**REACT**

**React is a JavaScript library for building user interfaces.**

React has been designed from the start for gradual adoption, and you can use as little or as much React as you need. Whether you want to get a taste of React, add some interactivity to a simple HTML page, or start a complex React-powered app

Perhaps you only want to add some “sprinkles of interactivity” to an existing page. React components are a great way to do that. The majority of websites aren’t, and don’t need to be, single-page apps. With a few lines of code and no build tooling, try React in a small part of your website. You can then either gradually expand its presence, or keep it contained to a few dynamic widgets.

* Step 1: Add a DOM Container to the HTML
* Step 2: Add the Script Tags
* Step 3: Create a React Component

That’s It! You have just added the first React component to your website.

**React particularity**

* Virtual DOM (Virtual Document Object Model)
* Using JSX (a combination of JavaScript and XML code)
* Babel (JavaScript compiler)
* SPA (Single-page Application)
* Reconciliation

A single-page application is an application that loads a single HTML page and all the necessary assets (such as JavaScript and CSS) required for the application to run.

Babel is a JavaScript compiler.

JSX is a syntax extension to JavaScript. It is similar to a template language, but it has full power of JavaScript. JSX gets compiled to React.createElement() calls which return plain JavaScript objects called “React elements”.

The virtual DOM (VDOM) is a programming concept where an ideal, or “virtual”, representation of a UI is kept in memory and synced with the “real” DOM by a library such as ReactDOM.

When a component’s props or state change, React decides whether an actual DOM update is necessary by comparing the newly returned element with the previously rendered one. When they are not equal, React will update the DOM. This process is called “reconciliation”.

React implements a heuristic O(n) algorithm based on two assumptions:

1. Two elements of different types will produce different trees.
2. The developer can hint at which child elements may be stable across different renders with a key prop.

**Thinking in React**

React is the premier way to build big, fast Web apps with JavaScript. It has scaled very well for us at Facebook and Instagram. One of the many great parts of React is how it makes you think about apps as you build them.

An element describes what you want to see on the screen.

Applications built with just React usually have a single root DOM node. If you are integrating React into an existing app, you may have as many isolated root DOM nodes as you like.

React elements are immutable. Once you create an element, you can’t change its children or attributes. An element is like a single frame in a movie: it represents the UI at a certain point in time.

React DOM compares the element and its children to the previous one, and only applies the DOM updates necessary to bring the DOM to the desired state.

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.

All React components must act like pure functions with respect to their props. Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen.

Extracting components might seem like grunt work at first, but having a palette of reusable components pays off in larger apps. A good rule of thumb is that if a part of your UI is used several times (Button, Panel, Avatar), or is complex enough on its own (App, FeedStory, Comment), it is a good candidate to be a reusable component.

In React, you can create distinct components that encapsulate behavior you need. Then, you can render only some of them, depending on the state of your application.

Conditional rendering in React works the same way conditions work in JavaScript. Use JavaScript operators like if or the conditional operator to create elements representing the current state, and let React update the UI to match them.

In rare cases you might want a component to hide itself even though it was rendered by another component. To do this return null instead of its render output. Returning null from a component’s render method does not affect the firing of the component’s lifecycle methods.

React has a powerful composition model, and we recommend using composition instead of inheritance to reuse code between components.

At Facebook, we use React in thousands of components, and we haven’t found any use cases where we would recommend creating component inheritance hierarchies. Props and composition give you all the flexibility you need to customize a component’s look and behavior in an explicit and safe way. Remember that components may accept arbitrary props, including primitive values, React elements, or functions.

**JSX**

This funny tag syntax is neither a string nor HTML. It is called JSX, and it is a syntax extension to JavaScript. We recommend using it with React to describe what the UI should look like. JSX may remind you of a template language, but it comes with the full power of JavaScript. JSX produces React “elements”.

Embedding Expressions in JSX

JSX is an Expression Too

Specifying Attributes with JSX

Specifying Children with JSX

JSX Represents JavaScript-Objects

**Without JSX**

Do I need to use JSX with React?

JSX is not a requirement for using React.

Each JSX element is just syntactic sugar for calling React.createElement(component, props, ...children). So, anything you can do with JSX can also be done with just plain JavaScript.

This code written with JSX

This code does not use JSX