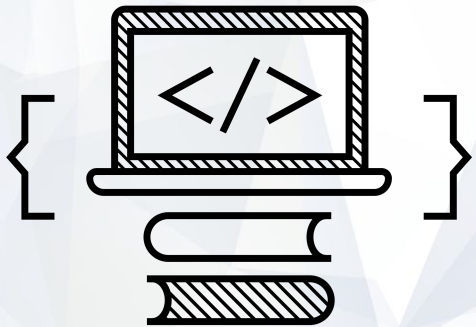
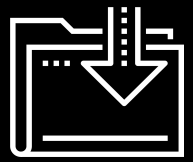


Props, Lists, and Stateful Components

Skills Bootcamp in Front-End Web Development

Lesson 13.2




WELCOME

Learning Objectives

By the end of class, you will:

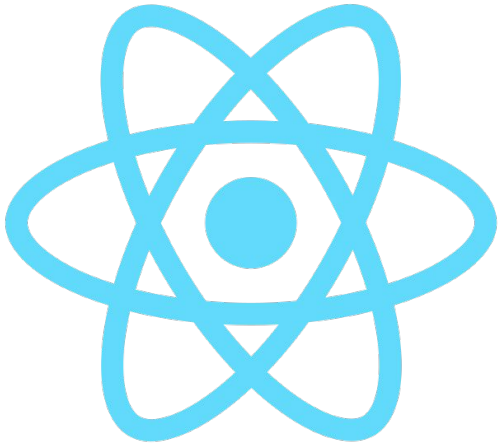
Deepen your understanding of passing props between React components.

Gain a firm understanding of the concept of child–parent relationships in React.

Programmatically render components from an array of data.

Understand the concepts of class components and component state.

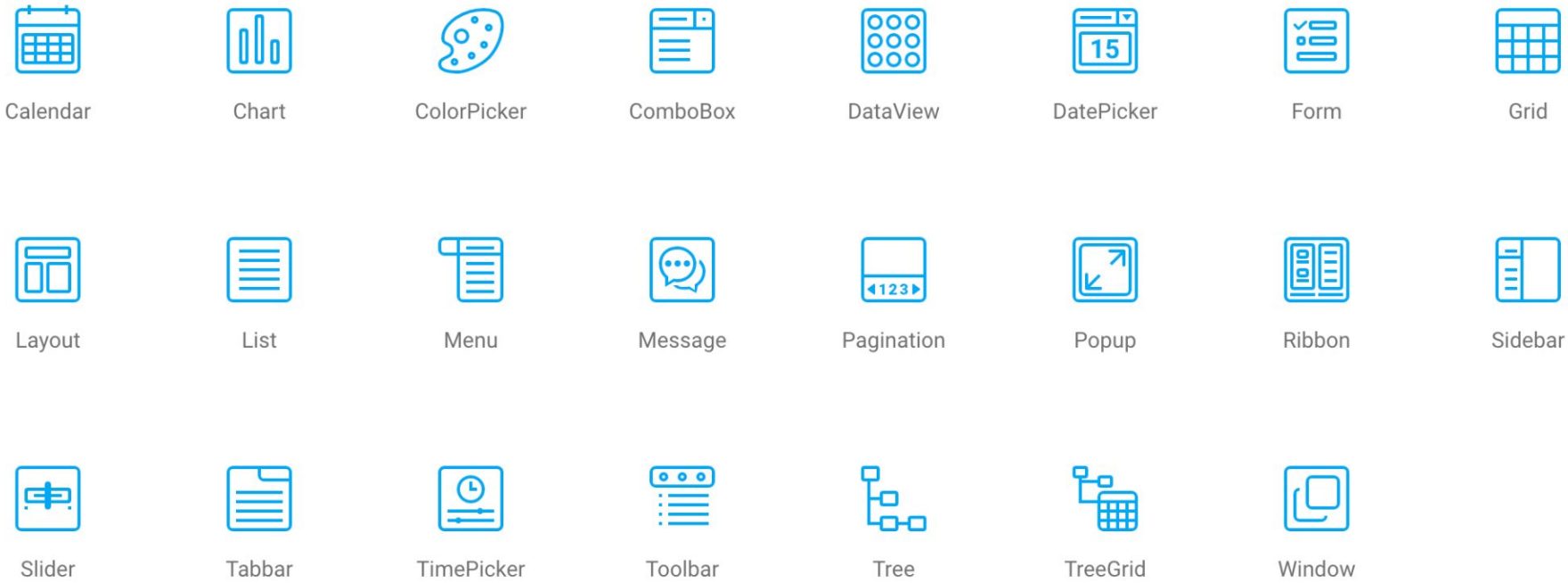
We can conceptualize React components as JavaScript functions.



Component **Function**

Props

It’s a component’s job to describe and *return* some part of our application’s UI.



