THE PRAGMATIC INTRO TO REACT

Clayton Anderson

thebhwgroup.com

WEB AND MOBILE APP DEVELOPMENT

AUSTIN, TX

REACT

"A JavaScript library for building user interfaces"

But first...

HOW WE GOT HERE

OR: A BRIEF HISTORY OF WEB DEVELOPMENT

- Our apps vary greatly in size and needs
- Developers work on multiple projects
- (Not everyone's history will be the same)

I'm going to walk through broad eras of web development history, and describe pros and cons of each

THE FIRST ERA: ALL SERVER RENDERED

For most of the history of the web this was the norm. Web pages were simple, JavaScript support was bad, and there was little dynamic client behavior.

Capabilities were limited during this era, but it was much easier to understand your entire application

1. ALL SERVER RENDERED

PROS

- Less required knowledge. No JS
- Easy to reason about and scale
- Good search engine optimization

1. ALL SERVER RENDERED

CONS

- All interactions require full round trip and server render
- Limited ability for highly dynamic pages

THE SECOND ERA: SERVER RENDERED, BUT WITH INCREMENTAL JAVASCRIPT

With the introduction of libraries like jQuery, it had become much easier to add cool client-side features like Ajax requests, form validation, or animations.

This is really when we started thinking about "web apps" vs "websites". Browsers were starting to get better too

2. SERVER RENDERED, BUT WITH INCREMENTAL JAVASCRIPT

PROS

- Gentle learning curve
- Easy to drop in small pieces of jQuery as necessary
- Good SEO
- Huge plugin ecosystem

2. SERVER RENDERED, BUT WITH INCREMENTAL JAVASCRIPT

CONS

- Likely to repeat some view logic between server and client
- Selector hell and entangled JS and HTML
- Hard to reason about
- Difficult to scale
- Hard to test
- Huge plugin ecosystem

As a note before continuing, jQuery is a wonderful and useful tool. These pain points aren't so much a criticism of jQuery itself but rather with trying to build large apps with it.

THE THIRD ERA: THE MONOLITHIC FRAMEWORK

For us, this was AngularJS. While it was far better suited to building large apps than plain jQuery, it introduced a long list of its own problems.

3. THE MONOLITHIC FRAMEWORK

PROS

- No more element selectors!
- If you were making an SPA, you had a cleaner separation between server and client
- Easier to scale

Imperative vs declarative

With jQuery, we thought "What actions do I perform to get my view to look the way I want?". But with Angular, now we ask "What data do I change so that my view looks the way I want?"

3. THE MONOLITHIC FRAMEWORK

CONS

- Poor performance
- Extremely steep learning curve
- Hard to debug
- Contains conflicting approaches for accomplishing the same task
- Wants to own your entire app
- Little to no SEO

The point isn't to dunk on Angular, but instead to show the history so we can learn from it

THE FOURTH ERA: REACT!

The goal here isn't to say that React is perfect or will be used forever, but rather that it does a tremendous job of addressing all the pain points of the prior eras, all while being fun use and learn.

4. REACT

PROS

- Easy to learn and reason about
- Easy to test
- Good SEO, from shared client / server rendering
- Maintains clean separation between SPA and API server
- Good performance, and clear path to tuning when it goes awry
- Straightforward integration with a larger app or other libraries

For this presentation, we are going to focus on React being easy to learn and reason about

4. REACT

CONS

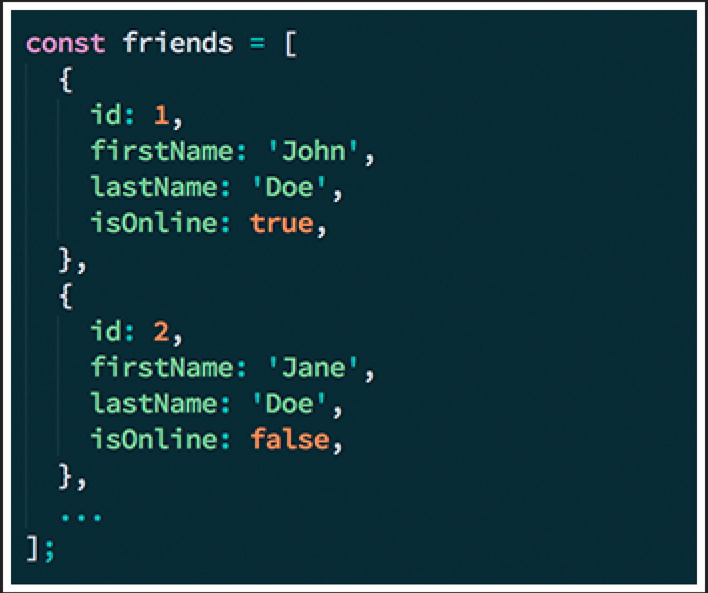
- React server rendering still isn't as fast as the traditional methods
- Some people prefer a more all encompassing framework
- ???

