REACT WORKSHOP

INTRO

DAVID WOLVERTON

david@grandcircus.co

@dwolverton (LinkedIn & Twitter)

INTRODUCTIONS

Introduce yourself to your neighbor. What interested 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

- Code-Alongs and Demos
- Mini Lectures
- Exercises
- Ask Questions

HOW WE'LL SURVIVE

We will have a short break around 10:30.

WHAT WE'LL BUILD

See Exercises.

TOPICS

DAY 1

- React Components & JSX
- Functional Components and Hooks
- Data in State
- Handling Events

TOPICS

DAY 2

- Props
- Displaying lists
- Calling APIs

ZOOM

- Question: Just unmute and ask.
- Breakouts during Exercises
- "Ask for Help" button

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

SETUP

TWO SETUP OPTIONS

1. Quick start: Sign into codesandbox.io and start a React sandbox.
2. Install and develop a project on your computer. (See next slides.)

OPTION 2...

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

OPTION 2...

STEP 2: CREATE A NEW REACT PROJECT.

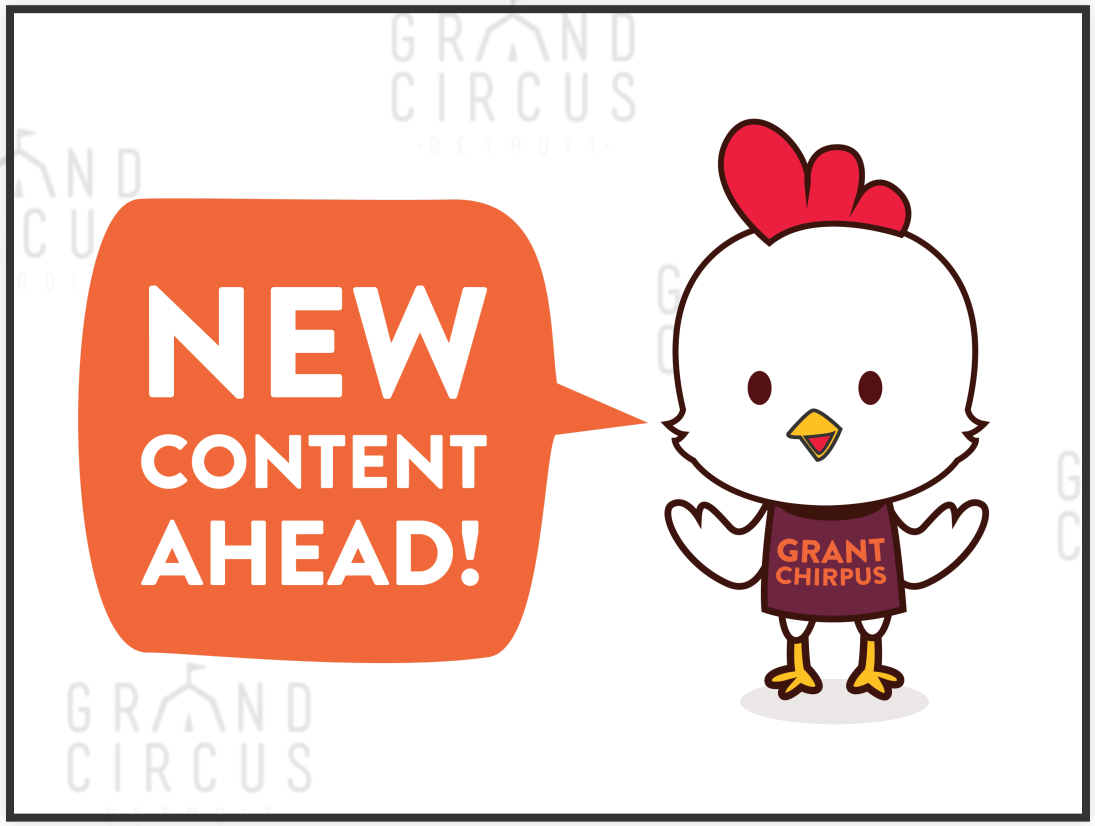
- `cd` to the folder where the project will live (parent folder)
- Run `npx create-react-app react-workshop`. This will create a new folder named `react-workshop` with your project in it.
- `cd` into this folder. Also open up the folder in your IDE.

OPTION 2...