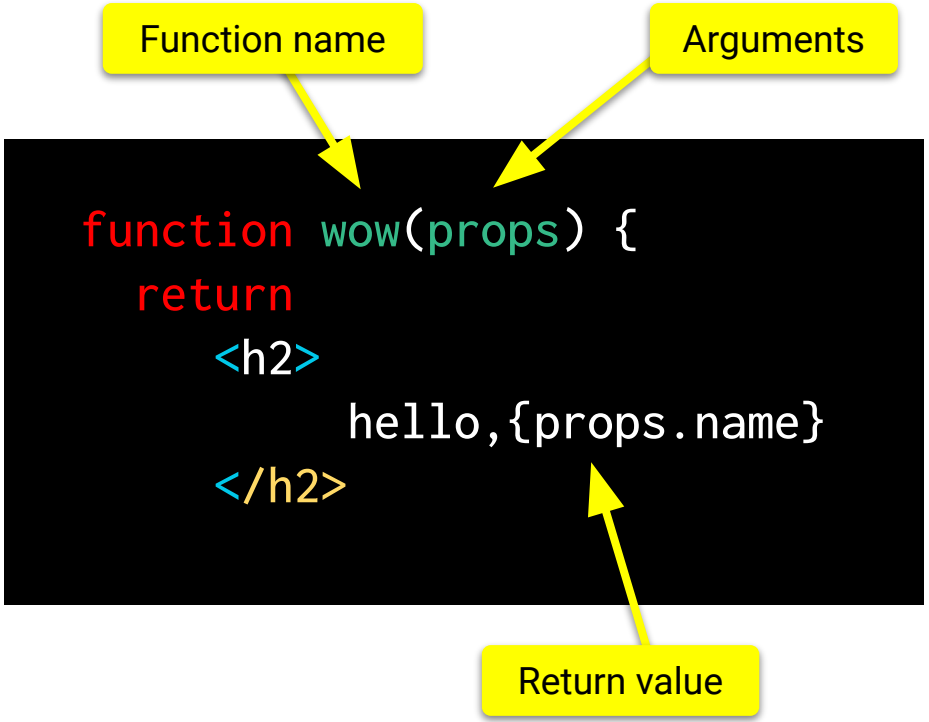
If a component is a function that *returns* some data, what else might a component be able to do?

Props

Since it’s a function, a component can also receive arguments.

This allows us to write components that behave differently based on the arguments that they receive.

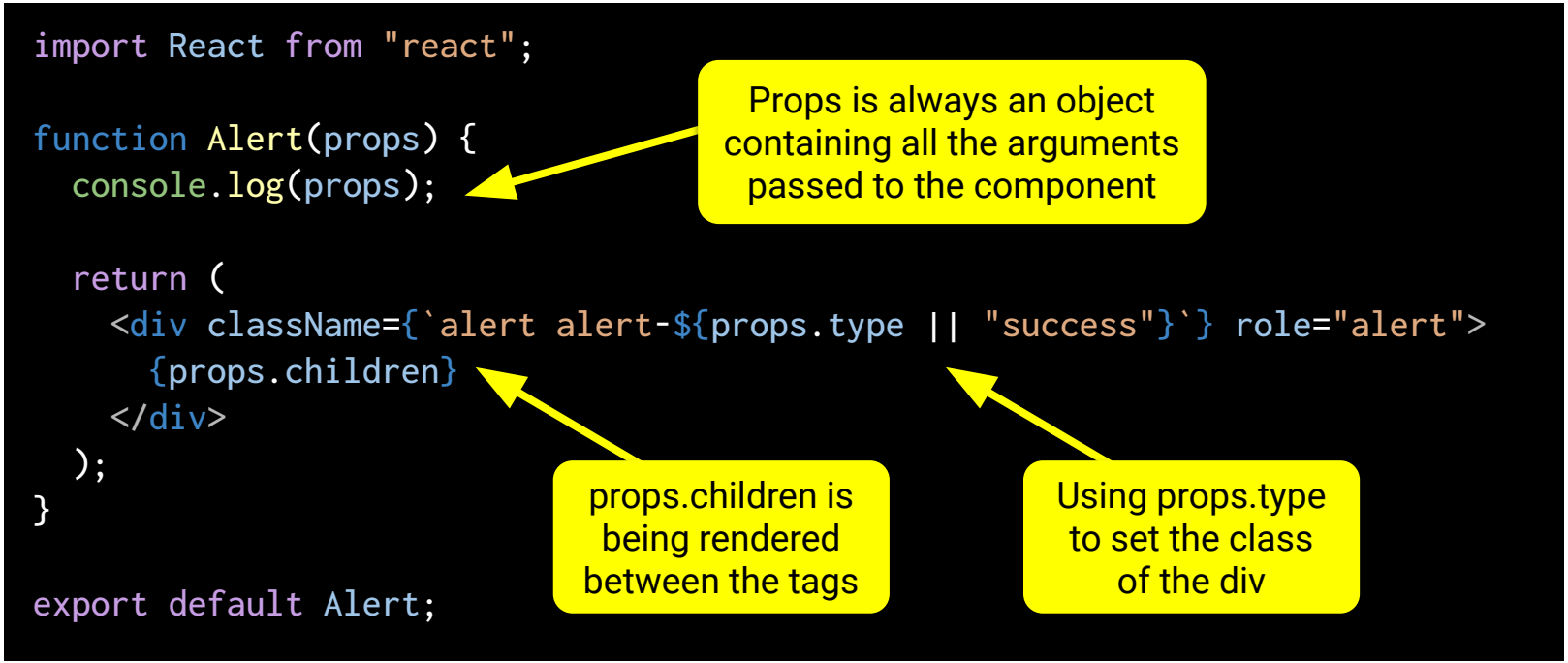
We call the arguments that we pass into React components props.



Props are like function arguments that you can pass to components for them to use.

Props

Every component has access to a `props` argument. Props are always objects containing all of the values passed to the component.



