Web Front-End Development
Week 9: React Basics and Setup

As web sites have grown into full blown web apps, developers are turning to frameworks to handle their complexity. We're going to look at one of the most popular frameworks to see its approach to creating web apps.
What is a web app and how does it differ from a web site?
There's really no difference technically, and people use different definitions, but typically web apps refer to web sites that are very interactive and often all on one page. Being highly interactive, performance and scalability become key issues.

Install
Install Node.js https://nodejs.org/en/ if you don't already have it installed (LTS version is fine).
Node.js is a JavaScript runtime built on Chrome's JavaScript engine.
This also installs npm, the node package manager.
Windows instructions: http://nodesource.com/blog/installing-nodejs-tutorial-windows/

React
React is a JavaScript framework created by Facebook and was initially released in 2013.
React was created to solve the problems that arise when developing web apps:

1. Traversing and manipulating the DOM is slow and when your DOM grows, and your app has to access it a lot, you'll run into performance issues
2. Keeping your data in sync with your user interface can be cumbersome

React has two main goals:

1. Manage the creation of complex, large-scale user interfaces
2. Handle web apps with data that frequently changes

React is not only used by Facebook and Instagram, but also by Netflix, Dropbox, the New York Times, Yahoo, the BBC, Khan Academy, PayPal, Reddit, and many more.
As we get into React and understand its approach to front-end web development you'll see that it takes a very different approach that challenges many of the de-facto standards in web development.

You'll also notice that the terms library and framework are often used interchangeably. I believe there is a big difference. A library offers some functionality by providing a set of functions that you can easily use in any web site without having to change its overall structure. A framework provides a broader approach to development and imposes a structure for the pattern you use in your development. Because React focuses on user interface components it is not considered a full application framework so some refer to it as a library. But because it imposes control over the structure of your code many consider it a framework.

React Key Concepts
Some of the key concepts behind React will feel like the opposite of how we've been approaching web development. You'll need to understand, and embrace, this structure in order to develop React applications.

ReactDOM is the package that contains several React-specific methods, all of which manage DOM elements on a web page.
https://reactjs.org/docs/hello-world.html

Elements
https://reactjs.org/docs/rendering-elements.html
React elements are the building blocks of React apps and can be almost any HTML tag.
Unlike browser DOM elements, React elements are plain objects, and are cheap to create. React handles updating the DOM to match the React elements.
React elements are rendered using `ReactDOM.render().`
`ReactDOM.render(what, where)` renders *what* you want to *where* you want it in your document.
`ReactDOM.render()` can only render one root element so if you have multiple elements you'll need to wrap them in one parent element first, such as a div.
React apps only call `ReactDOM.render()` once.
Although the elements look like HTML, it's actually a special syntax called JSX which makes it easier to write these structures.

Virtual DOM
When using React, React is responsible for building the entire web page.
Because at scale manipulating the DOM is slow, React uses a virtual DOM.
The Virtual DOM is an abstraction of the HTML DOM. It is lightweight and detached from the browser-specific implementation details.
So instead of directly manipulating the document object after every change, React does all the work in its virtual DOM using React elements.
React keeps track of the difference between the current virtual DOM (computed after some data changes), with the previous virtual DOM (computed before some data changes). React then updates the browser's DOM only with the necessary changes.

JSX
https://reactjs.org/docs/introducing-jsx.html
To use its virtual DOM, React uses JSX, an extension to JavaScript that produces React elements. Although you could use regular JavaScript, it's not the React way, and once you get used to JSX it's more streamlined as well.
JSX expressions are converted to JavaScript objects after compilation so the result is still the core web technologies - HTML, CSS, and JavaScript.
JSX is a language that allows you to easily mix JavaScript and HTML-like tags to define user interface (UI) elements and their functionality. So instead of separating technologies by putting markup (HTML) and logic (JavaScript) in separate files, React separates concerns by coupling the UI and its logic into components (more on these next week).

- HTML-like syntax (use lower case)
- JavaScript expressions by wrapping them in curly braces.
- React classes

JSX elements are treated as JavaScript expressions. They can go anywhere that JavaScript expressions can go.

That means that a JSX element can be saved in a variable, passed to a function, stored in an object or array.
JSX elements can have attributes, just like HTML elements can.
A JSX attribute is written using HTML-like syntax: a name, followed by an equal sign, followed by a value. The value should be wrapped in quotes.
Since JSX is closer to JavaScript than HTML, React DOM uses camelCase property naming convention instead of HTML attribute names.
Because JSX is JavaScript, you can't use JavaScript reserved words such as 'class' and 'for'. Instead we use 'className' and 'htmlFor' that get converted in the JavaScript equivalent.
In JSX self-closing tags must include the / so for example <br> must be <br />.