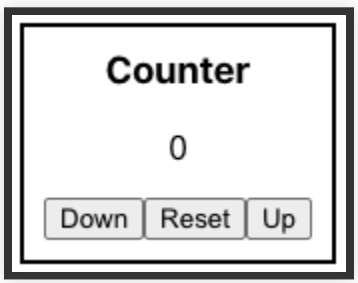
STEP 3: START RUNNING IT

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.


NEW
CONTENT
AHEAD!
GRANT
CHIRPUS

CODE ALONG



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

CODE ALONG

1. Clear out `App.css`.
2. In `App.js` clear out the JSX inside the `<div>` so that only `<div className="App"></div>` is left.
3. You should now see a blank page in the web browser.
4. We'll build the rest together.

KEY TERM: JSX

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

FUNCTIONAL COMPONENTS

Components are defined as functions that return 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

```jsx
// Bring in stuff from the React code library. You always* need this line.
import React from 'react';

// Difine the component using a function. This component is named Greeting.
function Greeting() {
  // The function returns the JSX for this component.
  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.

`styles.css` or `index.css`

QUESTIONS?

EXERCISE 1

NON-FUNCTIONING SWITCH 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 old-fashioned browser-safe 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

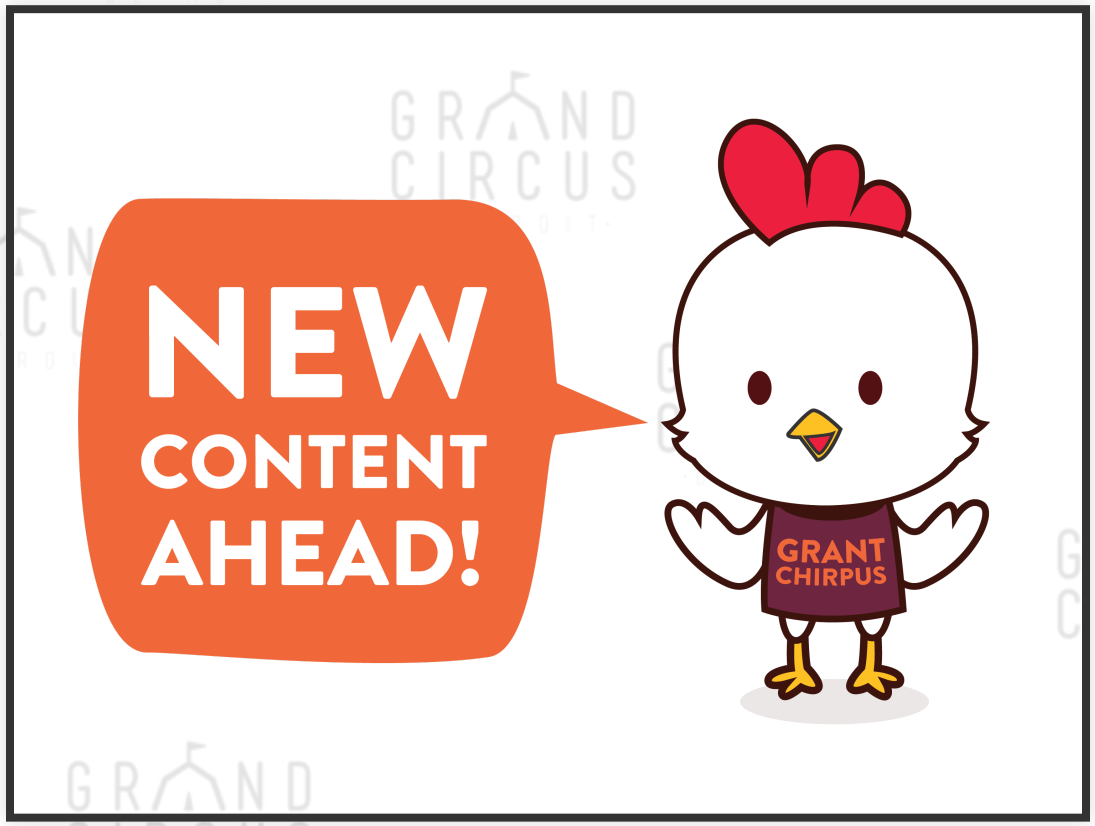
Look at index.js. This is where everything starts.

