React JS Lesson

By: Ostavo Palacios

Objectives

1. Learn a few terminal command lines
2. Create a react app
3. Translate our HTML/CSS code from workshop 1 to React Format
4. Differentiate Javascript syntax from React
5. Learn React JS
    5.1. Components
    5.2. Props
    5.3. States
    5.4. Event Handle
    5.5. Conditional Rendering
    5.6. Lists and Map

Starting

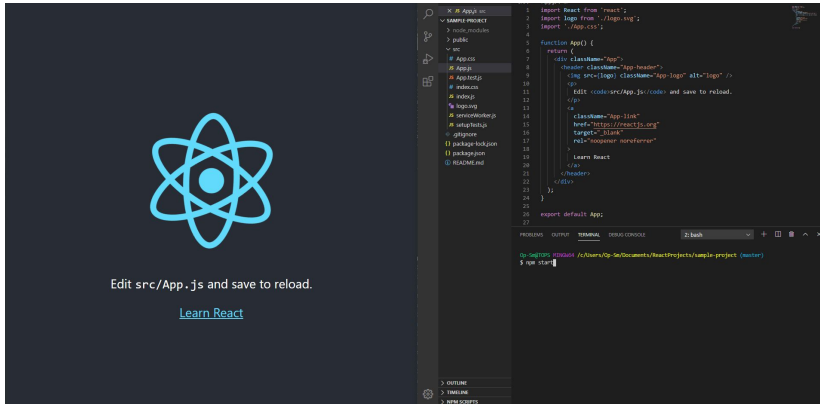
In the Terminal `npx create-react-app my-app`

`cd my-app`

`npm install`

Open file on your editor and in the terminal where your package-lock.json file is run `npm start`

You should get the picture on the left



Quick Differences

Arrow Functions are more efficient to use, they allow for better use of *this* keyword

```
handleClick = () => {
  this.setState({ counter: inc });
};
```

To use javascript in html code you use curly braces to call functions

```
{this.thisOutput()}
```

To use other Components or CSS files you use import keyword

```
import React from "react";
import ReactDOM from "react-dom";
import "./index.css";
import App from "./App";
```

Syntax Component: import *NameOfcomponent* from “*filelocation*”

Syntax CSS : Import “file location”

Components

Functional Components

```jsx
import React from 'react';
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

return() is needed in order to output to screen

Output

import allows you to call other component or files

Benefits: Usually easier to read, for shorter amounts of code.

Class Components

```jsx
import React from "react";
import logo from "./logo.svg";
import "./App.css";

class App extends React.Component {
  render() {
    return (
      <React.Fragment className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <p>
            Edit <code>src/App.js</code> and save to reload.
          </p>
          <a
            className="App-link"
            href="https://reactjs.org"
            target="_blank"
            rel="noopener noreferrer"
          >
            Learn React
          </a>
        </header>
      </React.Fragment>
    );
  }
}

export default App;
```

In Class Components you can can have state variables and allow for more complex components.

Props and Components

Props can be passed down components these relationships are usually known as a child component and Parent Component where Parent component passes down props/attributes which the children inherit.

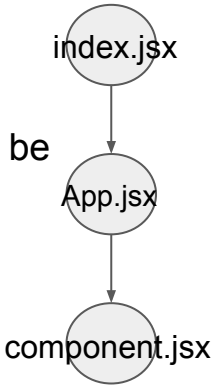
Props are immutable, they are variables you can pass down when you call the component States are not so they can be changed even outside their respective components.

In parent component Index.js, it is calling another component(child) App.js

```
<App name={"ostavo"} />
```

Props

Calling a component can be done as so



Child Component

```
class App extends React.Component {
  constructor(props) {
    super(props);
  }

  render() {
    return (
      <React.Fragment>
        <header className="App-header">
          {this.props.name}
```

To use the props.name you call as so in the component that will render it.

States and handling event

States can be used the same as props but they are not immutable unlike props. So you can change the value of state. You can only make states in class components

```
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      counter: 0
    };
```

To declare a state variable you would do the same as making an object. This allows for the same manipulation as objects

There are Components like Button which you have to control the event that it will trigger. you do that by creating a function which does that commands you ask of it

```
<button onClick={this.handleClick}>this is a button</button>
```

```
this.setState({ counter: inc });
```

setState() allows to change the value of state variable

```
handleClick = () => {
  var inc = this.state.counter;
  inc++;
  this.setState({ counter: inc });
};
```

```
<ButtonOuput
  var={this.state.counter}
  buttonState={this.state.buttonState}
/>
```

You can pass down state variables like props and use them as props in child components

Conditional Rendering

Using conditionals you can have some pieces of html code show at a time. You can return html code by calling return and inputting what html code you want outputted depending on the condition.

```
buttonClickedText = () => {
  var props = this.props.buttonState;
  if (props === 2) {
    return <h4>Reset button Was clicked!!</h4>;
  } else if (props === 1) {
    return <h4>incrementing value</h4>;
  } else {
    return <h4>Press button</h4>;
  }
};
```

Here we have a prop *buttonState* that is controlling the output when a certain button is clicked.

Lists and Map

We can output arrays more easily by using a map() function. This function returns a new array and lets us call on the properties in an object. This is also minimizes the lines of code.

Note: when using map it is important to assign a key value for each value in the array.

To use map() you need to input an arrow function and return code in that arrow function

```
this.hwLists = [
  {
    name: "HTML",
    number: 1
  },
  {
    name: "Javascript",
    number: 2
  }
];
```

An array of objects each with a *name* and a *number* attribute

Attributes

```
<ul>
  {this.hwLists.map(hwlists => (
    <li key={hwlists.number}>
      <h2> Hw# {hwlists.number}: </h2> <h3> {hwlists.name} </h3>
    </li>
  ))}
</ul>
```