Example:
https://repl.it/@aileenjp/react-hello-world-basic
index.html

- div with the id root which is where we'll render our React elements

index.js
Basic render w/JSX

```
const element = <div><h1>Hello World</h1></div>;
```

- ReactDOM.render(what, where)
- render() returns a virtual DOM representation of the browser DOM element(s)
- render() must return one element so to return multiple elements wrap them in a div and make them children of the div element.

Embedded JavaScript expression
{ } are *markers* that signal the beginning and end of a JavaScript injection into JSX. Use them for anything you want to have evaluated.

```
const name = "Diana Prince";
const element = <div><h1>Hello, {name}</h1></div>;
```

JavaScript objects
You can also use a JavaScript object and dot notation. Remember the curly brackets anytime you want JavaScript evaluated in JSX.

```
const user = {
   firstName: "Clark",
   lastName: " Kent"
};
const element = <div><h1>Hello, {user.firstName + " " +
user.lastName}</h1></div>;
```

You can either use quotes for string values as attributes or curly braces to embed a JavaScript expression but not both in the same attribute.
A multi-line JSX expression should always be wrapped in parentheses.

Browsers don't understand JSX so it is converted to JavaScript at runtime. After compilation, JSX expressions become regular JavaScript objects.
JSX is compiled down to `React.createElement()` calls. So these two code snippets are really identical.

```
const element = (
  <h1 className="greeting"> Hello, world! </h1>
);

const element = React.createElement(
  'h1', {className: 'greeting'}, 'Hello, world!'
);
```

Then ReactDOM.render() renders the elements and updates the DOM.

Comments in JSX are similar to JavaScript if they're in the middle of a tag but if they're a separate child tag you need to wrap them in curly braces.

You can not inject an `if` statement or loop into a JSX expression. Instead use JSX inside of `if` statements and `for` loops, assign it to variables, accept it as arguments, and return it from functions.

Setup
There are two ways to use React:

- Use a JavaScript library to automatically convert JSX to JavaScript. You specify your JSX directly just like you would any old piece of JavaScript, and your browser takes care of the rest.
  - This is initially quicker but not used for real development
  - You'll see this a lot online including in Codepen and JSfiddle
- Set up a development environment around Node.js and a handful of build-tools. In this environment, every time you perform a build, all of your JSX is automatically converted into JS and saved on disk for you to reference like any plain JavaScript file.
  - Although this requires some setup, this is what we'll be using and that's why I had you install Node.js

Create React App is the best way to start building a new React single page application. It sets up your development environment so that you can use the latest JavaScript features, provides a nice developer experience, and optimizes your app for production.
https://create-react-app.dev/
Go into a directory where you want to create your React app.
https://create-react-app.dev/docs/getting-started
Create a project called hello-world:

```
npx create-react-app hello-world
```

If you get permission errors you'll need to do it as the root user.

```
sudo npx create-react-app hello-world
```

If you have an old version of create-react-app

```
npm uninstall -g create-react-app
npm install react-scripts@latest
```

I still had problems possibly related to an old version of create react-app somewhere and used

```
npx --ignore-existing create-react-app hello-world
```

Most recently I got the error:

```
gyp: No Xcode or CLT version detected!
```

To reset the path:

```
sudo xcode-select --reset
```

Then try to install again.

Then change into your new app directory and start your React app.

```
cd hello-world
npm start
```

This starts a development build and should open a browser window to http://localhost:3000/

Keep this window open so as you create your app and save your files you can refresh and see your changes here.

Control C to stop the process.

Go into the directory for your app (hello-world) and see what's there.
In public look at index.html.
It's a blank page right now but notice the div with id="root". That's where our React app will be rendered.
Only files inside public can be used from public/index.html.

Back in your app's directory go into the src directory. You'll see some css files and some js files.
Open the index.js and App.js files, these are the ones you'll be working in.
You CANNOT move or rename the public/index.html or src/index.js files.
You need to put any JS and CSS files inside src. You may create subdirectories inside src.

Note that you can't see the JavaScript source through view source.

Note: Previous versions of create-react-app included a service worker so you didn't need to refresh the web page. To set up your app with a service worker follow these instructions instead.

```
npx create-react-app my-app --template cra-template-pwa
```

In index.js update the service worker registration

```
serviceWorkerRegistration.register();
```

In your app's home directory (not src) create a file called .env with one line

```
FAST_REFRESH=false
```

Now when you start the process the service worker will automatically update the browser window everytime you make and save changes to your app.

DevTools
The React Devtools extension for Chrome and Firefox lets you inspect a React component tree in your browser devtools.
After installing it, you can right-click any element on the page, click "Inspect" to open the developer tools, and the dev tools will include Components and Profile tabs on the right.

Chrome:
Chrome web store, search for react developer tools
Add to Chrome
Add Extension
You should now see the React logo on the right side of the toolbar.

Firefox:
Menu | Add-ons
Search for react developer tools
Install, give permission to add
You should now see the React logo on the right side of the toolbar.