We don't have the historical perspective yet

SO WHAT IS REACT?

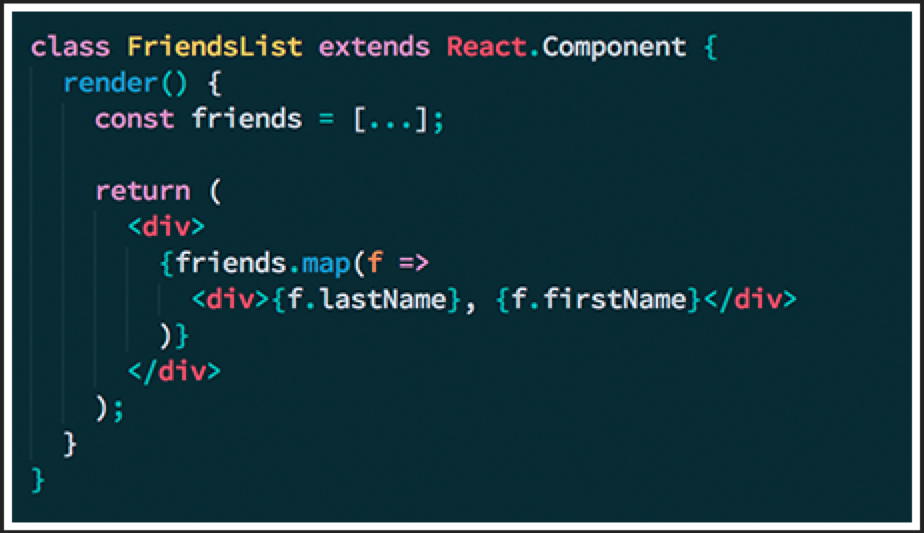
React is a view library that abstracts away DOM manipulations. Instead of the developer having to imperatively modify or insert into the DOM as the state of their app changes, React uses declarative `components` that do the messy work for them, in a performant way. This lets the developer focus on what really matters: Given some input data, what is the rendered output of their app?

LET'S MAKE A FRIENDS LIST WIDGET



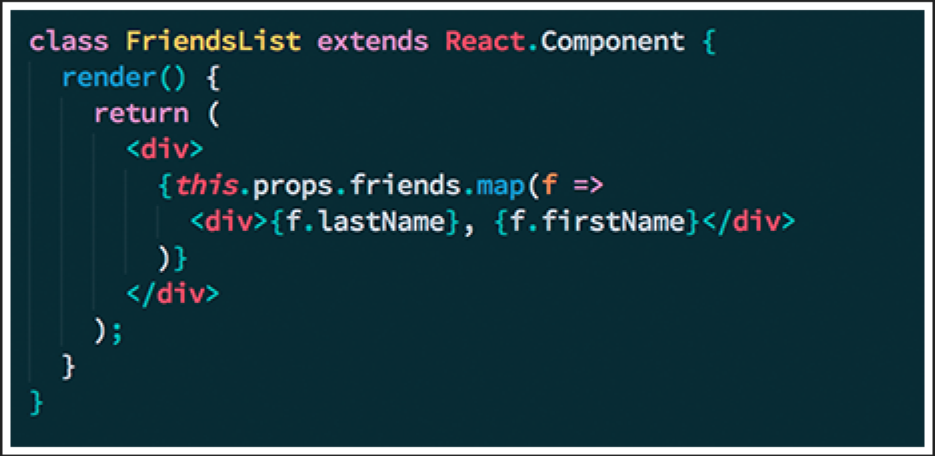
```
const friends = [
  {
    id: 1,
    firstName: 'John',
    lastName: 'Doe',
    isOnline: true,
  },
  {
    id: 2,
    firstName: 'Jane',
    lastName: 'Doe',
    isOnline: false,
  },
  ...
];
```

