Web Front-End Development
Week 11: React Component Architecture

<u>Component architecture</u>
https://reactjs.org/docs/components-and-props.html
Components are the main building blocks of all React apps.
React separates "concerns" (or features) into components. Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.
Instead of separating markup, styling, and logic as we do in regular web development, React couples these together into components.
Components are basically JavaScript classes or functions. They accept inputs and return React elements describing what should appear on the screen.
A component should ideally do one thing, have one responsibility. If it ends up growing, it should be broken up into smaller subcomponents.
Components can refer to other components and render them. This lets us use the same component abstraction for any level of detail. A button, a form, a dialog, a screen: in React apps, all those are expressed as components.
Like classes, always start component names with a capital letter.
A component must return a single root element in render().
You will see Components created 3 different ways:

- Functional components: function App(){}
- ES5 syntax: var App=React.createClass({render: function(){}});
- ES6 syntax: class App extends Component {render(){}}
    - We will be using ES6 classes as they have some additional features we'll eventually need

Example:
hello-world (index1.js and App1.js)

- App component (un-comment out import App statement)
    - Create a component named App (App.js)
    - render() determines what is returned when that component is rendered
    - return can only return one parent element
    - It looks like it's returning HTML but it's actually JSX which gets translated to regular JavaScript at run time.
    - The Babel transpiler compiles JSX down to JavaScript React.createElement() calls
        - React.createElement(type, props, children)
        - Nested calls to React.createElement() for each element
        - Use of className
    - Note that the App component is exported in its file and imported in index.js. Use the component in render()

<u>Props</u>
Components store data as properties in an object called props.
Props holds information about that component.

To create a property in a component you define an attribute by giving it a name, just like you do with variables.

```
<Greeting firstName='Aileen' />
```

This creates a prop called firstName. You can create as many props as you want.
Props is an object that can have as many key/value pairs as you want to describe your data.
Components access props through {this.props.*propertyname*}

```
{this.props.firstName}
```

Props can be used within a component or to pass data to other components.
Components pass data to another component using an attribute that has the name of the property.
Properties are passed as a single argument called props.
Through props properties can be accessed with whatever property name they were given. All the properties are passed in through props.
You can also set default values for props using a class property called defaultProps.

```
Greeting.defaultProps = {firstName: 'yo'};
```

The default will be used if no data is passed to the prop firstName.

Props are read only and components should never modify props.
You can also pass functions as props.

```
keepScore(){}

<Greeting score={this.keepScore} />
```

All React components must act like pure functions with respect to their props.

- Pure functions do not attempt to change their inputs, and always return the same result for the same inputs.

Example:
hello-world (index2.js and App2.js)

- Props provide data properties to be sent to a component
    - You can name your properties anything as long as you're consistent
    - The curly braces is the syntax to let the JSX parser know that it needs to evaluate the expression instead of treat them as a string and just print them
- The ability to reuse components is the whole reason behind components

Styling

You can link to an external CSS file for CSS for the whole site.
React favors an inline approach for styling content within a component. It's designed to help make your visuals more reusable. The goal is to have your components be independent units where everything related to how your UI looks and works gets stashed there.
The way you specify styles inside your component is by defining an object whose content are the CSS properties and their values.

Because it's an object it's a key/value pair and the values are strings so they're in quotes.
fontSize: "50px"

```
<h1 style={{ fontSize: "50px"}}>
```

Here there are 2 sets of curly brackets – the inside one is the object, the outside one is for JSX to evaluate it. This method can get messy and cumbersome.
Instead you can define a variable and assign the object to it.

```
var headerStyle = { fontSize: "50px"}
```

Once you have an object defined, you assign that object to the JSX elements you wish to style in your component by using the style attribute.

```
<h1 style={headerStyle}>
```

In React CSS properties are similar to when we used them in JavaScript.

- Single word CSS properties (like padding, margin, color) remain unchanged.
- Multi-word CSS properties with a dash in them (like background-color, font-family, border-radius) are turned into one camelcase word with the dash removed and the words following the dash capitalized.
    - For example, background-color becomes backgroundColor, font-family becomes fontFamily, and border-radius becomes borderRadius.
- In React if you write a style value as a *number*, then the unit `"px"` is assumed.
    - fontSize: 50

You can also keep your styles in a separate js file, export them, and then import them in the components where you want to use them.

Example:
hello-world (index3.js and App3.js)

- Styling
    - Object of CSS properties and values
    - Using props

Example:
Favorite #1 (App1.js and Favorite1.js)
Restructured from our hello-world example so index.js renders the App component which renders the Favorite component

- App.js
    - Add {Component} in import statement.
    - class App extends Component { … }
    - add className="App" to the H1 tag
- Create a Favorite component
    - In App.js import Favorite from './Favorite'
    - Favorite.js
        - Notice the "export default Favorite" at the bottom of the Favorite.js file. This allows it to be imported in another file, as we do at the top of App.js.
            - There can only be one default export per file. In React it's a convention to export one component from a file, and to export it is as the default export.
            - You could also have the export statement before the class
                - Export default class Favorite extends Component{…}