ES6 ARROW FUNCTIONS

- Before we jump into react, let’s take a look at some new ES6.
- As we talked about in the last class, in 2015 there was a major update to Javscript. One of the changes that was introduced was a different way to write functions. It’s pretty similar to the way we write the function expression but with some funky syntax:

ES5 function expression

```
var sayHi = function(name) {
 return ‘hi’ + name;
}
```

Vs

ES6 function expression

```
var sayHi = (name)=>{
 return ‘hi’ + name;
}
```

ES6 ARROW FUNCTIONS

- One thing to know about arrow functions is that they remove contextual binding of this.
- We’ll talk a bit more about this, but you can use arrow functions anywhere you can use other functions, EXCEPT you most likely won’t want to use them in classes.
- Using an arrow function in a class will make it so that your ‘this’ won’t work in any of the objects made from the class.
- We will be using arrow functions in React. More on that later.

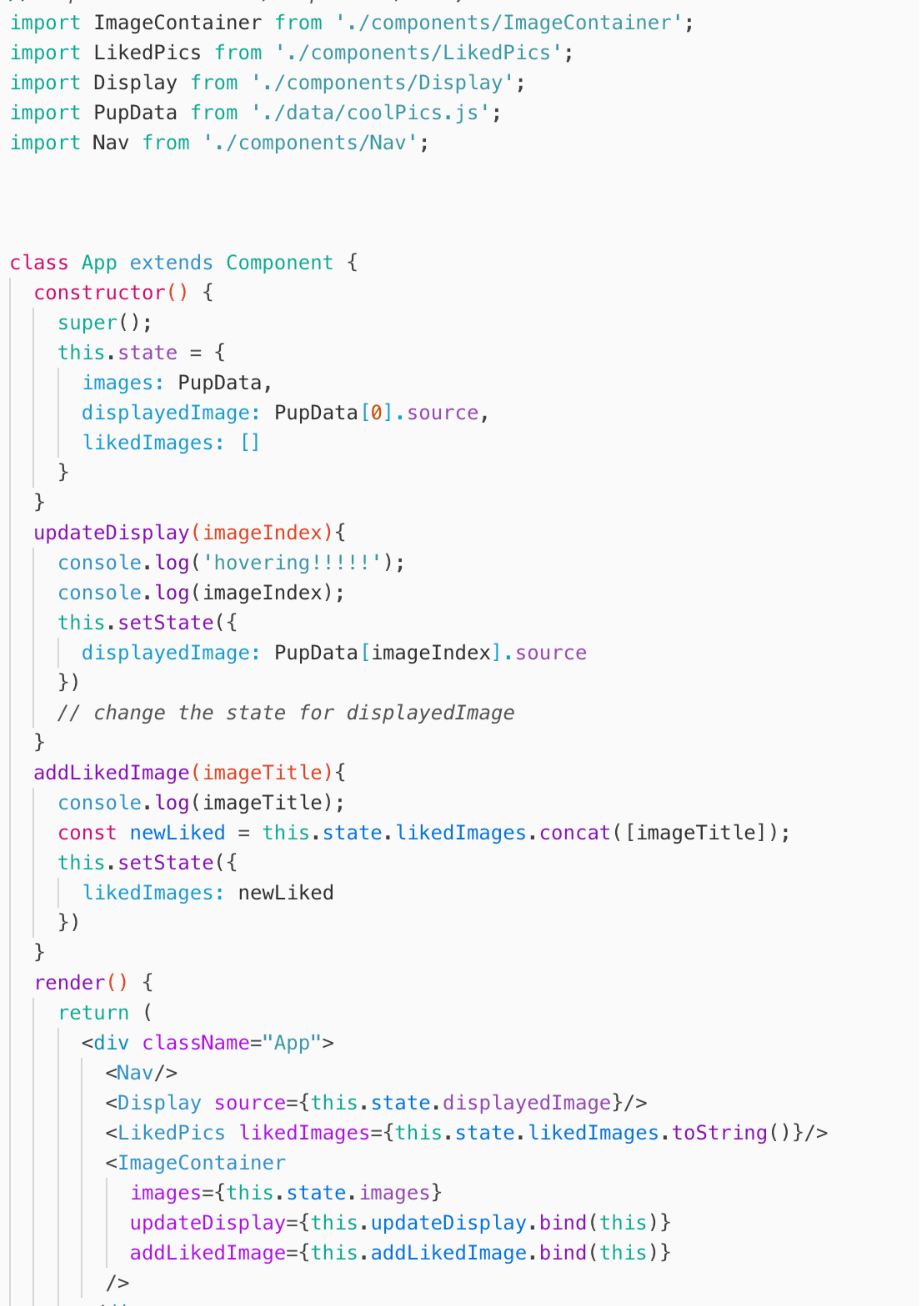
REACT

WHY REACT

- With the evolution of user experiences going towards single page applications there became a need to make single-page applications faster and more scaleable.
- Working with jQuery is fantastic for smaller to medium-large applications, but when we get to mega-scale applications (think facebook) it can be hard to keep your pages neat
- React is also faster than classic options. This speed benefit is more noticeable at larger scale and with slower computers, things that large websites have to worry about.

WHY REACT

- React has a focus on making things into “components”, or re-usable bits that we can re-use over and over again.
- You remember the Harry Potter homework? We had to create and recreate pieces of code over and over again.
- React inherently has a solution for this problem. More on this later, but generally speaking encourages reusable pieces.



```
import ImageContainer from './components/ImageContainer';
import LikedPics from './components/LikedPics';
import Display from './components/Display';
import PupData from './data/coolPics.js';
import Nav from './components/Nav';



class App extends Component {
  constructor() {
    super();
    this.state = {
      images: PupData,
      displayedImage: PupData[0].source,
