State

Read this in pairs for 15 mins: https://beta.reactjs.org/learn/state-a-components-memory

useState is a Hook – a kind of function in React (see above docs).

What is State?

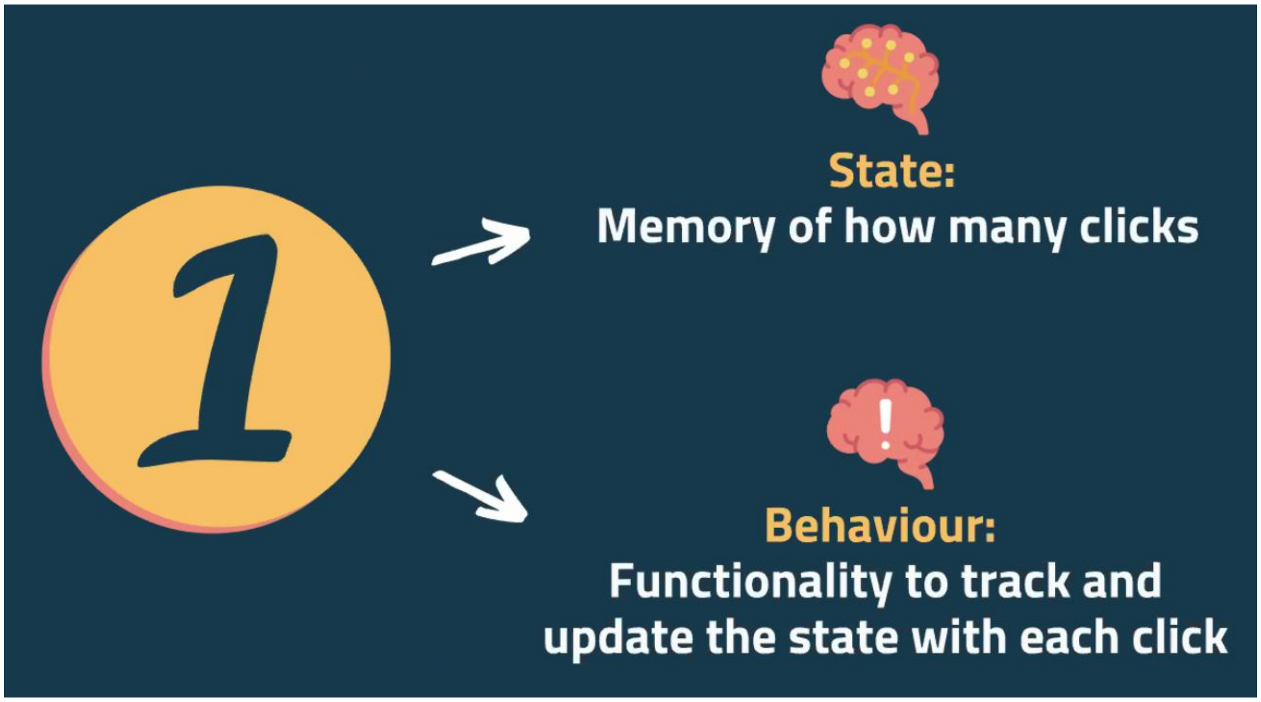
Using a "state" is a way of giving your app a "memory" and managing anything you want to control or track.

Smart component vs Dumb component = Smart controls its own state, Dumb reacts to what it's handed. E.g. Parent components tend to be smarter than children as parent controls the props to pass in and the children just use those props they've been handed

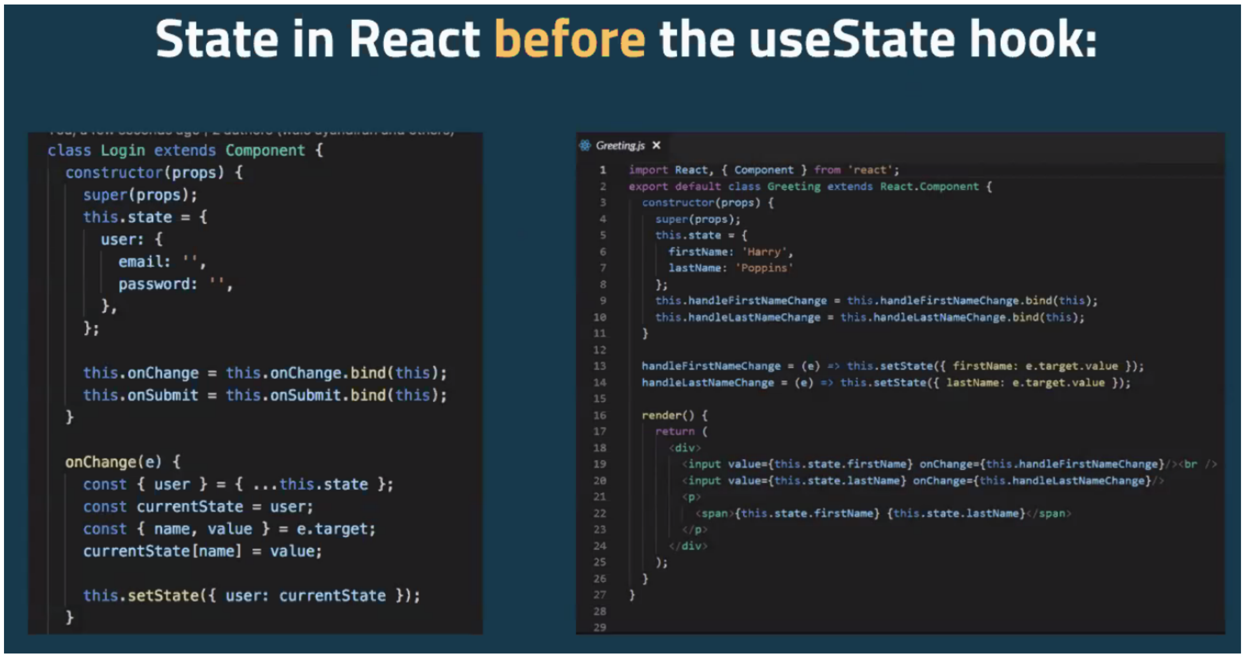
It is more common to want more dumb components – remember separation of concerns and only needing to change something once, in one place. So only use Smart components where you really need them.

How useState works at a basic level:

Imagine a counter button – it displays the number in it and each time you click it, the number increases by 1. To make this work, it needs to remember the previous value, and needs a way to increase that value by 1.


1
State:
Memory of how many clicks
Behaviour:
Functionality to track and update the state with each click

Don’t worry about taking this in, it’s a shock tactic to show us what it used to look like using a “this” and “bind” keyword:



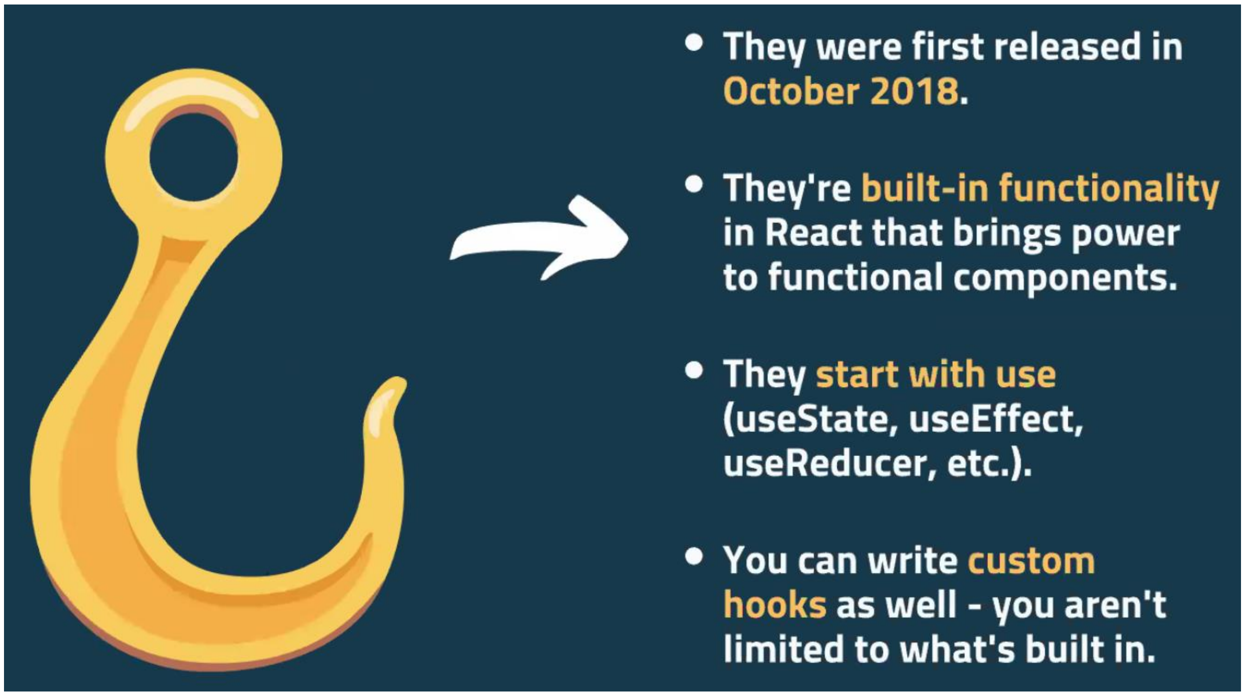
State in React before the useState hook:

```
class Login extends Component {
  constructor(props) {
    super(props);
    this.state = {
      user: {
        email: '',
        password: '',
      },
    };

    this.onChange = this.onChange.bind(this);
    this.onSubmit = this.onSubmit.bind(this);
  }

  onChange(e) {
    const { user } = { ...this.state };
    const currentState = user;
    const { name, value } = e.target;
    currentState[name] = value;

    this.setState({ user: currentState });
  }
```

Greeting.js

```
import React, { Component } from 'react';
export default class Greeting extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      firstName: 'Harry',
      lastName: 'Poppins'
    };
    this.handleFirstNameChange = this.handleFirstNameChange.bind(this);
    this.handleLastNameChange = this.handleLastNameChange.bind(this);
  }

  handleFirstNameChange = (e) => this.setState({ firstName: e.target.value });
  handleLastNameChange = (e) => this.setState({ lastName: e.target.value });

  render() {
    return (
      <div>
        <input value={this.state.firstName} onChange={this.handleFirstNameChange}/><br />
        <input value={this.state.lastName} onChange={this.handleLastNameChange}/>
        <p>
          <span>{this.state.firstName} {this.state.lastName}</span>
        </p>
      </div>
    );
  }
}
```

Hooks in React:



- They were first released in October 2018.
- They're built-in functionality in React that brings power to functional components.
- They start with use (useState, useEffect, useReducer, etc.).
- You can write custom hooks as well - you aren't limited to what's built in.

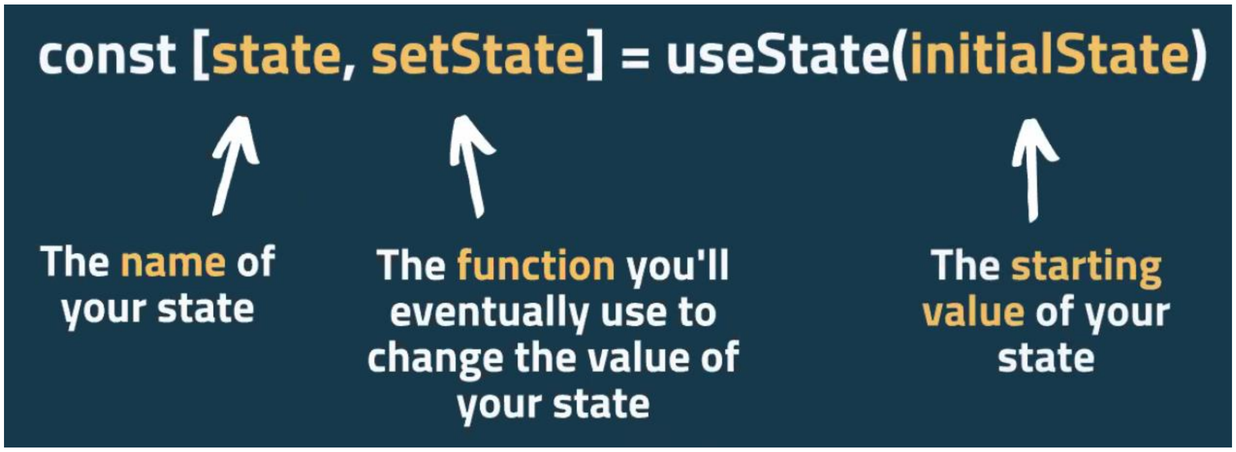
How do we use useState?

