GRAND CIRCUS

CODING · BOOTCAMPS

REACT WORKSHOP

INTRO

DAVID WOLVERTON

david@grandcircus.co

@dwolverton (LinkedIn & Twitter)

CHANDLER KEYES

My YouTube Channel: CodingWithChandler

INTRODUCTIONS

Introduce yourself to your neighbor. What insterested you in this class?

OBJECTIVE

By the end of this course, you should have the foundations to start building your own web apps and dynamic web UIs with React. There are plenty of additional techniques and details as you get deeper into React in the future, but here you will get the necessary framework of understanding to start the adventure.

HOW WE'LL LEARN

- Mini Lectures and Examples
- Code-Along
- Exercises
- Ask Questions
- Optional readings

HOW WE'LL SURVIVE

We will have short breaks throughout and an hour break for lunch.

WHAT WE'LL BUILD

Some of these Components

TOPICS

- React Components & JSX
- Data in State & Props
- Handling Events
- Displaying images & lists
- Using Forms
- Performance & Immutability

BONUS TOPICS

As time allows...

- Component Lifecycle
- AJAX
- Functional Components
- React Hooks

REACT WORKSHOP

TECHNOLOGY SURVEY

Click Here

SETUP

STEP 1: GET YOUR COMPUTER READY

You only have to do this once, ever.

- Install Node.js
- Pick an IDE. I'm using Visual Studio Code

PRO-TIP

If you're using Atom, install the `language-babel` plugin. This will greatly improve your JSX experience.

STEP 2: CREATE A NEW REACT PROJECT.

- `cd` to the folder where the project will live (parent folder)
- Run `npx create-react-app workshop-demo`. This will create a new folder named `workshop-demo` with your project in it.
- `cd` into this folder. Also open up the folder in your IDE.