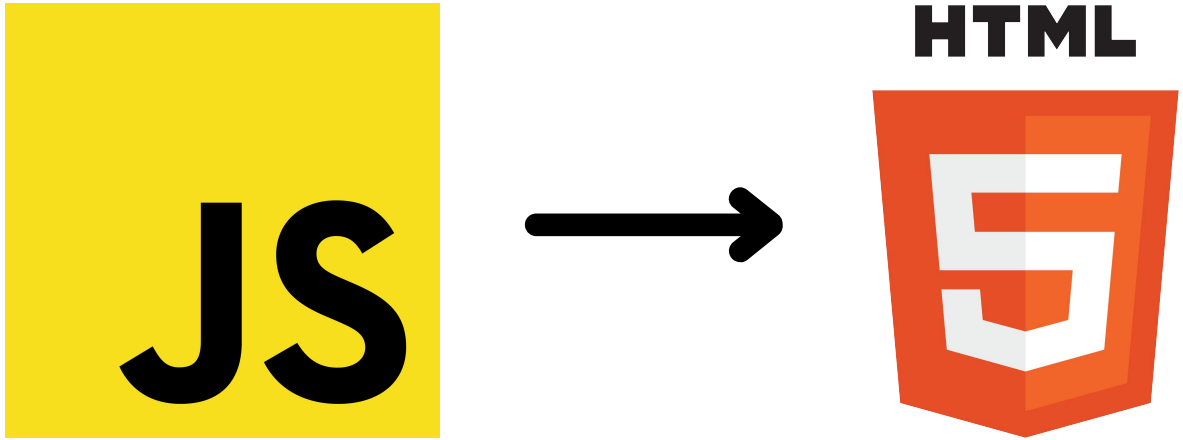
Props

These props are being passed into the Alert component.

We have two ways of passing props to components:

```
function App() {
  return <Alert type="danger">Invalid use id or password</Alert>;
}
```

Having this familiar syntax for passing props to our components is another way for JSX to be similar to HTML.



Props

Props allow us to customize our components so that we can reuse them in different situations.

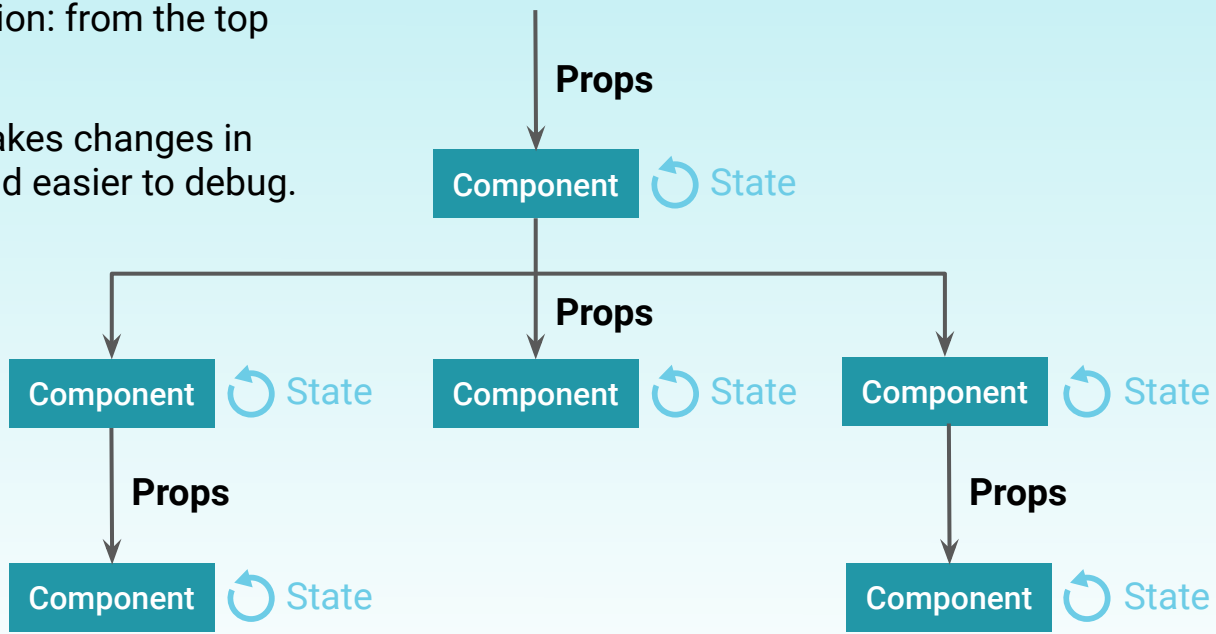
For example, we might use this `Alert` component on a sign-in page and render a different alert depending on whether or not a user has successfully logged in to their account.

Props

Props are the primary means by which we pass data around our React apps.

React utilizes a unidirectional data flow, meaning that data only flow in one direction: from the top down, parent to child.

This unidirectional data flow makes changes in React apps more predictable and easier to debug.



