To Do
Add Item
Submit your exercise
This is an item
Another item