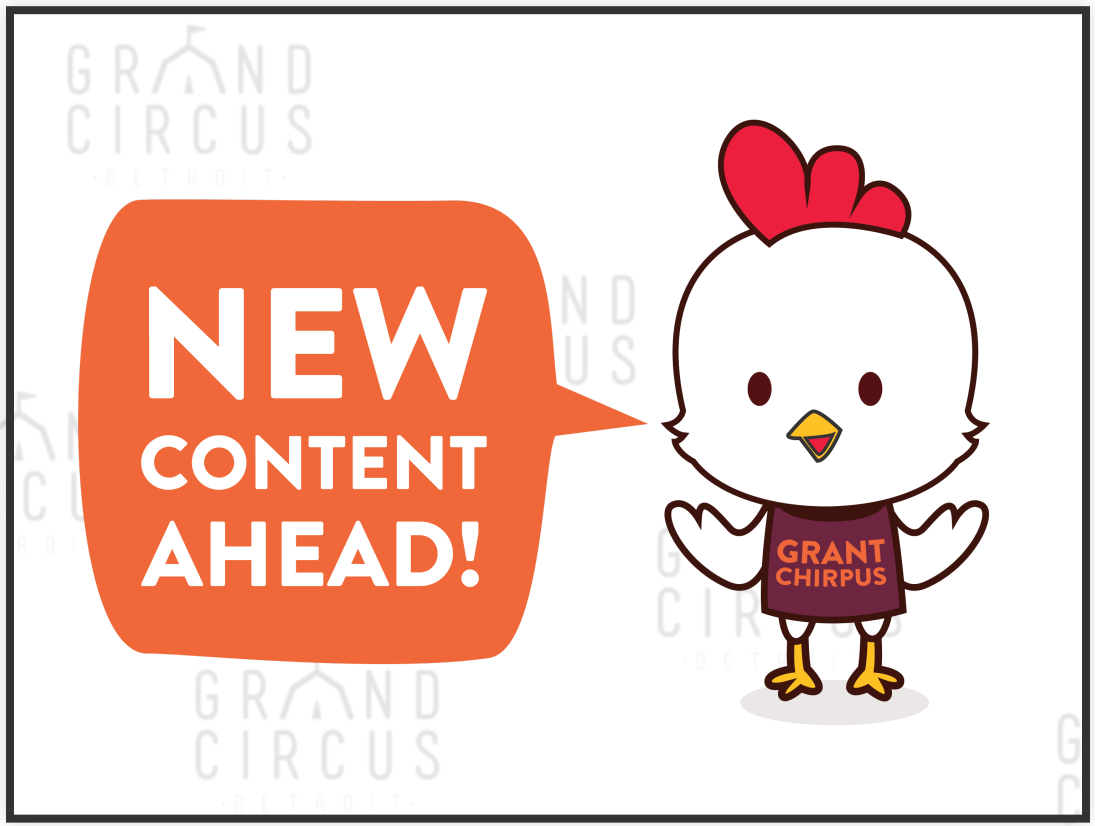
STEP 3: START RUNNING IT

Run `npm start`.

This will start the whole React system. It will open a browser tab for http://localhost:3000. They've already started you off with a simple React app.

STEP 4: MAKE IT OUR OWN.

1. Clear out `App.css`.
2. In `App.js` clear out the JSX inside the `<div>` so that only `<div classNameAppAppApp


NEW
CONTENT
AHEAD!
GRANT
CHIRPUS

WHAT IS REACT?

It is a *JavaScript library* with a tool set that will help us make complicated web UIs more easily. For example...

This or This

LET'S JUMP RIGHT INTO IT!

CODE ALONG

If you get behind, don't worry. There will be time at the end to catch up.

KEY TERM: JSX

JSX looks (mostly) like HTML. We use it to tell React what HTML to render to the page.

RENDER

Components have a `render` method that returns JSX. The JSX here tells React what HTML to insert for each Counter.

CLASSNAME

One little trick about JSX. It usually looks just like HTML, but there are a few exceptions. Instead of `class` we have to use `className`. (Because `class` is a reserved word in JavaScript.)

Similarly `for="id"` becomes `htmlFor="id"`.

KEY TERM: COMPONENT

Components are the pieces and parts that make up a webpage. One Counter correlates to one div on the screen. We can use it to stamp out as many Counters as we need.

In React, we always want to break up our code into a bunch of small components that are each responsible for their own little part.

EXAMPLE COMPONENT

```
// Bring in stuff from the React code library
import React, { Component } from 'react';

// Define the component using ES6 class syntax. This component is named Greeting.
class Greeting extends Component {
  // Every React Component has a render method.
  // The render method returns the JSX for this component.
  render() {
    return (
      // This JSX tells React what HTML to render
      <div className="Greeting">
        <h2>Hello Class!</h2>
      </div>
    )
  }
}

// Always export the component so that other components can use it.
export default Greeting;
```

CSS

Components often need CSS. There are actually a lot of different ways to specify CSS for a React Component.

In this workshop we create a CSS file for the component and link it with an `import`.

CAUTION: These CSS files still apply to the whole page. Use selectors specific to your component.

CSS

There is also a global CSS stylesheet for styles that are not specific to a component.

`index.css`

QUESTIONS?

EXERCISE 1

NON-FUNCTIONING TIMER COMPONENT

LET'S TALK ABOUT IT

WE'RE USING JSX AND OTHER ADVANCED JAVASCRIPT

The first thing we'll notice is that there's JSX embedded in our JavaScript files. We're also using advanced JS features, such as import/export. This is possible because all our JavaScript for React is *transpiled* with *Babel* to plain old JavaScript that can run anywhere, even in old browsers.

KEY TERM: TRANSPILE

Transpile means we run a program that takes leading edge JavaScript syntax that we write and turns it into browser-safe old-fashioned JavaScript that does the exact same thing.

The pro is that we can use awesome, convenient and powerful new features and JSX. The con is we can't just run directly in the browser. We have to run the transpiler first.

TRANSPILE

For example...

```
// this
const add = (x, y) => x + y;

// transpiles to something like this
function add(x, y) {
  return x + y;
}
```

Want to learn more about ES6 arrow functions? Read this.

KEY TERM: BABEL

Babel is the program that we're going to use to do the transpiling. It's so easy to use, we don't really have to think about it. And with the setup we'll use, it's already installed and configured.

KEY TERM: WEBPACK

Webpack is the tool that brings all our many source files together. It takes all our JavaScript, runs Babel on it, adds all our imported libraries, and wraps it into one big JS file, which it automatically links from index.html.