      likedImages: []
    }
  }
  updateDisplay(imageIndex){
    console.log('hovering!!!!!');
    console.log(imageIndex);
    this.setState({
      displayedImage: PupData[imageIndex].source
    })
    // change the state for displayedImage
  }
  addLikedImage(imageTitle){
    console.log(imageTitle);
    const newLiked = this.state.likedImages.concat([imageTitle]);
    this.setState({
      likedImages: newLiked
    })
  }
  render() {
    return (
      <div className="App">
        <Nav/>
        <Display source={this.state.displayedImage}/>
        <LikedPics likedImages={this.state.likedImages.toString()}/>
        <ImageContainer
          images={this.state.images}
          updateDisplay={this.updateDisplay.bind(this)}
          addLikedImage={this.addLikedImage.bind(this)}
        />
```

WHAT DOES REACT LOOK LIKE?

- Let's open this up in a text editor so we can get a closer look.
- What do we see?

WHAT IS IT LIKE TO USE REACT?

- It’s very challenging at first.
- React is very restrictive in the way you can do things.
- In the big picture this is a benefit. Once you know the “React way” you know what to do when you approach a new problem.
- It feels like solving a Rubic’s cube (not that I can do that). There are a few different patterns to memorize, and once you have it down you can recognize the problem and apply a pattern to most situations.

WHY IS REACT FAST? VIRTUAL DOM

- In the old times, re-rendering one thing meant re-rendering everything. This had negative implications on processing power and ultimately user experience, which at times became glitchy and laggy.
- A lot of the times, even with jQuery the easiest solution can be to re-render a large portion of the DOM even for simple tasks.
- Enter the Virtual DOM.

VIRTUAL DOM

- Remember the DOM, we had described it as an object interface for a rendered site.
- React adds another layer between the DOM and our javascript. The virtual DOM.
- When changes are made to the view we want to show, we update the Virtual DOM first, which then checks the diffs between what was changed to what is currently rendered, and changes ONLY the pieces that need to be changed, rather than re-rendering the entire page. You can think about it like a staging area for changes that will eventually be implemented.

