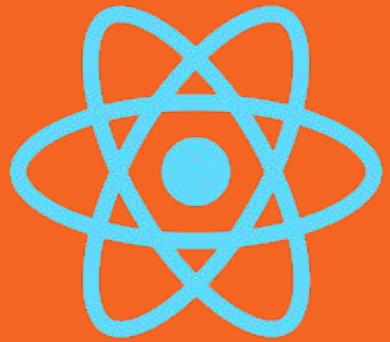
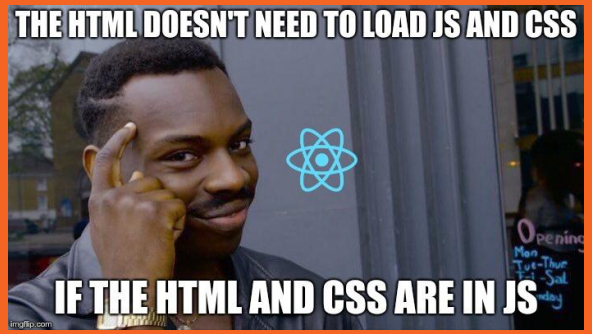
# Welcome to ReactJS



# What is React?



# ReactJS is...

A Javascript for building user interfaces(built by Facebook)

Declarative: The state of the application determines what the User Interface(UI) looks like.

Composable: create building blocks that are used to make bigger and more complicated structures

# Why React?

There are so many javascript frameworks(Angular, Vue, etc). Why do we need another?

- ➔ Speed: React is fast!
- ➔ Declarative: Makes code easier to read
- ➔ Composable: easier to build modular and reusable code
- ➔ Learning how to write a React Web App give you the experience to write an app for another platform
  - ◆ iOS
  - ◆ Android
  - ◆ Windows 10

# The ReactJS Community

- ➔ React has a large community backing it
- ➔ Tons of support by Facebook, other companies, and engineers
- ➔ Smart people are always updating and fixing React
- ➔ If you have a problem, it is probably on Stack Overflow

# Hello World!


```
<!DOCTYPE html>
<html>
<head>
    <title>Hello World</title>
    <script src="https://unpkg.com/react@15.3.2/dist/react.js"></script>
    <script src="https://unpkg.com/react-dom@15.3.2/dist/react-dom.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.24/browser.min.js"></script>
</head>
<body>
    <div id="container"></div>

    <script type="text/babel">
        var HelloWorld = React.createClass({
            render: function() {
                return <div> Hello World </div>
            }
        });

        ReactDOM.render(<HelloWorld />, document.getElementById('container'));
    </script>
</body>
</html>
```

# What's Going On Here?

**`createClass`**
Used to create a React Component(More on this to come)

**`render function`**
Minimal property that needs to be defined on an object passed to the `createClass`

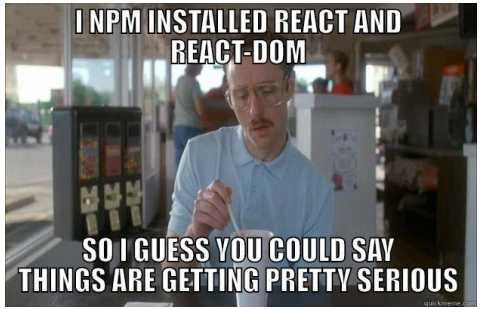
**HTML IN JAVASCRIPT?!!?!? What the huzzzaaa?**

This is not actually HTML, but something similar called JSX

# Lets get set up...(with ES6)

We are not too focused on build processes today, so let's build our app with the `create-react-app` node module

Let's get to engineering!




```
npm install -g create-react-app
create-react-app app-name
cd app-name/
npm start
```

# Hello World + ES6

```
//App.js
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class HelloWorld extends Component {
  render() {
    return <h1>Hello World!</h1>
  }
}

class App extends Component {
  render() {
    return <HelloWorld />
  }
}

export default App;
```

# How does it all work?

**Virtual DOM**

**Diffing**

# Intro to the Virtual DOM

# The Virtual DOM

React utilizes the concept of the virtual DOM, an in-memory representation of the actual DOM.

When we define the render function in React, we are creating the virtual DOM, not the actual DOM.

Any time a state variable changes, React updates its virtual DOM to reflect the new state.

When defining the `render` function, we are not trying to create the actual DOM, but the virtual DOM

- ➔ React will handle rendering the actual DOM (aka painting the DOM)
- ➔ JSX is the language we use to generate the virtual DOM

# Diffing

In React, when a change is detected in the virtual DOM, it performs a process called "diffing."

What is a diff?

- ➔ Diffing involves comparing the previous virtual DOM with the new virtual DOM to identify the differences between them.
- ➔ It determines the exact locations and nature of the changes that have occurred.
- ➔ Once the differences are identified, React updates or "paints" the corresponding parts of the actual DOM on the page.
- ➔ Performing the diffing process in the virtual DOM is efficient and cost-effective, making React highly performant.
- ➔ Manipulating the actual DOM directly is expensive in terms of performance

# JSX

React uses a special syntax called JSX

- ➔ JSX is a language written by Facebook that simplifies the amount of code we need to write
- ➔ JSX allows us to XML inside of our Javascript
- ➔ Similar to XML, JSX consists of tags that can have attributes and children.