It does the same thing with all imported CSS files. It even adds various browser prefixes automatically.

WEBPACK

Just like Babel, we don't really have to think about Webpack. It's already installed and configured for us.

REACTDOM.RENDER

Look at index.js.

```
ReactDOM.render(<App />, document.getElementById('root'));
```

This line takes the App component from App.js and actually inserts it into the HTML page.

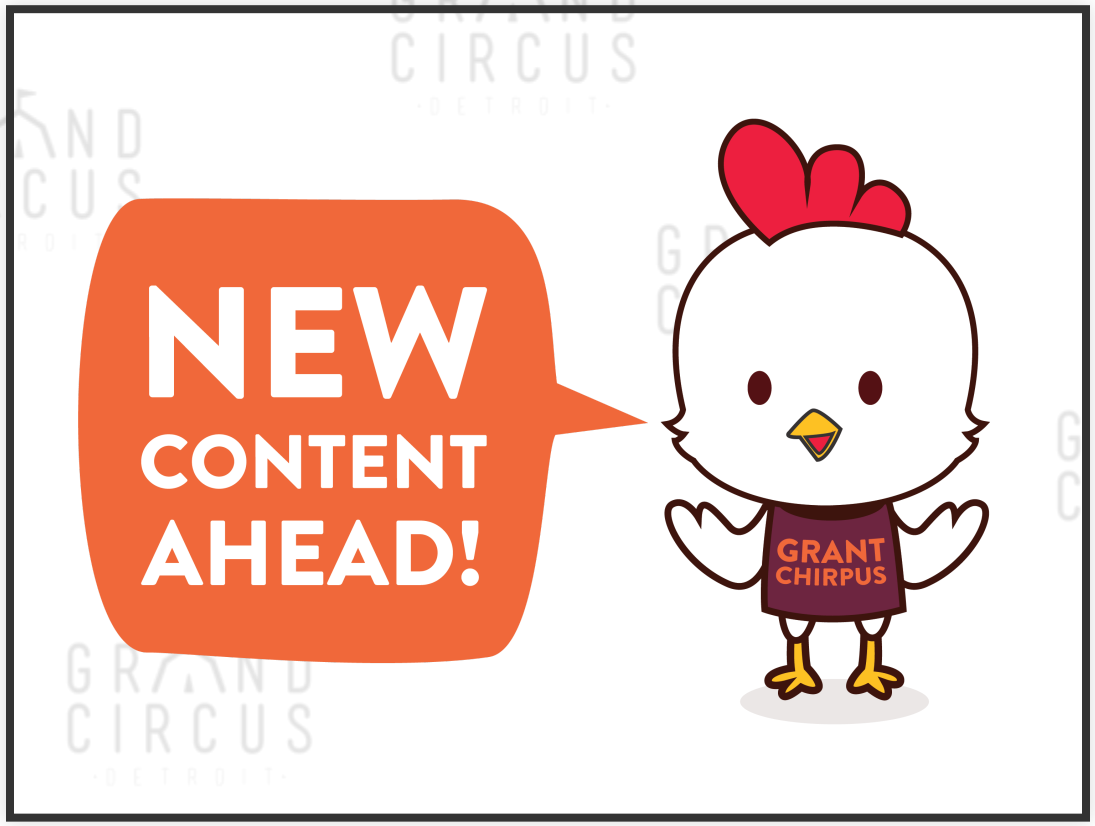
APP

It will be our convention in this class to always have an `App` component that serves as the root of our entire application. All other components are included within it. Then we will add that App component to the page here in index.js.

THAT'S A LOT OF INDEXES

- index.html
- index.js
- index.css

It's a modern JavaScript convention to call your main JavaScript file index.js just like your main HTML is index.html. We'll also extend the convention to our CSS just for kicks!


NEW
CONTENT
AHEAD!
GRANT
CHIRPUS

STATE

CODE ALONG

If you get behind, don't worry. There will be time at the end to catch up.

KEY TERM: STATE

Each React component can have state.