```
ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  rootElement
);
```

These lines take the App component from App.js and actually insert 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.


NEW
CONTENT
AHEAD!
GRANT
CHIRPUS

STATE

CODE ALONG

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

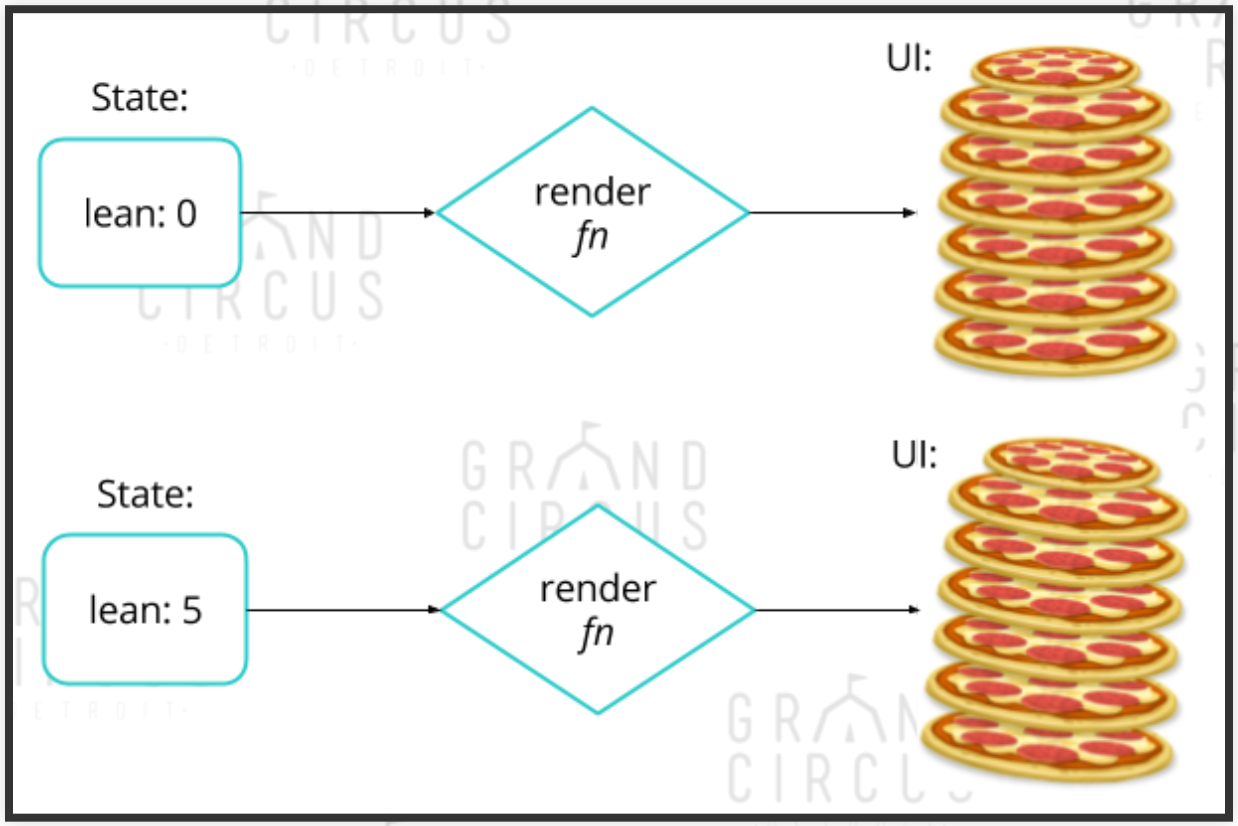
KEY TERM: STATE

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

KEY PHILOSOPHY OF REACT:

UI IS A FUNCTION OF STATE

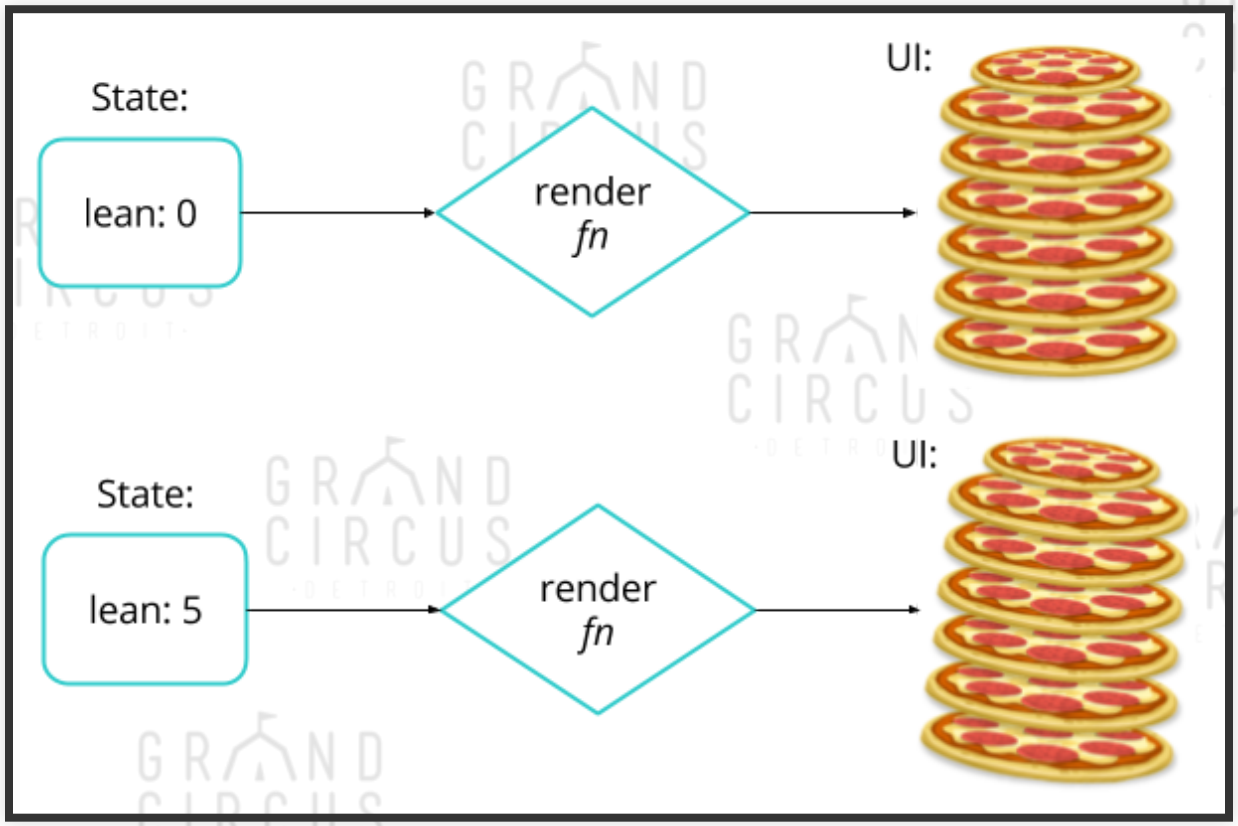
UI IS A FUNCTION OF STATE



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 component function 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 component function and keeps the HTML on your page fully up-to-date. This is some of the real power of React.

HOW TO SET INITIAL STATE

The `0` passed to useState is the initial value when the component first loads.

