Cubstart Lecture 9 [optional!]

React.js

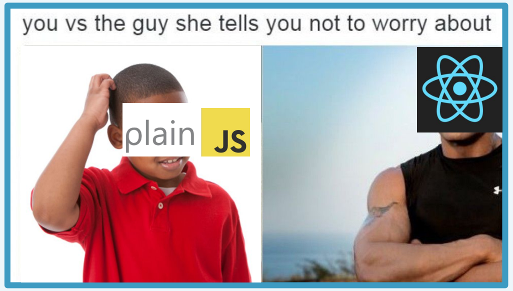
Administrivia

- Final Project Checkpoint 1 due date EXTENDED to SUNDAY NOV 12th
- HW 8: Create your own API (last required hw) due Friday Nov 10th
- HW 9: Social Media Website is OPTIONAL
- NO LAB THIS FRIDAY because of Veteran's Day

React.js Overview

React.js

- Frontend JavaScript Framework
  (Framework = prewritten code)
- "React to Changes"
- No more page refreshing :)
- Key word: **Components**



Alternatives:

Popularity

Uber Eats

NETFLIX

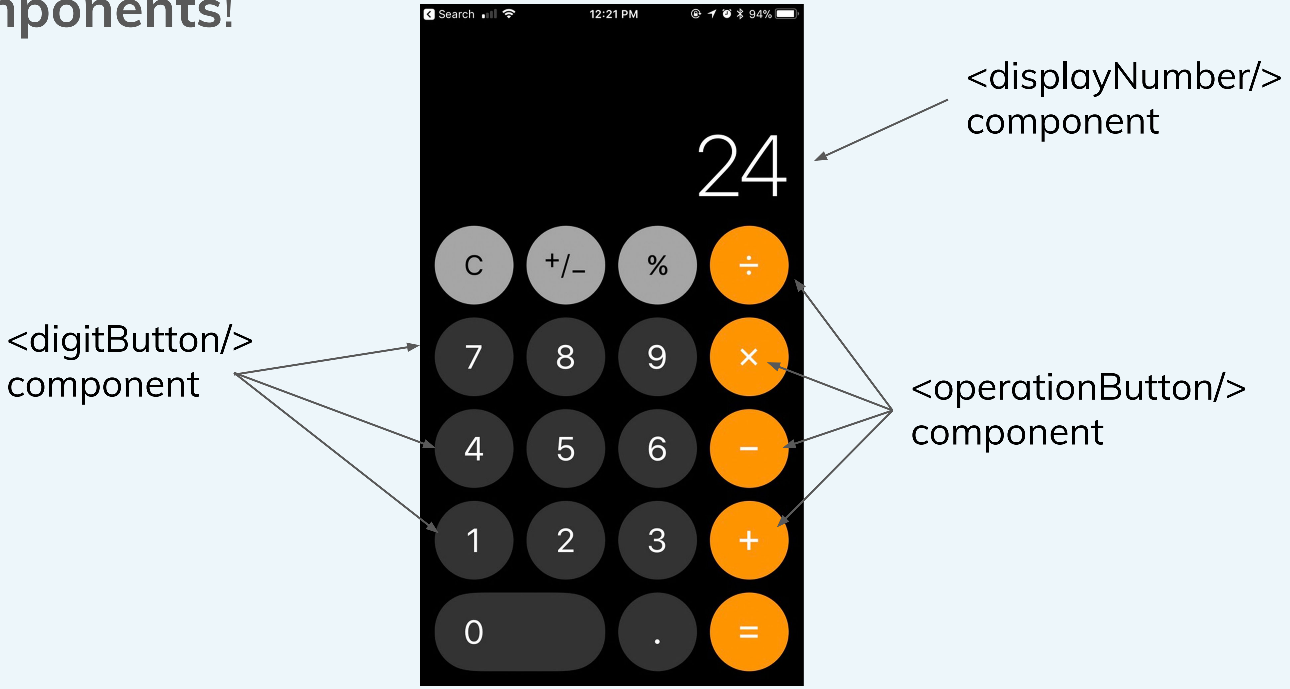
airbnb

PayPal

Khan Academy

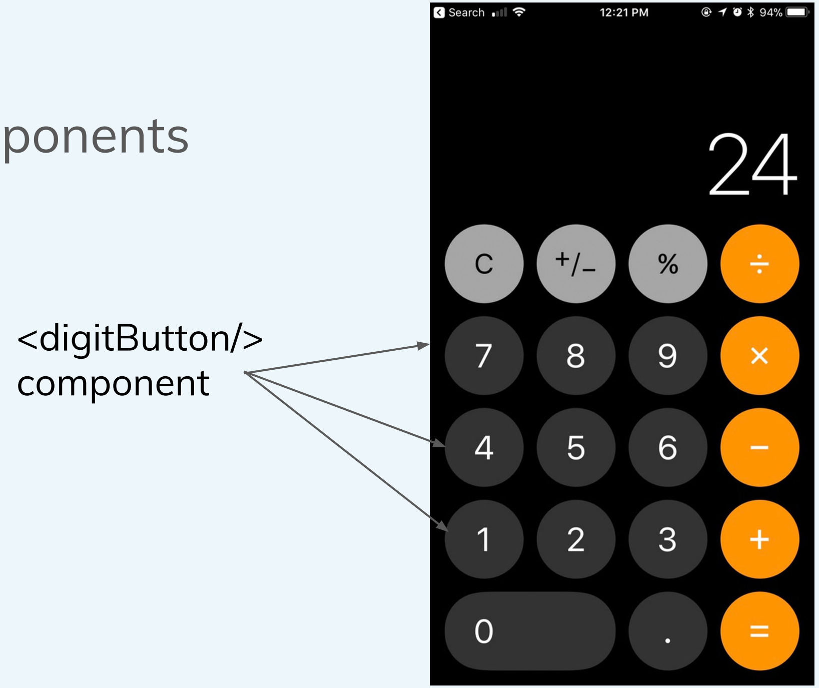
React Components

- Big idea: breaks down complex front-end user interfaces down into **components**!



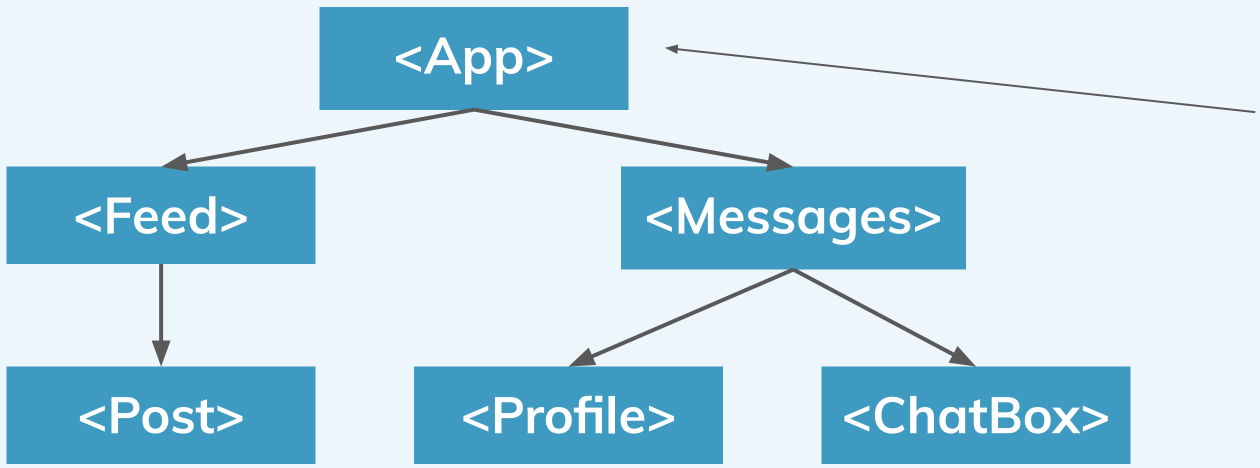
React Components

- Pieces of code that represent **a part** of your front-end web app
- Everything is made of components