Source: Tarun Sharma

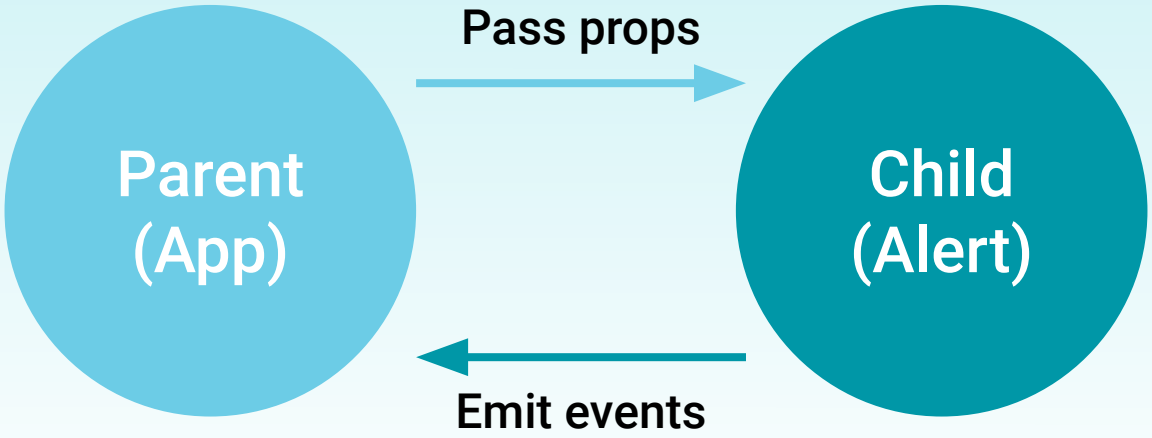
If a prop inside of our component isn’t what we expect it to be, where could we look to find out why?

Props

We could look at the component's parent.

In this example, App and Alert have a parent–child relationship.

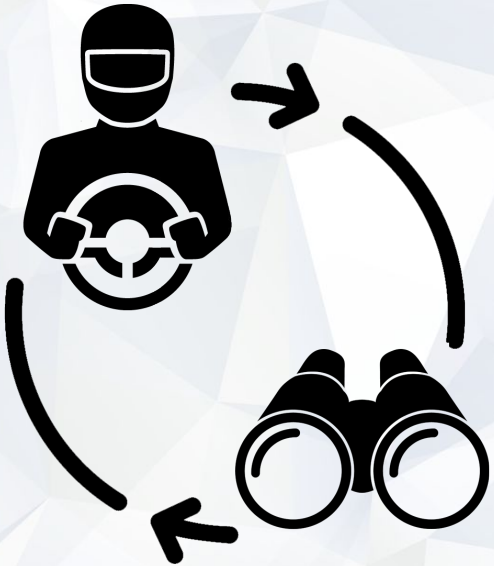
Alert is being rendered inside of App, and App is passing props to Alert.



Instructor Demonstration

Props

Questions?



Pair Programming Activity:

Calculator Props

In this activity, you will work with a partner to write a component that accepts props, performs arithmetic, and renders the result.

Suggested Time:

10 Minutes

Time’s Up! Let’s Review.

Review: Calculator Props

We're passing each `Math` component three props:

- `num1`
- `operator`
- `num2`

```
import React from "react";
import Math from "./Math";

// Calculator renders the Math component 4 times with different props
function Calculator() {
  return (
    <div>
      {/* Math renders a span tag containing the result */}
      {/* Each span is the font-size of the result in pixels */}
      <p>
        19 + 142 = <Math num1={19} operator="+" num2={142} />
      </p>
      <p>
        42 - 17 = <Math num1={42} operator="-" num2={17} />
      </p>
      <p>
        100 * 3 = <Math num1={100} operator="*" num2={3} />
      </p>
      <p>
        96 / 4 = <Math num1={96} operator="/" num2={4} />
      </p>
    </div>
  );
}

export default Calculator;
```

The numbers are wrapped in JSX curly braces, but the operator is in quotes.

Why do you think this is?

Review: Calculator Props

The operator is a string literal, and we can express that shorthand just by using quotes without curly braces. The following are equivalent:

```
<Math num1={19} operator={"+"} num2={341} />
```

This shorthand only works for string literals. All other values that we pass as props need to be in JSX curly braces.

```
<Math num1={19} operator="+" num2={341} />
```

Review: Calculator Props

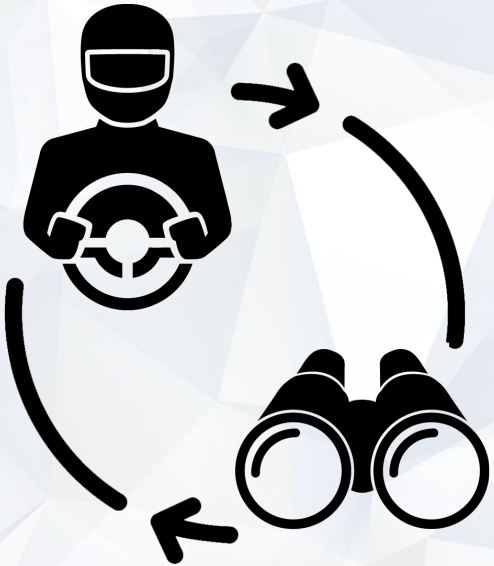
- The `props` argument should be an object containing all of the values passed to the rendered `Math` component in the `Calculator.js` file.
- At the bottom of the function, we're returning `<span>{value}</span>`.

```
// The Math function component accepts a props argument
function Math(props) {
  let value;

  // Assign value based on the operator
  switch (props.operator) {
  case "+":
    value = props.num1 + props.num2;
    break;
  case "-":
    value = props.num1 - props.num2;
    break;
  case "*":
    value = props.num1 * props.num2;
    break;
  case "/":
    value = props.num1 / props.num2;
    break;
  default:
    value = NaN;
  }

  // Return a span element containing the calculated value
  // Set the fontSize to the value in pixels
  return <span style={{ fontSize: value }}>{value}</span>;
}
```

Questions?



Pair Programming Activity:

Props Review

In this activity, you will work with a partner to make an existing React application DRIER through the use of reusable components and props.

Suggested Time:

15 Minutes

Time's Up! Let's Review.