State is internal data which affects the rendering of a component.

definition from https://medium.com/react-tutorials/react-state-14a6d4f736f5

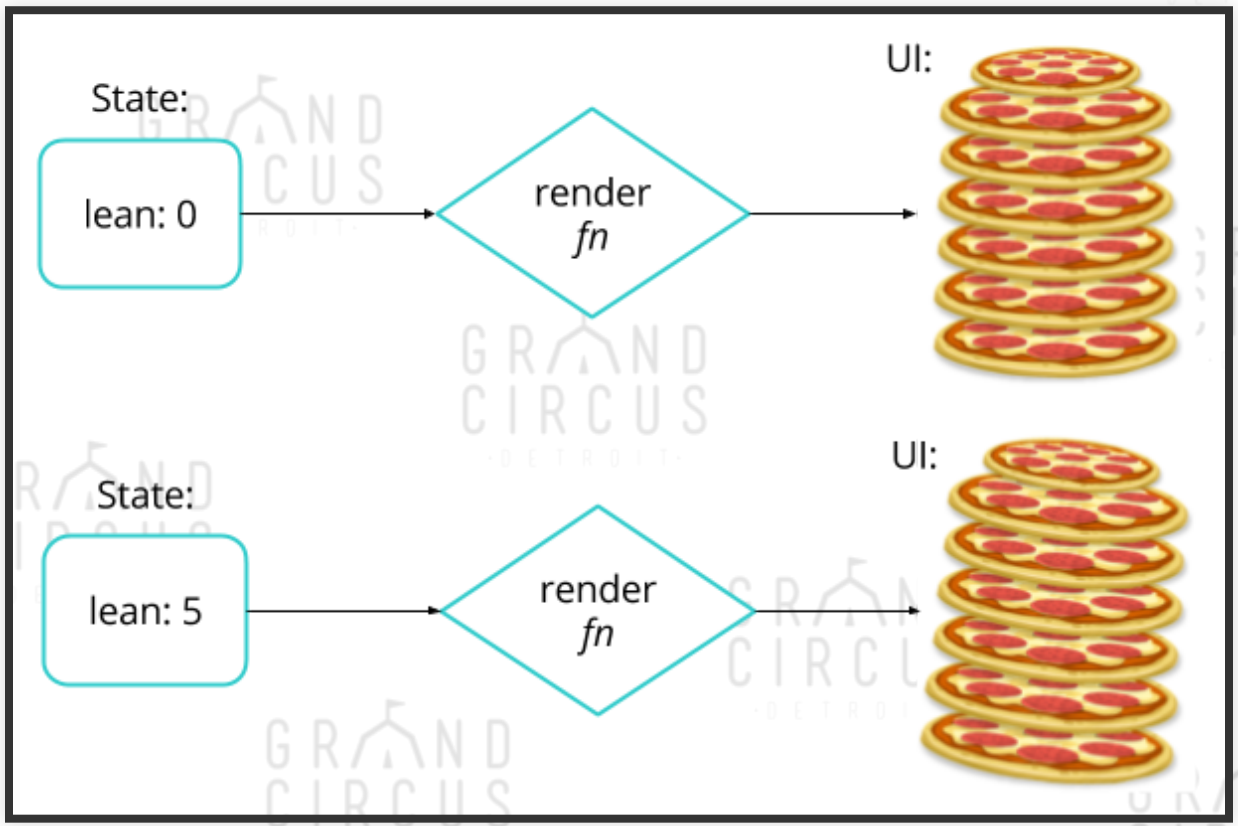
KEY TERM: STATE

State is an object that allows the component to own and control values for itself. In this case, each Counter controls its count.

KEY PHILOSOPHY OF REACT:

UI IS A FUNCTION OF STATE

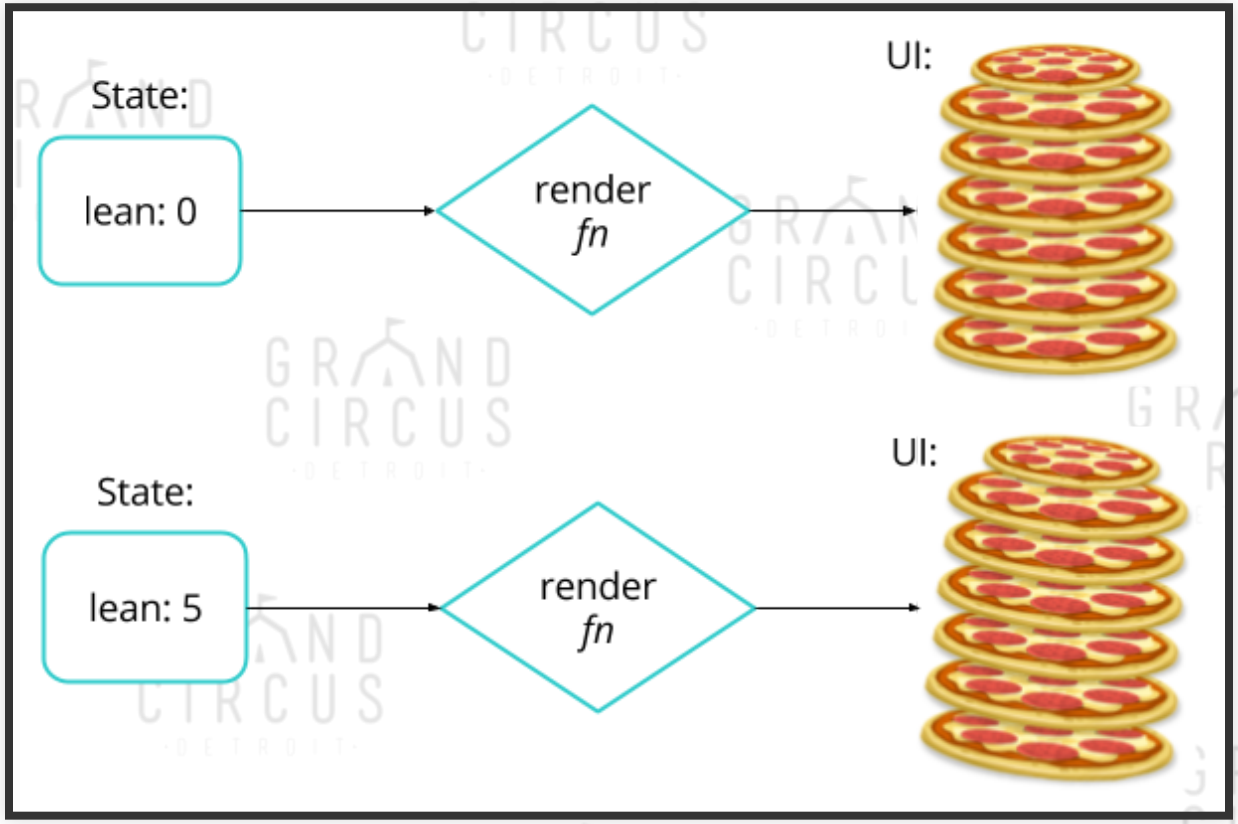
UI IS A FUNCTION OF STATE



UI IS A FUNCTION OF STATE

Unlike jQuery or basic JavaScript, we do not manually add and remove elements, modify attributes or set content. *We only modify the data in the state.* And in the render method we tell React what we want the HTML to look like given any state. React makes all the connections to keep the HTML up to date with each change.

UI IS A FUNCTION OF STATE



CHANGES TO STATE

So how does it work?

Anytime state is changed, react automatically re-runs the render method and keeps the HTML on your page fully up-to-date. Here's some of the real power of React.

HOW TO SET INITIAL STATE

Set a state property on the class.

```
state = {
    count: 0
}
```

HOW TO MODIFY STATE

```
this.setState({
    prop: value,
    anotherProp: anotherValue,
    ...
});
```

Set one or more properties at once. This changes specified properties only. You don't have to include every property of the state.

UPDATING PREVIOUS STATE

If we need to know the old state to figure out the new state, we need to use the *callback function form* of `setState`. This is because state updates are asynchronous, and this guarantees we're getting the most up-to-date previous state.

```
this.setState(prevState => {
    // update the count in state
    return { count: prevState.count + 1 };
});
```

COMPARE...