Data structure used in next couple code samples. Don't worry about styles, just show the list of friends' names, and filter whether they are online



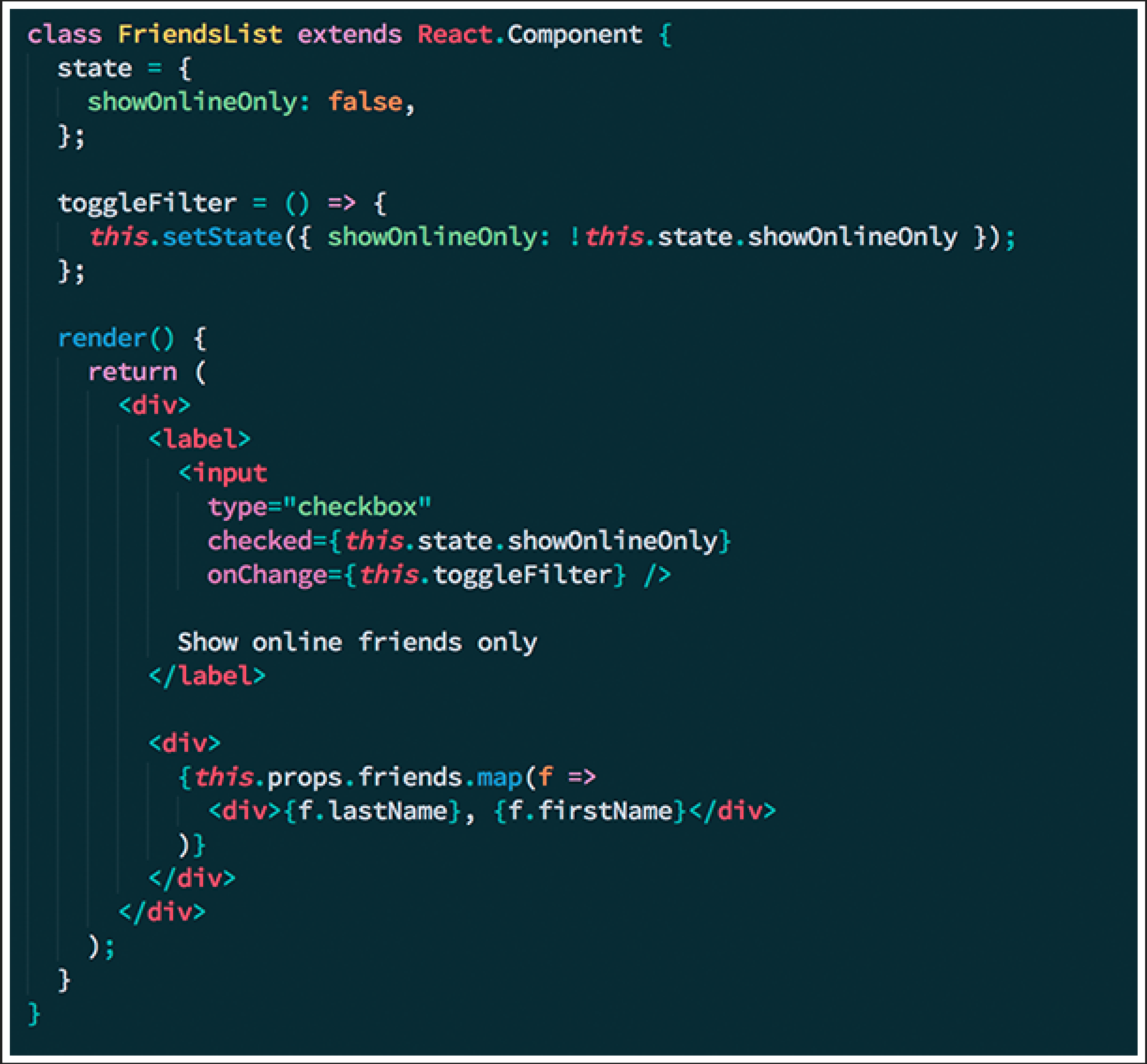
```
class FriendsList extends React.Component {
  render() {
    const friends = [...];

    return (
      <div>
        {friends.map(f =>
          <div>{f.lastName}, {f.firstName}</div>
        )}
      </div>
    );
  }
}
```

- Render describes how the DOM should look
- Called by React whenever something changes
- Render is the only requirement of a component
- How do we pass data in?