# JSX(continued)

- ➔ JSX is not something browsers understand, so we have transpile it to Javascript before serving it to the browser.
- ➔ We previously transpiled ES6 code into ES5 code using Babel
- ➔ We can also use Babel to transpile our JSX code into Javascript(more on this to come…)

# JSX Again!

**Without JSX**

```
// React without JSX
render() {
  return React.createElement(
    "div",
    {className: "foo"},
    "Hello ",
    this.props.name
  );
}
```

**With JSX**

```
// React with JSX
render() {
  return <div className="foo">Hello {this.props.name}</div>;
}
```

# Yes More JSX!!!!

- ➔ React can render HTML tags and React Components. Below is an example of rendering a `<div>` tag.

- ➔ HTML tags are lowercase. React distinguishes between HTML tags and React Components by their casing.

```
var exampleDivElement = <div>
        <h1> Welcome </h1>
        <MyComponent className="foo">Hello World</MyComponent>
    </div>
```

# Guess What???... JSX

Why `className` instead of good old `class`?

- ➔ Since JSX is written in Javascript, identifiers such as `class` and `for` are discouraged as XML attribute names.
- ➔ Instead, React DOM components expect DOM property names like `className` and `htmlFor`, respectively.

# JSX... Last JSX Slide I Promise!

You can use Javascript expressions in JSX by using curly braces ... { }

```
<h1>Hello {this.user.name}</h1>
```

and also

```
<h1>Total Amount {2 + 2}</h1>
```

# Let’s get Reacty

# Exercise

Length: 10 mins

Timer

- ➔ Get yourself set up with `create-react-app`
- ➔ Create-react-app to make a sample application
- ➔ Get the Hello World application working

# Components

# Component Based Architecture

- ➔ Declare a state and tie it to specific UI component
- ➔ Think small to big! Single purpose components and isolated scope make it easier to debug and maintain… separation of concerns
- ➔ Small components are reusable through out your app
  - ◆ DRY - Don’t Repeat Yourself
- ➔ You might have a component for different features on your site such as:
  - ◆ Navbar
  - ◆ Search Form
  - ◆ Create comment form
  - ◆ Slider

# What does a component look like?

Minimum Requirements:

- ➔ Extends the `Component` class
- ➔ Defines the `render` function

```
class HelloWorld extends Component {
  render() {
    return <h1>Hello World!</h1>
  }
};
```

# Gotchas!

- ➔ render should return a single HTML element
- ➔ Every self-closing tag in JSX needs an ending
  - ◆ <br> needs be <br/>

# Data and Properties

# What about the Data?


There are two ways of getting data into our components:

1. Props
2. State

# Props aka Properties

- props is the data that is passed into your component
- props are immutable, meaning they can’t be changed
- You can think of props as the default data of your component
- Objects/Arrays can’t be passed into props
- Doesn’t take objects either

Component Documentation

# Example of props in action...

Component with JSX

```
class HelloWorld extends Component {
  render() {
    return <h1>Hello {this.props.name}!</h1>
  }
};
```

Render to DOM with prop

```
ReactDOM.render(<HelloWorld name="Tom" />)
```

# State

How can we make UI changes if we can’t mutate the data?

With **State**!

**State** is an object defined on the Component that can be used in the render function to define data that will change

# Example of State

constructor

super(props)

this.state

this.setState

```
class HelloFriend extends Component {
    constructor(props) {
      // props.name is default data; in our first example we passed in 'David'
      // need to call super(props) when using 'this' in constructor
      super(props);
      this.state = {
        name: props.name
      }

      setTimeout(this.updateName.bind(this), 2000)
    }

    updateName() {
        this.setState({name: 'Jeff'});
    }

    render() {
      return <h1>Hello {this.state.name}!</h1>
    }
};
```

# Proptypes

- Defined as the `propTypes` property on the `Component`
- Essentially a dictionary where you define what props your component needs
- Also, what type(s) they should be
  - A type is `Number, Object, String,` etc

# Proptypes Example # 1

- `id` prop should be a type Number
- `id` is **required**
- message is not required, and is type `String`

```
MyCoolComponent.propTypes = {
  id: React.PropTypes.number.isRequired,
  message: React.PropTypes.string
}
```

# Proptypes Example # 2

imports

constructor

render

propTypes

```
import React, { Component, PropTypes } from 'react';

class MyCoolComponent extends Component {
  constructor(props) {

  }

  render() {
    return <div id={this.props.id}>{this.props.message}</h1>
  }
};

MyCoolComponent.propTypes = {
  id: PropTypes.number.isRequired,
  message: PropTypes.string
}
```

# Exercise # 2 - Increment-Decrement

State

- Define a ‘Increment-Decrement’ (call it whatever makes most sense to you) component that will display a number an “Increment” button, and a “Decrement” button next to it.
- The start number should come from props.
- The number should have a Number prop type and be required in the component.
- “Increment” should increase the number by 1
- “Decrement” should decrease the number by 1
- When the number reaches zero, clicking on “Decrement” should show an alert to the user stating “Cannot be less than zero” and not decrement into less than zero

# Housekeeping TBD