Simple form takes an object.

```
this.setState({
  count: 0
});
```

Callback form takes a function.

```
this.setState(prevState => {
    return { count: prevState.count + 1 };
});
```

STATE IN JSX

Use `{ }` in JSX to insert JavaScript expressions. These can be used in content and as attribute values.

```
<p className="info">
  <a href={link}>Welcome to {this.state.parkName}!</a>

  You're the visitor number {this.state.index + 1}.
</p>
```

DOM EVENTS

To handle DOM events, we usually add a method to our component and call it from an `on__` attribute in the JSX.

```
<button onClick={this.handleClear}>

handleClear = () => {
  ...
}
```

DOM EVENTS

In order to pass arguments to the method, use an arrow function.

```
<button onClick={() => this.handleCount(+1)}>

handleCount = (change) => {
  ...
}
```

A COMMON MISTAKE

Like this...

```
<button onClick={() => this.handleCount(+1)}>
```

Not this...

```
<button onClick={this.handleCount(+1)}>
```

BINDING `THIS`

```
handleClear = () => {
  ...
}
```

For event handlers, it's common to use this syntax for setting an arrow function as a property. This ensures that `this` is bound correctly when the method is called.

BINDING THIS

Like this

```
handleClear = () => {
  ...
}
```

Not this

```
handleClear() {
  ...
}
```

BINDING `THIS`

Another solution is to redefine each method with a `.bind(this)` in the constructor.

```
constructor(props) {
  super(props);
  this.handleClear = this.handleClear.bind(this);
}
```

CODE ALONG

`Timer` Component

EXERCISE 2

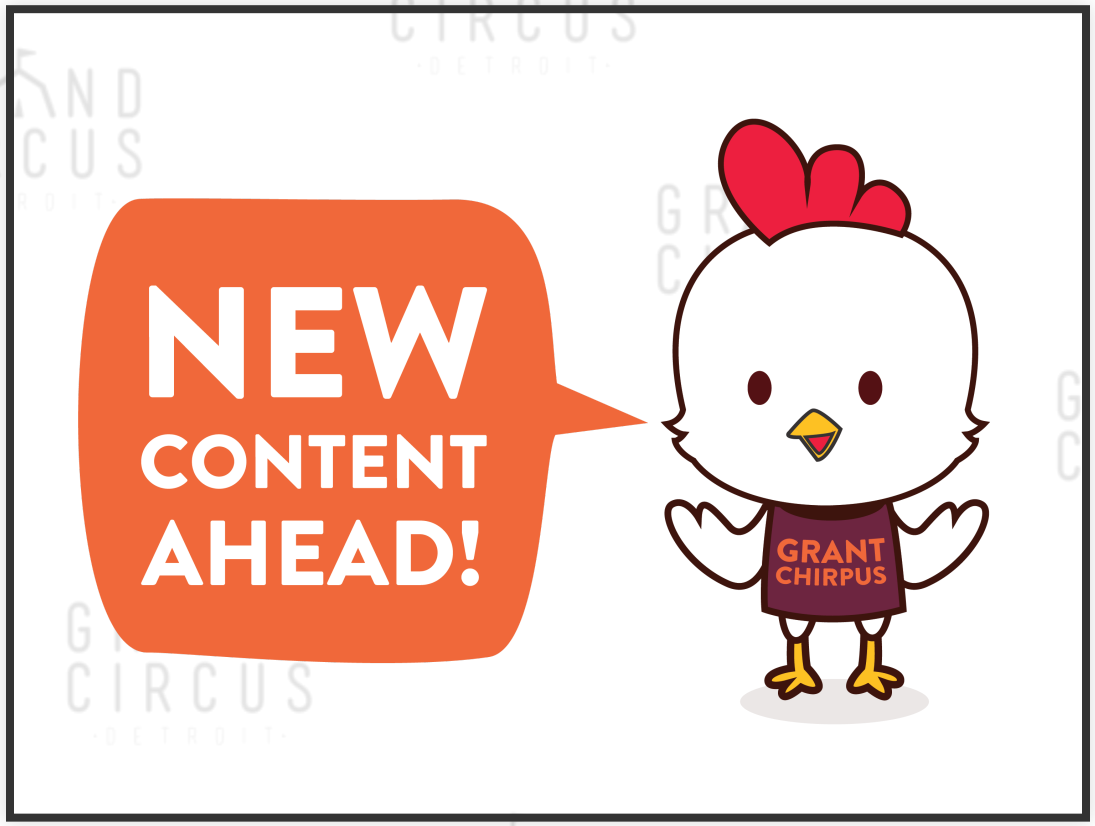
WORKING TIMER & COUNTER COLOR

EXERCISE 3

VOTES

CODE ALONG

Hide and show `Counter` and `Timer` buttons.