Component Tree

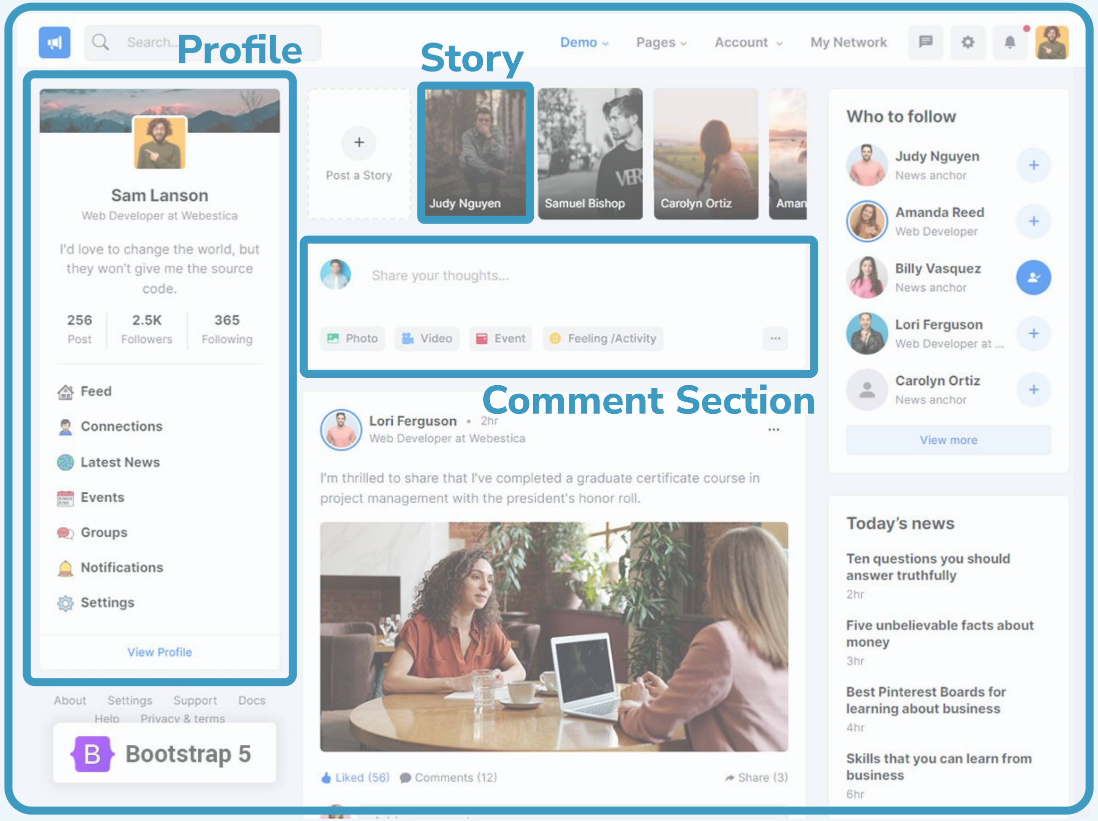
- Big Idea: EVERYTHING is a component.
- Some components just contain other components



App.js is a component too, even though it represents your entire web application and contains many other components!

Example

App



I'd love to change the world, but they won't give me the source code.

I'm thrilled to share that I've completed a graduate certificate course in project management with the president's honor roll.

Ten questions you should answer truthfully
2hr

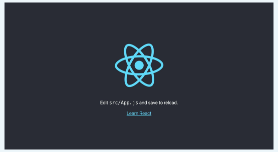
Five unbelievable facts about money
3hr

Best Pinterest Boards for learning about business
4hr

Skills that you can learn from business
6hr

Getting started with React

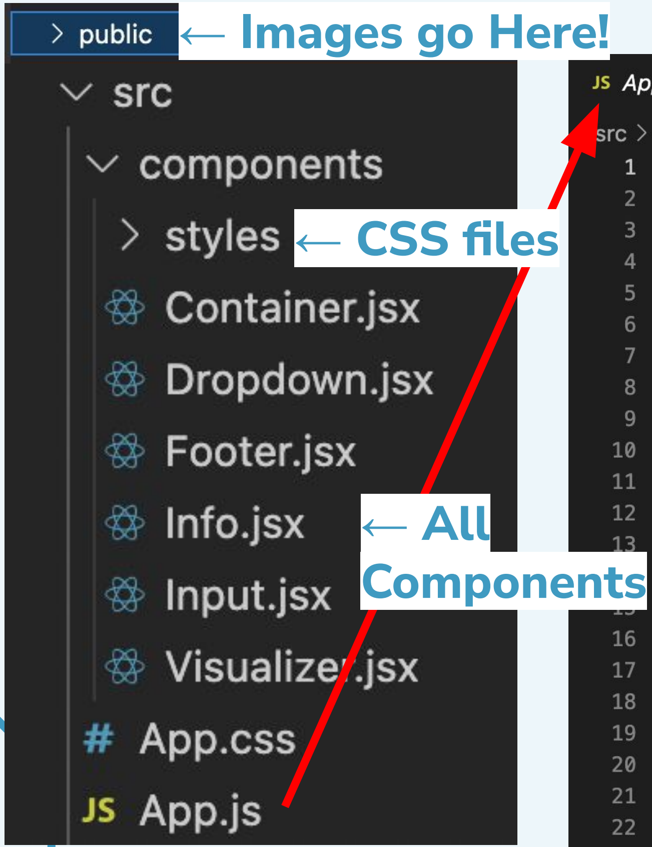
Steps

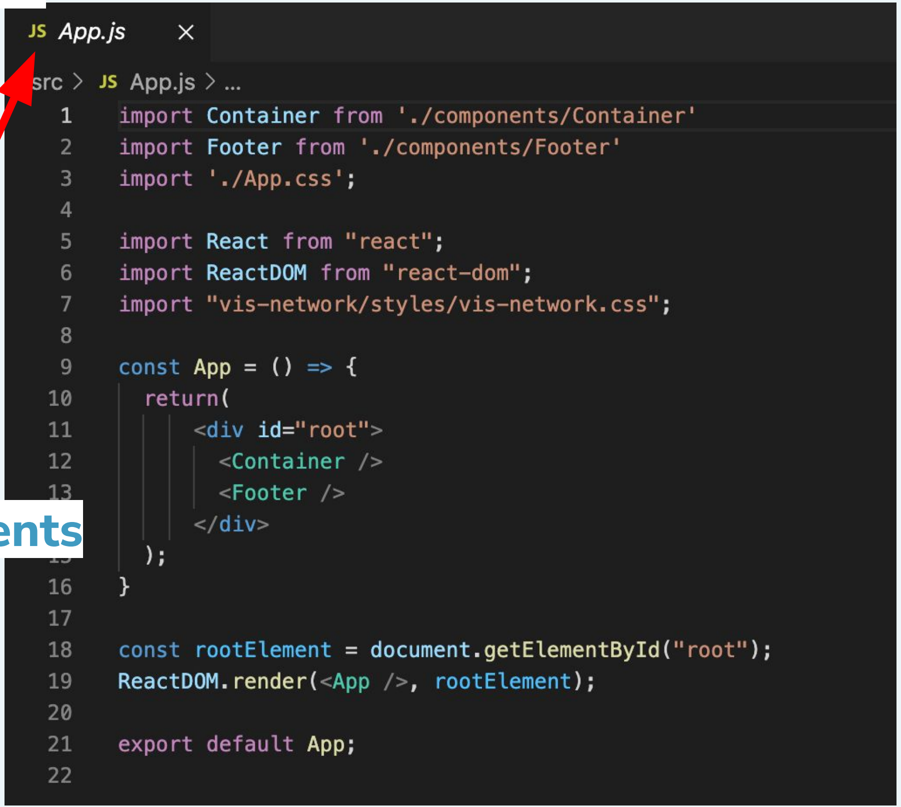
1. Open your terminal.
2. Type: “**cd Desktop**”
3. Type: “**npx create-react-app lecture9**” to create a React app
4. Type: “**cd lecture9**”
5. Type: “**npm start**” to run your React app
6. This will open “**localhost:3000**” containing: 
Edit `src/App.js` and save to reload.