Review: Props

The application being rendered to the browser doesn't look any different from the unsolved version, but now we've made our code DRIER by creating a reusable component, `FriendCard`, to render each friend with the appropriate prop inside of the App component.

Friends List

Name: SpongeBob

Occupation: Fry Cook

Location: A Pineapple Under the Sea

Name: Mr. Krabs

Occupation: Restaurant Owner

Location: A Giant Anchor

Name: Squidward

Occupation: Cashier

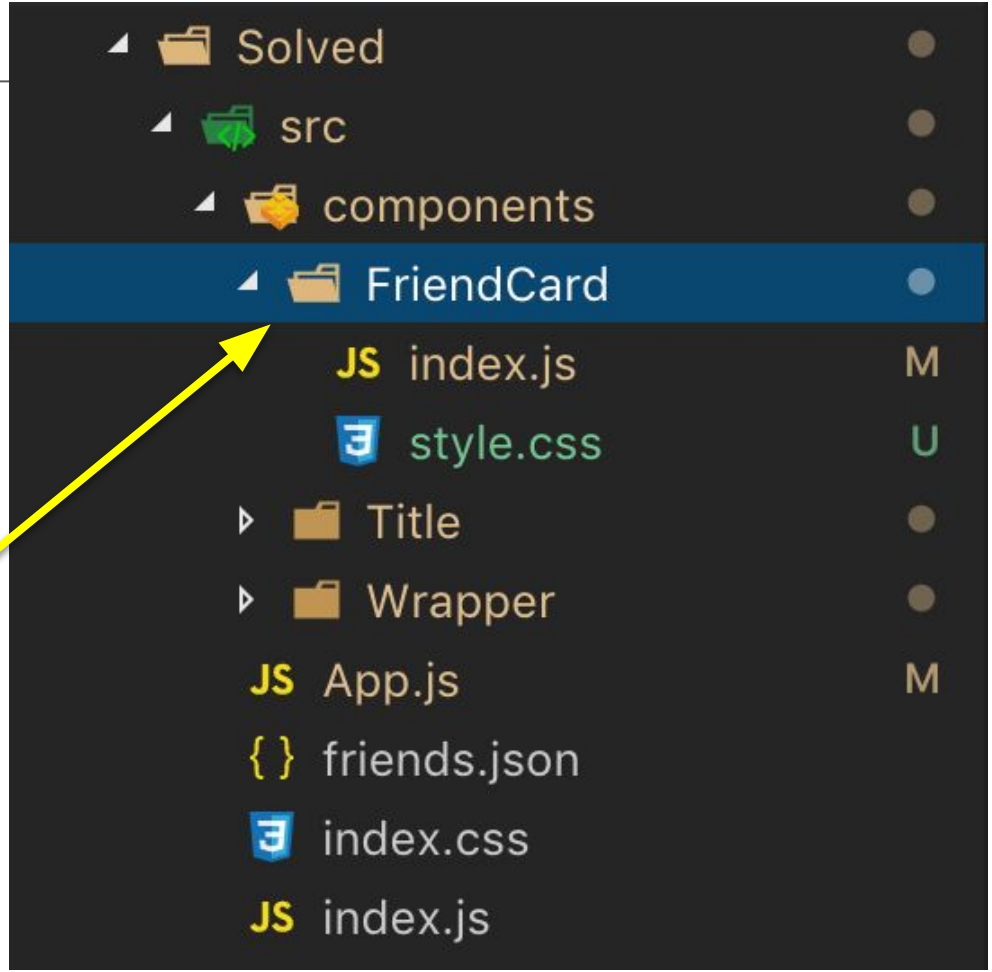
Location: An Easter Island Head

In a real application, where might all of the friend JSON data come from?

Review: Props

Normally, we might receive the friend JSON from an AJAX request and probably won't know ahead of time which friends will need to be rendered.

Each component is contained inside of its own folder containing a CSS file and an `index.js` file.



Why are we using `index.js` to hold the component instead of `FriendCard.js`?

Review: Props

Whenever we require/import a folder instead of a file, the folder's `index.js` file is required/imported by default (if it exists).

This allows us to keep our paths for importing these components short. For example, we can do:

```
import FriendCard from "./components/FriendCard";
```

instead of:

```
import FriendCard from "./components/FriendCard/FriendCard";
```

Giving all of our components their own folder is another option for organizing our React apps. Each folder could contain any CSS or other dependencies that the component will need.

Activity: Component Map

In this activity, you will utilize the map method in order to render JSX from an array of objects.

Suggested Time:

10 Minutes

Time's Up! Let's Review.

Review: Component Map

The array of grocery objects is passed into the `List` component from inside of `App`, making it available inside of the `List` component as `props.groceries`.

```
// Whenever we try to render an array containing JSX, React knows to render each JSX element separately
const List = props => (
  <ul className="list-group">
    {props.groceries.map(item => (
      <li className="list-group-item" key={item.id}>
        {item.name}
      </li>
    ))}
  </ul>
);
```

