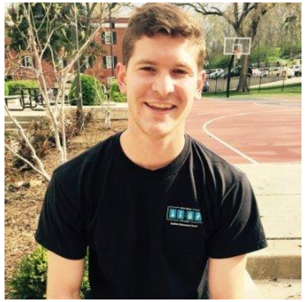
ACM@KU Workshop

Learning React

Facebook’s front end Javascript framework

About Me



- I am Evan
- Senior CS student
- Freelance web/mobile developer
  - KU Business School, KU Med Center, Directional Drilling Company
- React/React Native/GatsbyJS
- After graduation: EngRes @ Google NYC
- KU basketball junkie, live music fan, graphic design, journaling

React Roadmap

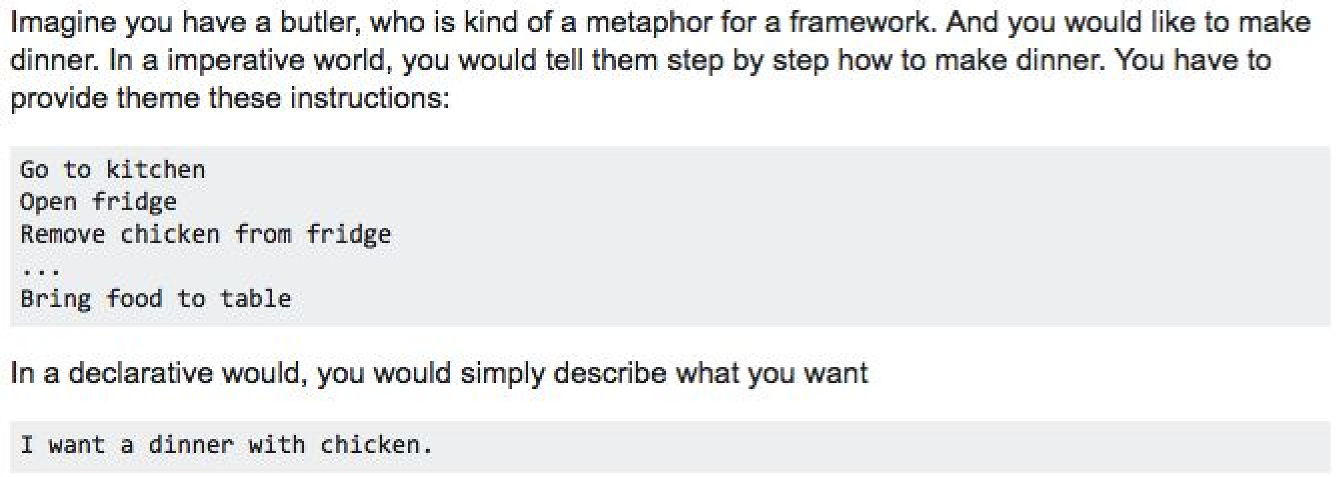
- Overview
- Declarative Programming
- Components
- Props
- State
- Lots of Examples
- References/Further Reading
- Q&A

Overview

- **React is a declarative, efficient, and flexible JavaScript library for building user interfaces.**
- Lots of big companies use React (Airbnb, BBC, Chegg, Cerner, Imgur, etc)
- React allows you to easily manage application state and create concise, logical, reusable components.
- One-way data flow: Parent → Child
- Virtual DOM (Advanced Topic: Read More Here if interested)
  - TL;DR: React saves us from doing low-level, imperative interactions with the DOM, and the virtual DOM allows React to do this very efficiently through “diffing”
- Open Source (download the source code + tons of examples!)

Declarative Programming

- Declarative vs. Imperative: what’s the difference?



Imagine you have a butler, who is kind of a metaphor for a framework. And you would like to make dinner. In a imperative world, you would tell them step by step how to make dinner. You have to provide theme these instructions:

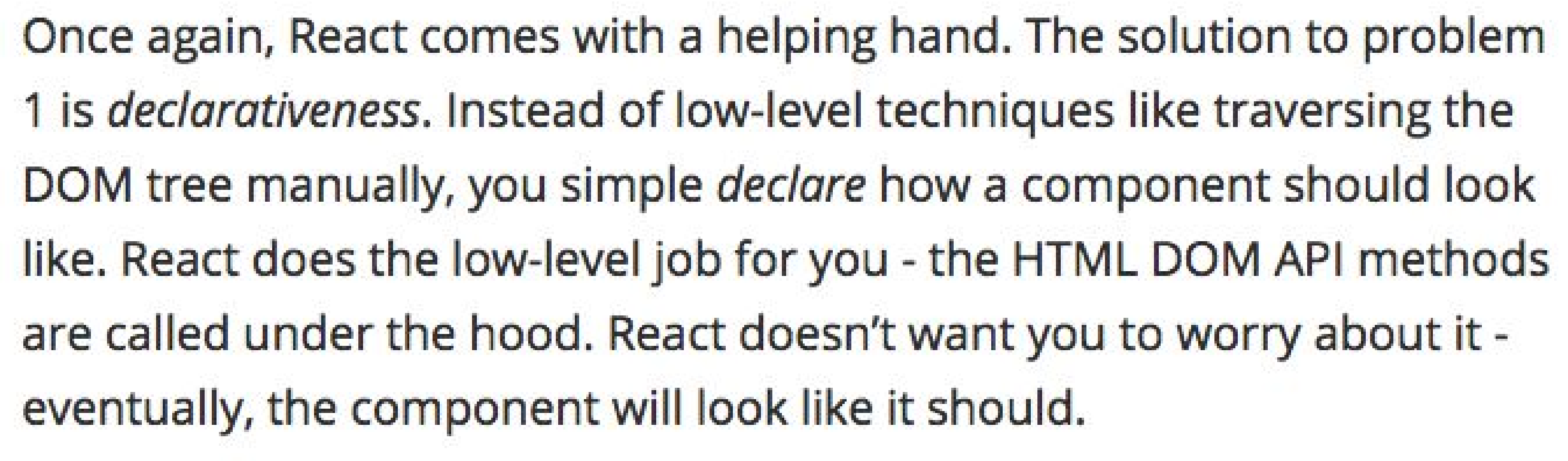
```
Go to kitchen
Open fridge
Remove chicken from fridge
...
Bring food to table
```

In a declarative would, you would simply describe what you want

```
I want a dinner with chicken.
```

- Declarative programming “abstracts” away the imperative instructions: we tell it what we want and leave the implementation to the framework.

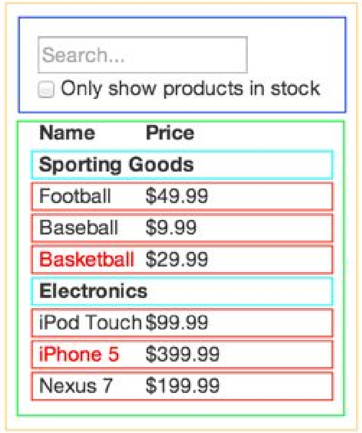
Declarative Programming Con’t



Once again, React comes with a helping hand. The solution to problem 1 is *declarativeness*. Instead of low-level techniques like traversing the DOM tree manually, you simple *declare* how a component should look like. React does the low-level job for you - the HTML DOM API methods are called under the hood. React doesn’t want you to worry about it - eventually, the component will look like it should.

Components

- Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.
- Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called "props") and return React elements describing what should appear on the screen.
- Example: “Componentize” a Blog Post

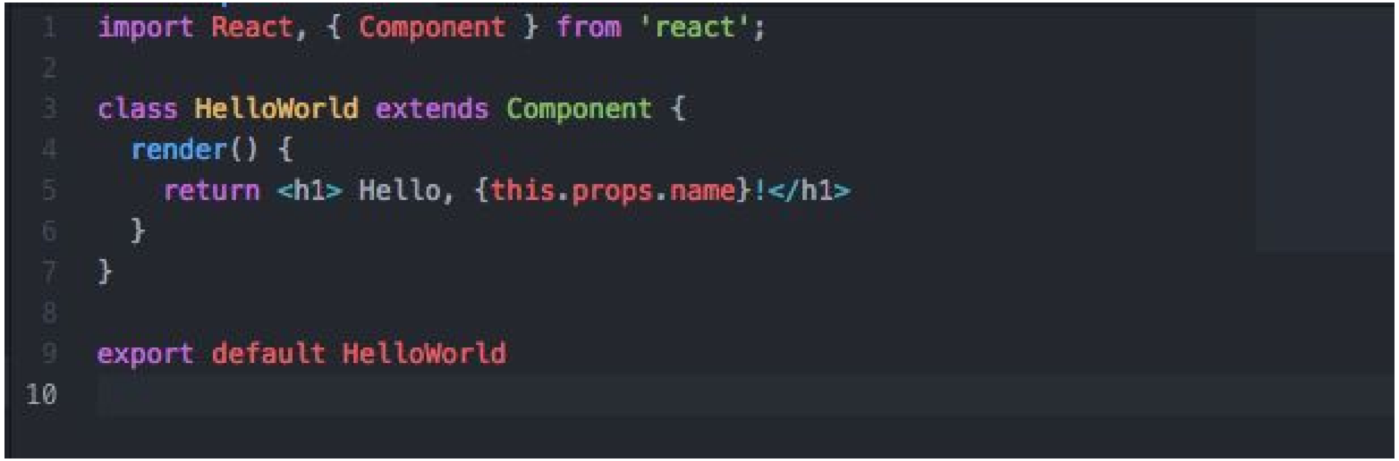


Search...

