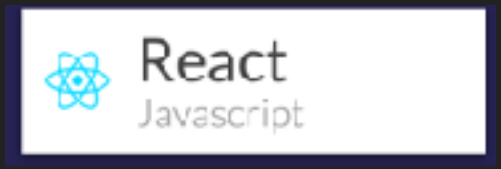
WORKSHOP #1

# THE COUNTER

## STACKBLITZ.COM

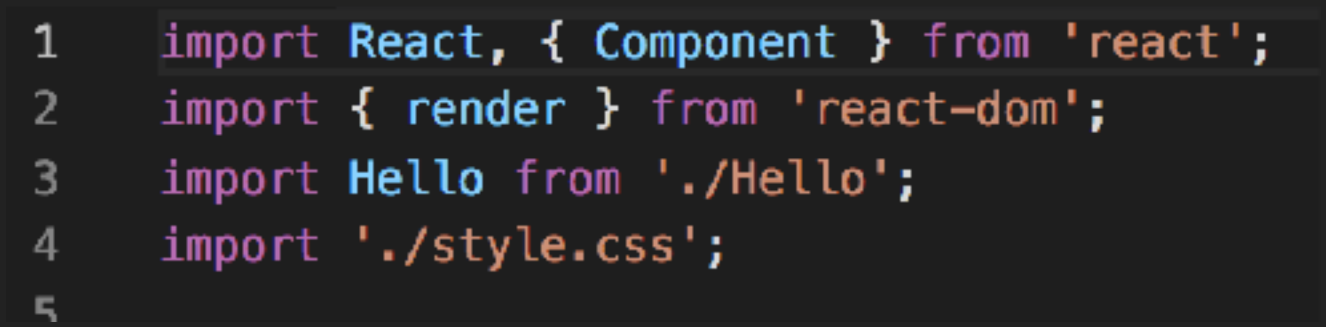
- Lets get started by navigating to stackblitz.com
- Stackblitz is a online editor that allows you to create React applications without running a IDE on your local machine.
- Click on the  button to start a new project. You should find this as you scroll down the main page.

## HELLO WORLD

- Now we have to honor the tradition of first saying “Hello World”
- Lets inspect the boilerplate that Stackblitz gives us.

1. Click on index.js in your file tree on the left of the page. This is the main file you will be programming your application in for now.
2. Lets take a line by line look at the code.

## HELLO WORLD



```
1    import React, { Component } from 'react';
2    import { render } from 'react-dom';
3    import Hello from './Hello';
4    import './style.css';
5
```

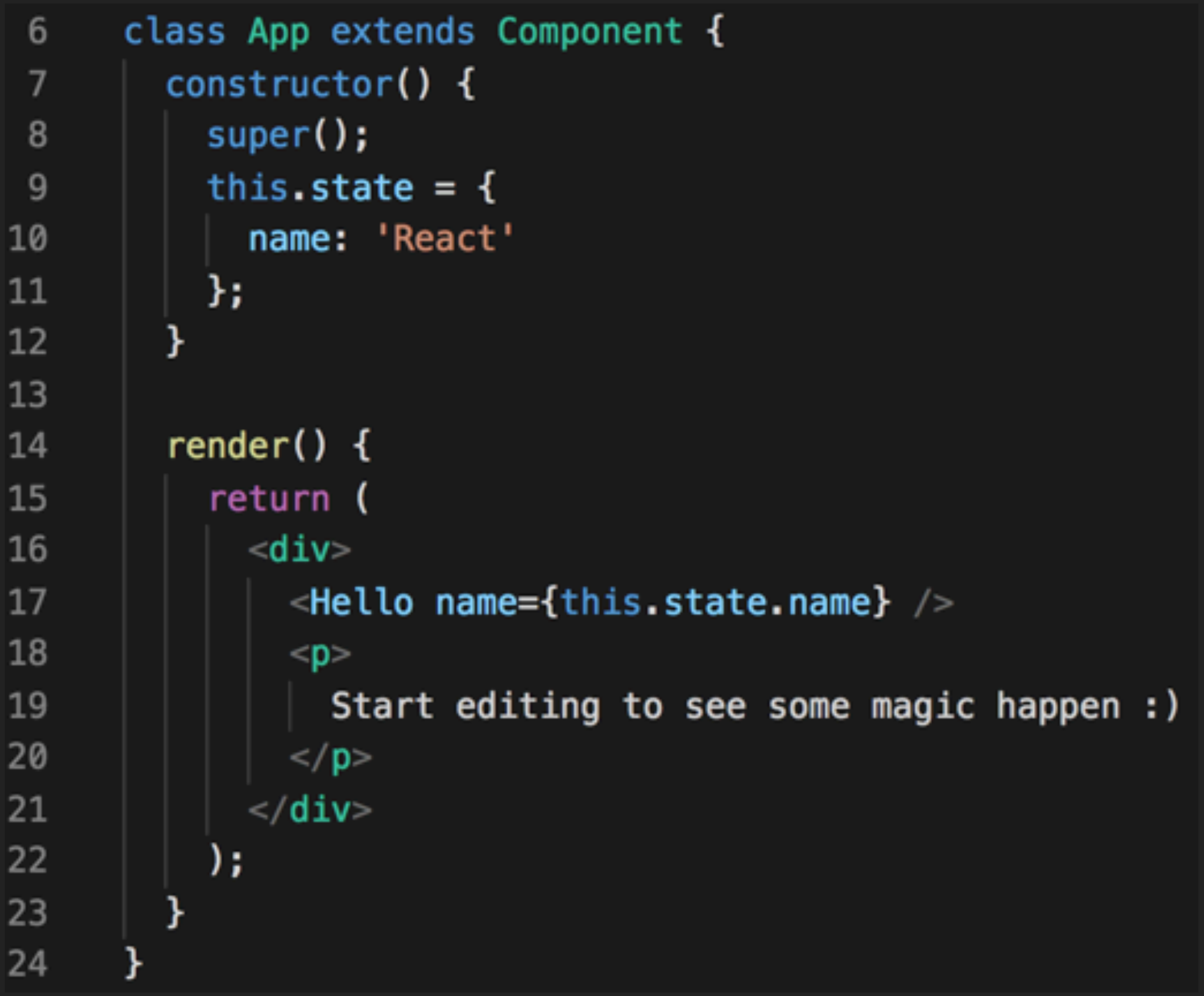
Line 1: We import React to let the interpreter know that this Javascript file is using the React library.