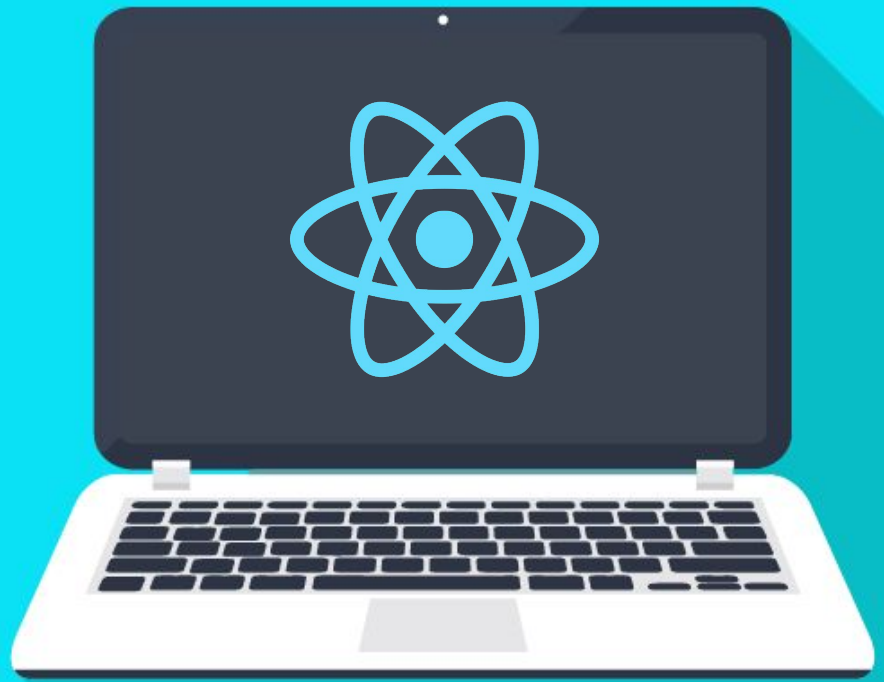
Inside of the `List` component, we insert JSX curly braces inside of the `ul` element. We map over `props.groceries` and return one `il` tag for every element in `props.groceries`.

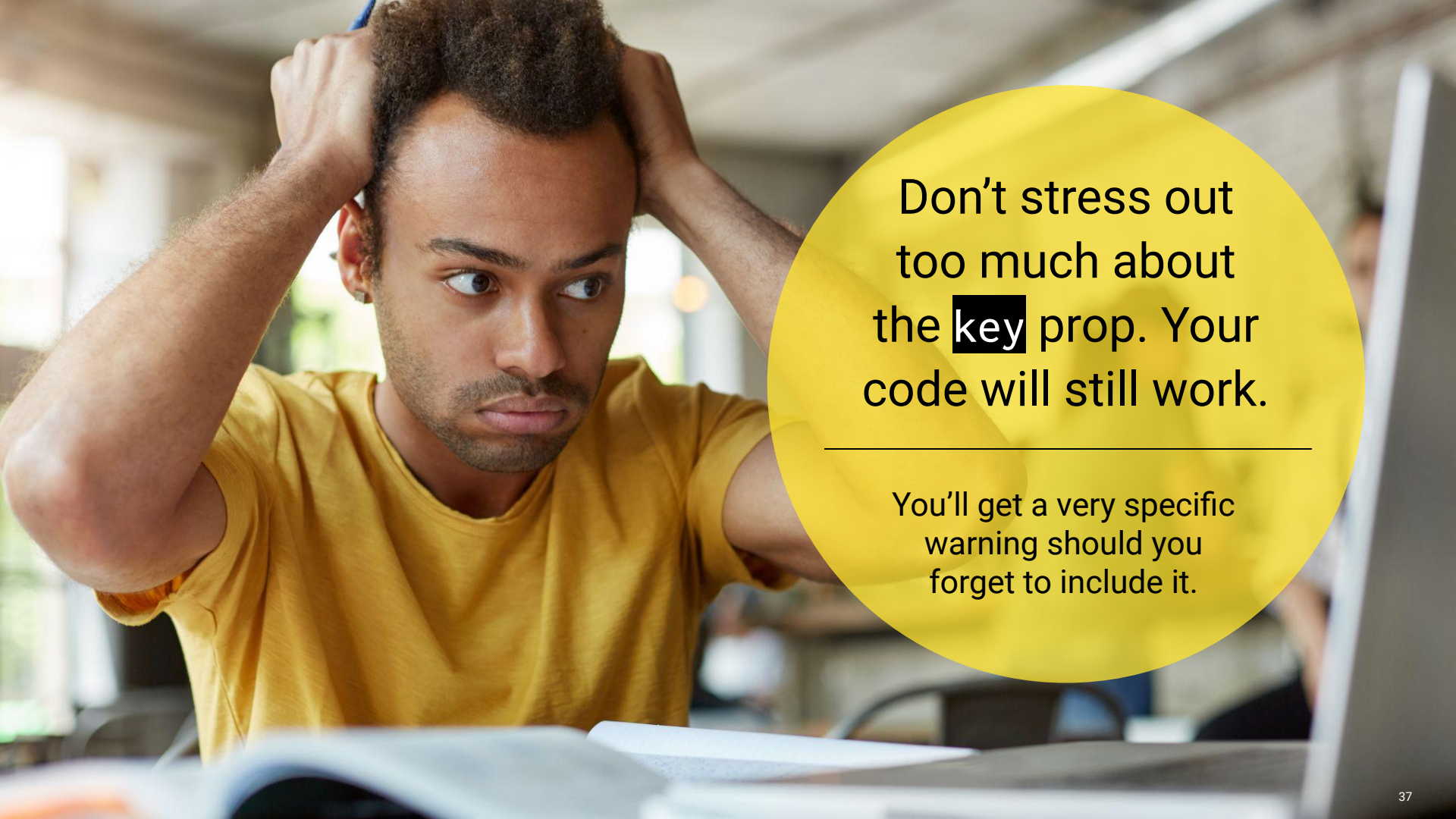
What type of value is returned by the map method here?