```
const [ count, setCount ] = useState(0);
```

HOW TO MODIFY STATE

```
setCount(8)
```

Call the set*** function you got from useState.

UPDATING PREVIOUS STATE

```
setCount(prevCount => prevCount + 1)
```

Use a callback function with the previous value to calculate the new value using the set*** function.

```
count++ // <-- INCORRECT!
```

Calling the set*** function is the only way to modify state.

STATE IN JSX

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

```
const [link, setLink] = useState('https://example.com');
const [parkName, setParkName] = useState('Jurassic Park');
const [index, setIndex] = useState(0);
```

```
return (
  <p className="info">
    <a href={link}>Welcome to {parkName}!</a>

    You are the visitor number {index + 1}.
  </p>
);
```

MULTIPLE STATE VARIABLES

You can use as many different state variables as you want. Just repeat the same formula.

```
const [height, setHeight] = useState(400);
const [width, setWidth] = useState(300);
const [color, setColor] = useState('pink');
```

HOOKS

You may see the term "hooks" related to React. `useState` is an example of a hook. Hooks were introduced in React 16.8 in February 2019, and provide a much streamlined development experience.

There are many hooks built into React. They all start with "use". We will see one other hook in this workshop, `useEffect`.

DOM EVENTS

To handle DOM events, we create a function and use it in an `on__` attribute in the JSX.

```
function clear() {
  ...
}
```

```
<button onClick={clear}>
```

DOM EVENTS

In order to pass arguments to the function, use an inline function.

```
function handleCount(change) {
  ...
}
```

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

A COMMON MISTAKE

Correct...

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

Incorrect...

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

EXERCISE 2

WORKING SWITCH

Finish Task 1. Then try Task 2.

REVIEW EXERCISE 2

How to get that background color working?

CODE ALONG

Hide and show `Counter` reset button.

We'll use the JSX Cheatsheet.

EXERCISE 3

OPTIMIZED SWITCH

EXERCISE 4

ICE CREAM VOTES

This is totally optional homework.