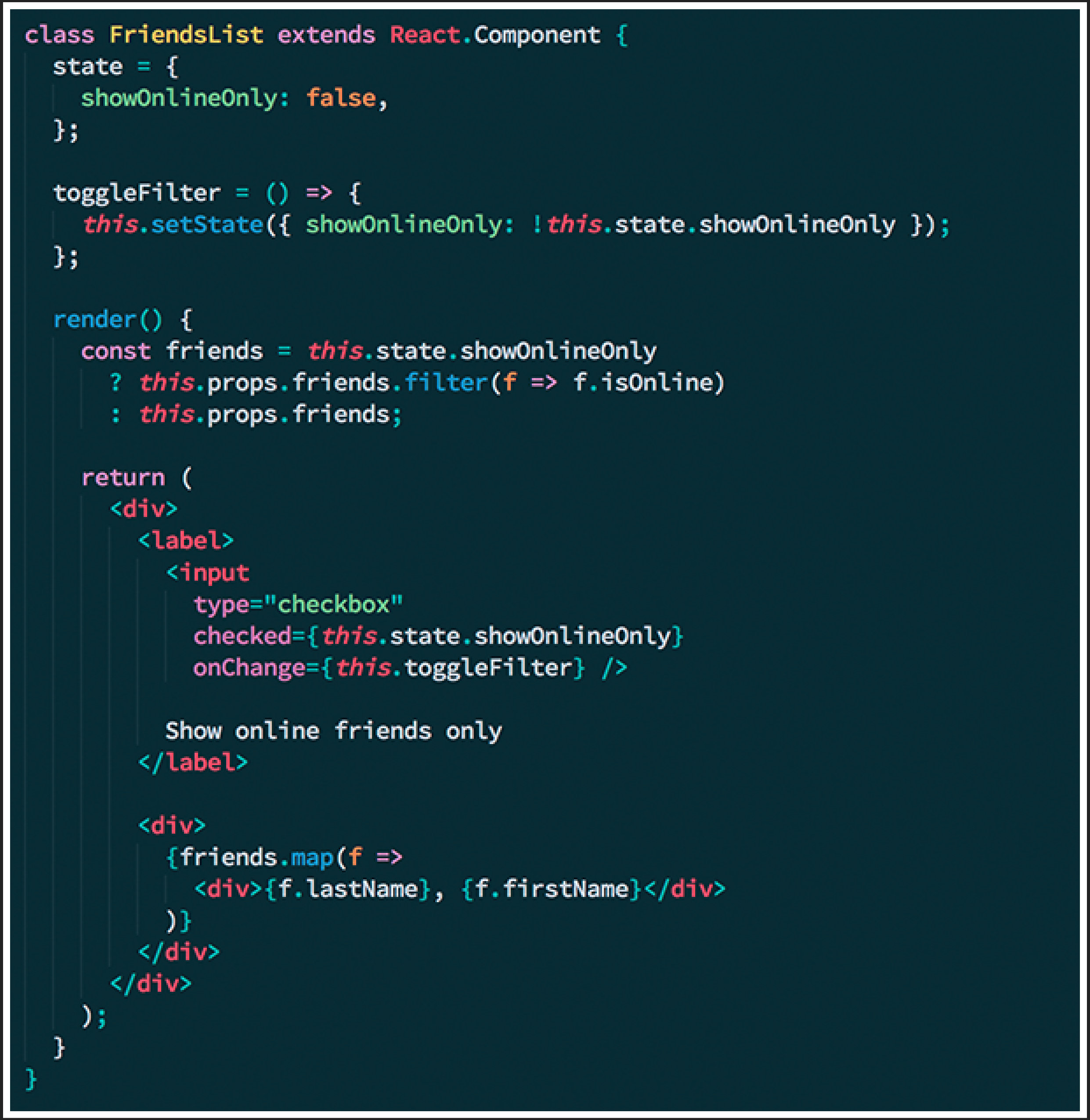
```
class FriendsList extends React.Component {
  render() {
    return (
      <div>
        {this.props.friends.map(f =>
          <div>{f.lastName}, {f.firstName}</div>
        )}
      </div>
    );
  }
}
```