Line 2: We import and deconstruct the render method out from the react-dom library so we can use it to render our view.

Line 3: We import the Hello functional component.  This allows us to call

Line 4: We import our style sheet that is on the same directory level to be able to apply styles to our JSX.

## HELLO WORLD



```
 6  class App extends Component {
 7    constructor() {
 8      super();
 9      this.state = {
10        name: 'React'
11      };
12    }
13
14    render() {
15      return (
16        <div>
17          <Hello name={this.state.name} />
18          <p>
19            Start editing to see some magic happen :)
20          </p>
21        </div>
22      );
23    }
24  }
```

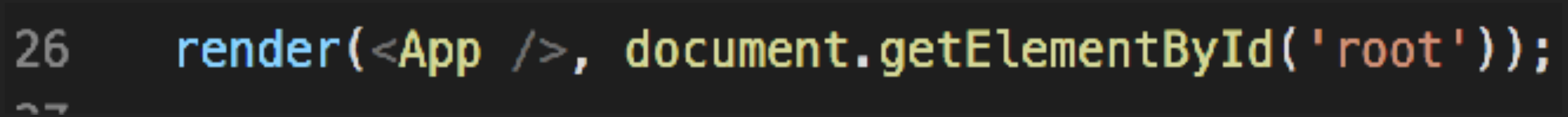
Line 6: This is where we define our class component by giving it a name “App” in this instance, as well as were we declare that it extends the Component Library.

Line 7-12:  We define our constructor and set the initial data or state for our component to use.  All I will say about Super on Line 8, is that by calling that method we then are able to set the “this” context and are able to use the “this” keyword.

Line 14-24: The render method is called, and in this method we want to return the JSX that we wish to be rendered to the DOM.

## HELLO WORLD

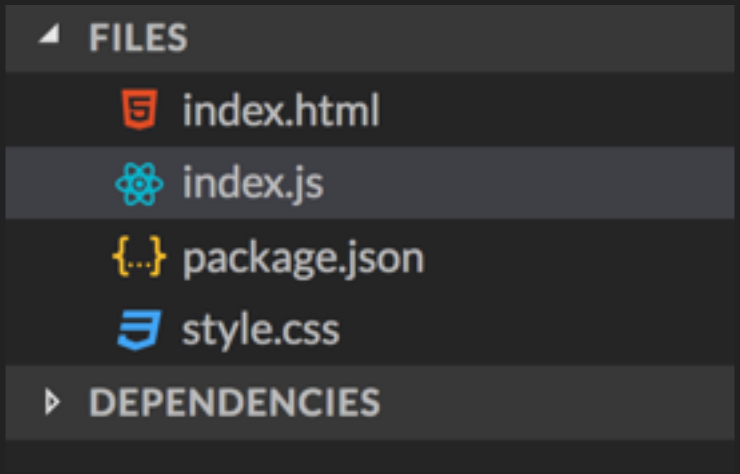
Line 26: Lastly we use the render method, and use this to render the App component to the root element of the DOM



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

## COUNTER APPLICATION

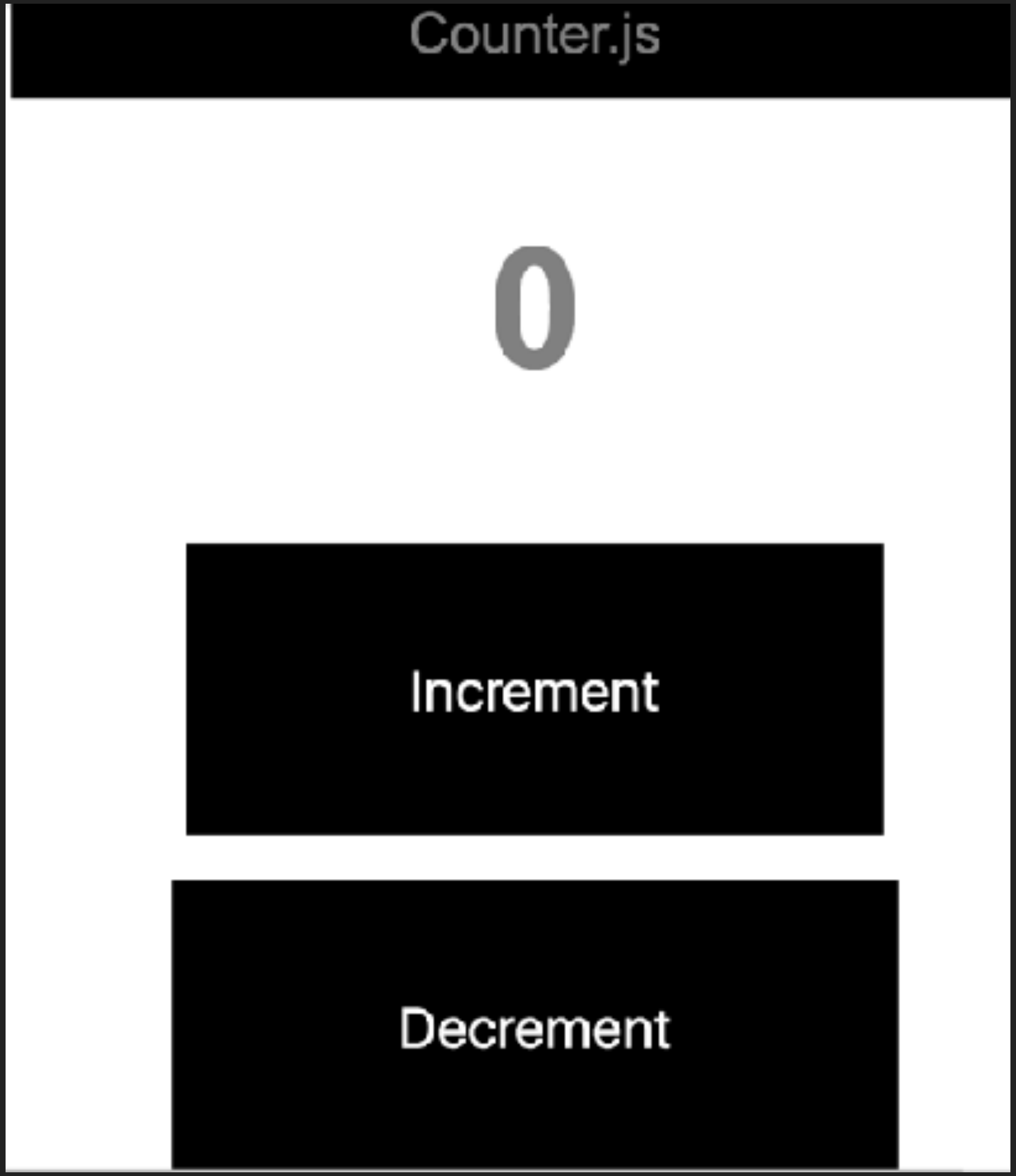
- First off… lets take a moment and say goodbye to our HelloWorld code by deleting every line of code in index.js. You want to become comfortable with writing components from scratch. So what better time to start than now. Lets also delete the Hello.js file from our file list on the left side.



