The Virtual DOM

When talking about React, it's important to understand how it renders information to the screen. One of the most vital concepts to understand with React is something called the Virtual DOM. In order to more easily understand this, let's talk about some other concepts outlined below:

Virtual DOM vs Traditional DOM

What we traditionally see in pure JavaScript is the DOM (Document Object Model). This is what we worked with for the last few weeks, and this is the main backbone of web technology for applications in browsers. There are some things to understand about the two first:

Real DOM

- What is directly on the screen. We see this when we change things with an `element.style.backgroundColor = 'blue';`
- Only way to actually change the UI for the user
- Because it doesn't handle comparison of changes well, it can lead to re-rendering of elements that didn't need to be changed. For example, look at these steps:
    1. You have a list of 5 items
    2. You change one item in the list using some sort of JS interaction
    3. The ENTIRE list on the browser gets re rendered directly via DOM Manipulation

With the above, it's not a huge issue with only a couple items in a list, but what happens if you have 100 items and you're continuously making changes to only one at a time? If you had 100 items in a list and made changes to 20 of them, that is (including the initial render of the 100 list items): 100 items rendered then 100 items rendered again for each of the 20 changes. In total that would be 2100 items rendered. That is a TON of work for the browser when only 20 items changed after initial render. Even more work than necessary if we were only changing a background color of each of those list items as opposed to the whole item itself.

Virtual DOM

- Sounds weird, but it's a representation of the DOM that is kept in JavaScript memory
- JS performs very quickly when not having to continuously render elements to the page, so the Virtual DOM can make changes to itself at a much faster speed
- Is used with some form of front-end framework such as React
- It CANNOT change the UI directly, but it is used in conjunction with the Real DOM. It tells the Real DOM when to re-render, and what the only which items to re-render. So continuing the above example, the steps would look like:
    1. You have a list of 5 items
    2. The Virtual DOM creates a copy of the Real DOM
    3. You change one thing with some JavaScript code
    4. The Virtual DOM compares any changes to what it currently has in memory (through a process of diffing and reconciliation)
    5. The Virtual DOM tells the DOM to re-render ONLY the item(s) that changed, and ONLY the property of the item that needed to change.

6. The Real DOM updates the page

So expanding on the above, because the Virtual DOM can tell the DOM when to render, it can massively improve performance. There are some issues to keep in mind with this, though, as there could be accidental renders if you're not careful with how you set up your React components. We'll talk more about that as we progress, but assuming you had the same 100 list items as above, the Virtual DOM would allow you to have only 120 items rendered. The initial 100 and then ONLY the 20 items that were changed. That is a MASSIVE improvement over the 2100 renders that happen with just the Real DOM.

Diffing and Reconciliation

When you're talking about the Virtual DOM you might be asking, well how does it know what to change, and when to tell the DOM to render items again? There is A LOT that goes into it, but the simple version is that whenever there is any change / function run / etc, React uses it's diffing algorithm to compare the Virtual DOM to the Real DOM. It does this by comparing the top level elements, any references to attributes, any `key` (which are a way to tell React that two items are effectively the same, even if they are in different parent elements), some other things, and then decides whether or not to tell the DOM to render something again.

Re-renders

One thing to keep in mind is WHEN the DOM is actually re-rendered. Don't worry too much about these for right now; I only wanted to include them so you'll have them as reference later. We talked a bit about diffing and reconciliation above, but an easier way to remember is it will re-render if one of the following happens:

1. State Changes - States are like variables, but for React.
2. Prop Changes - Properties are values passed to children to help them render data needed. Think of them like parameters in functions.
3. Parent Renders - This is the most straight forward. If a parent component renders, it will force a re-render to all of the components inside of it.
4. Context Change - Context is like state for your whole application.