- Props are data passed from the parent
- Props are immutable



```
class FriendsList extends React.Component {
  state = {
    showOnlineOnly: false,
  };

  toggleFilter = () => {
    this.setState({ showOnlineOnly: !this.state.showOnlineOnly });
  };

  render() {
    return (
      <div>
        <label>
          <input
            type="checkbox"
            checked={this.state.showOnlineOnly}
            onChange={this.toggleFilter} />

          Show online friends only
        </label>

        <div>
          {this.props.friends.map(f =>
            <div>{f.lastName}, {f.firstName}</div>
          )}
        </div>
      </div>
    );
  }
}
```

- State is mutable data managed by the component
- State at one level could be a prop at another level
- Meaning there is only one location where a given piece of data can be changed!



```
class FriendsList extends React.Component {
  state = {
    showOnlineOnly: false,
  };

  toggleFilter = () => {
    this.setState({ showOnlineOnly: !this.state.showOnlineOnly });
  };

  render() {
    const friends = this.state.showOnlineOnly
      ? this.props.friends.filter(f => f.isOnline)
      : this.props.friends;

    return (
      <div>
        <label>
          <input
            type="checkbox"
            checked={this.state.showOnlineOnly}
            onChange={this.toggleFilter} />

          Show online friends only
        </label>

        <div>
          {friends.map(f =>
            <div>{f.lastName}, {f.firstName}</div>
          )}
        </div>
      </div>
    );
  }
}
```

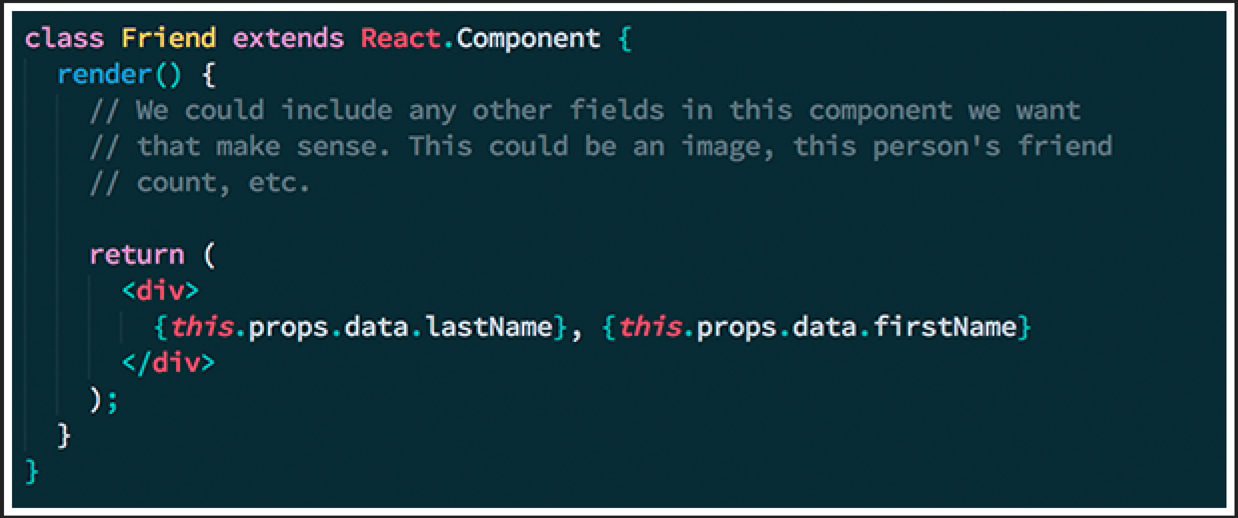
Explain data filtering

REACT IS EASY TO LEARN

- Builds upon JavaScript fundamentals: We used the native array's map and filter functions
- Most components consist of nothing more complicated than render, props, and state
- Your data structures are plain objects and arrays