7. Change Something … You no longer have to refresh!
8. Go back to your terminal and press “**ctrl+c**” to stop running.

React Project Structure

React.js Project Structure





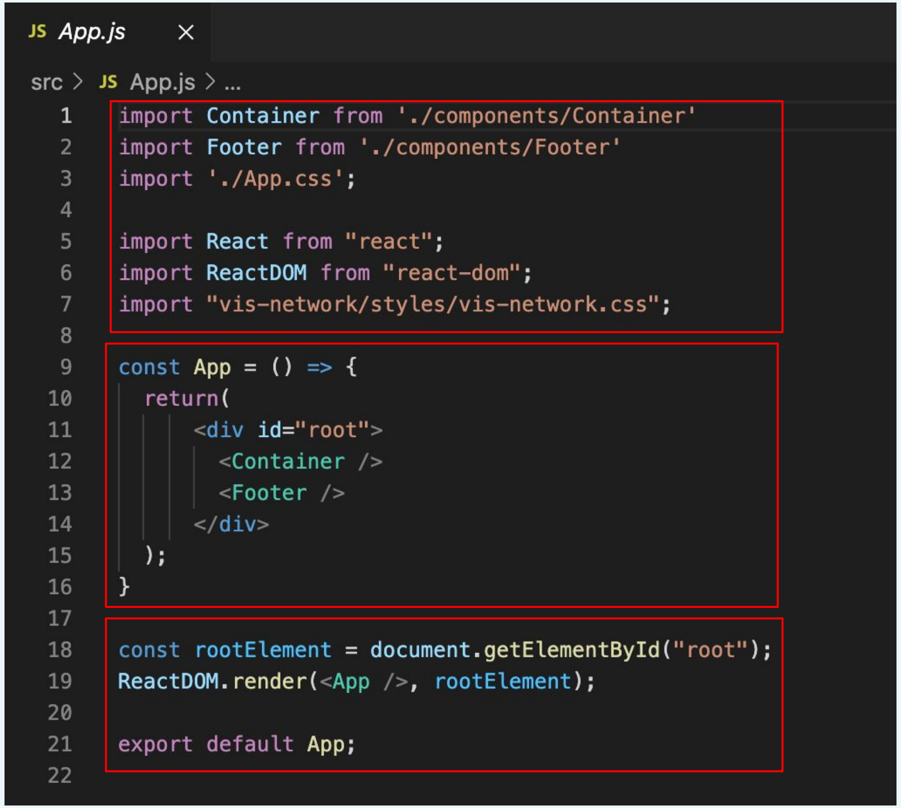
```
import Container from './components/Container'
import Footer from './components/Footer'
import './App.css';

import React from "react";
import ReactDOM from "react-dom";
import "vis-network/styles/vis-network.css";

const App = () => {
  return(
    <div id="root">
      <Container />
      <Footer />
    </div>
  );
}

const rootElement = document.getElementById("root");
ReactDOM.render(<App />, rootElement);

export default App;
```

Component Structure

1. Imports (Your components + 3rd Party Libraries)

2. Your Component Function

3. Exporting For Use Outside

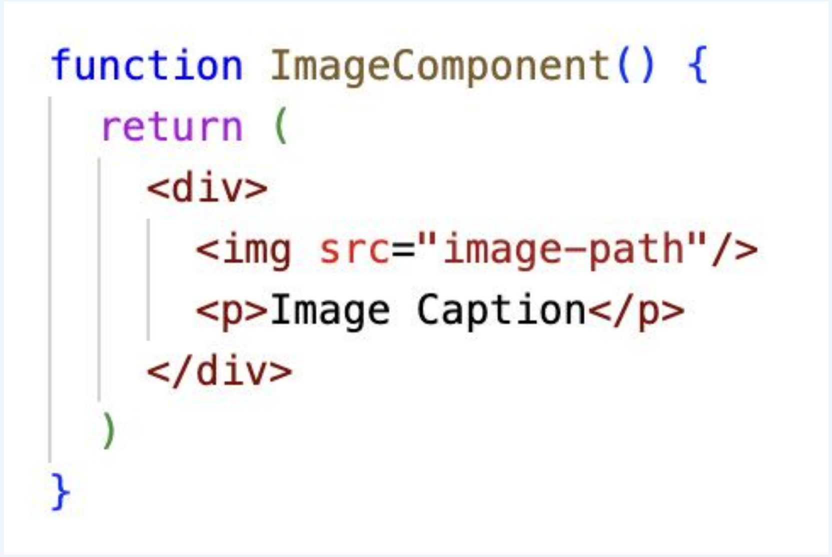


```javascript
import Container from './components/Container'
import Footer from './components/Footer'
import './App.css';

import React from "react";
import ReactDOM from "react-dom";
import "vis-network/styles/vis-network.css";

const App = () => {
  return(
      <div id="root">
        <Container />
        <Footer />
      </div>
  );
}

const rootElement = document.getElementById("root");
ReactDOM.render(<App />, rootElement);

export default App;
```

JSX

JSX

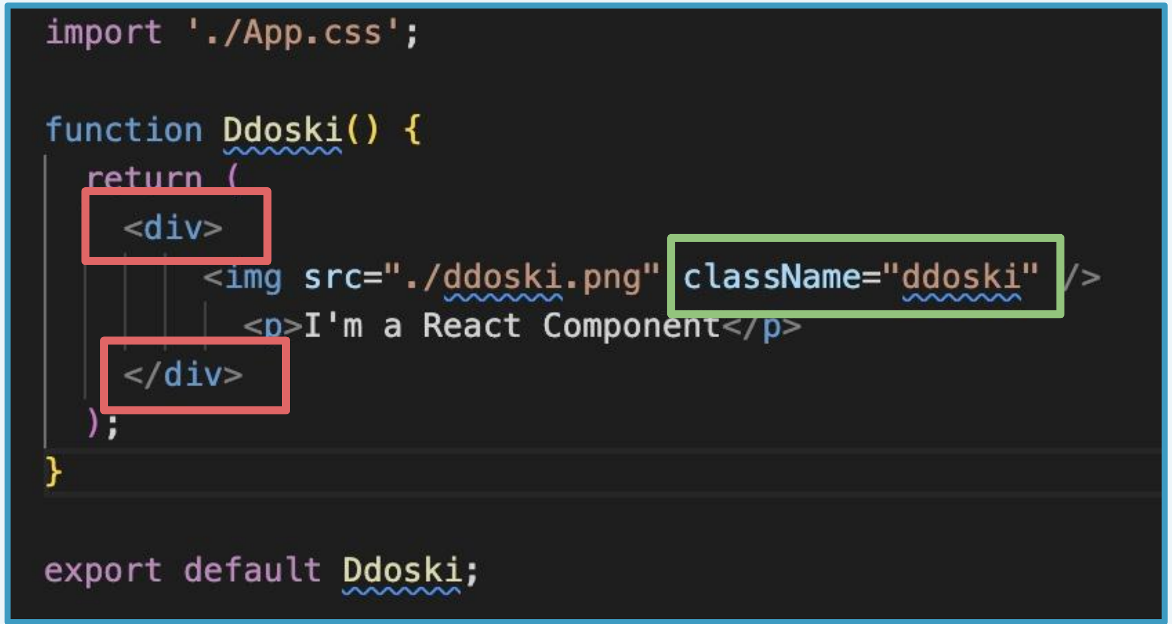
- Syntactic extension for Javascript
- JSX = HTML for JavaScript



```
function ImageComponent() {
  return (
    <div>
      <img src="image-path"/>
      <p>Image Caption</p>
    </div>
  )
}
```

We have written a JavaScript function that returns HTML!

