This document contains notes from the React lecture on November 12th 2019.

# Setting Up

StackBlitz and CodePen are easy ways to create a React App online without having to install anything on your computer.

Please refer to the React Lab, available on the course site, for information on getting set up on your computer.

# HTML Review

HTML has elements; HTML code is made up of tags. In the example below, `<head>` is an opening tag and `</head>` is the matching closing tag:

```
<head>
    <title>Hello</title>
</head>
```

In the below example, color is the name and "blue" is the value.

```
<div color="blue">
```

# Why React?

Lots of websites have repetitive elements. React can help avoid lots of copying and pasting of code.

If you want to change something, you'll have to change it in every single element. React helps avoid this.

Sometimes a piece of data has dependencies (e.g. the amount of money you have left is based on the number of cookies you bought). When one piece of data changes, others should change if necessary. With React, you don't have to manually encode these changes.

React avoids having to refresh a page to update the number of likes a post has after you like it. Instead, it responds automatically right when you click it.

# React Coding

Before you begin: install React Developer Tools, which will be super helpful for debugging. First, you'll need to import libraries in the JavaScript file (index.js).

```
import React from "react"
import {Component} from "react"
import ReactDOM from "react-dom"
```

OR, instead of

```
import React from "react"
import {Component} from "react"
```

you can use

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

HTML goes in index.html. We can ignore package.json, which just contains some dependencies. The rest of the code goes in index.js. We'll be working in the HTML file now.

Next, we'll create a React element and render it. Always make sure you call render so that your element will show up!

```
const App = () => React.createElement("h1", null, "Hi class")
ReactDOM.render(<App /> document.getElementById("main"))
```

# Functions

First, we define a function that takes in an element and returns whether it should be filtered or not.

```
// using old style declaration
function filterFunc(num){
return num > 2;
}

// OR new style declaration
const filterFunc = (num) => {return num > 2;}
```

```
// OR super clean style
const filterFunc = num => num > 2;
(Jeff used this style in the live demo)
```

## Components

Components are the heart of React. They're used to factor out common code that will be used repeatedly. They're very similar to classes in Java!

Components take in two types of input: state and props, and then output the interface.

Every component needs a render() function that basically tells the component what to draw.

```
class App extends Component {
    render() {
        return ( // wrapping everything in parentheses prevents errors later
            <div>
                <h1>Howdy</h1>
                <p>I love dogs.</p>
            </div>
        )
    }
}
```

## Material UI

First we have to import Material UI's libraries:

```
import MenuIcon from '@material-ui/icons/Menu'
import { AppBar, Typography, Toolbar, IconButton } from '@material-ui/core'
```

To create an AppBar for your app, add `<AppBar>` tags to your `render` function's return:

```
render() {
    return (
        <AppBar>
```

```
        </AppBar>
    );
}
```

To add some text, just toss it between the tags:

```
render() {
    return (
        <AppBar>
            Portfolio Items To Do
        </AppBar>
    );
}
```

To make your text follow Material UI's typography, wrap it in a `Typography` tag:

```
render() {
    return (
        <AppBar>
            <Typography>
                Portfolio Items To Do
            </Typography>
        </AppBar>
    );
}
```

If we want to add a menu button, we can make the entire AppBar a ToolBar, and add the menu button alongside the text:

```
render() {
    return (
        <AppBar>
            <ToolBar>
                <SomethingIcon>
                    <MenuIcon/>
                </SomethingIcon>
                <Typography>
                    Portfolio Items To Do
                </Typography>
            <ToolBar>
        </AppBar>
    );
}
```

# Functional Components

A somewhat cleaner syntax to create a Component is:

```
const TopBar = () => (
    // HTML HERE
)
```

# Props

If you have a component, you can pass attributes into it, which become props (short for properties). For example, if you have an `App` component, writing `<App number="7"/>` passes in a prop called 'number' with the value of 7. Inside of the `App` component, you can access this property with `props.number`.

You can pass in numbers, strings, and even functions as properties to your components. For example, if you've written `this.filter`, you can pass it to a component like so:

```
<App filterFunction={this.filter}>
```

and we can call it within the component with `this.props.filterFunction`:

```
this.props.filterFunction(stuffToFilter)
```

# More React Coding

To prevent the default action (e.g. a page refresh), use preventDefault(), e.g.:

```
addItem = event => {
event.preventDefault()
}
```

The `flat()` method turns an array of arrays [[3, 4], 3] into an array [3, 4, 3]
e.g. `const items = [this.state.items, newItem]` OR, you can use the ellipse syntax:

`[...this.state.items, new Items]` turns an array into a list

When setting the state:

```
this.setState({items:items})
```

can be written as simply

```
this.setState({items})
```

This doesn't look like it should work, but it does!

The map function applies the same function to all the items in an array:

```
createTasks = item => {
return (<h1>{item.text}</h1>)
}

const todoEntries = this.props.entries;
const listItems = todoEntries.map(this.createTasks)
```