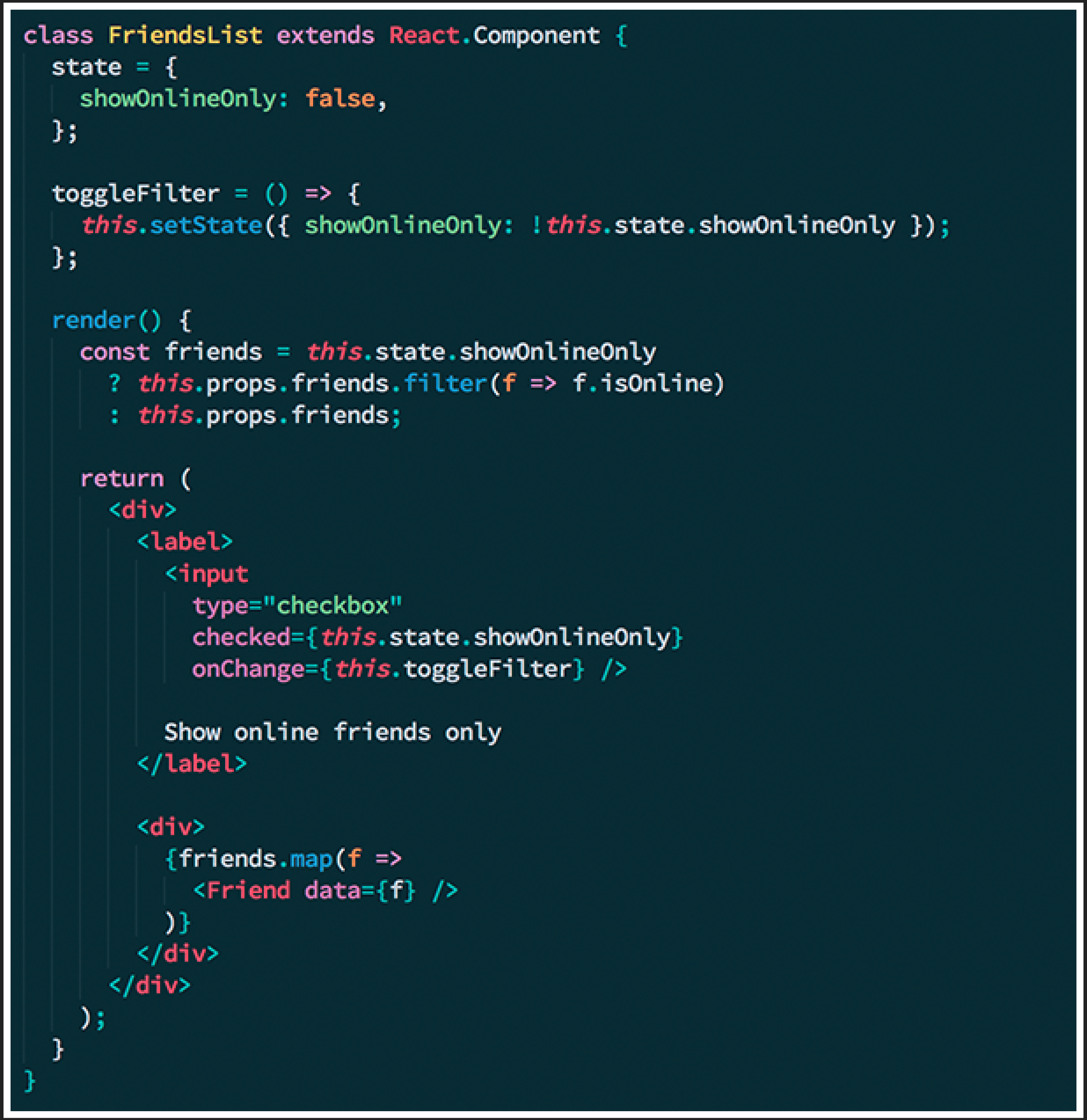
If you know JS, you'll be productive with React quickly. If you don't, it'll help teach you JS

HOW WOULD WE EXTEND THE FRIENDS LIST COMPONENT?



```
class Friend extends React.Component {
  render() {
    // We could include any other fields in this component we want
    // that make sense. This could be an image, this person's friend
    // count, etc.

    return (
      <div>
        {this.props.data.lastName}, {this.props.data.firstName}
      </div>
    );
  }
}
```