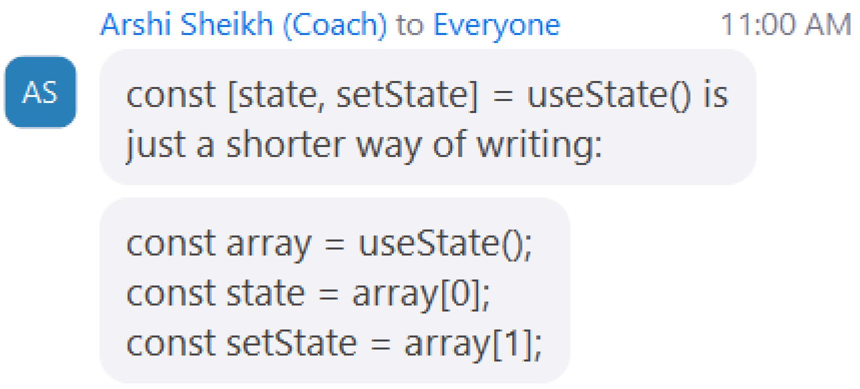
Use it at the top of the function component. Using const as it acts like a variable (even though the value of state is going to update!).

useState returns an array – so the left hand side of = is destructuring the array that useState provides (like object destructuring from before).

Call "state" something semantic and declarative, e.g. count in the earlier example. Convention is to name the second item *set* then whatever you named the first thing (e.g. setCount in earlier example). In the earlier example, we would write useState(0) to set the initial count value as 0.

Your state can be any data type e.g. boolean, string, number etc.


const [state, setState] = useState(initialState)
The name of your state
The function you'll eventually use to change the value of your state
The starting value of your state


Arshi Sheikh (Coach) to Everyone 11:00 AM

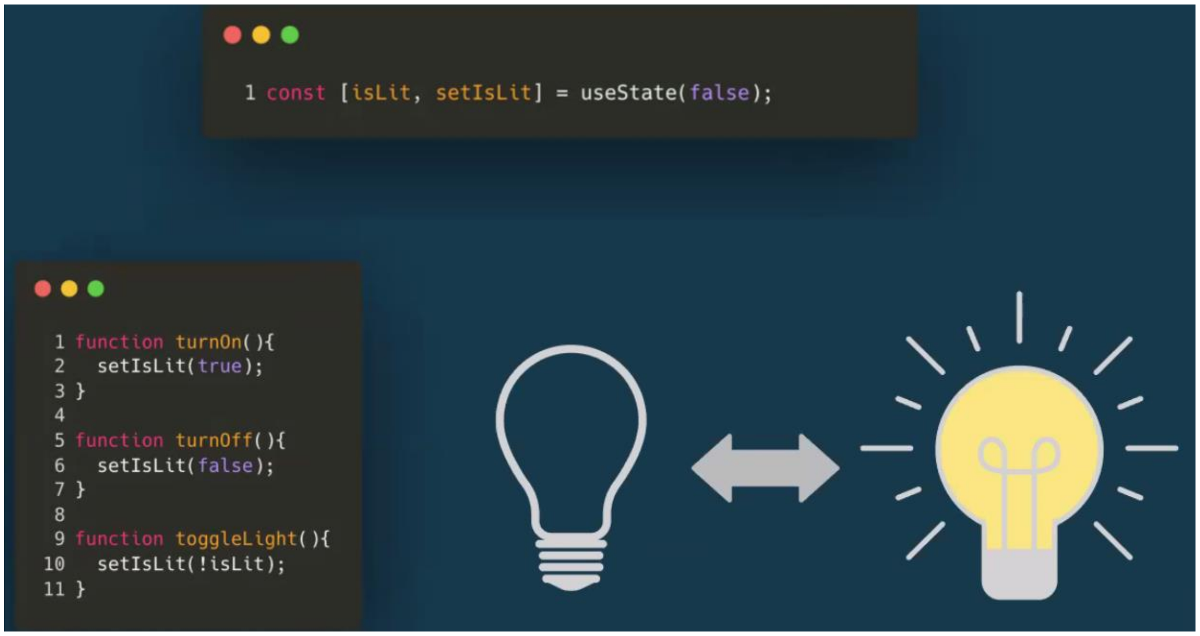
const [state, setState] = useState() is just a shorter way of writing:

```
const array = useState();
const state = array[0];
const setState = array[1];
```

Note from Arshi:

Pseudo code to demo how this works in practice:

Top code is the proper code for using State. Bottom left code is what's happening under the hood in pseudo terms (functions to turn it on or off and line 9-11 sets it to whatever it is not (using ! operator)).



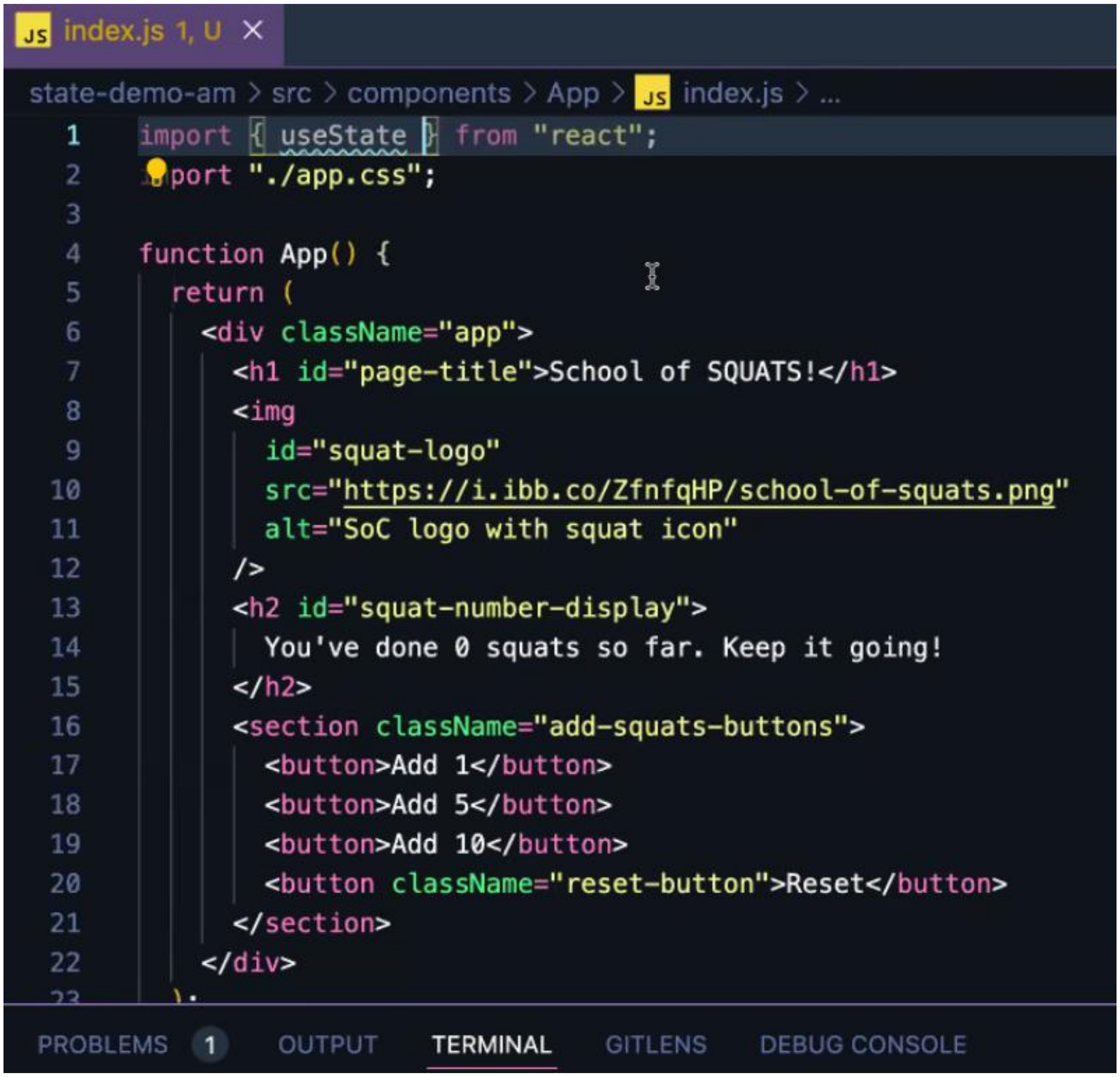
```
const [isLit, setIsLit] = useState(false);
```

```
function turnOn(){
  setIsLit(true);
}

function turnOff(){
  setIsLit(false);
}

function toggleLight(){
  setIsLit(!isLit);
}
```

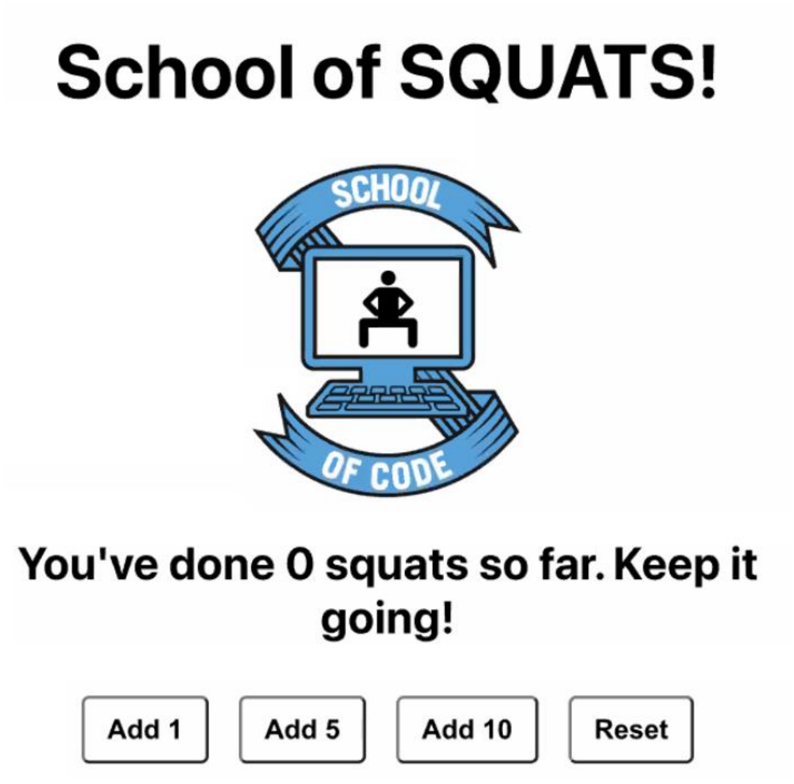
Side note – convention to name Booleans is to precursor it with the word "is". So itLit semantically suggests it is a Boolean.