☐ Only show products in stock

Name	Price
Sporting Goods	
Football	$49.99
Baseball	$9.99
Basketball	$29.99
Electronics	
iPod Touch	$99.99
iPhone 5	$399.99
Nexus 7	$199.99

Our First Components: HelloWorld.js

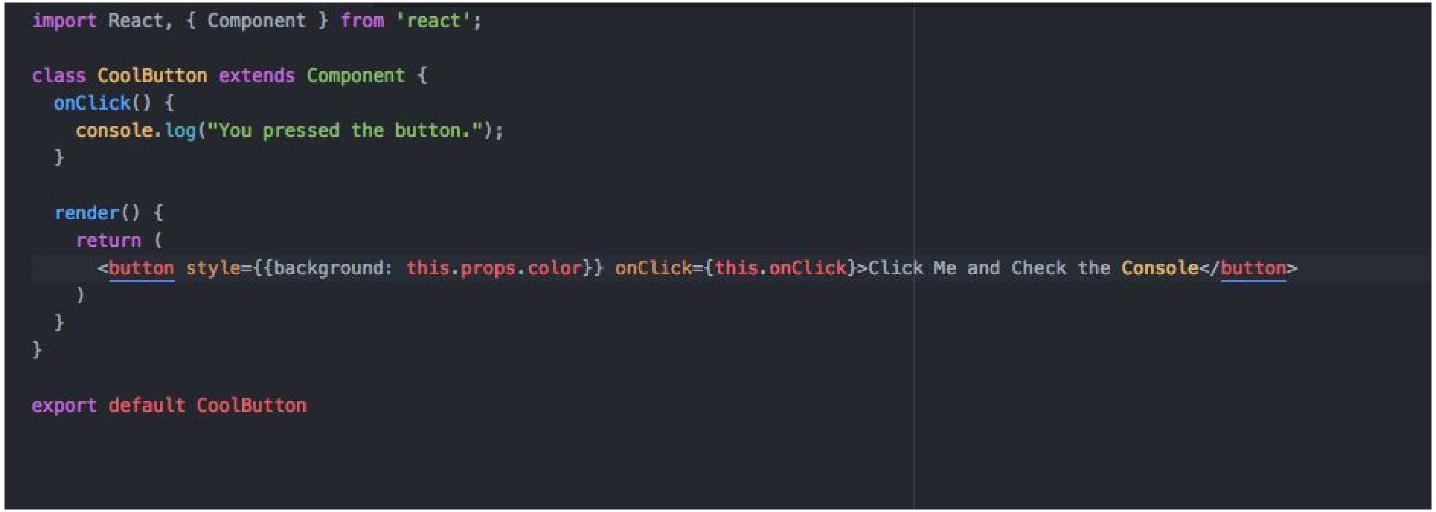


```
import React, { Component } from 'react';

class HelloWorld extends Component {
  render() {
    return <h1> Hello, {this.props.name}!</h1>
  }
}

export default HelloWorld

```

- render( ) method
- *this*… what is *this*…?
- What's export do? *"Take this class and let us use it in other files"*
- TO DO: Add an adjective prop

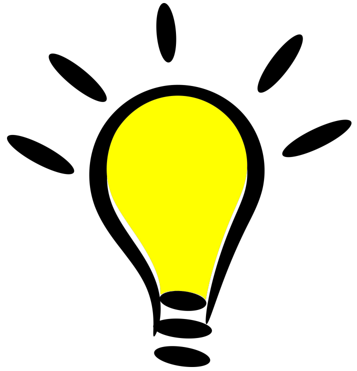
What are Props?

- “Arbitrary inputs” to a React Component
- Really, arbitrary
- E.g. “Title,” “color,” “Year,” “Data”
- Example: CoolButton



```
import React, { Component } from 'react';

class CoolButton extends Component {
  onClick() {
    console.log("You pressed the button.");
  }

  render() {
    return (
      <button style={{background: this.props.color}} onClick={this.onClick}>Click Me and Check the Console</button>
    )
  }
}

export default CoolButton
```

What is State?