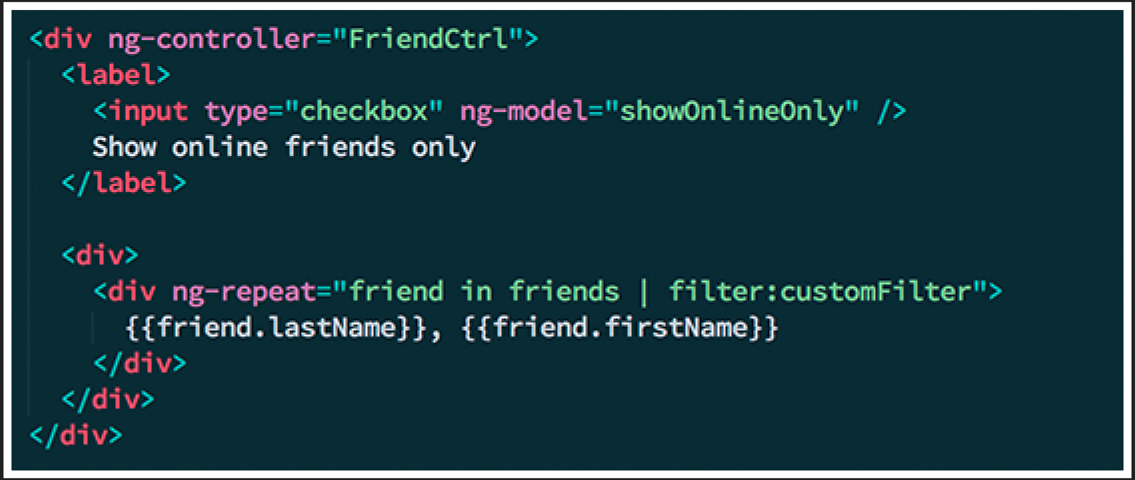
We are hypothetically adding quite a bit to each friend, so it deserves its own component



```
class FriendsList extends React.Component {
  state = {
    showOnlineOnly: false,
  };

  toggleFilter = () => {
    this.setState({ showOnlineOnly: !this.state.showOnlineOnly });
  };

  render() {
    const friends = this.state.showOnlineOnly
      ? this.props.friends.filter(f => f.isOnline)
      : this.props.friends;

    return (
      <div>
        <label>
          <input
            type="checkbox"
            checked={this.state.showOnlineOnly}
            onChange={this.toggleFilter} />

          Show online friends only
        </label>

        <div>
          {friends.map(f =>
            <Friend data={f} />
          )}
        </div>
      </div>
    );
  }
}
```