This is what your Files tree should look like after you delete Hello.js

## COUNTER APPLICATION

- This is what our counter application will look like if complete correctly



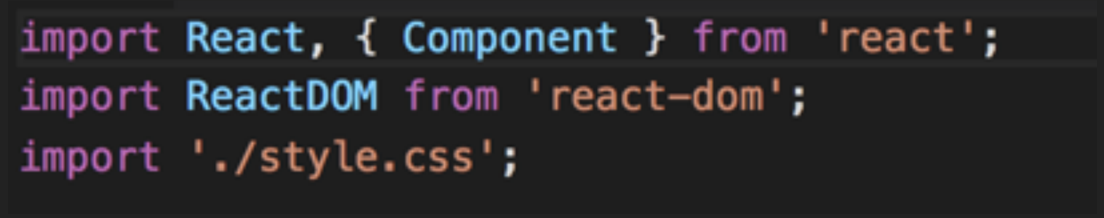
- So now that we have an idea on what we want to create. Take a moment and think about where should we start. See if you can answer these questions.
- What is the major functionality of the application?
- What type of component should we implement based on the functionality of the application?  I.E. Stateful class component, or a pure functional component?
- How will you structure the JSX and styles to accomplish this view or UX?

## COUNTER APPLICATION

- We know that this application will display a count based on the number of times a increment and decrement button is clicked.
- Our counter has to start at 0.  Usually, and almost always, when you have values that are changing in your views you are going to need State.  This should help you in making your decision between a pure functional component or a stateful class component.
- Lets not worry this round about how to style and set up this application.  I'll provide the CSS and guide you along the way on how to add in styling.

- Okay, phew… that was a lot of reading. Now lets start coding.
- Lets start by importing the proper libraries.
- Import React and the Component so we can create our stateful class component and use JSX. Then we want to import ReactDOM from the react-dom library so we can attach our component to the DOM.

Lines 1-3

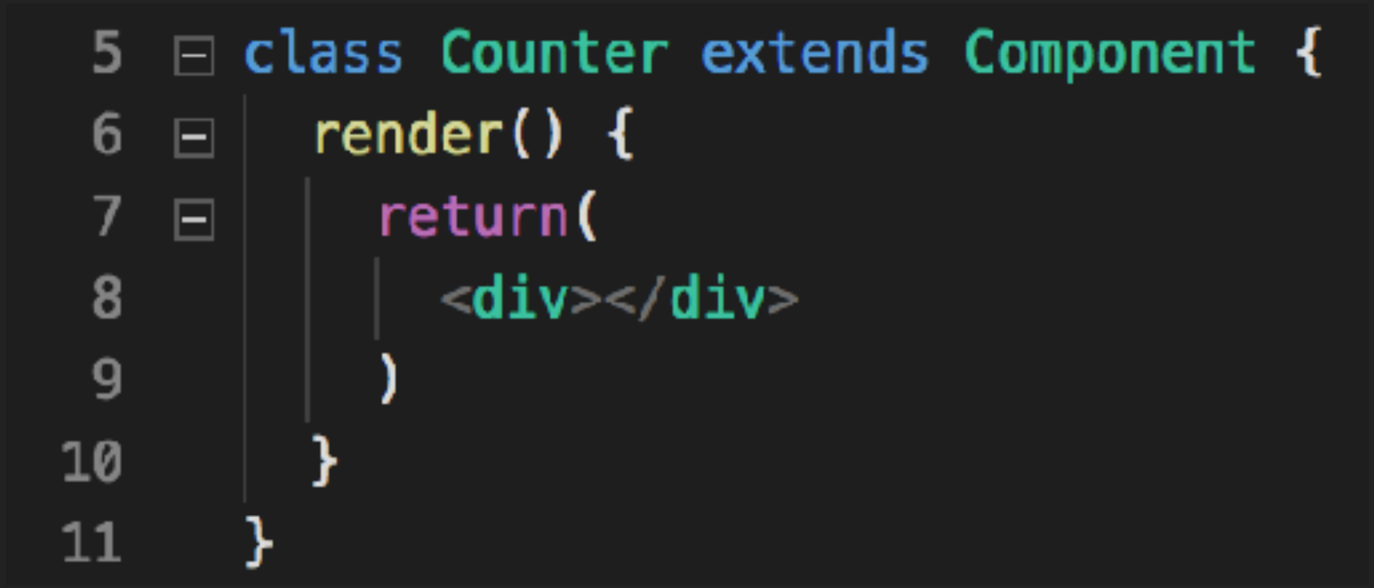


```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
import './style.css';
```

Importing style.css file allows us to use that file to add styling to our JSX

## COUNTER APPLICATION

- Now that we have our imports complete we need to create our first stateful class component.



```
class Counter extends Component {
  render() {
    return(
      <div></div>
    )
  }
}
```

Note that all of the JSX describing our view is being returned by a method on the class called render. This is a special method that all class components must have.

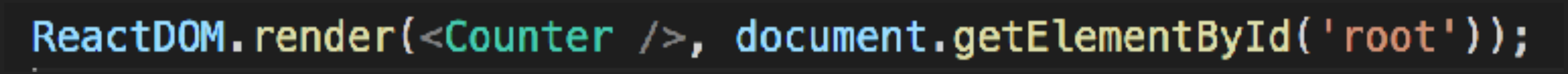
Tip:

When naming our class component make sure to use names that are clear and declarative.  Like classes in Javascript, make sure to capitalize the first letter of the class name.