JSX

- React has created something called a “javascript syntax extension” that you use while writing React apps called **JSX**.
- JSX stands for “Javascript and XML”.
- It is called this because the inspiration for it looks like Javascript and XML combined.
- Let’s open up that React page again to inspect what I mean.
- The upper part is just regular ES6 classes in Javascript, then at the bottom there is a render function where there is some html and some XML like parts to it.

JSX AND BABEL

- Inside of React there is a transpiler that converts all the JSX into plain ES5 (< 2015) javascript.
- We can go to Babel’s website and see how it works.
- You can see how everything we type in React (which we’ll know soon) is automatically converted with Babel.
- Key takeaway: we use JSX to make our coding easier. Ultimately this code is transpired by Babel into vanilla JS.

CREATE REACT APP

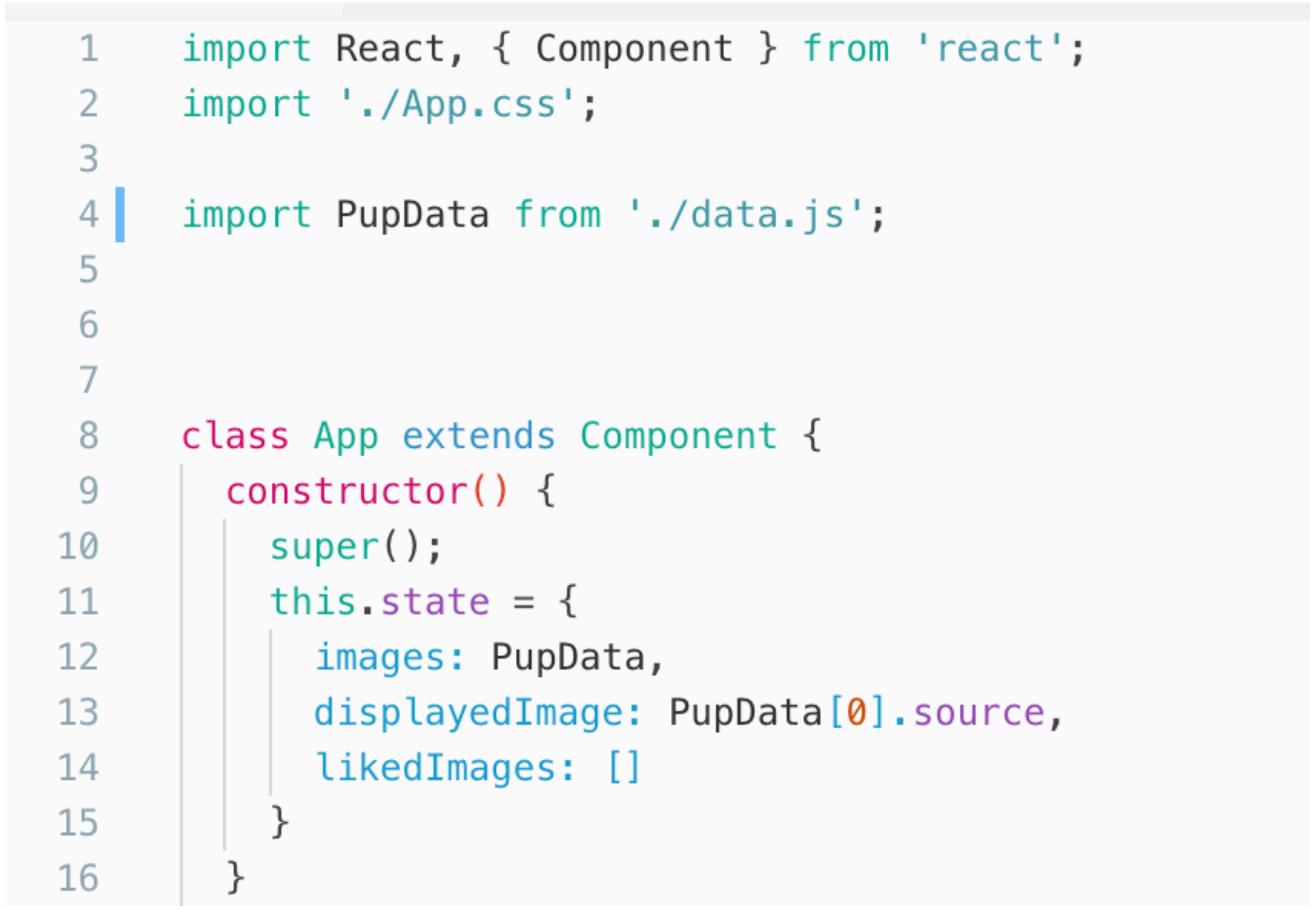
- React released a handy tool to build a React app quickly called Create React App.
- This handy tool lets us build a working scaffold of our React app in a single line.
- Let’s do this together!
- Type this into your console:
  - `npx create-react-app my-app`
- It takes some time, let’s just chill for a bit while that loads.
  - So uh… what’s your favorite animal?