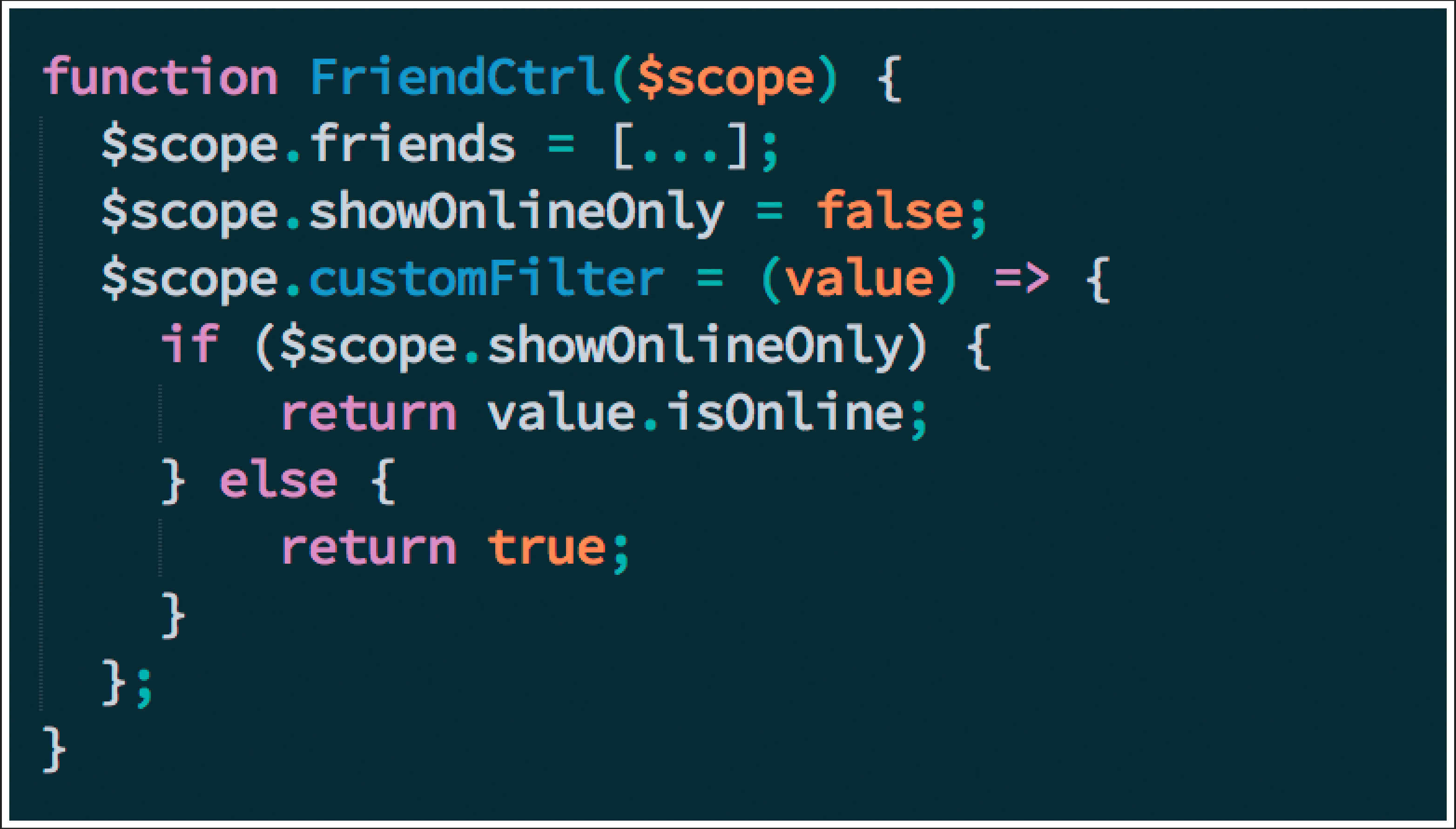
Component composition

ANGULAR IMPLEMENTATION



```
<div ng-controller="FriendCtrl">
  <label>
    <input type="checkbox" ng-model="showOnlineOnly" />
    Show online friends only
  </label>

  <div>
    <div ng-repeat="friend in friends | filter:customFilter">
      {{friend.lastName}}, {{friend.firstName}}
    </div>
  </div>
</div>
```

ANGULAR IMPLEMENTATION



```
function FriendCtrl($scope) {
  $scope.friends = [...];
  $scope.showOnlineOnly = false;
  $scope.customFilter = (value) => {
    if ($scope.showOnlineOnly) {
        return value.isOnline;
    } else {
        return true;
    }
  };
}
```

ANGULAR QUESTIONS

- What is `$scope`?
- What are these `ng-` attributes?
- What is the syntax for `filter`?
- And how would I extend or alter the behavior?

- Many of the changes you might make here require switching between two files
- There is very little plain JS. Learning Angular just teaches you Angular

STATISTICS TO CONSIDER

Metric	Angular	React
GitHub stars	47,454	37,347
GitHub watchers	4,196	2,741
StackOverflow questions	156,195	11,247

We can't tell exactly which is used more from these numbers, but they're in the same ballpark. But when you look at the number of SO questions, Angular has over 10 times as many

REACT IS EASY TO REASON ABOUT

With React, render output is a pure function of props and state. Given the same props and state, a React component should always render the same output

This is the single most important benefit of React. This is what makes React so simple to reason about, test, and debug.

- Consistency
- Static analysis

REACT IS EASY TO REASON ABOUT

- Colocation of concerns
- Your entire view layer is a tree of components
- Synchronous render
- Clear control & data flow
- Performance: Easy to prune branches of tree that don't need to re-render

CONCLUSION

There has never been a better time to start using React! It has been proven in production, and there is a great community putting out free educational resources.

- Facebook React Tutorial
- Removing User Interface Complexity, or Why React is Awesome

THANK YOU!

QUESTIONS?