## COUNTER APPLICATION

- Render you Counter component to the DOM by declaring it as the first argument in your ReactDOM.render. Note that we write the component as a self-closing JSX tag.

This should be outside of our Counter components code block.

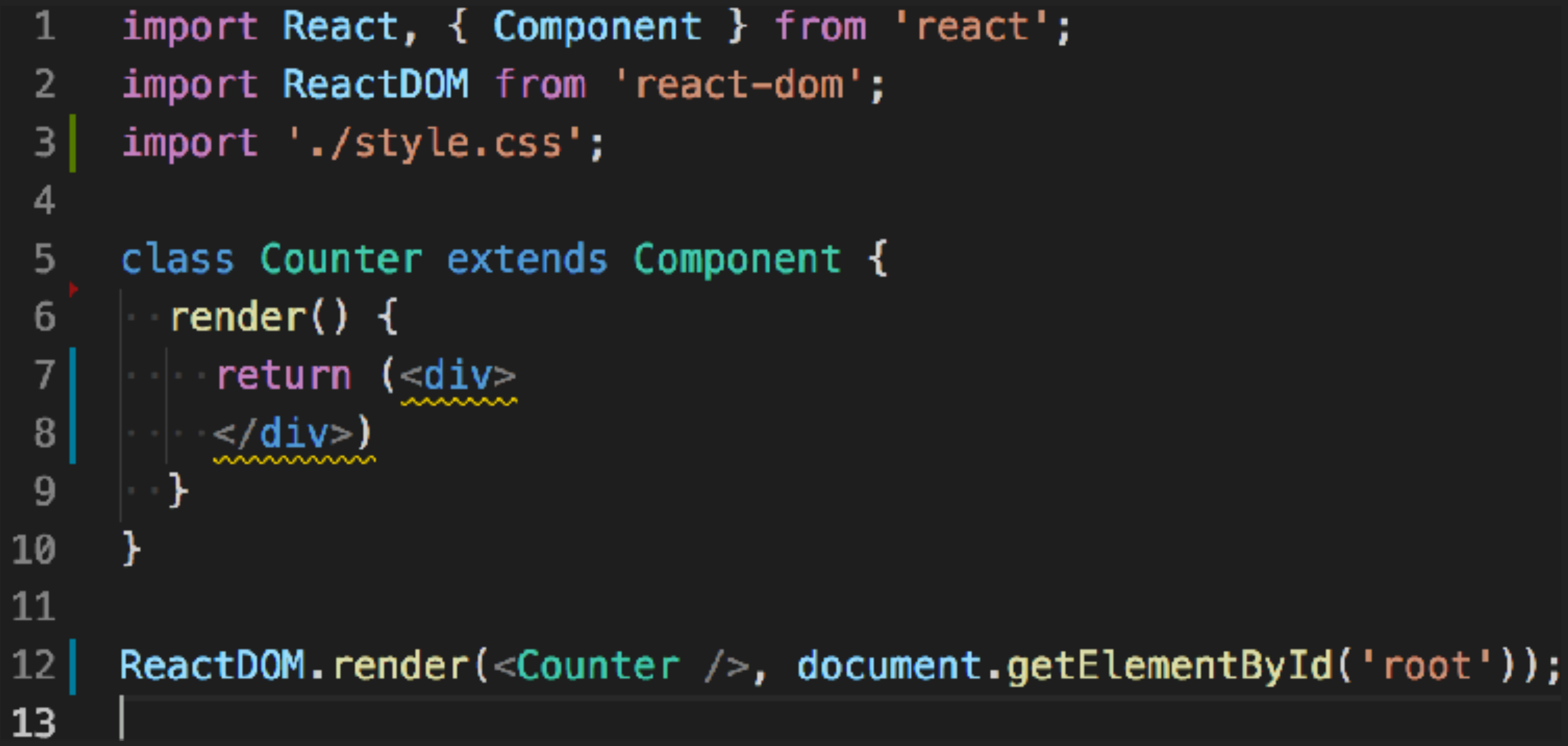


```
ReactDOM.render(<Counter />, document.getElementById('root'));
```

The first argument for the render method is the component you wish to be attached to the DOM, and the second argument is what HTML you want to inject to a specific DOM node. So for our example we want to inject this component to the HTML DOM node that has an id of 'root'

## COUNTER APPLICATION

- At this point your index.js file should look exactly like this.

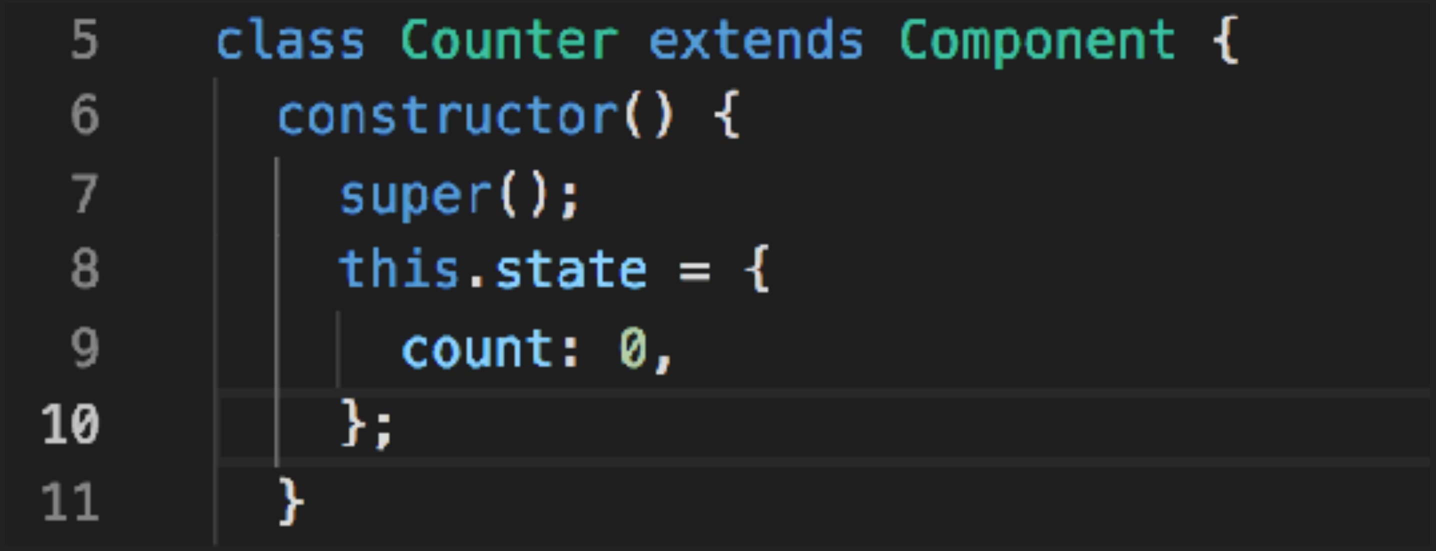


```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
import './style.css';

class Counter extends Component {
  render() {
    return (<div>
    </div>)
  }
}

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