Review: Component Map

The map method will always return an array—in this example, it's returning an array of JSX elements.

React is smart enough to know that whenever we're rendering an array containing JSX, it should deconstruct the array and render each element inside of its parent.




Don’t stress out too much about the key prop. Your code will still work.
You’ll get a very specific warning should you forget to include it.

Questions?


Break

Stateful Components

What we've been working with so far are known as stateless, functional components

Stateless Components Can...

Stateful vs. Stateless Components

In a React application, **most** components should be stateless components.

Stateless

These are easy to test, debug, and they tend to be more reusable—even across applications—because they usually don't depend on how the rest of the application works.

Stateful

These special components aren't created using plain JavaScript functions, but with ES6 classes (which, if we want to get technical, are still JavaScript constructor functions once compiled).

Stateful Components

`state` is a special type of property attached to a class component that can contain data that we want to associate with that component.

Values stored on a component's state are different from regular variables because, unlike regular variables, when a component updates its `state`, the React application will update itself in the browser to reflect the change wherever necessary.

A component can set and update its own state, whereas its props are always received from up above and considered immutable (can't/shouldn't be changed).

Instructor Demonstration

Basic State

Questions?

Stateful Components

Takeaways

01

We can use state to associate data with our components and keep track of any values that we want to update the UI when changed.

02

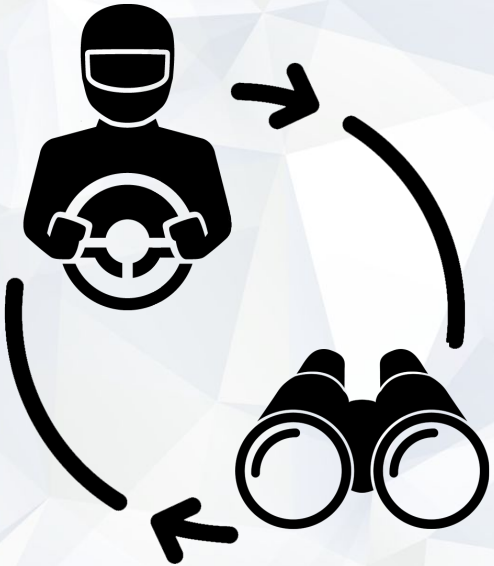
We can define methods on a class component and pass them as props.

03

The onClick prop can be used to set a click event listener to an element.

Stateful vs. Stateless Components

Use a stateless component when:	Use a stateful component when:
You just need to present the prop.	Building an element that accepts user input.
You don’t need a state or any internal variables.	Building an element that is interactive on the page.
Creating an element that does not need to be interactive.	Dependent on state for rendering, such as fetching data before rendering.
You want reusable code.	Dependent on any data that cannot be passed down as props.



Pair Programming Activity:

Decrement Counter

In this activity, you will will add a “Decrement” button and event handler to the previous click counter example.

(Instructions sent via Slack)

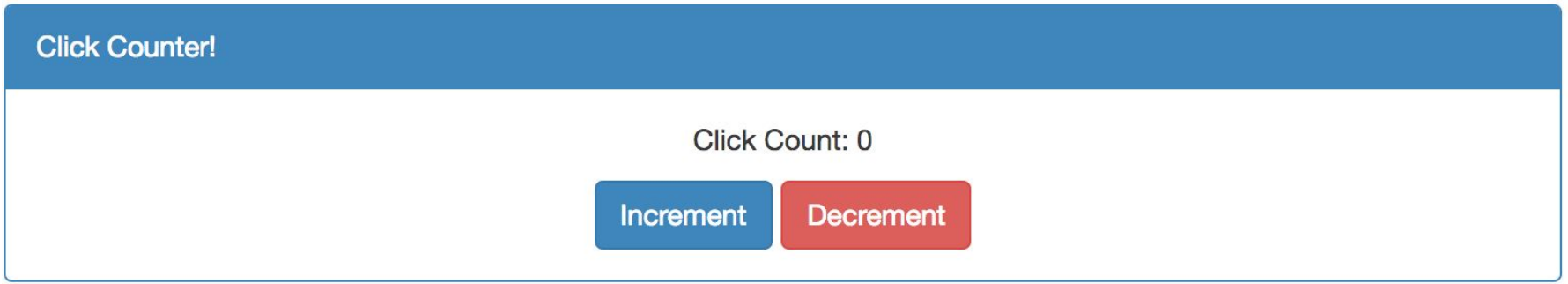
Suggested Time:

10 Minutes

Time's Up! Let's Review.

Review: Decrement Counter

You'll get more practice with working with class components.



Questions?

Activity: Friend Refactor

In this activity, you will further refactor the friends list application from earlier to use class components, events, and programmatically render the `FriendCard` components.