JSX: Conventions



```
import './App.css';

function Ddoski() {
  return (
    <div>
        <img src="./ddoski.png" className="ddoski" />
          <p>I'm a React Component</p>
    </div>
  );
}

export default Ddoski;
```

Wrap every output in a single component

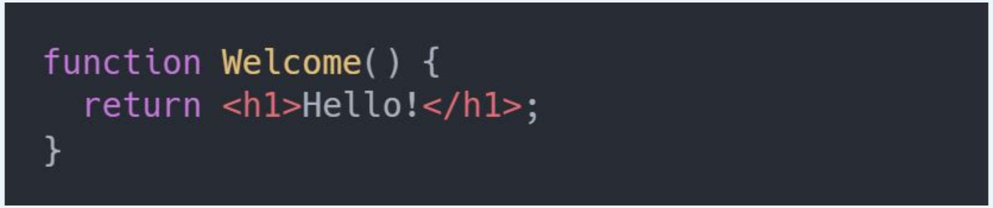
Use **className=""** instead of class=""

Components in Action

Functional Components

Think of a component as: “A Function That Returns HTML”

In component file:



```
function Welcome() {
  return <h1>Hello!</h1>;
}
```

Props

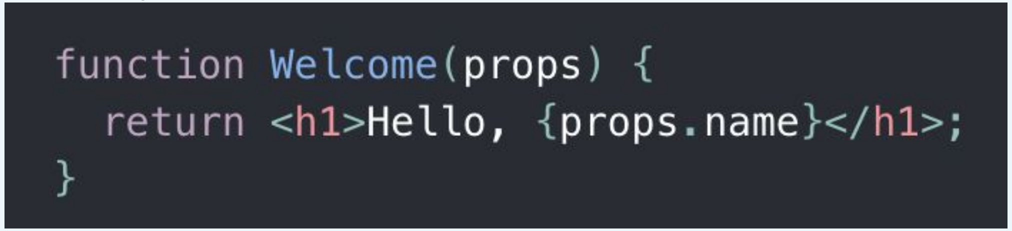
Props

- Stands for "properties" of a components
- Arguments passed into components

Why use props?

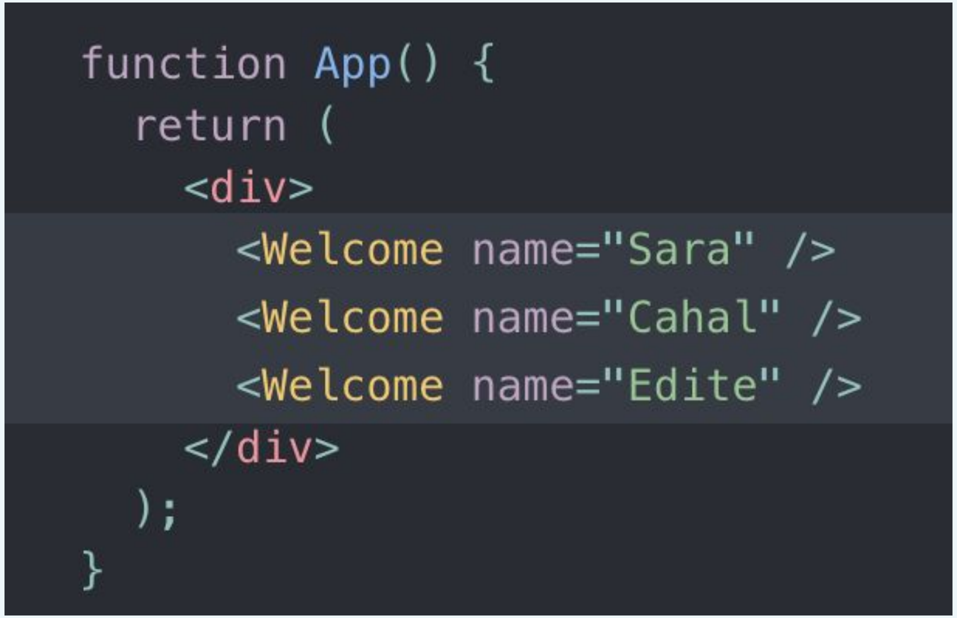
- Allows you to reuse the same component but customize it!

In component file:



```jsx
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

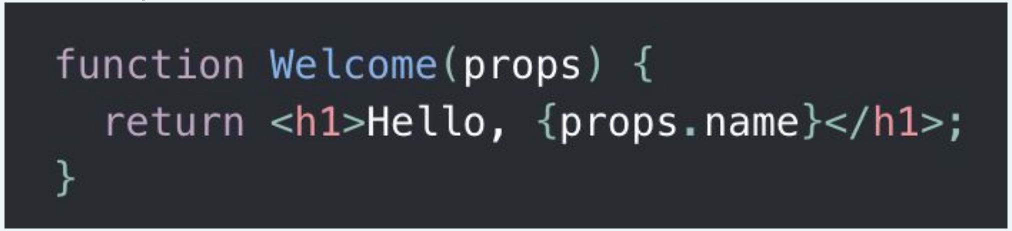
In App.jsx:



```jsx
function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}
```

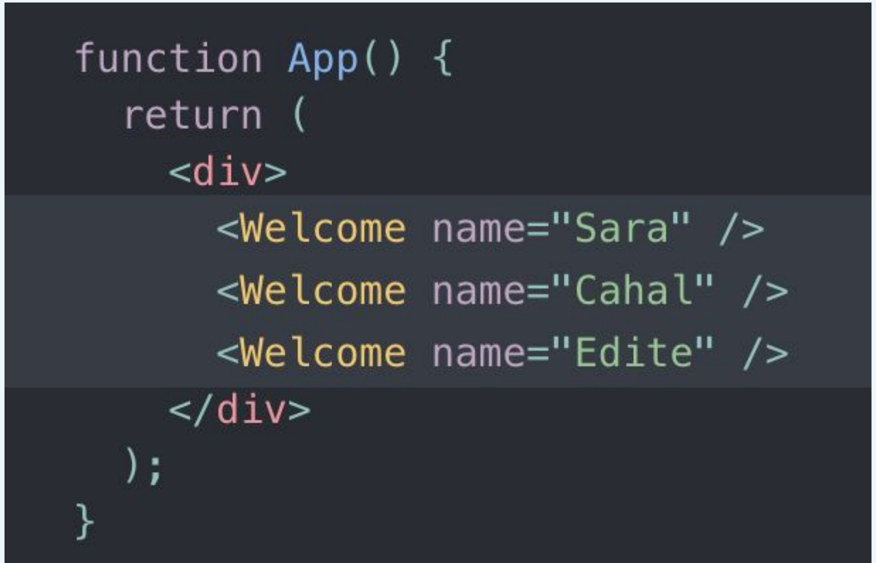
Rendering Components

In component file:



```jsx
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

In App.jsx:

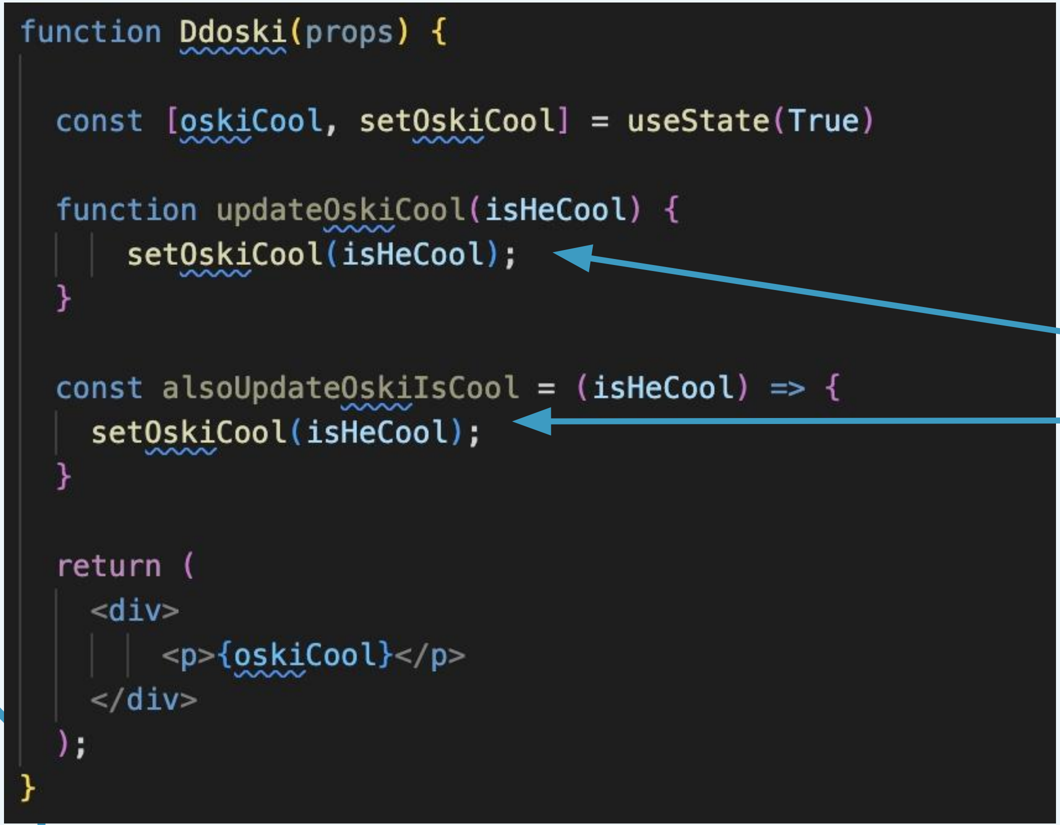


```jsx
function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}
```

In our App component, we render the **Welcome** component three times with three different names!

Methods and Event Handlers

Where do we put functions/methods?



```
function Ddoski(props) {

  const [oskiCool, setOskiCool] = useState(True)

  function updateOskiCool(isHeCool) {
      setOskiCool(isHeCool);
  }

  const alsoUpdateOskiIsCool = (isHeCool) => {
    setOskiCool(isHeCool);
  }

  return (
    <div>
        <p>{oskiCool}</p>
    </div>
  );
}
```

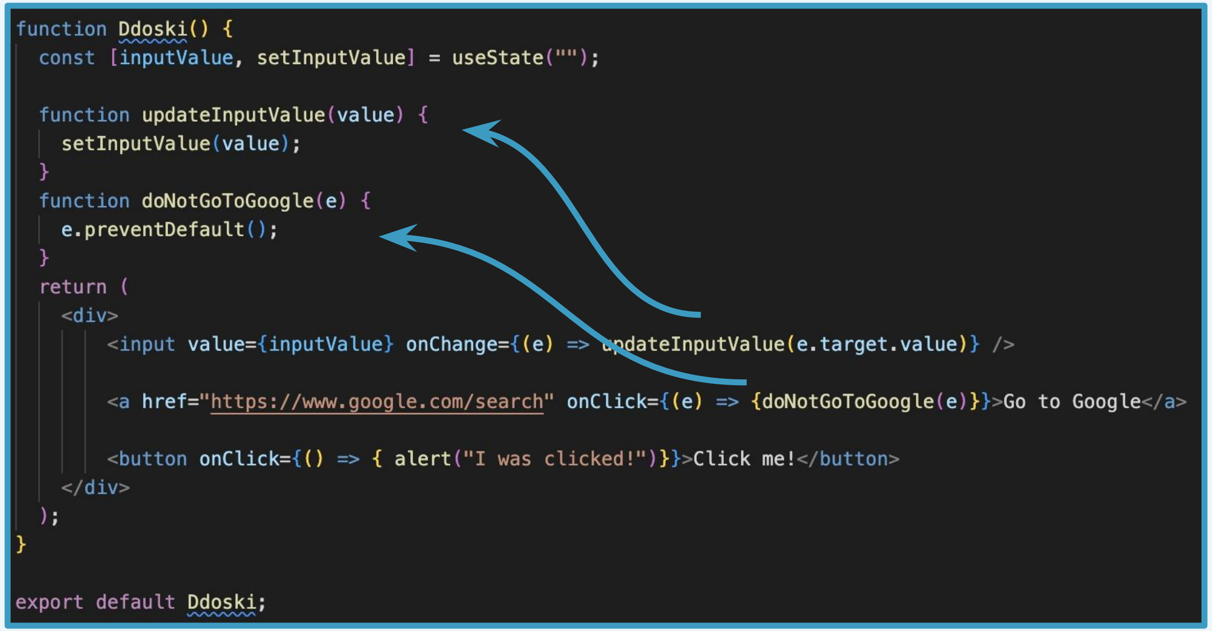
Inside the component function, before the return statement!

Two ways to write the same function for a functional component

Event Handlers

Event handlers are methods that run when someone interacts with a JSX (HTML) element.

Example Event Handlers



```
function Ddoski() {
  const [inputValue, setInputValue] = useState("");

  function updateInputValue(value) {
    setInputValue(value);
  }
  function doNotGoToGoogle(e) {
    e.preventDefault();
  }
  return (
    <div>
        <input value={inputValue} onChange={(e) => updateInputValue(e.target.value)} />

        <a href="https://www.google.com/search" onClick={(e) => {doNotGoToGoogle(e)}}>Go to Google</a>

        <button onClick={() => { alert("I was clicked!")}}>Click me!</button>
    </div>
  );
}

export default Ddoski;
```

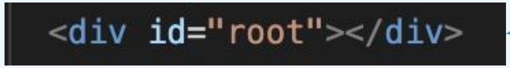
TONS of Events...

```
onClick onContextMenu onDoubleClick onDrag onDragEnd onDragEnter onDragExit
onDragLeave onDragOver onDragStart onDrop onMouseDown onMouseEnter onMouseLeave
onMouseMove onMouseOut onMouseOver onMouseUp
```

Routing

Routing

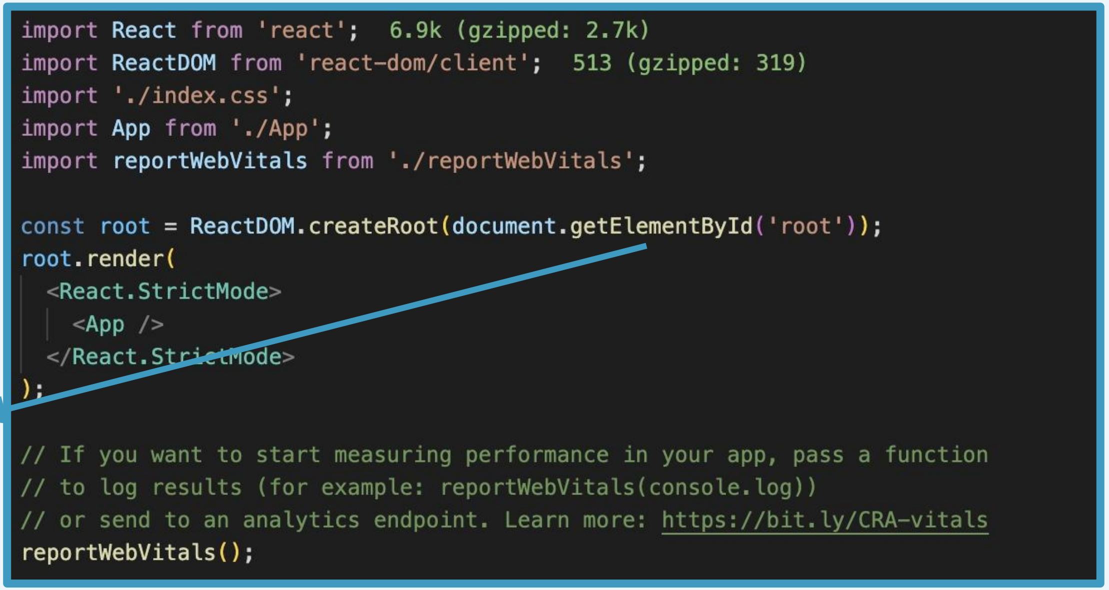
- Use React Router: has changed over the years
- Prob the most annoying part, but once it works it works!



```
<div id="root"></div>
```

