React.js

React is a javascript library.

Disadvantage of Framework:

1. Not Flexible
2. Wants you to code in a certain way. If you want to deviate from that way, it ends up fighting you about it.
3. It has lot of features. If you want to use some small portion of it, you have to include the whole thing anyway.

React follow the Unix philosophy. Because it is small library and focus on one thing and on doing that thing extremely well.

User Interface: It is anything we put in front of users to have them interact with a machine. Since Web browser understand javascript, we can use React to describe Web UIs.

React is declarative. We say to react that what we want but not say that how to do that. React introduced the virtual DOM. DOM is “Document Object Model”. It’s the browsers’ programming interface for HTML (and XML) documents that treats them as tree structures. The DOM API can be used to change a document structure, style, and content.

If someone asked you to give **one** reason why React is worth learning, this outcomes-based UI language is it. I call this language “the React language”.

Any expensive operation on the DOM means a slow and janky experience for the user. It is extremely important that your applications do the absolute minimum operations and batch them where possible. React came up with a unique concept to help us do exactly that!

**How react works (React Tree Reconciliation) :** When we tell React to render a tree of elements in the browser, it first generates a virtual representation of that tree and keeps it around in memory for later. Then it’ll proceed to perform the DOM operations that will make the tree show up in the browser. When we tell React to update the tree of elements it previously rendered, it generates a new virtual representation of the updated tree. Now React has 2 versions of the tree in memory! To render the updated tree in the browser, React does not discard what has already been rendered. Instead, it will compare the 2 virtual versions of the tree that it has in memory, compute the differences between them, figure out what sub-trees in the main tree need to be updated, and only update these sub-trees in the browser. This process is what’s known as the tree reconciliation algorithm and it is what makes React a very efficient way to work with a browser’s DOM tree.

This is React’s smart **diffing** algorithm in action. It only updates in the main DOM tree what actually **needs** to be updated while it keeps everything else the same. This diffing process is possible because of React’s virtual DOM representation that it keeps around in memory. No matter how many times the UI views need to be regenerated, React will take to the browser only the needed “partial” updates

**React Components:** In React, we describe UIs using components that are reusable, composable, and stateful. We define small components and then put them together to form bigger ones. All components small or big are reusable, even across different projects. React components are exactly the same; their input is a set of “props” and their output is a description of a UI. We can reuse a single component in multiple UIs and components can contain other components. The basic form of a React component is actually a plain-old JavaScript function.

React will simply **react** to the state changes and automatically (and efficiently) update the DOM when needed.

This is **JSX**. It’s an **extension** to JavaScript that allows us to write function calls in an HTML-like syntax. JSX is basically a **compromise**. Instead of writing React components using the React.createElement syntax, we use a syntax very similar to HTML and then use a compiler to translate it into React.createElement calls. A compiler that translates one form of syntax into another is known as a “transpiler”. To translate JSX we can use transpilers like Babel or TypeScript. For example, the jsComplete playground uses TypeScript to transpile any JSX you put into it. When you use [create-react-app](https://github.com/facebook/create-react-app), the generated app will internally use Babel to transpile your JSX. So a React component is a JavaScript function that returns a React element (usually with JSX).

Components created with functions used to be limited in React. The only way to make a component “stateful” was to use the class syntax. This has changed with the release of “React Hooks” beginning with React version 16.8, which was released in early 2019. The React hooks release introduced a new API to make a function component stateful (and give it many other features).

* You can group related stateful logic and separate it into self-contained composable and sharable units. This makes it easier to break complex components into smaller parts. It also makes testing components easier.

**Benefits of Components:** First, components make your code more **readable** and easier to work with. Second, when things get more complex, this parsing of HTML becomes harder so components allow us to quickly understand what a UI represent using the language that we’re comfortable with. Thirdly, React components can also be **reused** in the same application and across multiple applications.

**React Hooks:** React hook functions can only be used in function components. You can’t use them in class components.