(Instructions sent via Slack)

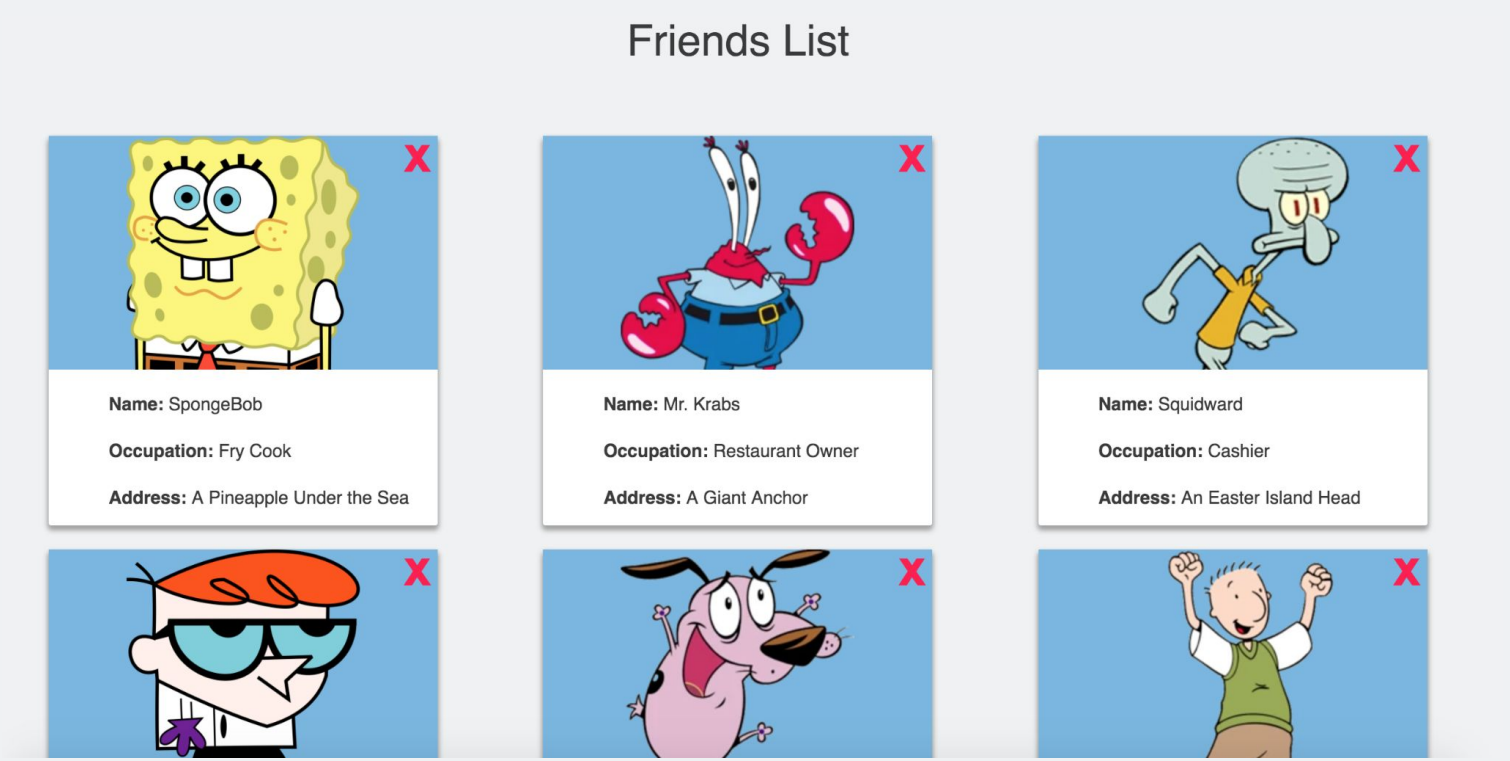
Suggested Time:

20 Minutes

Time's Up! Let's Review.

Review: Friend Refractor

We can remove friends by clicking the red **X** icon.



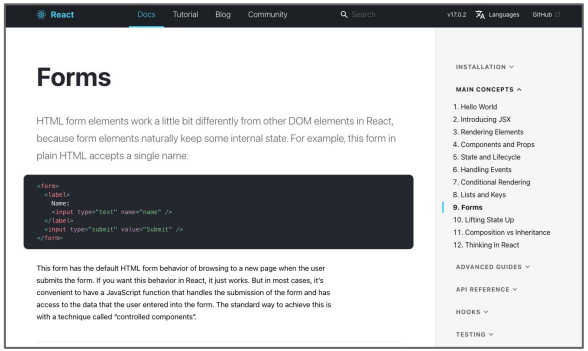
Questions?

We'll continue to go through forms in the next lesson.

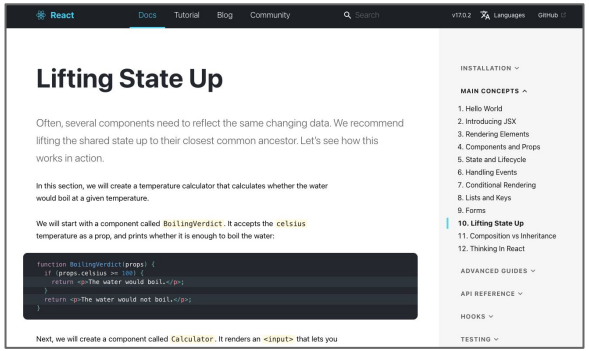
Next Class

Do your best to go through the following sections of the React documentation before the next class:

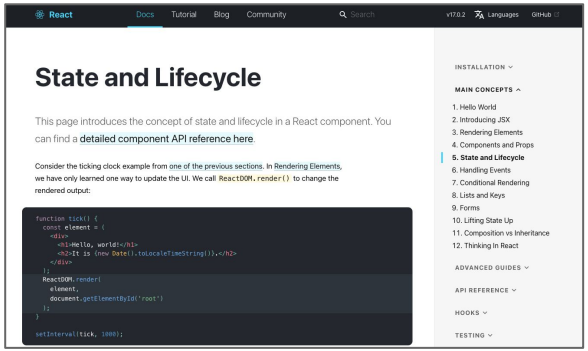
Forms



Lifting State Up



State and Lifecycle




The End