in public/index.html

Index.js in create-react-app

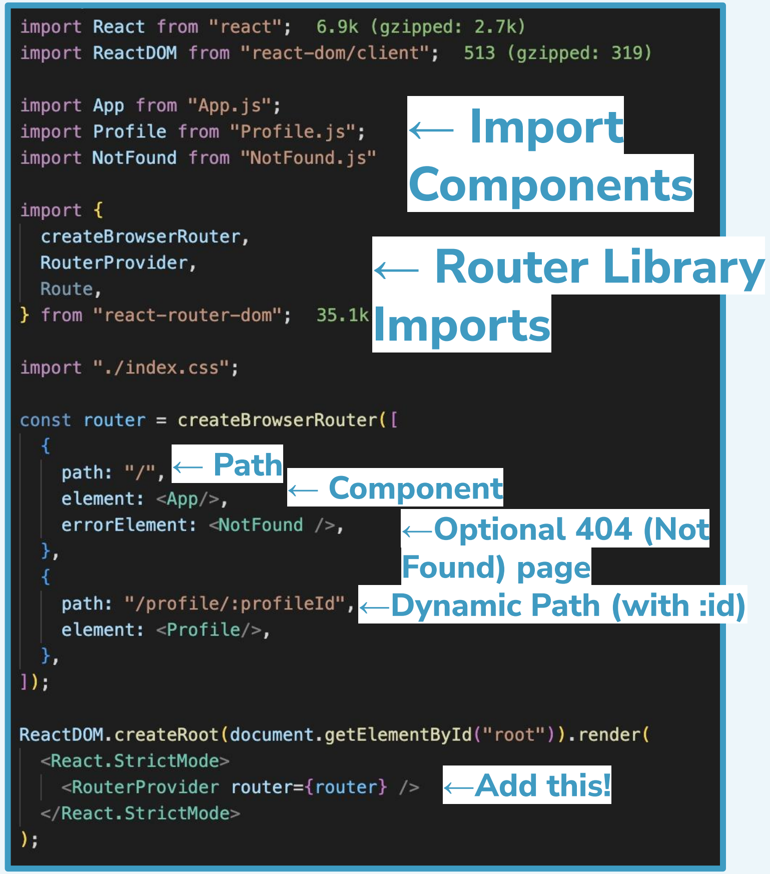


```
import React from 'react';  6.9k (gzipped: 2.7k)
import ReactDOM from 'react-dom/client';  513 (gzipped: 319)
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

Routing

For more information, go to reactrouter.com!

Update Index.js:



```
import React from "react";   6.9k (gzipped: 2.7k)
import ReactDOM from "react-dom/client";   513 (gzipped: 319)

import App from "App.js";
import Profile from "Profile.js";
import NotFound from "NotFound.js"

import {
  createBrowserRouter,
  RouterProvider,
  Route,
} from "react-router-dom";   35.1k

import "./index.css";

const router = createBrowserRouter([
  {
    path: "/",
    element: <App/>,
    errorElement: <NotFound />,
  },
  {
    path: "/profile/:profileId",
    element: <Profile/>,
  },
]);

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <RouterProvider router={router} />
  </React.StrictMode>
);
```

Routing: Switching Pages

Include this at the top of your component file:

```
import { Link } from "react-router-dom";
```

Plain JS😫 → **React**🤩

```
<a href="/path">Hi</a>
```

→

```
<Link to="/path">Hi</Link>
```

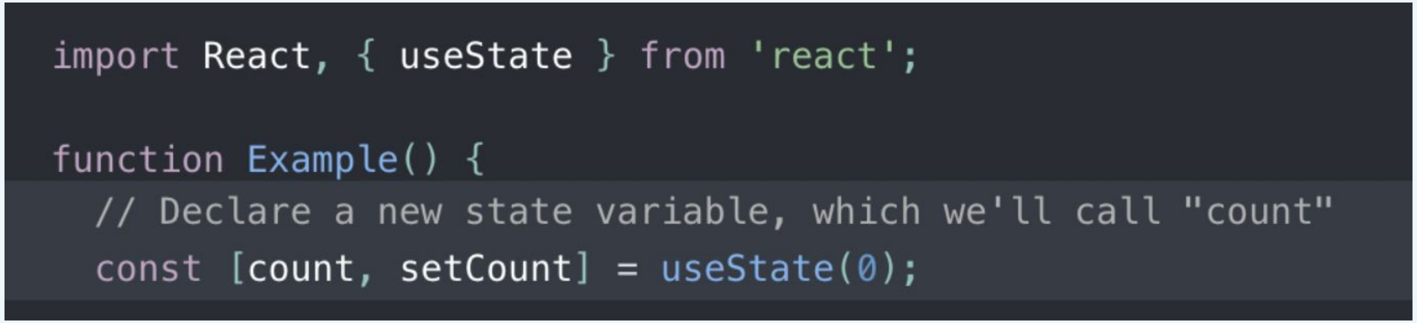
Hooks

📌useState Hook

Hooks allow you to 'hook' into components' states/lifecycle features!

useState Hook

Let's you add a state to your component!



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

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);
```

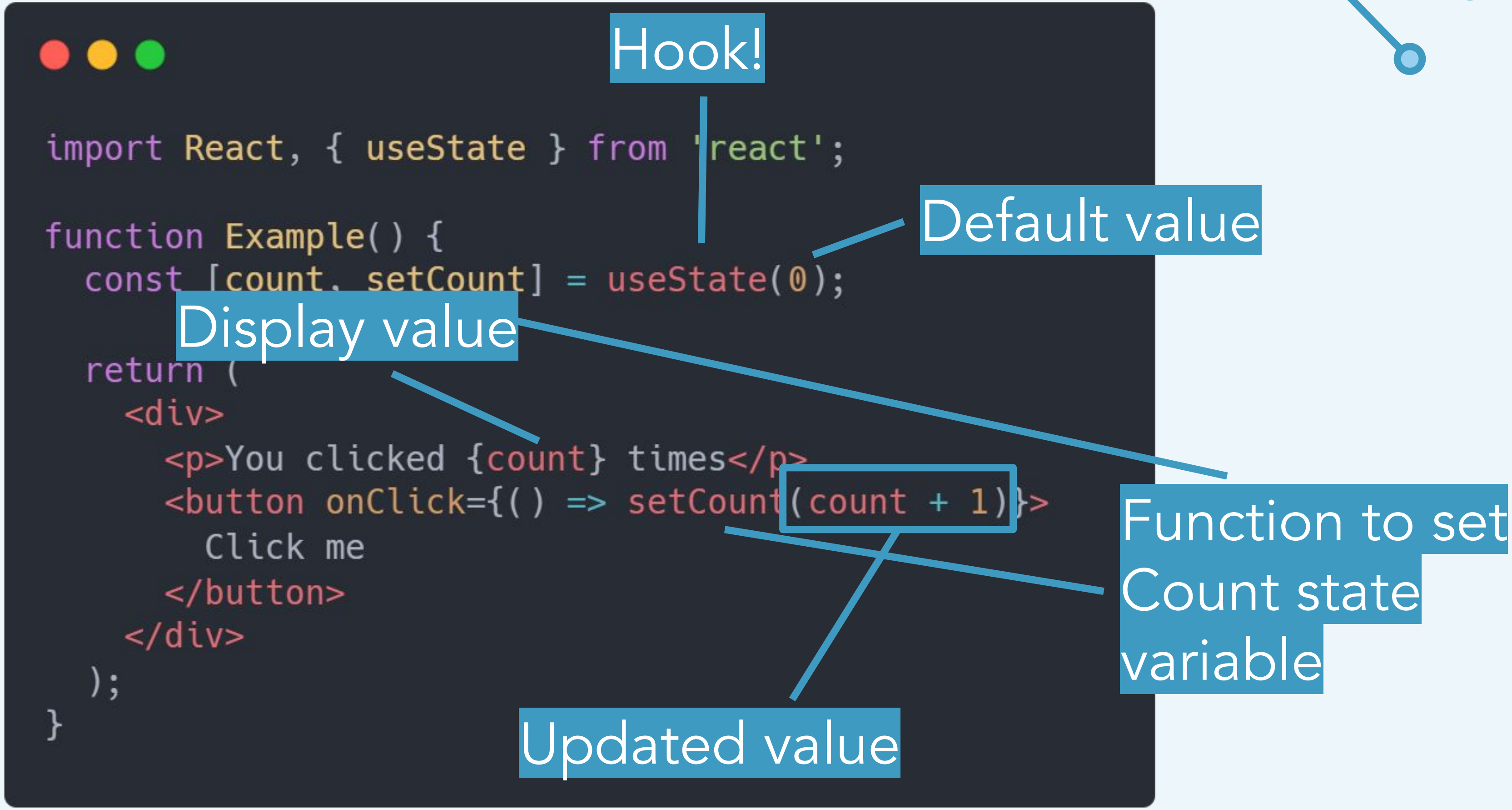
- useState() returns an array with two values: the current state and a function to update it
- We declare a **state variable** called **count**
- We name the function that will update it. We've called in **setCount**
- The argument passed in is the initial state (**count = 0**)
- Useful because state variables are *preserved* by React and won't disappear between function calls! (we always have access to **count**!)