```

- Remember, we’re building a stateful component.  Think for a moment: what should change about the view of our Counter app as user interact with it, and how could we represent that data?
- The data that changes in our UI is the number representing the count. We can represent this as a JavaScript number. Let's add state to our component, and some behavior that will cause that state to change.
- To do this we need to add a constructor to your counter class and define state on the component.  <u>Note that state is always defined as a Javascript object that have key value pairs.</u>

COUNTER APPLICATION

- Make sure to add your constructor directly under where we create the Counter class and before the render method.

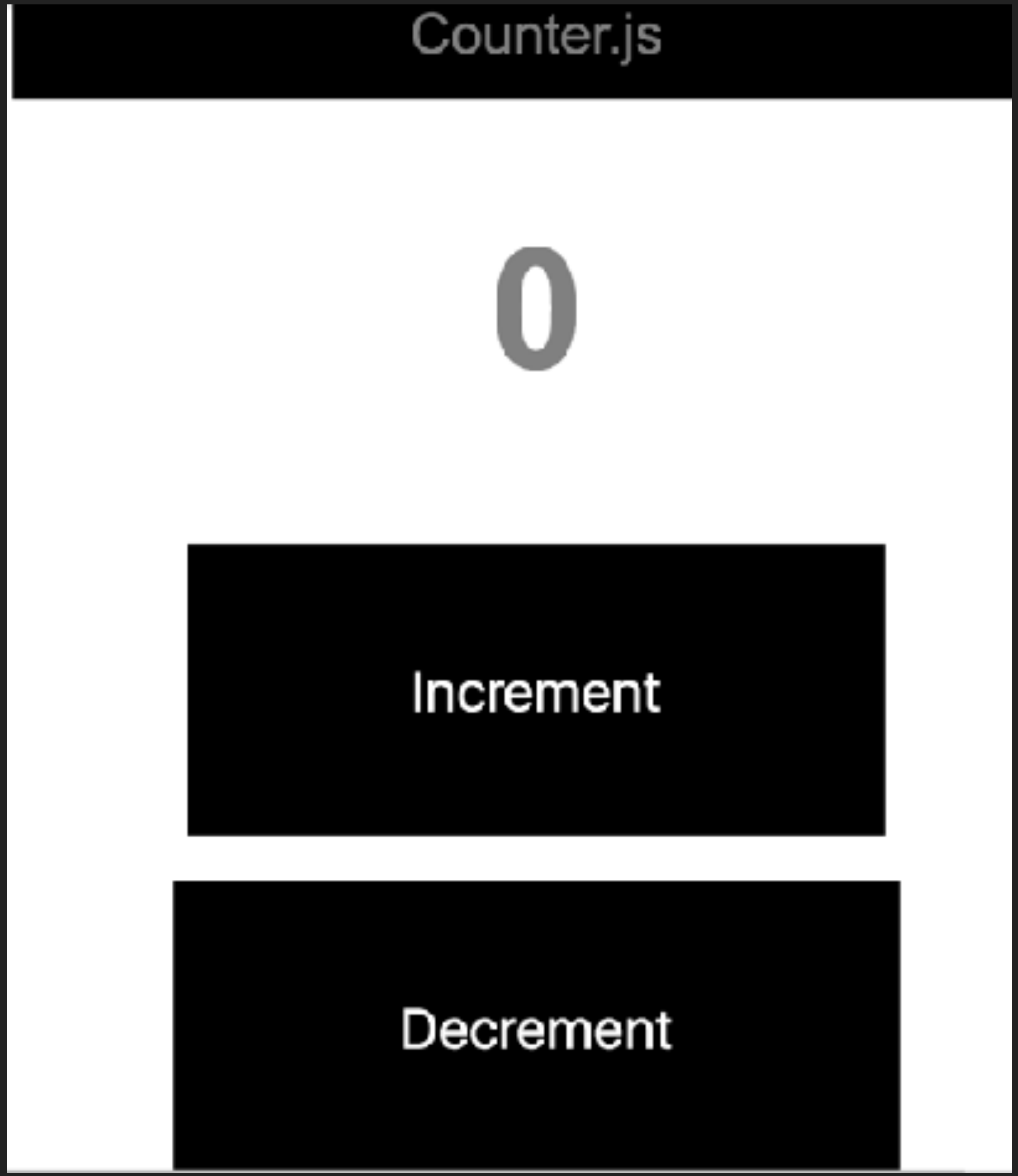


```
class Counter extends Component {
  constructor() {
    super();
    this.state = {
      count: 0,
    };
  }
```

We initialize our state by setting this.state to a Javascript object and the key and value we want our component to be initialized to.

## COUNTER APPLICATION

- Now that we have our component built, we should think about what we want our app to look like so we can start building the JSX.  Lets take a look at the end result again.