NEW
CONTENT
AHEAD!
GRANT
CHIRPUS

PROPS

CODE ALONG

If you get behind, don't worry. There will be time at the end to catch up.

KEY TERM: PROPS

We use props (short for "Properties") to pass data from one component to another. They are sent in attributes and received via `this.props.__`.

```
// Send
<NameTag name="Tigger" />

// Receive
<div>{this.props.name}</div>
```

SENDING PROPS

Props can also be sent using string literals or { } expressions.

```
<Contact name="Grant" email="hello@grandcircus.co" />
<Grade label="Geography Quiz" score={3} />
<Grade label={myQuiz} score={10 - 7} passing={false} />
<Grade label="Alaska Worksheet" score={myScore} passing={isPassingScore(myScore)} />
<Grade label={this.state.label} score={this.state.score1} />
```

SENDING PROPS

Also, a prop without a value is true.

```
<Star name="Sun" hasPlanets={true} />
```

is the same as

```
<Star name="Sun" hasPlanets />
```

STATE VS. PROPS

State	Props
Owned and controlled by the component. Not set from outside. Can change over time.	Controlled by a parent component. Not changed by this component.

EXERCISE 4

WEATHER

USING STATE & PROPS TOGETHER

EXERCISE 5

NEG-POS-NUMBER COMPONENT

IMAGES

USING IMAGES IN JS

Two options

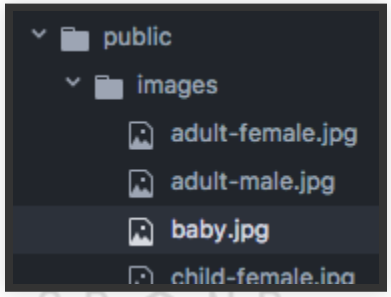
1. Include a string URL to an image in the public folder.
2. `import` the image directly. (*Preferred*)

USING IMAGES: PUBLIC FOLDER

Option 1: Include a string URL to an image in the public folder.

```
<img src="%PUBLIC_URL%/images/baby.jpg" alt="Baby" />
<img src={process.env.PUBLIC_URL + "/images/baby.jpg"} alt="Baby" />
```

The image should be somewhere in the `public` folder.

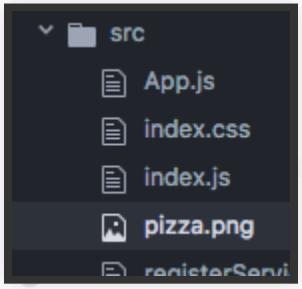


USING IMAGES: SRC FOLDER

Option 2: Import the image directly.

```
import pizzaImg from '../pizza.png';
<img src={pizzaImg} alt="Pizza" />
```

The image should be somewhere in the `src` folder.



ADVANTAGES OF IMPORTING IMAGES (OPTION 2)

- Scripts and stylesheets get minified and bundled together to avoid extra network requests.
- Missing files cause compilation errors instead of 404 errors for your users.
- Result filenames include content hashes so you don't need to worry about browsers caching their old versions.

Reference: create-react-app.dev

EXERCISE 6

ADD AN IMAGE

RECAP

WHAT HAVE WE LEARNED ABOUT REACT SO FAR?

- Components
- JSX
- render() method
- ES6, Babel transpiler (One does not simply open React code in a browser.)
- ES6 Modules: import / export
- Webpack
- create-react-app

WHAT HAVE WE LEARNED ABOUT REACT SO FAR?

It is a *JavaScript library* with a tool set that will help us make complicated web UIs more easily.

SUGGESTED READING

https://reactjs.org/docs/hello-world.html

This slide set covered sections "Hello World" through "Conditional Rendering".

Next time slide set covers "Lists and Keys" through "Lifting State Up".