Code for an app that counts squats, including what the page looks like. Buttons don't work at the moment:

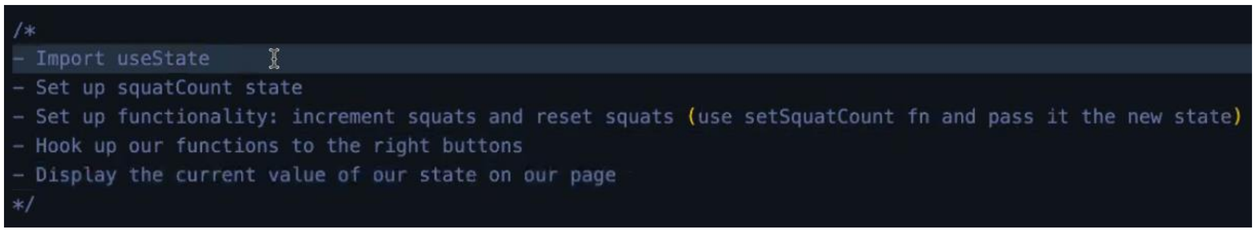
Don't need to import React from "react" inside of create-react-app anymore (used to have to do that), just {useState} hook for now.



```
import { useState } from "react";
import "./app.css";

function App() {
  return (
    <div className="app">
      <h1 id="page-title">School of SQUATS!</h1>
      <img
        id="squat-logo"
        src="https://i.ibb.co/ZfnfqHP/school-of-squats.png"
        alt="SoC logo with squat icon"
      />
      <h2 id="squat-number-display">
        You've done 0 squats so far. Keep it going!
      </h2>
      <section className="add-squats-buttons">
        <button>Add 1</button>
        <button>Add 5</button>
        <button>Add 10</button>
        <button className="reset-button">Reset</button>
      </section>
    </div>
  );
```



Plan:

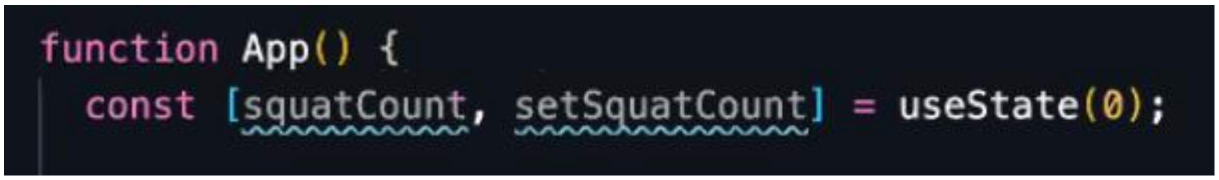


```
/*
- Import useState
- Set up squatCount state
- Set up functionality: increment squats and reset squats (use setSquatCount fn and pass it the new state)
- Hook up our functions to the right buttons
- Display the current value of our state on our page
*/
```

<u>Set up squatCount state</u>

Declare state at the top of the function. (After line 4 in this case)

const [squatCount, setSquatCount] = useState(0)



```
function App() {
  const [squatCount, setSquatCount] = useState(0);
```

<u>Set up functionality</u>

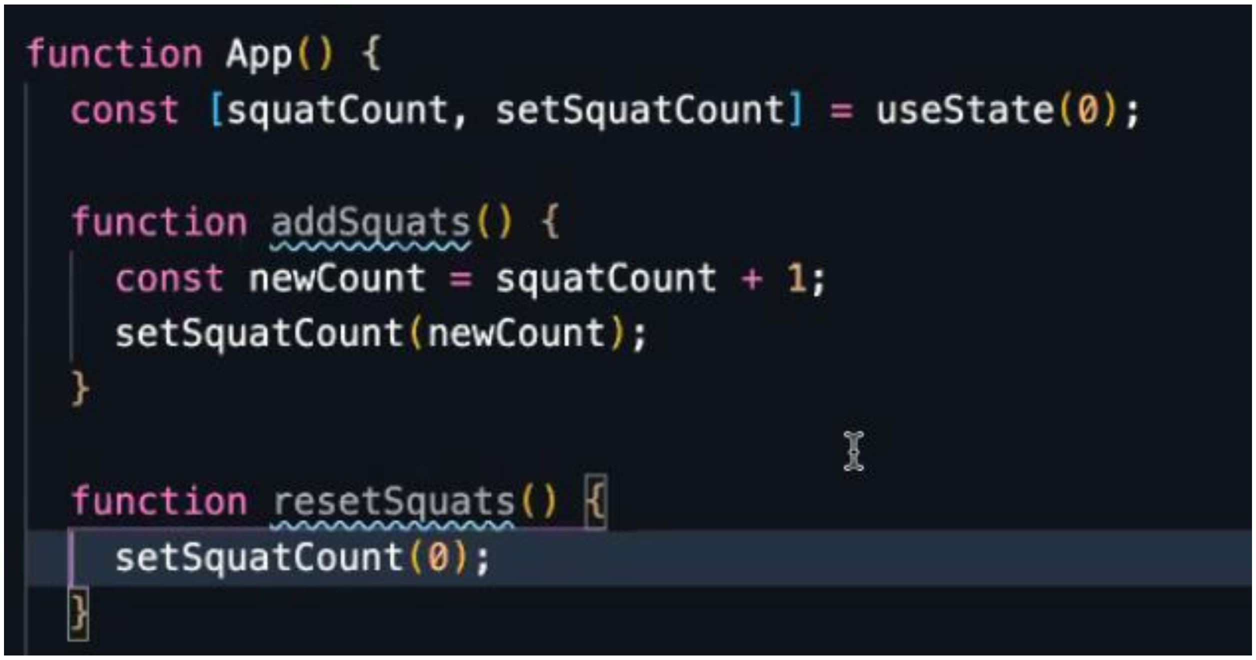
```
function addSquats(){
        const newCount = squatCount + 1;
        setSquatCount(newCount);
}
```

(Side note - You could write the above like the below, but above is more semantic to Liz and she finds it more readable:

```
        function addSquats(){
                setSquatCount = squatCount + 1;
        }
)
```

```
function resetSquats(){
        setSquatCount(0);
}
```

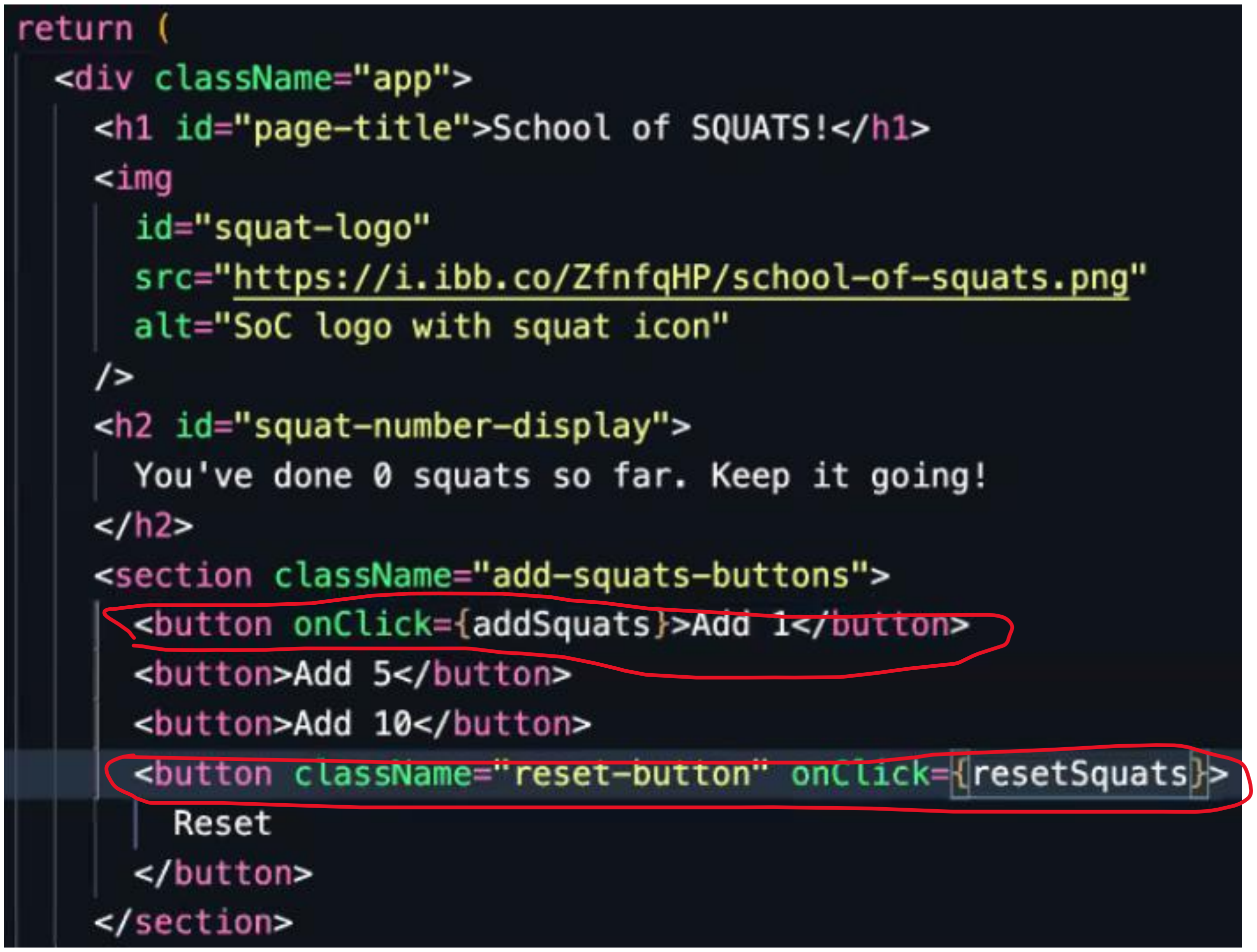
(we're only incrementing by 1 and resetting for now)



```
function App() {
  const [squatCount, setSquatCount] = useState(0);

  function addSquats() {
    const newCount = squatCount + 1;
    setSquatCount(newCount);
  }

  function resetSquats() {
    setSquatCount(0);
  }
```

<u>Hook up functionality:</u>

Modify JSX for those buttons to include onClick={addSquats} / onClick={resetSquats}

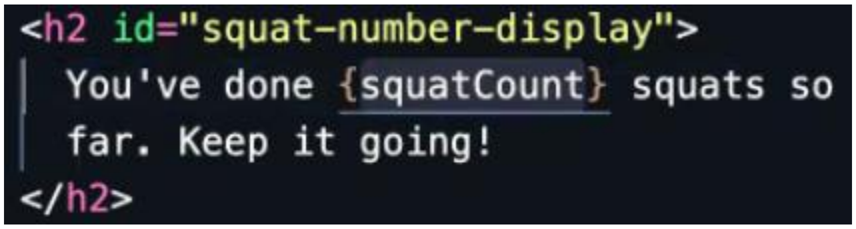


```
return (
  <div className="app">
    <h1 id="page-title">School of SQUATS!</h1>
    <img
      id="squat-logo"
      src="https://i.ibb.co/ZfnfqHP/school-of-squats.png"
      alt="SoC logo with squat icon"
    />
    <h2 id="squat-number-display">
      You've done 0 squats so far. Keep it going!
    </h2>
    <section className="add-squats-buttons">
      <button onClick={addSquats}>Add 1</button>
      <button>Add 5</button>
      <button>Add 10</button>
      <button className="reset-button" onClick={resetSquats}>
        Reset
      </button>
    </section>
```

Display the current value of our state on the page

Update the h2 to inject squatCount in to where 0 was previously hardcoded:

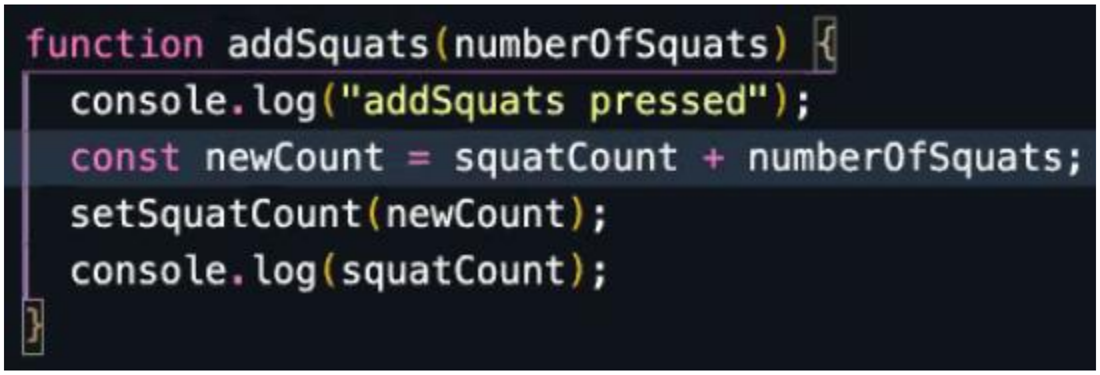
<h2 id=”squat-number-display”>You’ve done {squatCount} squats so far. Keep it going! </h2>



```
<h2 id="squat-number-display">
  You've done {squatCount} squats so
  far. Keep it going!
</h2>
```

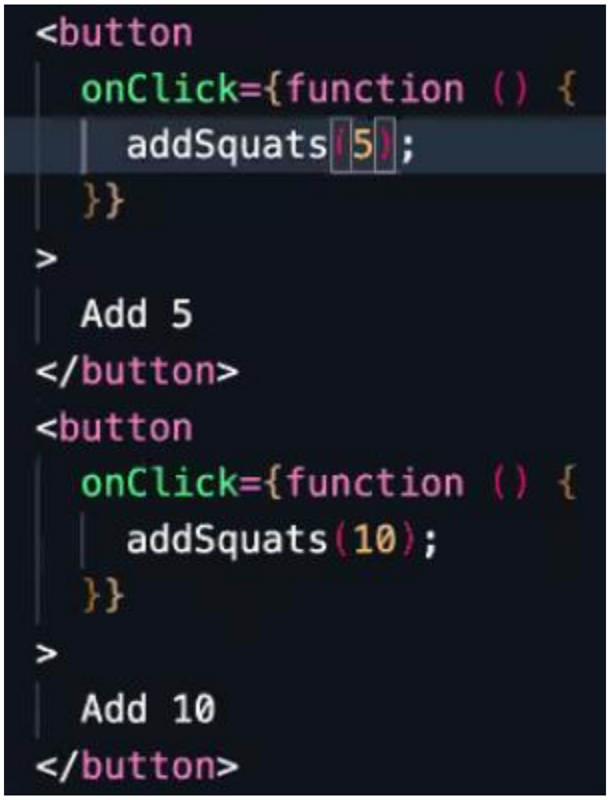
Then customised the addSquats function to increase by any number of squats and changed onClick functionality of each Add button to add the right amount:

Pass in numberOfSquats to the function:



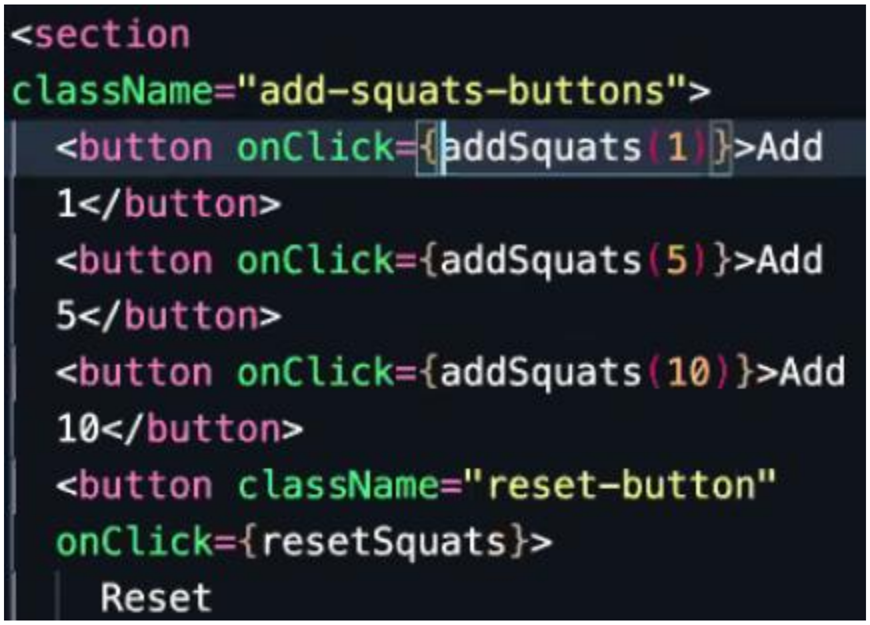
```
function addSquats(numberOfSquats) {
  console.log("addSquats pressed");
  const newCount = squatCount + numberOfSquats;
  setSquatCount(newCount);
  console.log(squatCount);
}
```

Refactor the onClick attribute of buttons to have a callback function that tells it to call addSquats and pass in the number:



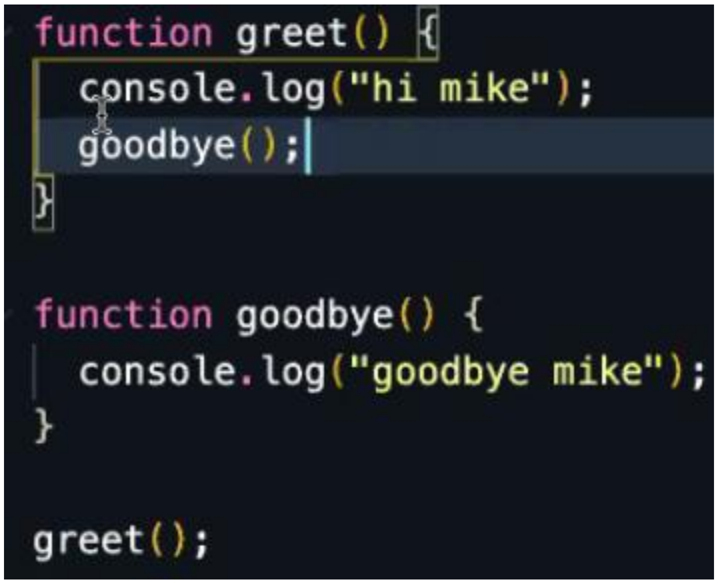
```
<button
  onClick={function () {
    addSquats(5);
  }}
>
  Add 5
</button>
<button
  onClick={function () {
    addSquats(10);
  }}
>
  Add 10
</button>
```

DON'T use smooth brackets like this otherwise you will get an infinite loop and the app won't even load (as JS is calling the function on page load, then React is instantly re-rendering as state is changing):



```
<section
className="add-squats-buttons">
  <button onClick={addSquats(1)}>Add
  1</button>
  <button onClick={addSquats(5)}>Add
  5</button>
  <button onClick={addSquats(10)}>Add
  10</button>
  <button className="reset-button"
  onClick={resetSquats}>
    Reset
```

This is because wrapping a function call inside another function means it only gets called when the first function is called. E.g. Greet function is called immediately on file load but Goodbye function only called when Greet is called:

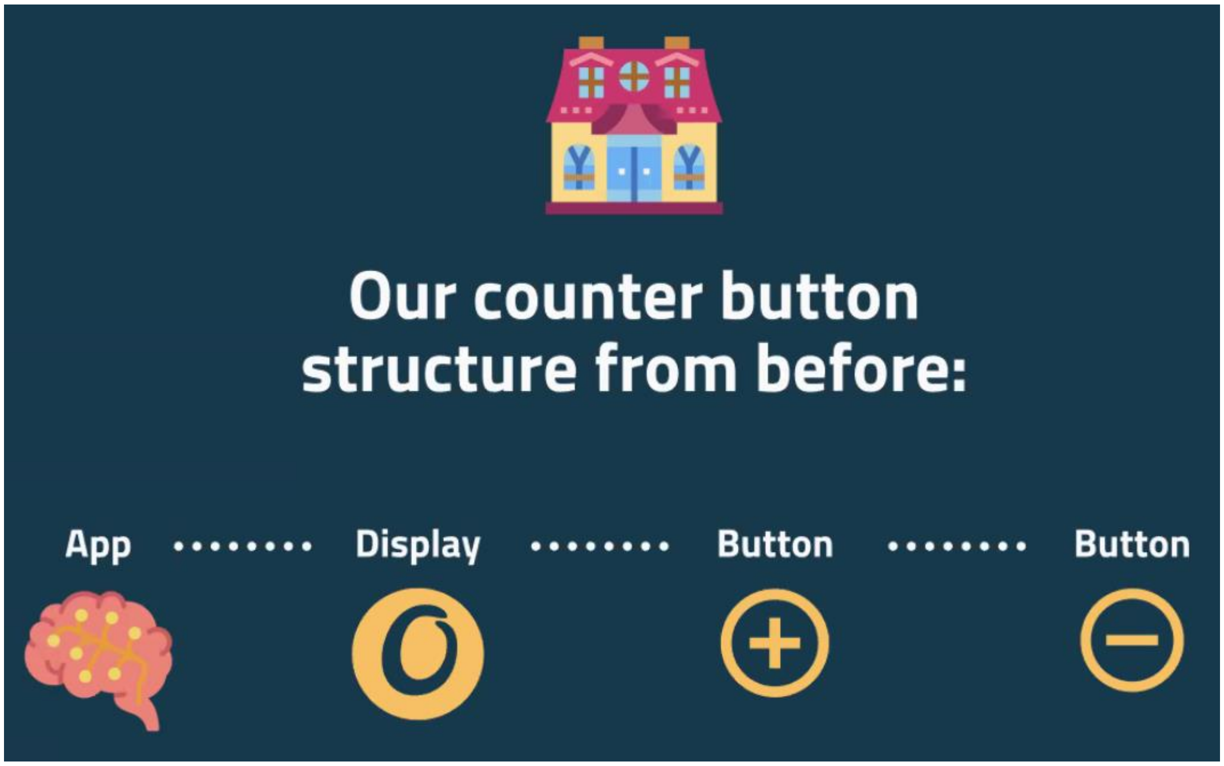


```
function greet() {
  console.log("hi mike");
  goodbye();
}

function goodbye() {
  console.log("goodbye mike");
}

greet();
```

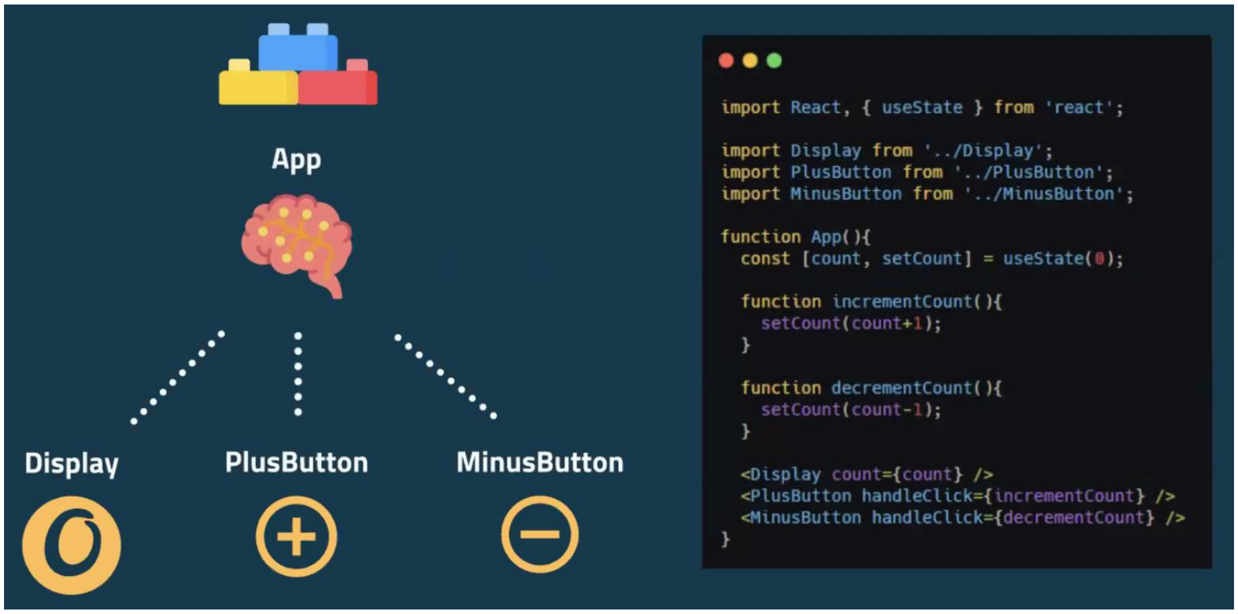
<u>Recycling Your Code</u>

Don't want to do everything inside our App.js.

This would be a doll house – fixed structure that you can't reuse bits of it in other apps (or elsewhere in your app).