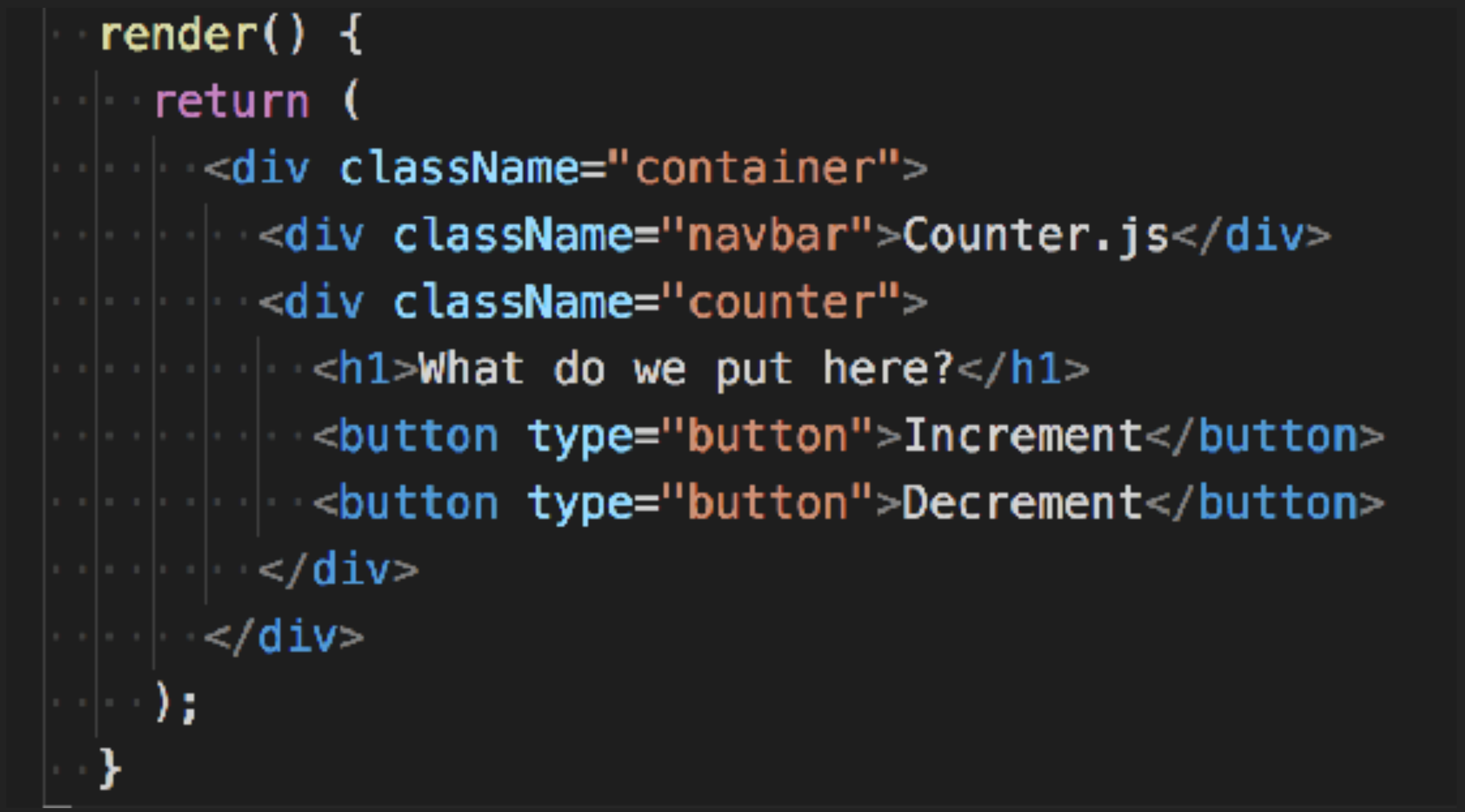
Here we will need a nav bar

Here we will want our count to be displayed

Here we will need two buttons one for increment and one for decrement.

## COUNTER APPLICATION

- To help get you started lets go ahead and build out what we know so far.

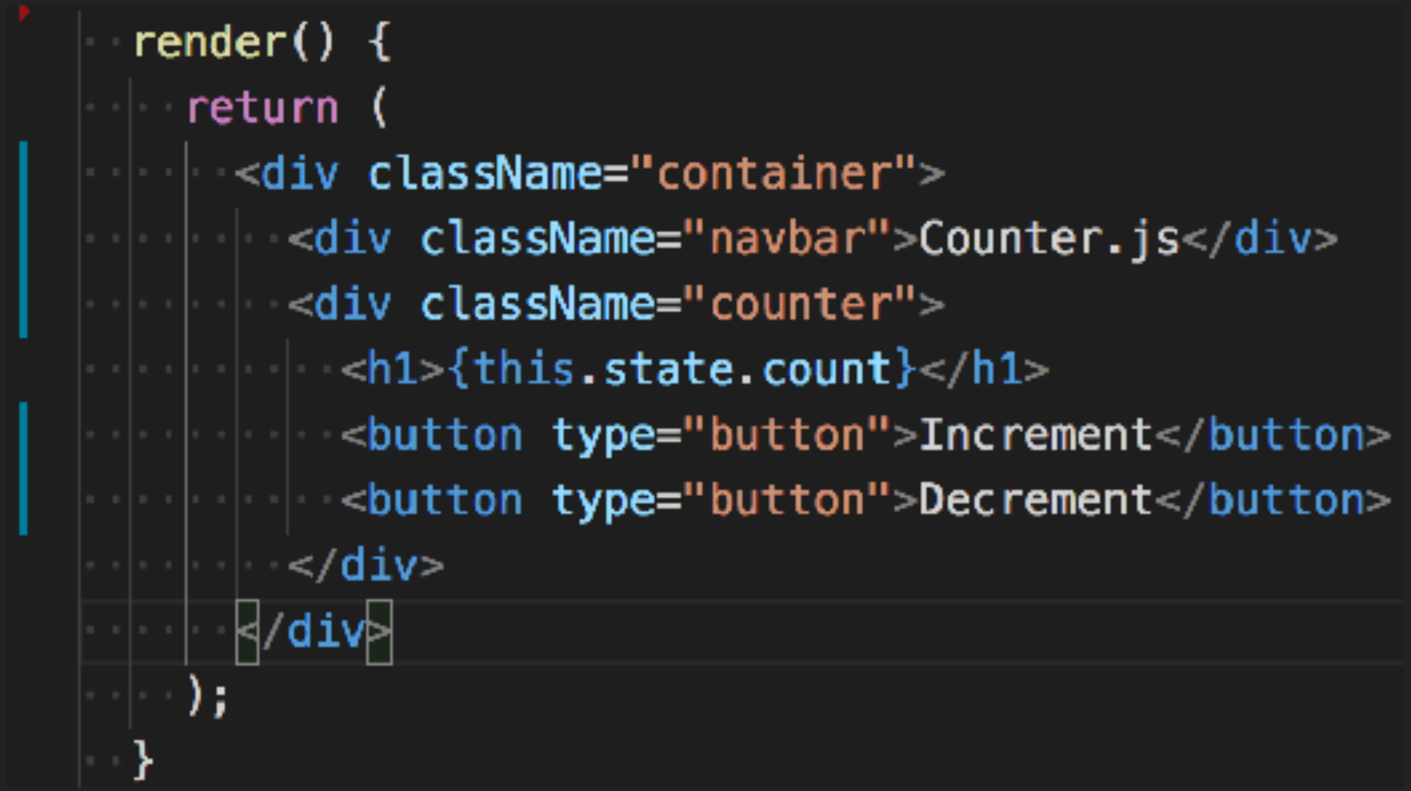


```
render() {
  return (
    <div className="container">
      <div className="navbar">Counter.js</div>
      <div className="counter">
        <h1>What do we put here?</h1>
        <button type="button">Increment</button>
        <button type="button">Decrement</button>
      </div>
    </div>
  );
}
```