CREATE REACT APP

- Once your app has been created with create-react-app you can start your app by typing:
  - `npm start`
  - In the console.
- We are greeted with a pleasant starter app welcoming us to React!
- We did it! We created our first React App.
- Let’s open up our new app and look at the files that it gave us.

STATE

- One of the core features of React is the idea of state.
- We’ve used the concept of state before in jQuery. Like setting a variable `var  menuOpen = true`, which we then toggle inside of some kind of event.
- This behavior is what makes React work!
- In React there is an object called state. There are different components of the application that watch state.
- ***Any time the state changes, React will automatically re-render anything on the page that involves that state.***



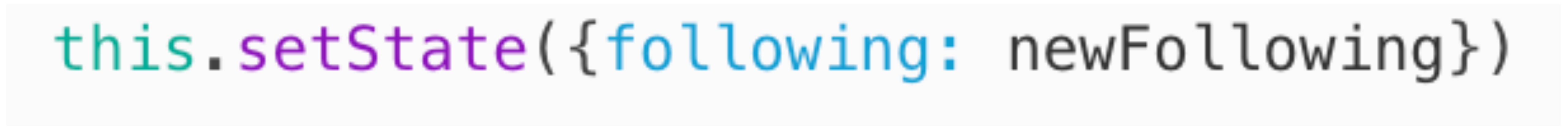
```
import React, { Component } from 'react';
import './App.css';

import PupData from './data.js';



class App extends Component {
  constructor() {
    super();
    this.state = {
      images: PupData,
      displayedImage: PupData[0].source,
      likedImages: []
    }
  }
```

DECLARING STATE AND IMPORTING DATA

- Here is the opening part of our main component in any React app.
- You can see at the top we are importing React and a `Component` class from the React library.
- We then have a new component that inherits from the Component class.
- Normal es6 class stuff - we have a constructor and a super.
- This in the constructor we use `this.state` to set the state for our application.
- Any changes to this state will cause re-rendering for any part of our app that needs it!