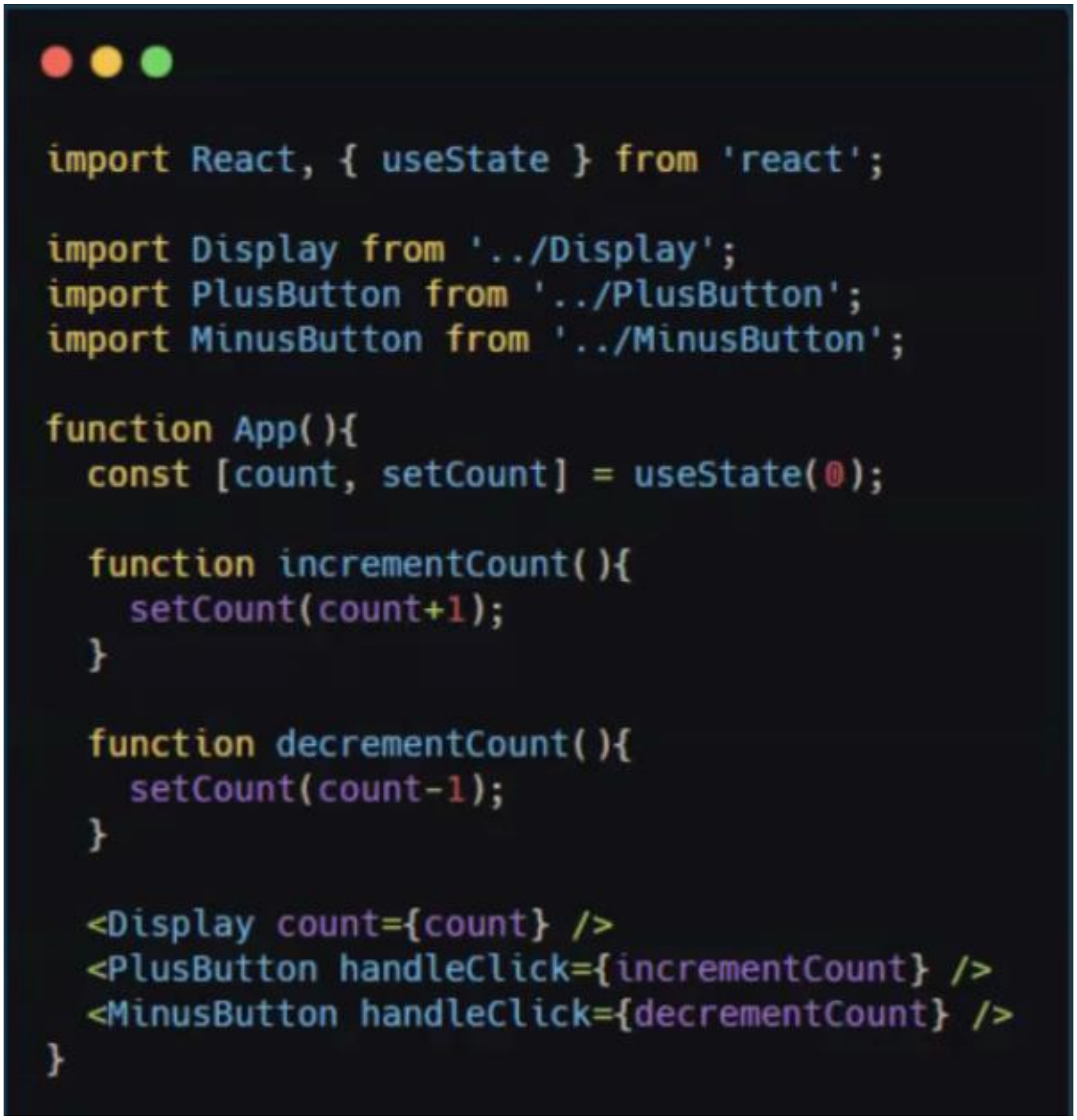


Make it more like lego blocks – separate components. This is abstraction/separation of concerns:



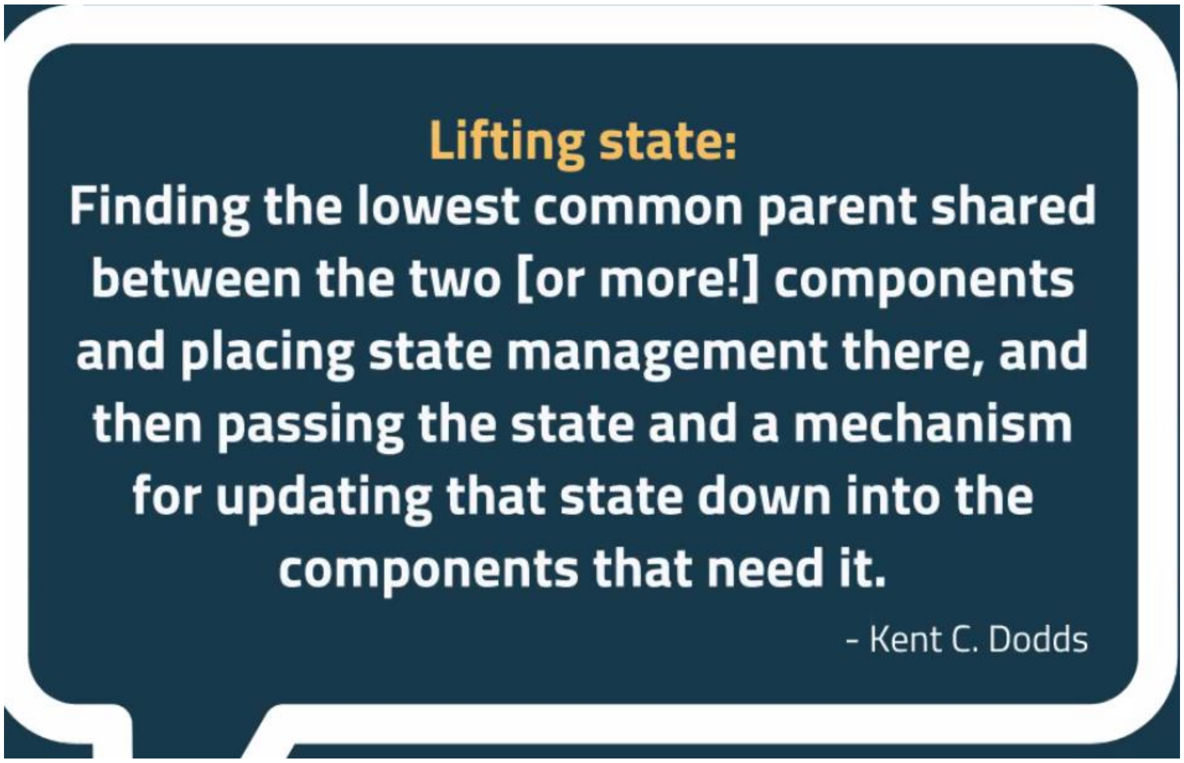
```
import React, { useState } from 'react';

import Display from '../Display';
import PlusButton from '../PlusButton';
import MinusButton from '../MinusButton';

function App(){
  const [count, setCount] = useState(0);

  function incrementCount(){
    setCount(count+1);
  }

  function decrementCount(){
    setCount(count-1);
  }

  <Display count={count} />
  <PlusButton handleClick={incrementCount} />
  <MinusButton handleClick={decrementCount} />
}
```

Set up 3 separate components for display, plus button and minus button, call them all inside app, passing in what they need. Keep the state in the App so all 3 are dumb components and you control state in just one place. Pass in the props to each component.
(Could even just have a button component without any sign in it to abstract even more)
Code from above screenshot zoomed in:



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

import Display from '../Display';
import PlusButton from '../PlusButton';
import MinusButton from '../MinusButton';

function App(){
  const [count, setCount] = useState(0);

  function incrementCount(){
    setCount(count+1);
  }

  function decrementCount(){
    setCount(count-1);
  }

  <Display count={count} />
  <PlusButton handleClick={incrementCount} />
  <MinusButton handleClick={decrementCount} />
}
```

This is called Lifting state:



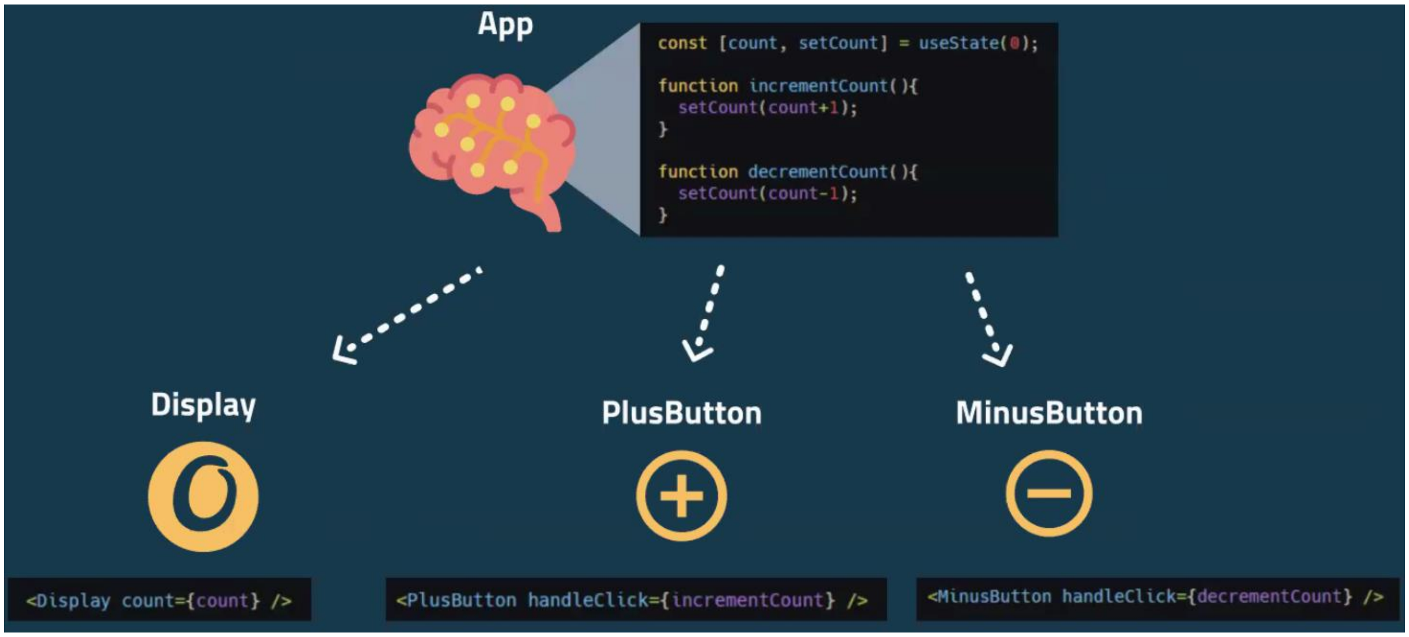
Lifting state:
Finding the lowest common parent shared between the two [or more!] components and placing state management there, and then passing the state and a mechanism for updating that state down into the components that need it.

- Kent C. Dodds

State management is just where you call the line of code to set state:
const [count, setCount] = useState(0)

(this is inside App.js in our example)

Same example broken down more:



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

function incrementCount(){
  setCount(count+1);
}

function decrementCount(){
  setCount(count-1);
}
```

```
<Display count={count} />
<PlusButton handleClick={incrementCount} />
<MinusButton handleClick={decrementCount} />
```

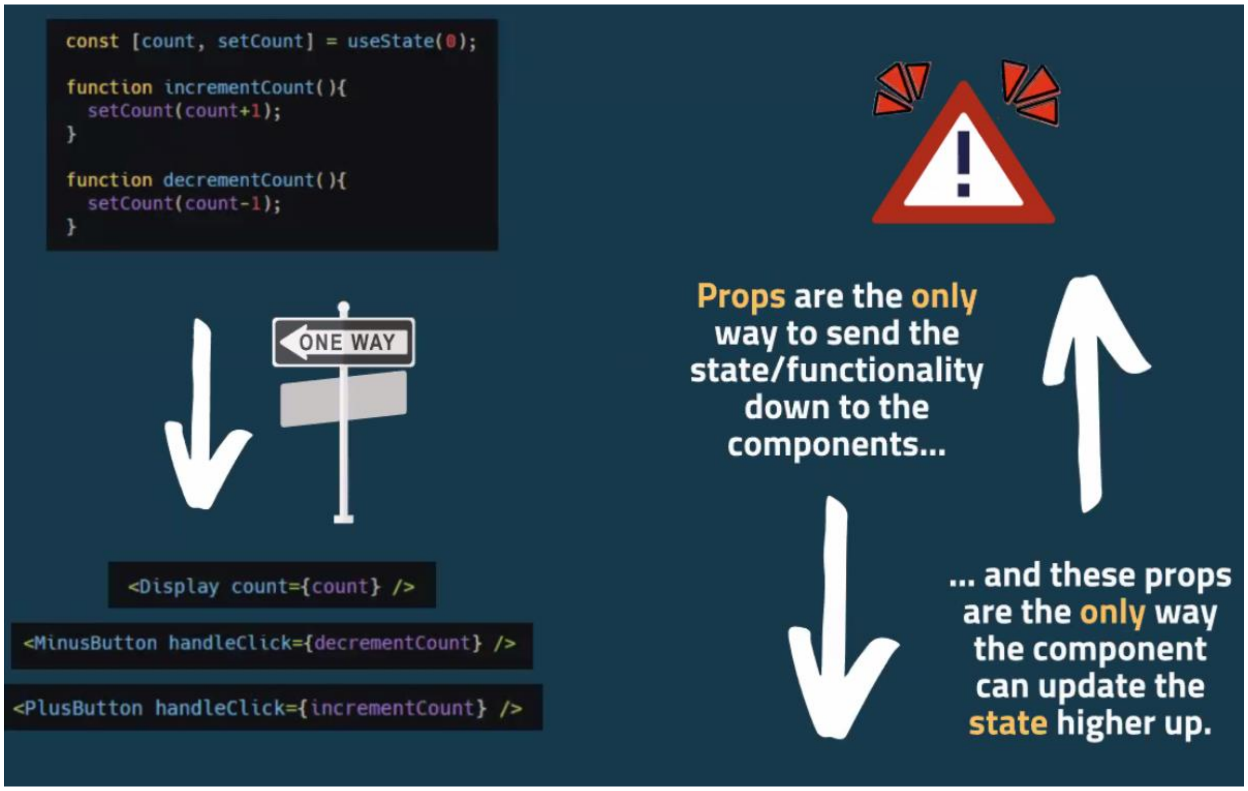
Passing count to the display component (so this will be re-rendered when count changes)

Setting up PlusButton to increment the count and MinusButton to decrement it (which will then cause Display to be re-rendered as above)

<u>Remember</u>

React can only respond to immutable changes – if you just reassign the value of count, React won't re-render (as that's a mutable change)! That's why we pass the new value to the setCount function to change it immutably – React then compares it's virtual DOM to the actual DOM and will notice the change and re-render accordingly.