In your return statements you can only have one parent JSX element.  A common pattern is to wrap everything in a div.

Note:  Look at what is between the h1 tags.
How can we represent our counters current state?

- Remember that we want our counters state to be represented in our JSX.  How do we do this?



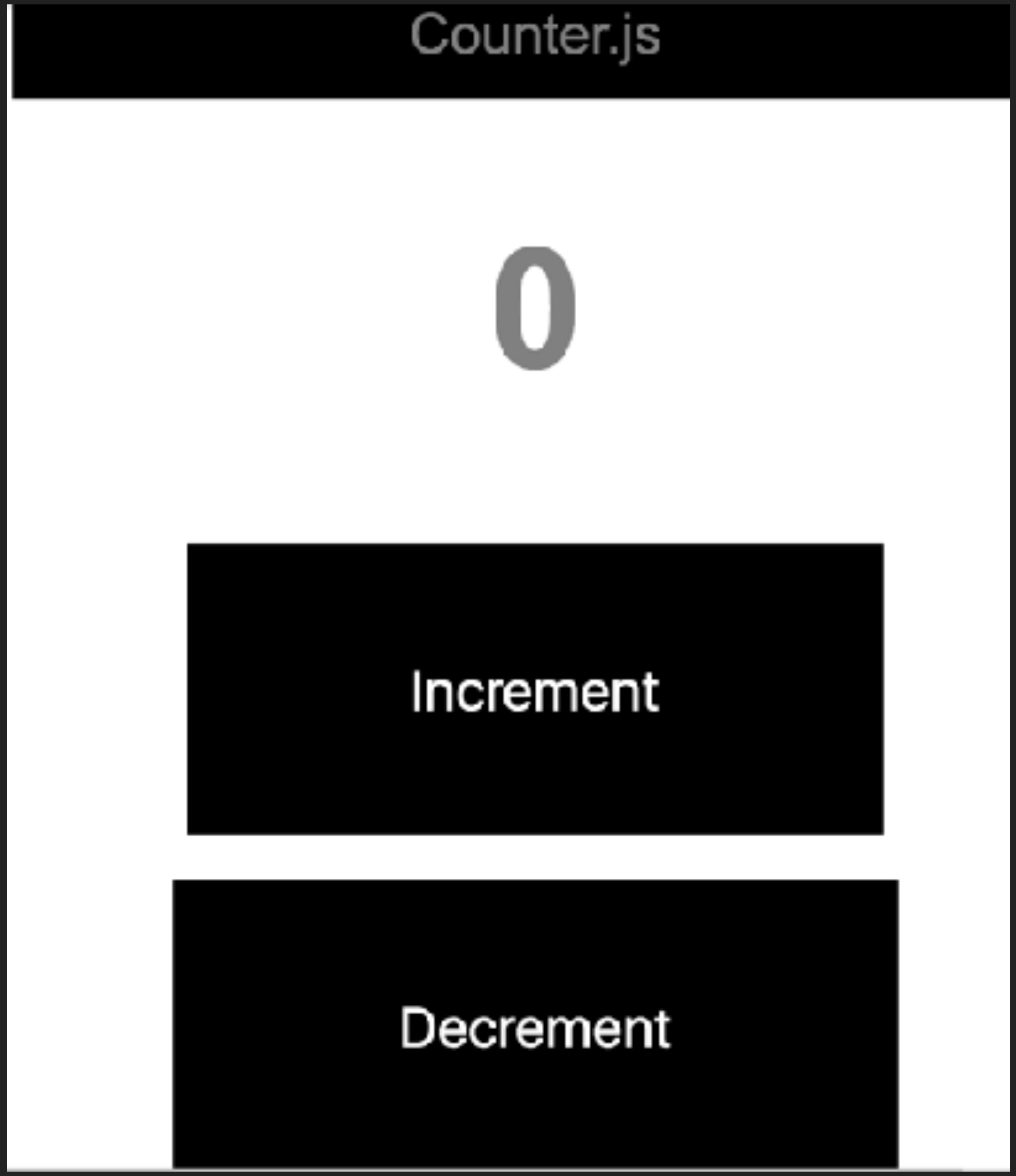
```
render() {
  return (
    <div className="container">
      <div className="navbar">Counter.js</div>
      <div className="counter">
        <h1>{this.state.count}</h1>
        <button type="button">Increment</button>
        <button type="button">Decrement</button>
      </div>
    </div>
  );
}
```

When adding javascript into our JSX.  We start by using open and closing brackets {}. This is a very powerful feature of React that allows us to add Javascript in our JSX.

Note:  Now look how we add the state in-between our h1 tags.  The count should be displayed and represent our initial value of 0.