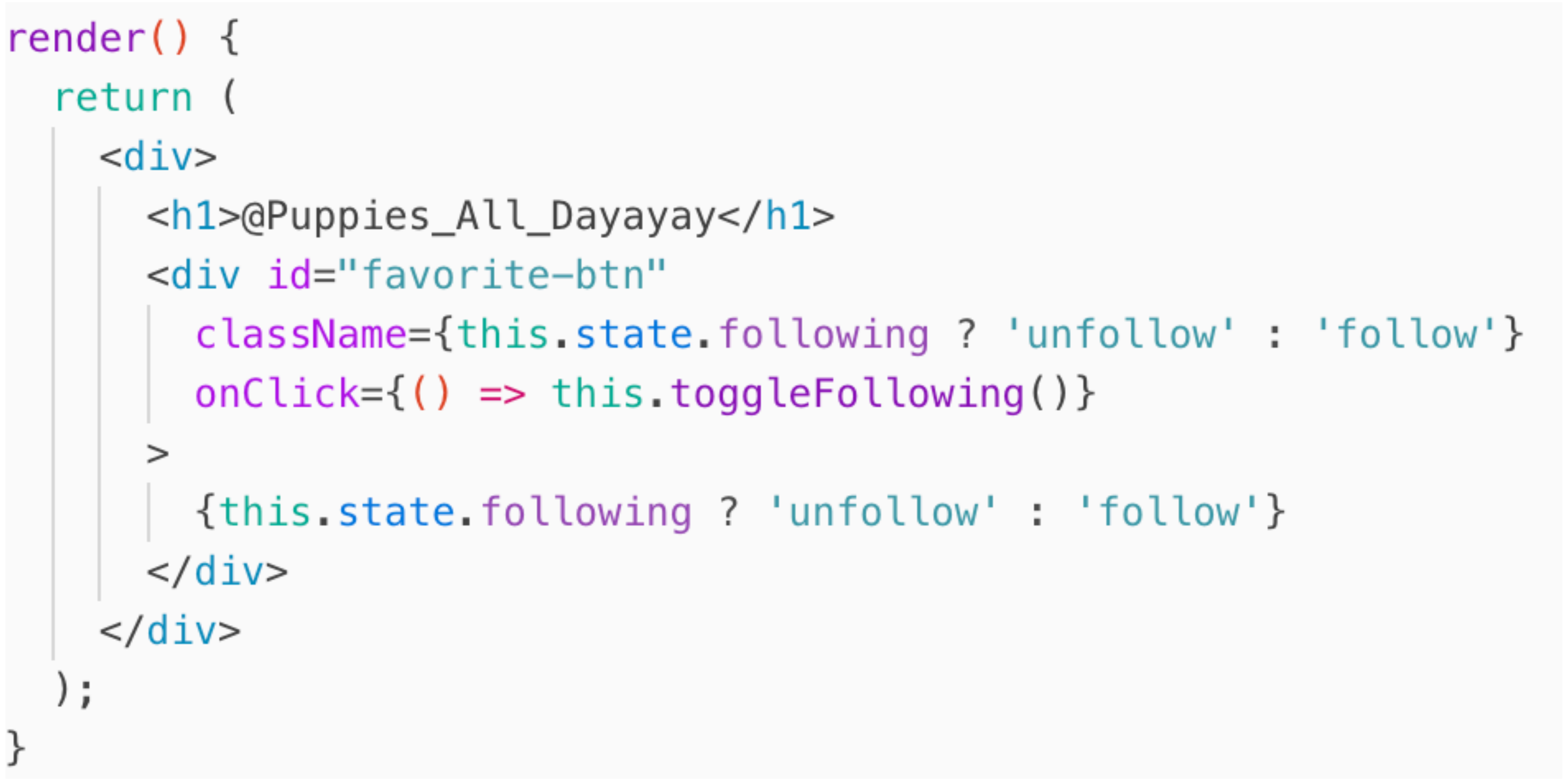
MODIFYING STATE

- You can change state by using the setState method:



```
this.setState({following: newFollowing})
```

- This will target a state in the constructor called “following” and set it to a new value. In this case the value comes from the parameter of a function.