useState Hook

- To update your state variable **count**:
    - Call **setCount(/* INSERT NEW VALUE */)**

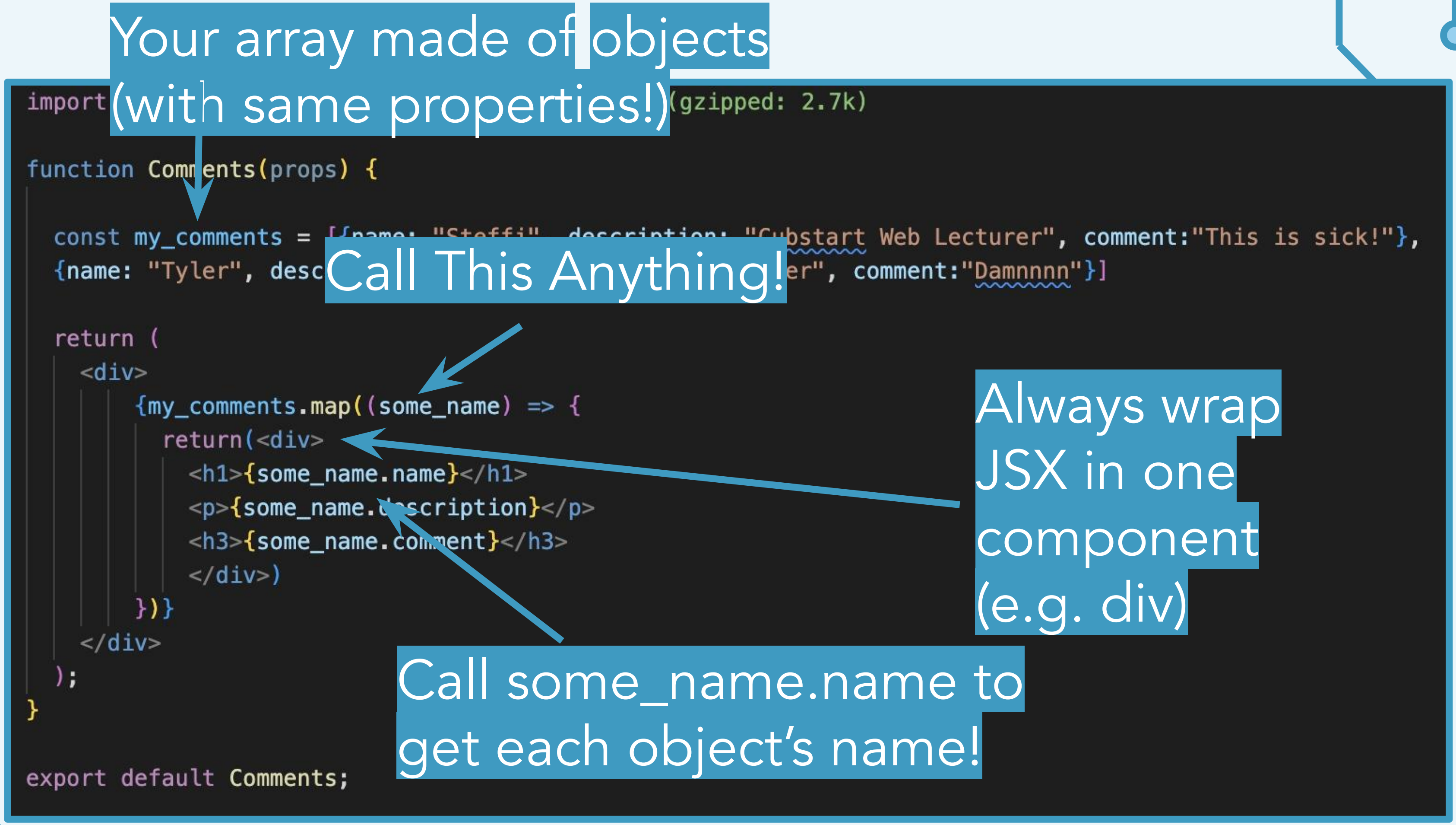
Example:

```
setCount(4)
setCount(count+1)
setCount(count * 642 - 4)
```



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

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

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Mapping



Your array made of objects (with same properties!)

Call This Anything!

Always wrap JSX in one component (e.g. div)

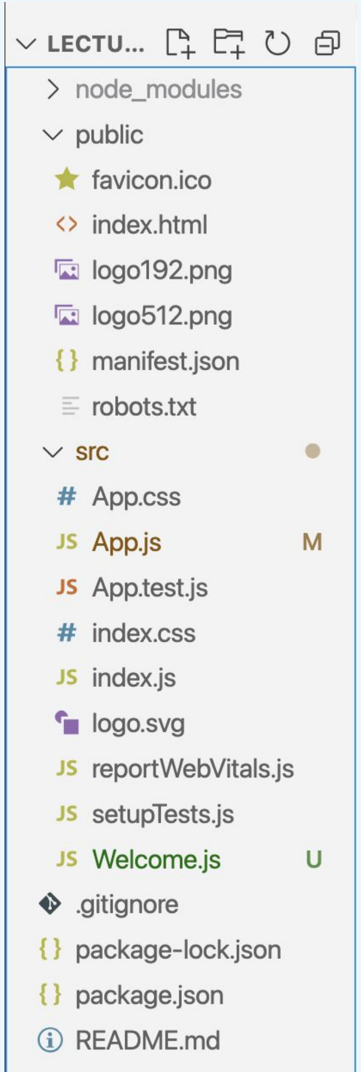
Call some_name.name to get each object's name!

```
import                                (gzipped: 2.7k)

function Comments(props) {

  const my_comments = [{name: "Steffi", description: "Cubstart Web Lecturer", comment:"This is sick!"},
  {name: "Tyler", desc                                    er", comment:"Damnnnn"}]

  return (
    <div>
        {my_comments.map((some_name) => {
          return(<div>
            <h1>{some_name.name}</h1>
            <p>{some_name.description}</p>
            <h3>{some_name.comment}</h3>
            </div>)
        })}
    </div>
  );
}

export default Comments;
```