Props are one way (from parent to child), but handing the function to change the count in the props allows the children to indirectly change the Count variable:



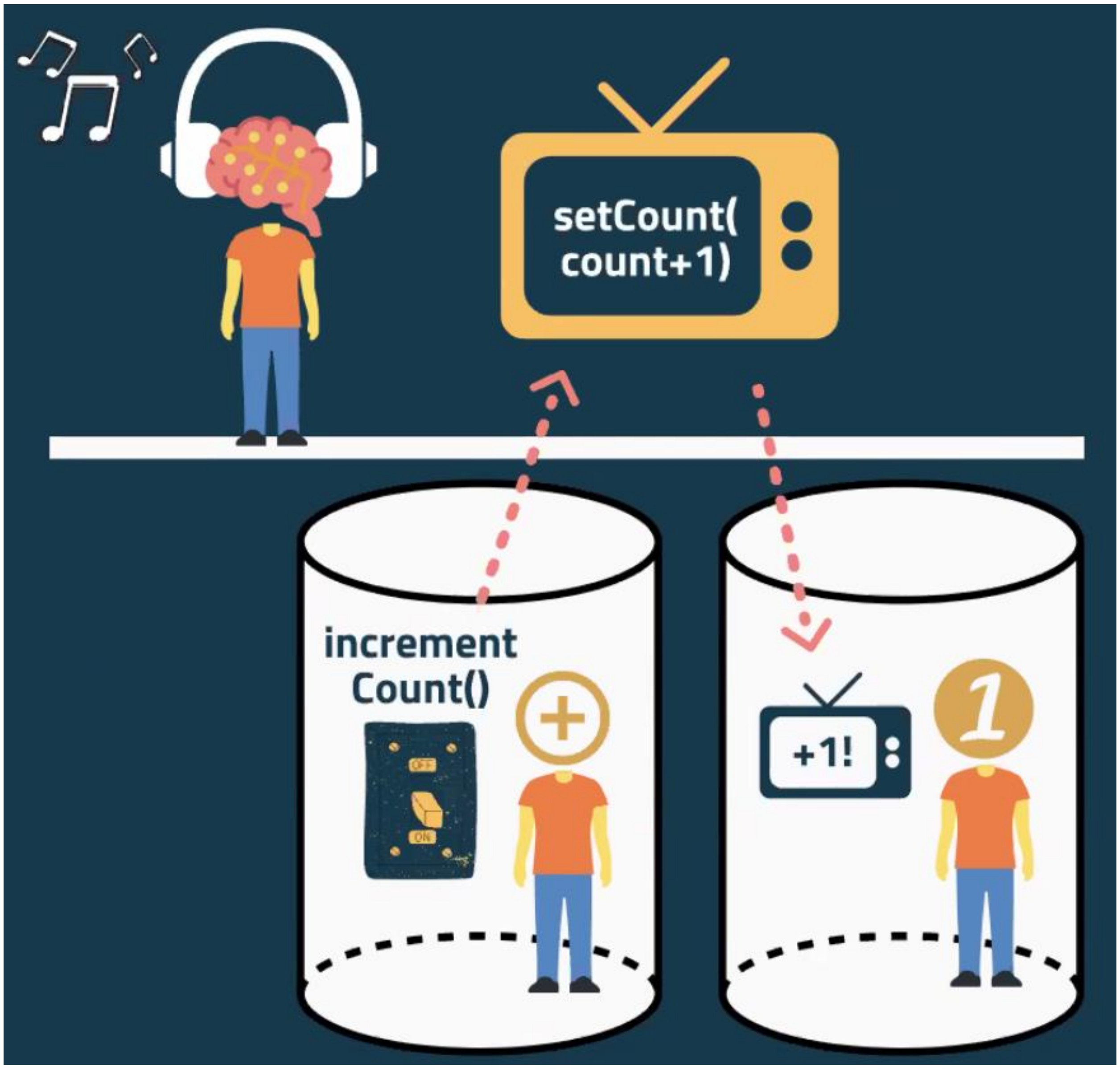
```
const [count, setCount] = useState(0);

function incrementCount(){
  setCount(count+1);
}

function decrementCount(){
  setCount(count-1);
}
```

```
<Display count={count} />
<MinusButton handleClick={decrementCount} />
<PlusButton handleClick={incrementCount} />
```

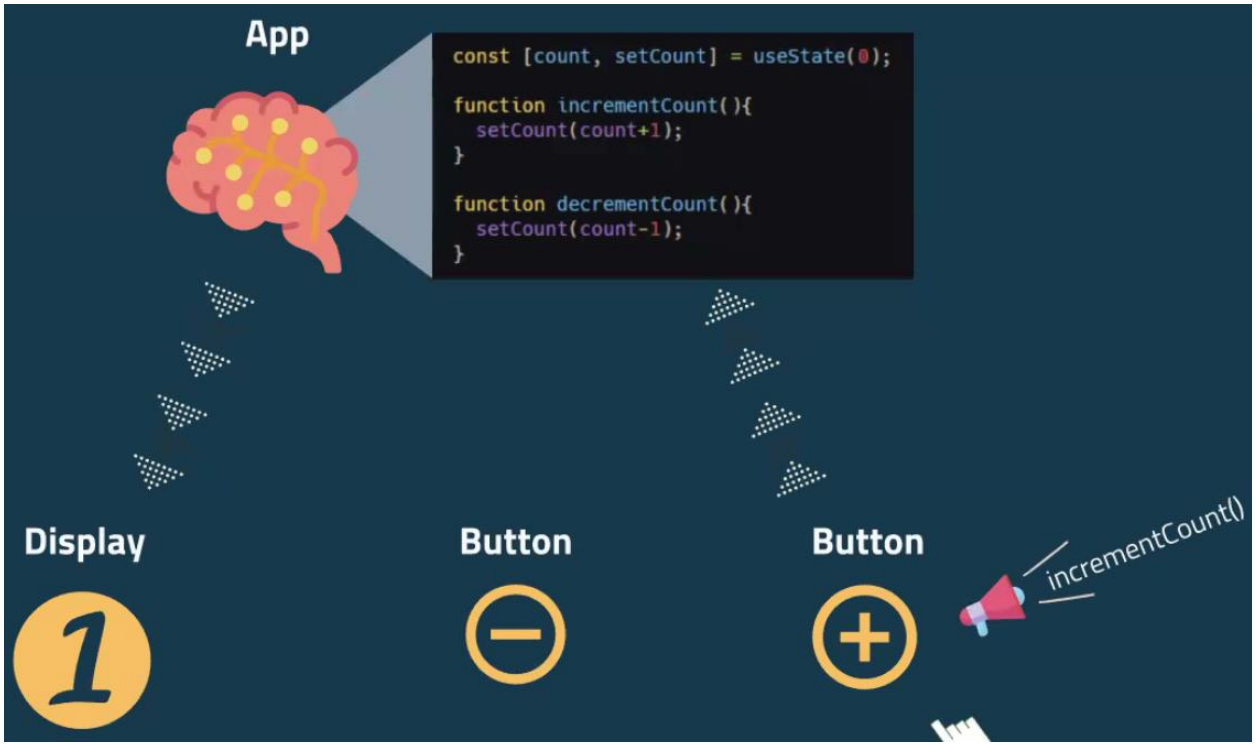
Props are the only way to send the state/functionality down to the components...

... and these props are the only way the component can update the state higher up.

Liz said this screenshot may confuse people, so it may not help!:
The plus button and display are in containers – can't talk to each other but plus button has been passed the function to increment the count (represented by the switch the switch).



Same example with previous code:



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

function incrementCount(){
  setCount(count+1);
}

function decrementCount(){
  setCount(count-1);
}
```

PM Workshop Demo

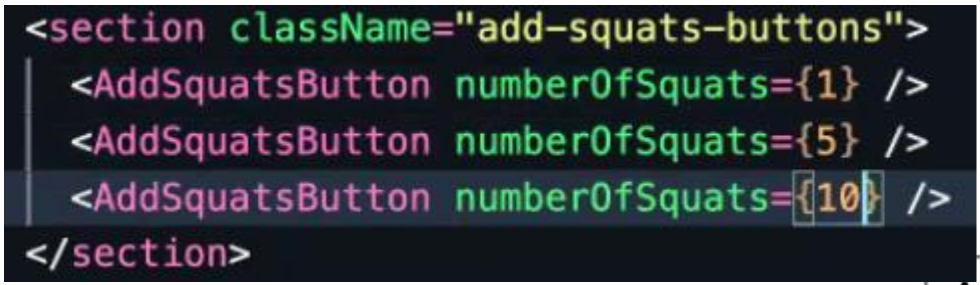
Original code just has h1 and SOC Squat image (no buttons or display).

Get the components to display (no functionality yet):

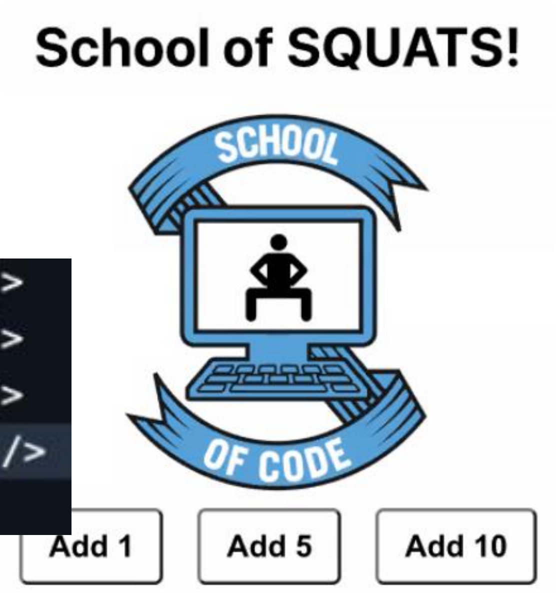
In App > index.js

Imported all components (Display, AddSquatsButton and ResetButton at top

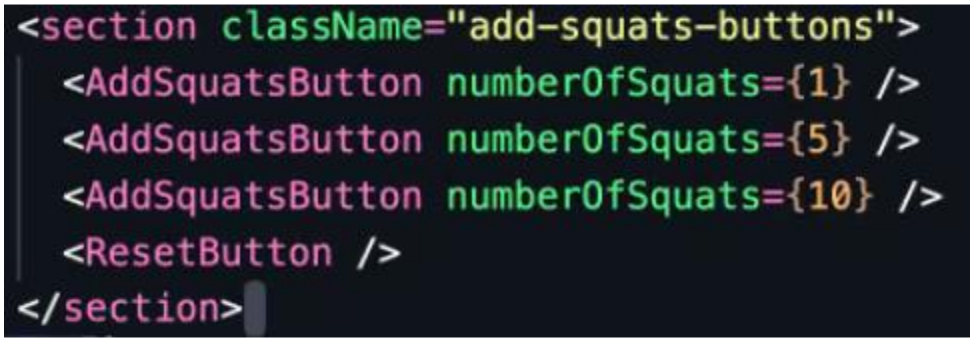
AddSquatButtons go inside section with className "add-squats-button" and you can give it a number and reuse it for however many different buttons you want (e.g. 1 squat, 5 squats, 10 squats):



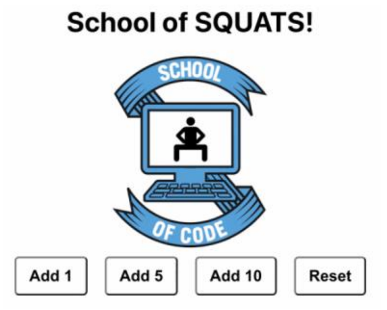
```
<section className="add-squats-buttons">
  <AddSquatsButton numberOfSquats={1} />
  <AddSquatsButton numberOfSquats={5} />
  <AddSquatsButton numberOfSquats={10} />
</section>
```



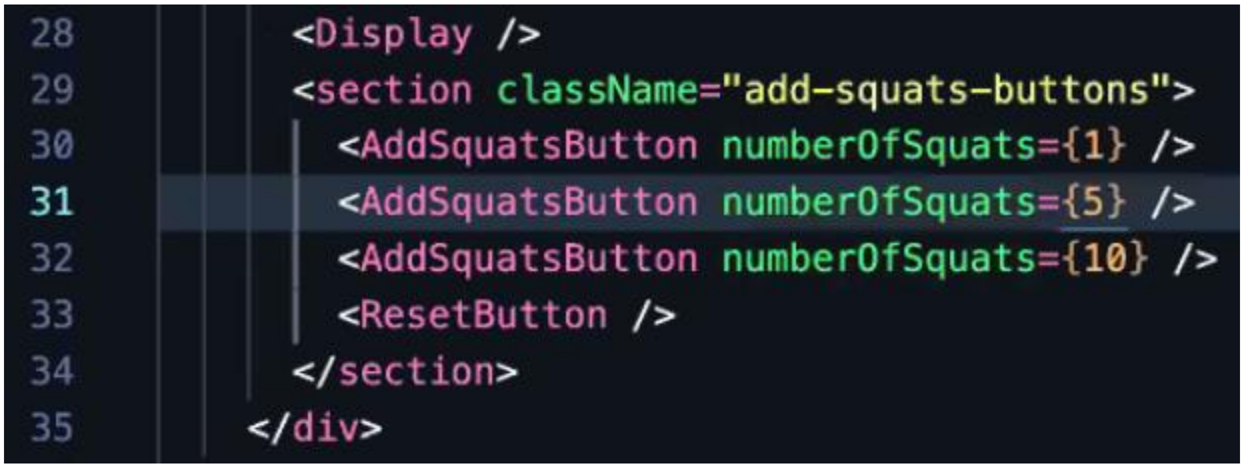
ResetButton also goes in there (doesn't need props):



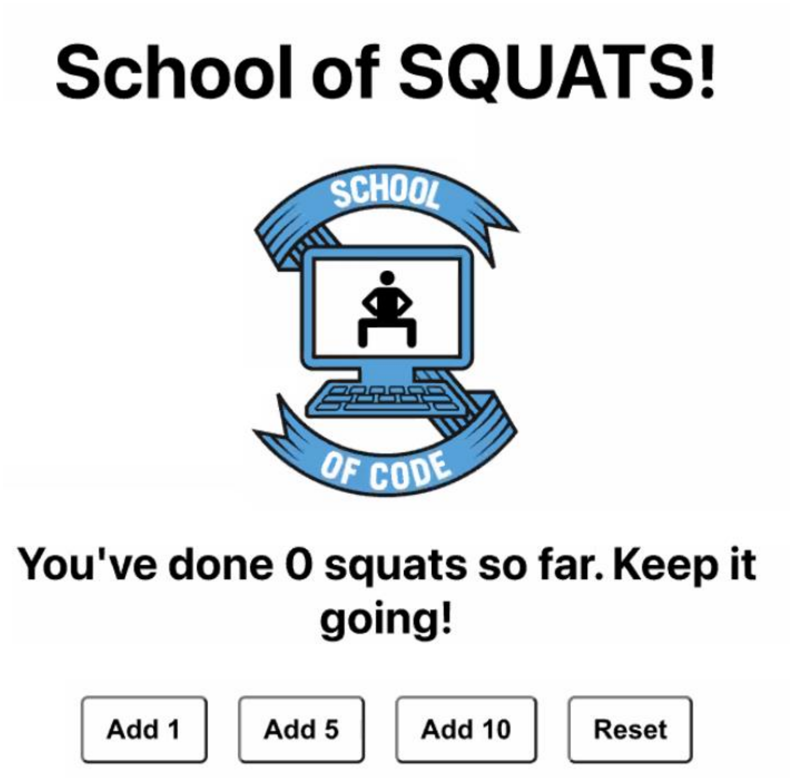
```
<section className="add-squats-buttons">
  <AddSquatsButton numberOfSquats={1} />
  <AddSquatsButton numberOfSquats={5} />
  <AddSquatsButton numberOfSquats={10} />
  <ResetButton />
</section>
```



Display component goes just above buttons section (line 28):



```
28        <Display />
29        <section className="add-squats-buttons">
30          <AddSquatsButton numberOfSquats={1} />
31          <AddSquatsButton numberOfSquats={5} />
32          <AddSquatsButton numberOfSquats={10} />
33          <ResetButton />
34        </section>
35      </div>
```

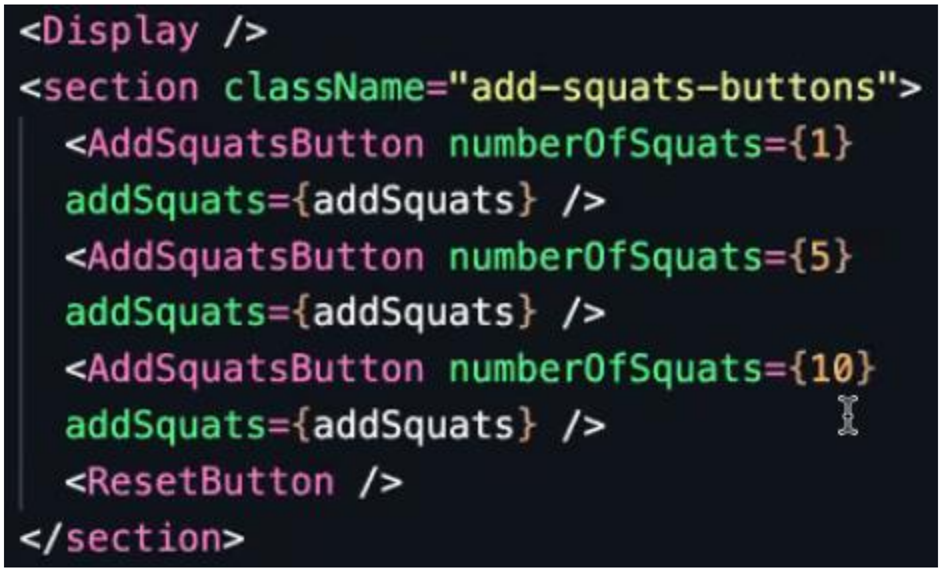

School of SQUATS!

You've done 0 squats so far. Keep it going!

No functionality yet.

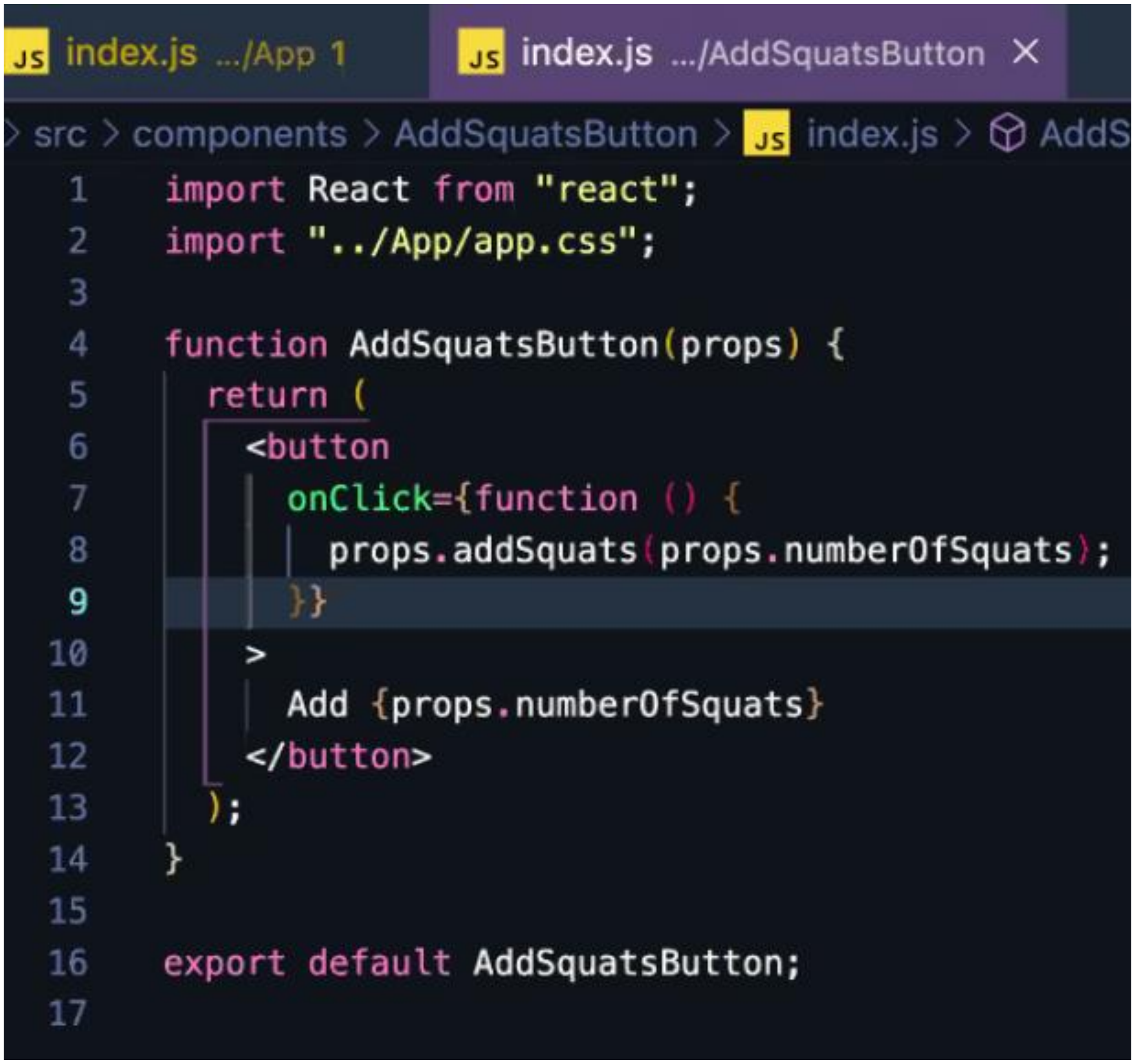
Add some functionality:

Left hand side doesn't have to be called addSquats, can be called anything, but Liz went for something semantic.



```
<Display />
<section className="add-squats-buttons">
  <AddSquatsButton numberOfSquats={1}
  addSquats={addSquats} />
  <AddSquatsButton numberOfSquats={5}
  addSquats={addSquats} />
  <AddSquatsButton numberOfSquats={10}
  addSquats={addSquats} />
  <ResetButton />
</section>
```

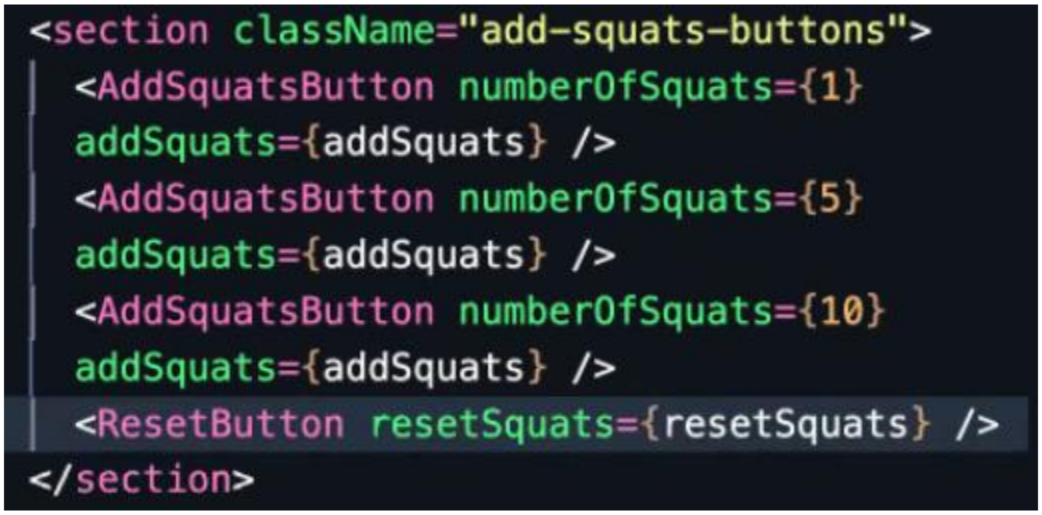
Inside the individual components, refactor to pass in props (no more object destructuring) and then accessing them by key name (remember to use anonymous function so it doesn't get called upon page load!). This updates all the "Add" buttons together:


index.js .../App 1
index.js .../AddSquatsButton
src > components > AddSquatsButton > index.js > AddS