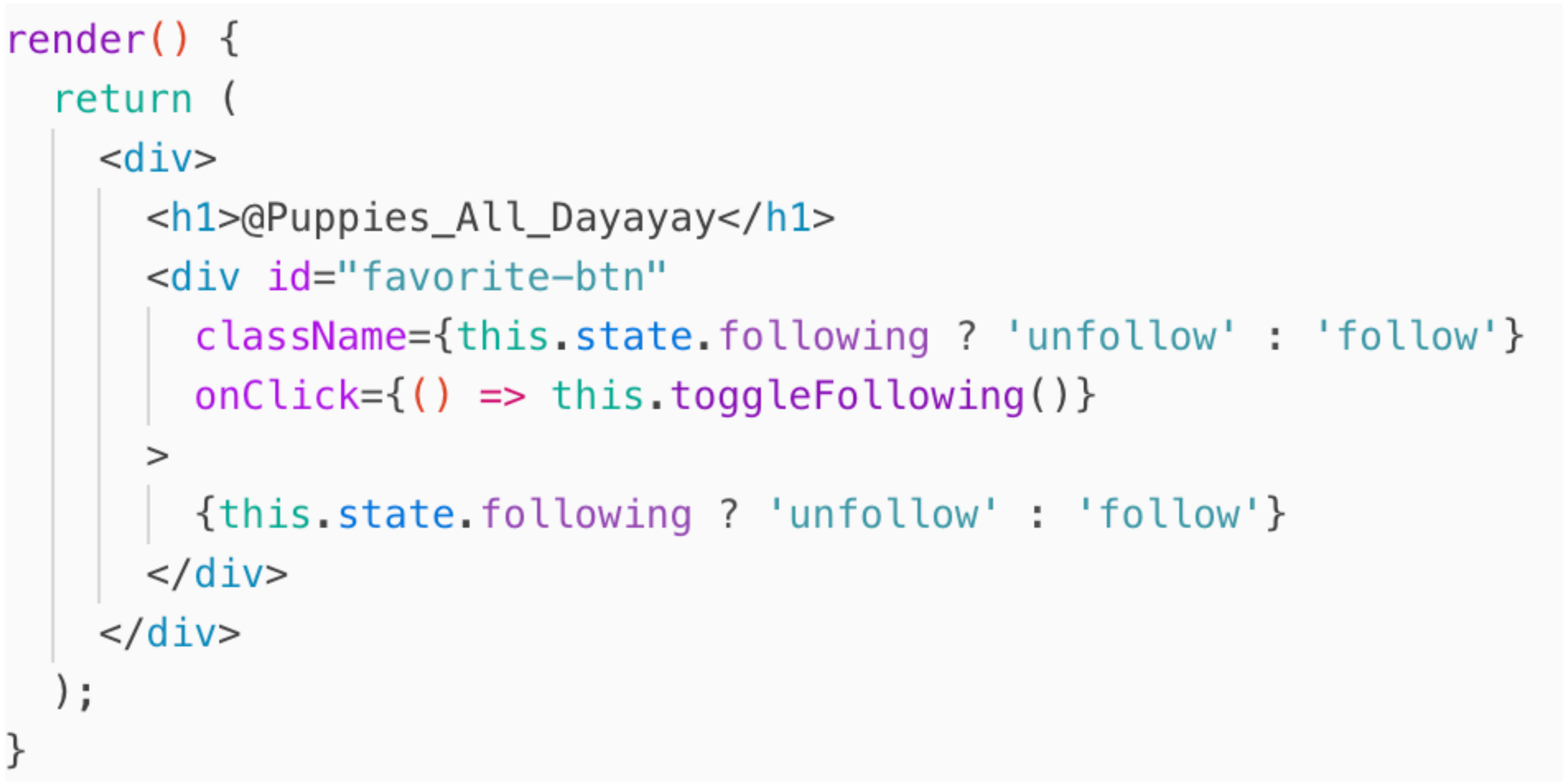


```
render() {
  return (
    <div>
      <h1>@Puppies_All_Dayayay</h1>
      <div id="favorite-btn"
        className={this.state.following ? 'unfollow' : 'follow'}
        onClick={() => this.toggleFollowing()}
      >
        {this.state.following ? 'unfollow' : 'follow'}
      </div>
    </div>
  );
}
```

RENDER FUNCTION

- React components will always have a render function at the bottom.
- This is where… you guessed it, the rendering happens.
- In the render function we can write html and render other React components.

EVENTS

- In React you can add your events inline on the elements that use it:
- 

```
render() {
  return (
    <div>
      <h1>@Puppies_All_Dayayay</h1>
      <div id="favorite-btn"
        className={this.state.following ? 'unfollow' : 'follow'}
        onClick={() => this.toggleFollowing()}
      >
        {this.state.following ? 'unfollow' : 'follow'}
      </div>
    </div>
  );
}
```

CODEALONG

Inputs with event listeners and state

LAB

Click events, buttons and state