React

Document Object Model (DOM)

- 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>
```

- The tags have a tree-like structure, where tags can have children or a parent tag.
- In the example below, the `<body>` tag is the parent of `<div>`, which is the parent of `<h1>` and `<p>`.

```
<head>
    <title>Hello</title>
</head>
<body>
    <div>
        <h1>Heading</h1>
        <p>Here is some text.</p>
    </div>
</body>
```

JavaScript/React vs. CSS

- In CSS, we can use `background-color` to manipulate background color.
- However, in JavaScript, this would translate to "background minus color".
- In JavaScript/React, use camel case. For example, instead of `background-color`, use `backgroundColor`.
- A useful function: `getElementById()` takes in an identifier and returns an element. For example: `document.getElementById("main")`

Why Use 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 ReactDOM from "react-dom"}
```

- 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

- The way that `App` is declared is different from the React lab. It follows this syntax: `<func name> = (args) => <func body>`
- The standard way of declaring a function `add()` is like this:

```
function add(a, b){
    return a + b;
}
```

or we could declare it as an anonymous function:

```
var add = function(a, b)
{return a+b}; // javascript goes inside braces
```

- We can add 3 and 4 using `console.log(add(3, 4))`.

Props

- Any attributes we pass into `App` are automatically assigned to props. For example, if you pass 7 into `App` using `<App number = "7" />`, then you can access 7 inside of `App` using `props.number` inside of `App`.
- Passing in arguments: if `App` takes in some `props`, we can render it like this.

```
function App(props) {
    return React.createElement("h1", null, "hey class " + props.number);
}
ReactDOM.render(<App number="7"/>, document.getElementByID("main"))
```

Filtering

- 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;
```

- Now we can create an array and filter it.

```
var arr = [1,2,3]
arr.filter(filterFunc)
const App = () => <h1 id="someElementID">{arr}></h1> //show array
```

React

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

- Inside a component is also the state! The state holds key:value pairs that can be used elsewhere in the component. The state gets defined in a component's constructor. Note that components are not required to have a constructor.

```
class App extends React.Component {
    constructor(props) {
        super(props)
        this.state = { // initialize state
            gold: 5,
            silver: 7
        }
    }
    render() {...}
    ...
}
```

Event Handling

- Event handlers allow you to make something happen when the user does something.

- If we want something to happen when a button is clicked, use `onClick`. There are two pieces to this:
  - We'll have a function called `iwasclicked` that prints to the console.
  - We specify that `iwasclicked` should be called when the button is clicked.

```
class App extends React.Component {
    // function to specify what happens on click
    iwasclicked = () => { console.log("I got clicked") }
    render() {
        <div>
            <button onClick={iwasclicked} name="mybutton"/>
            click me!</button>
        </div>
    }
}
```

- Now let's print to the console when some input is changed.

```
class App extends React.Component {
    iwaschanged = () => { console.log("I got changed") }
    render() {
        <div>
            <input onChange={this.iwaschanged} name="myinput"/>
        </div>
    }
}
```

An Example

http://nicolashery.github.io/example-ui-as-data/