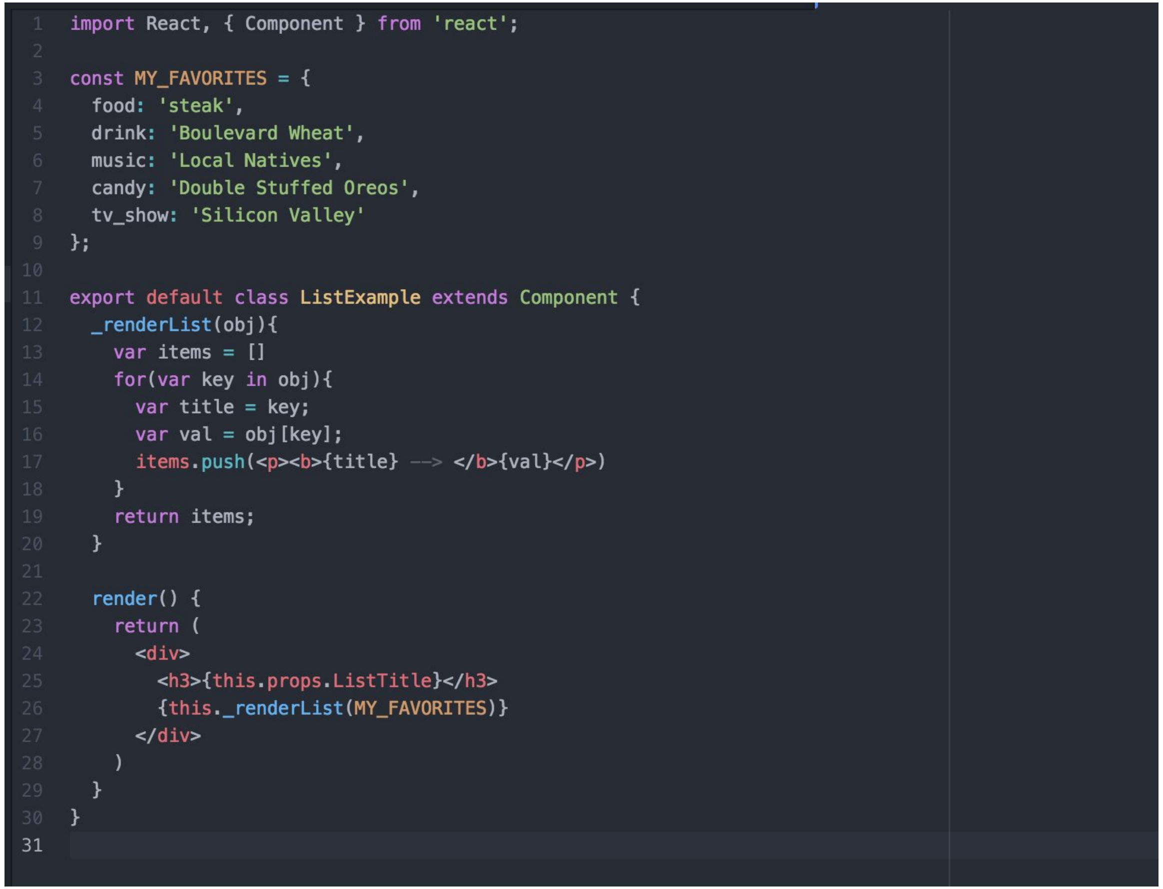


- A variable/changing value related to a React component
- Unlike props, it is NOT read-only, i.e. state gets updated
- (Advanced Topic: State and Lifecycle from the React Docs)
- Examples of when/how to use state
  - Your app is receiving data from an external database -- it needs to be loaded when the application “mounts”
  - User interaction/User Input (example: ClickCounter)
  - Timers
- Keep components stateless unless there is good reason not to.

Props vs. State: Recap

- We use props to pass *unchanging/static* values from parent components down to children components
- We use state to keep track of changing/updating values related to a component.
    - There are some built-in React functions related to the “lifecycle” of a component that we’ll save for another time.
- Use state only when needed. Things can get complex quickly.

Hands On: ListExample



```
import React, { Component } from 'react';

const MY_FAVORITES = {
  food: 'steak',
  drink: 'Boulevard Wheat',
  music: 'Local Natives',
  candy: 'Double Stuffed Oreos',
  tv_show: 'Silicon Valley'
};

export default class ListExample extends Component {
  _renderList(obj){
    var items = []
    for(var key in obj){
      var title = key;
      var val = obj[key];
      items.push(<p><b>{title} --> </b>{val}</p>)
    }
    return items;
  }

  render() {
    return (
      <div>
        <h3>{this.props.ListTitle}</h3>
        {this._renderList(MY_FAVORITES)}
      </div>
    )
  }
}

```

References/Further Reading

- React Docs: Hello World
    - https://facebook.github.io/react/docs/hello-world.html
- React Official Examples
    - https://github.com/facebook/react/tree/master/examples
- The Difference Between Virtual DOM and DOM
    - http://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/
- A collection of awesome things regarding React ecosystem
    - https://github.com/enaqx/awesome-react
- GatsbyJS: React-based static site generator
    - https://github.com/gatsbyjs/gatsby
-

Questions?