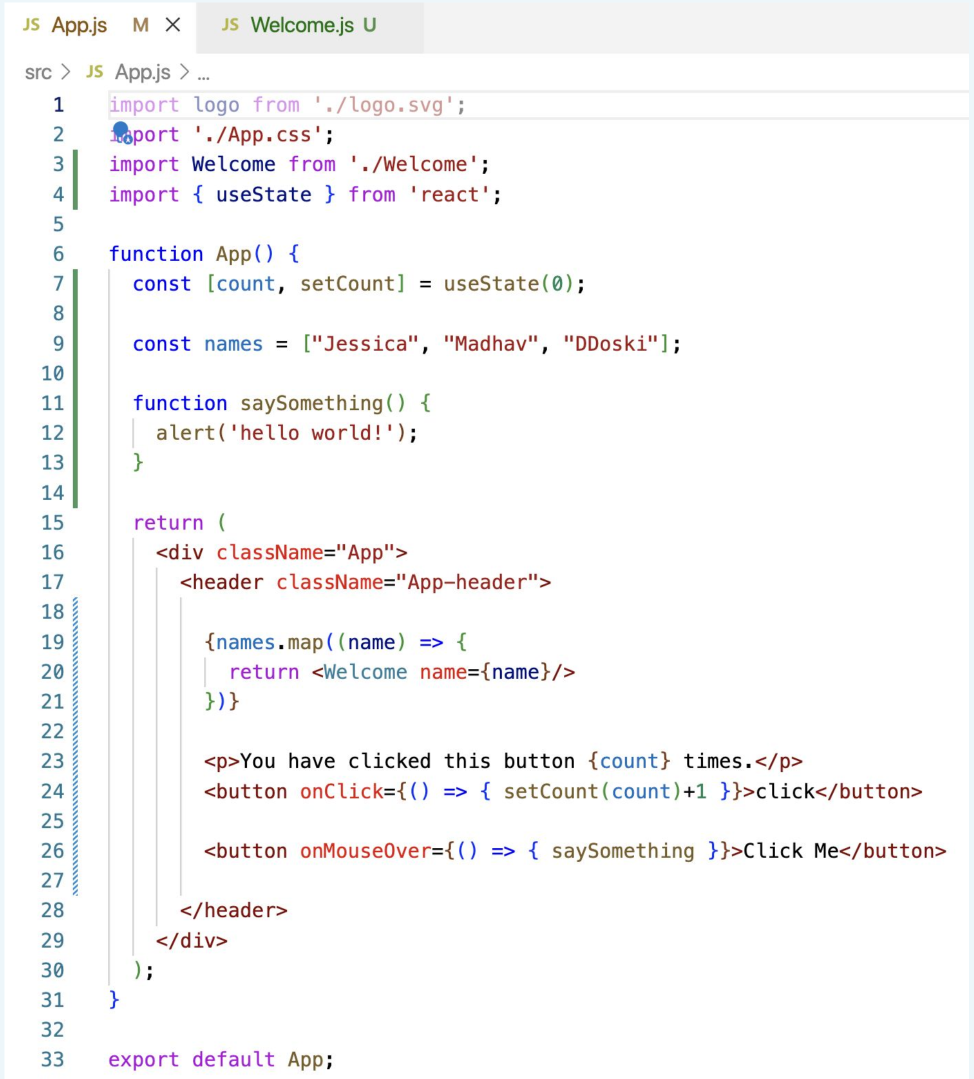
Recap

- React is centered around components
- Components are composed of other components.
- JSX allows you to write HTML inside JS code. JSX uses className="" and wraps everything in one component.
- We use props to pass parameters into sub components.
- useState() allows you to add states to components so you can update variables
- We use React Routing to create a navigation between pages.
- We can use map() to create a component for each object in an array

Demo from lecture: project structure and App.js

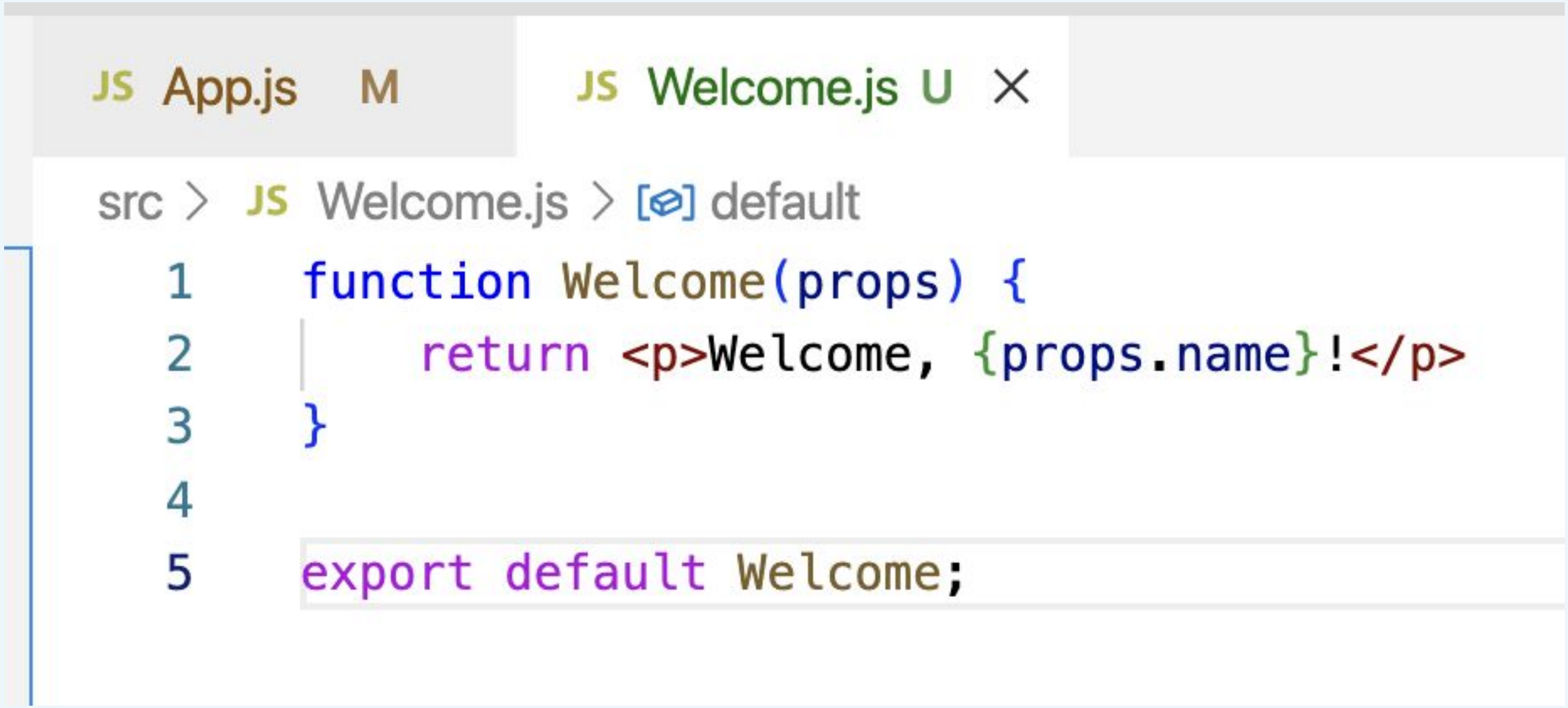


```
> node_modules
v public
    favicon.ico
    index.html
    logo192.png
    logo512.png
    manifest.json
    robots.txt
v src
    App.css
    App.js            M
    App.test.js
    index.css
    index.js
    logo.svg
    reportWebVitals.js
    setupTests.js
    Welcome.js        U
  .gitignore
  package-lock.json
  package.json
  README.md
```



```
import logo from './logo.svg';
import './App.css';
import Welcome from './Welcome';
import { useState } from 'react';

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

  const names = ["Jessica", "Madhav", "DDoski"];

  function saySomething() {
    alert('hello world!');
  }

  return (
    <div className="App">
      <header className="App-header">

        {names.map((name) => {
          return <Welcome name={name}/>
        })}

        <p>You have clicked this button {count} times.</p>
        <button onClick={() => { setCount(count)+1 }}>click</button>

        <button onMouseOver={() => { saySomething }}>Click Me</button>

      </header>
    </div>
  );
}

export default App;
```

Demo from lecture: Welcome Component



```
function Welcome(props) {
    return <p>Welcome, {props.name}!</p>
}

export default Welcome;
```

Ever Get Stuck in React?

USE THE REACT DOCS BY GOING TO REACT.DEV!