```
import React from "react";
import "../App/app.css";

function AddSquatsButton(props) {
  return (
    <button
      onClick={function () {
        props.addSquats(props.numberOfSquats);
      }}
    >
      Add {props.numberOfSquats}
    </button>
  );
}

export default AddSquatsButton;
```

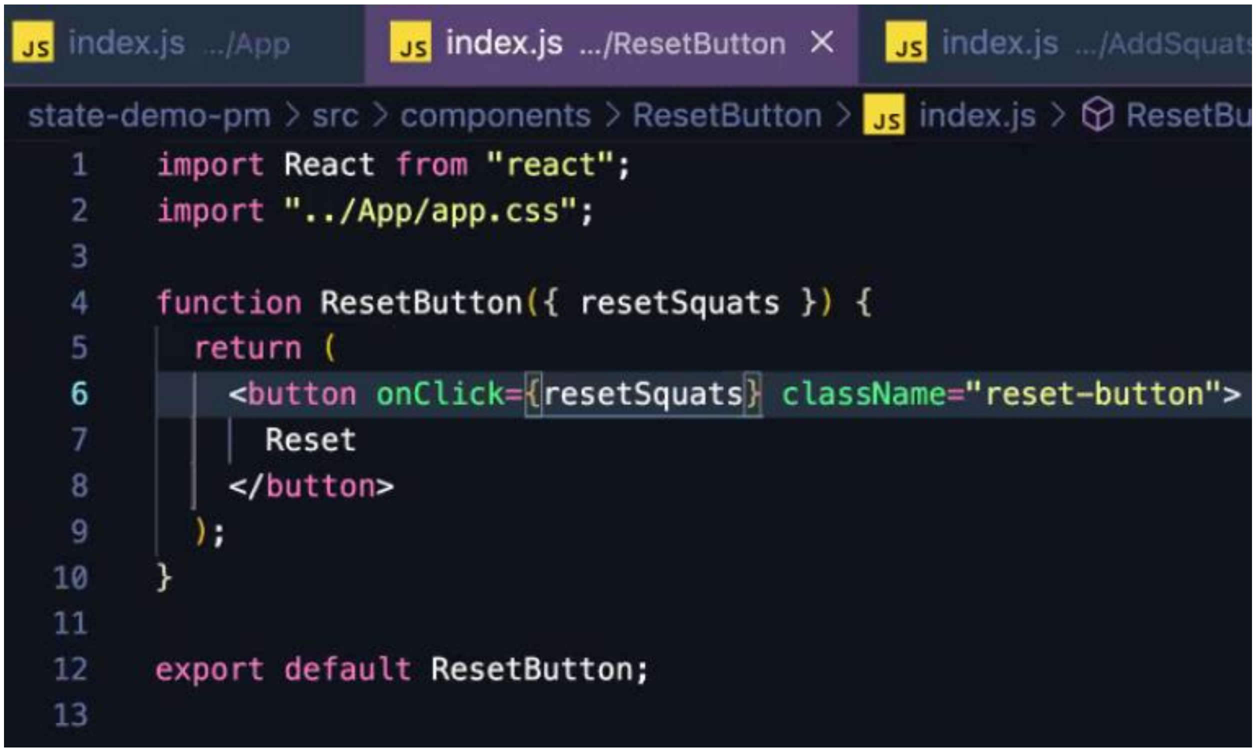
Hook up reset button in the same way:

Inside App.js:



```
<section className="add-squats-buttons">
  <AddSquatsButton numberOfSquats={1}
  addSquats={addSquats} />
  <AddSquatsButton numberOfSquats={5}
  addSquats={addSquats} />
  <AddSquatsButton numberOfSquats={10}
  addSquats={addSquats} />
  <ResetButton resetSquats={resetSquats} />
</section>
```

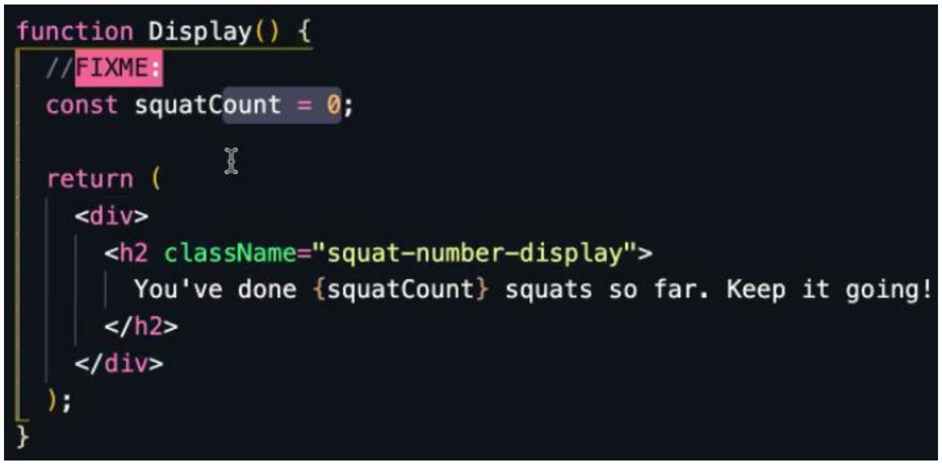
Inside ResetButton component:


index.js .../App
index.js .../ResetButton
index.js .../AddSquat
state-demo-pm > src > components > ResetButton > index.js > ResetBu

```
import React from "react";
import "../App/app.css";

function ResetButton({ resetSquats }) {
  return (
    <button onClick={resetSquats} className="reset-button">
      Reset
    </button>
  );
}

export default ResetButton;

```

Finally hooking up the display:

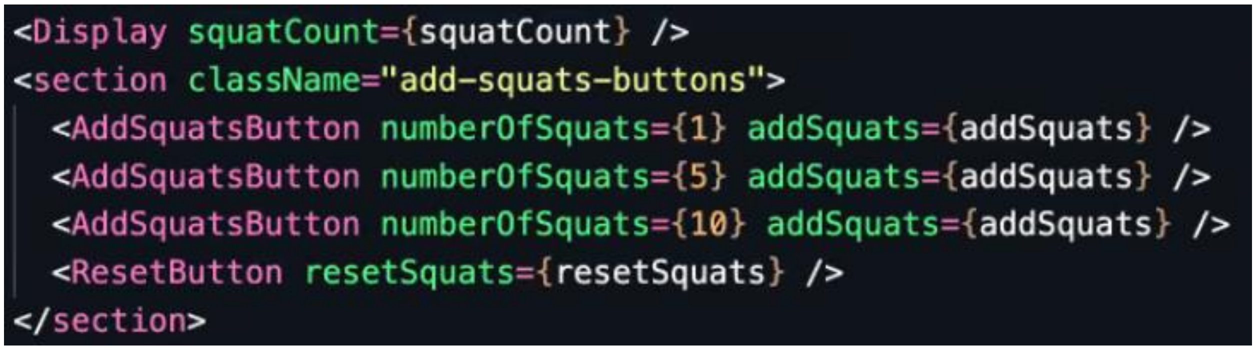
squatCount originally hardcoded to 0 always



```
function Display() {
  //FIXME:
  const squatCount = 0;

  return (
    <div>
      <h2 className="squat-number-display">
        You've done {squatCount} squats so far. Keep it going!
      </h2>
    </div>
  );
}
```

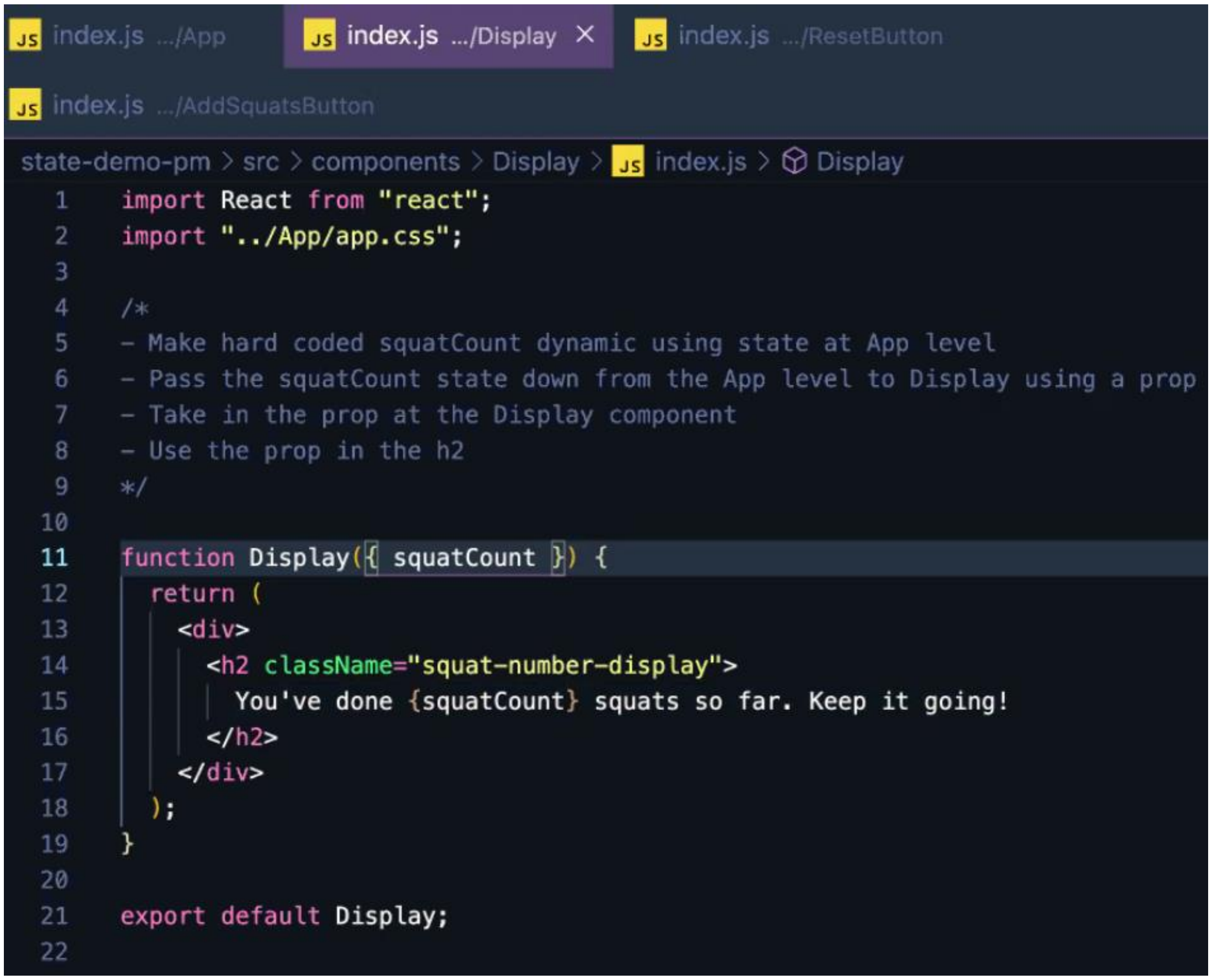
Changing it to a dynamic number based on what is being passed in via props:

In App.js, hand in squatCount prop to Display:



```
<Display squatCount={squatCount} />
<section className="add-squats-buttons">
  <AddSquatsButton numberOfSquats={1} addSquats={addSquats} />
  <AddSquatsButton numberOfSquats={5} addSquats={addSquats} />
  <AddSquatsButton numberOfSquats={10} addSquats={addSquats} />
  <ResetButton resetSquats={resetSquats} />
</section>
```

Inside display component:


index.js .../App | index.js .../Display | index.js .../ResetButton | index.js .../AddSquatsButton

state-demo-pm > src > components > Display > index.js > Display

```javascript
import React from "react";
import "../App/app.css";

/*
- Make hard coded squatCount dynamic using state at App level
- Pass the squatCount state down from the App level to Display using a prop
- Take in the prop at the Display component
- Use the prop in the h2
*/

function Display({ squatCount }) {
  return (
    <div>
      <h2 className="squat-number-display">
        You've done {squatCount} squats so far. Keep it going!
      </h2>
    </div>
  );
}

export default Display;
```

Functionality now sorted!