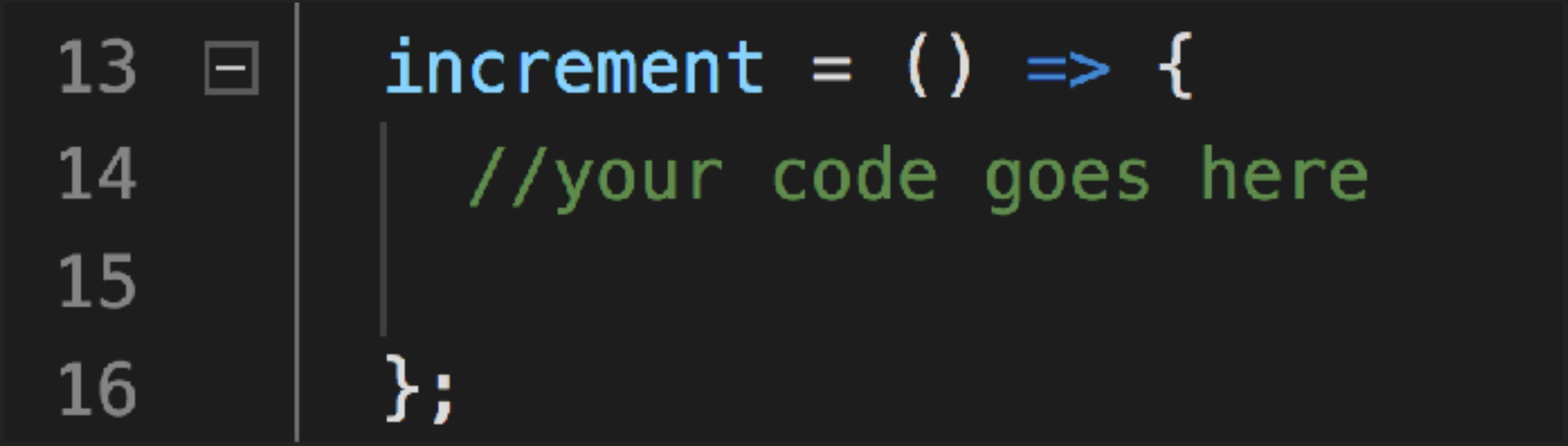
## COUNTER APPLICATION

- Now that we have some JSX built we need to add in the CSS.
- Copy in the CSS from styles.css in the workshop-1 folder



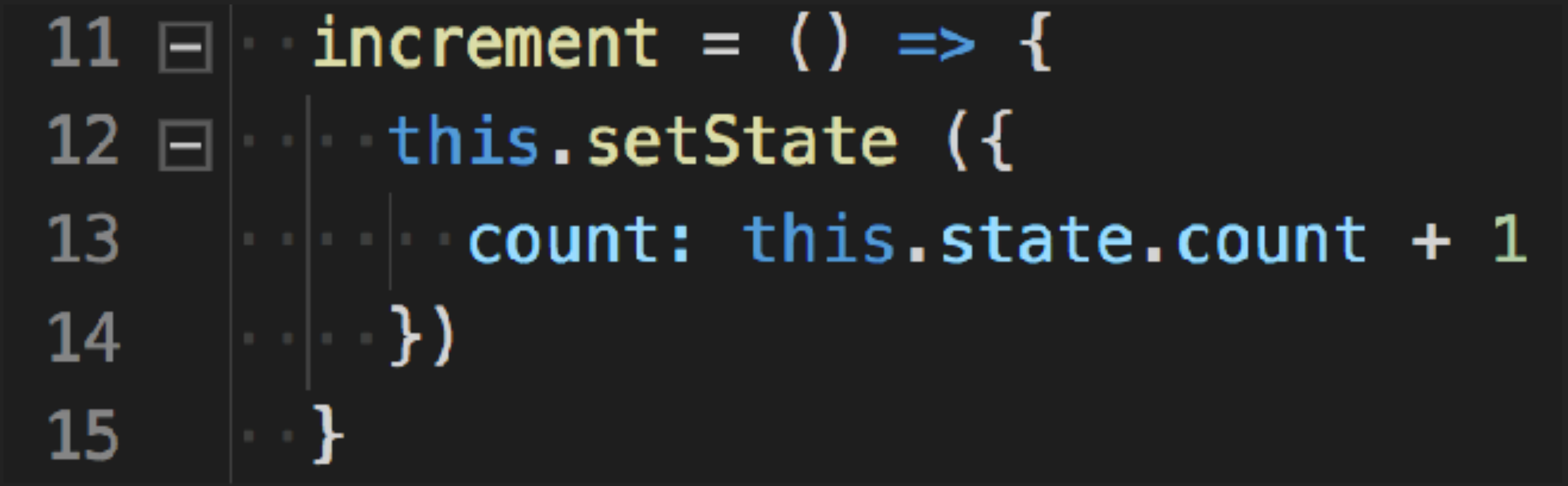
This is how your application should look.

- Now we need to increment the counter by clicking on the increment button.
- To do this we need to first make an increment method. Methods that will be added to our class component we will put below our constructors code block and before the render method.



```
increment = () => {
  //your code goes here

};
```

- Remember how we change our state? We accomplish this by calling the setState() method when we click the increment button.

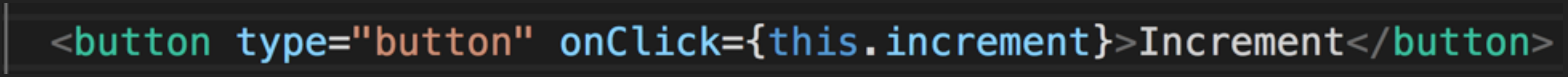


```
  increment = () => {
    this.setState ({
      count: this.state.count + 1
    })
  }
```

## COUNTER APPLICATION

- Now that we have our increment method.  How do we make sure that every time the increment button is clicked that this method will be evoked?

- We do this by adding an onClick listener to the increment button.  When this button is clicked the increment method will be invoked, and the counter will display the new count.



```
<button type="button" onClick={this.increment}>Increment</button>
```

## COUNTER APPLICATION

- Now, with your newfound knowledge of React, and how to create methods. Implement the decrement method on your own. Also, make sure that our counter cannot go below 0 and above 20.
- Create a button that when clicked will clear the counter back to 0.
- Make a button that will toggle between increments of 1 or 2 at a time, and will display “Double Increments” or “Single Increments” in the button name.

- If you’ve made it this far you’re doing great!
- Here are some materials you should look at before our next workshop.
  https://reactjs.org/docs/components-and-props.html
  https://reactjs.org/tutorial/tutorial.html