Glossary of React Terms

Single-page Application

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. Any interactions with the page or subsequent pages do not require a round trip to the server which means the page is not reloaded.

Though you may build a single-page application in React, it is not a requirement. React can also be used for enhancing small parts of existing websites with additional interactivity. Code written in React can coexist peacefully with markup rendered on the server by something like PHP, or with other client-side libraries. In fact, this is exactly how React is being used at Facebook.

ES6, ES2015, ES2016, etc

These acronyms all refer to the most recent versions of the ECMAScript Language Specification standard, which the JavaScript language is an implementation of. The ES6 version (also known as ES2015) includes many additions to the previous versions such as: arrow functions, classes, template literals, let and const statements. You can learn more about specific versions here.

Compilers

A JavaScript compiler takes JavaScript code, transforms it and returns JavaScript code in a different format. The most common use case is to take ES6 syntax and transform it into syntax that older browsers are capable of interpreting. Babel is the compiler most commonly used with React.

Bundlers

Bundlers take JavaScript and CSS code written as separate modules (often hundreds of them), and combine them together into a few files better optimized for the browsers. Some bundlers commonly used in React applications include Webpack and Browserify.

Package Managers

Package managers are tools that allow you to manage dependencies in your project. npm and Yarn are two package managers commonly used in React applications. Both of them are clients for the same npm package registry.

CDN

CDN stands for Content Delivery Network. CDNs deliver cached, static content from a network of servers across the globe.

JSX

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”. To get a basic introduction to JSX see the docs here and find a more in-depth tutorial on JSX here.

React DOM uses camelCase property naming convention instead of HTML attribute names. For example, tabindex becomes tabIndex in JSX. The attribute class is also written as className since class is a reserved word in JavaScript:

```
<h1 className="hello">My name is Clementine!</h1>
```

Elements

React elements are the building blocks of React applications. One might confuse elements with a more widely known concept of “components”. An element describes what you want to see on the screen. React elements are immutable.

```
const element = <h1>Hello, world</h1>;
```

Typically, elements are not used directly, but get returned from components.

Components

React components are small, reusable pieces of code that return a React element to be rendered to the page. The simplest version of React component is a plain JavaScript function that returns a React element:

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

Components can also be ES6 classes:

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Components can be broken down into distinct pieces of functionality and used within other components. Components can return other components, arrays, strings and numbers. 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. Component names should also always start with a capital letter (<Wrapper/> **not** <wrapper/>). See this documentation for more information on rendering components.

props

props are inputs to a React component. They are data passed down from a parent component to a child component.
Remember that props are readonly. They should not be modified in any way:

```
// Wrong!
props.number = 42;
```

If you need to modify some value in response to user input or a network response,
use state instead.

props.children

props.children is available on every component. It contains the content between the opening and closing tags of a component. For example:

```
<Welcome>Hello world!</Welcome>
```

The string Hello world! is available in props.children in the Welcome component:

```
function Welcome(props) {
  return <p>{props.children}</p>;
}
```

For components defined as classes, use this.props.children:

```
class Welcome extends React.Component {
  render() {
    return <p>{this.props.children}</p>;
  }
}
```

state

A component needs state when some data associated with it changes over time. For example, a Checkbox component might need isChecked in its state, and a NewsFeed component might want to keep track of fetchedPosts in its state.

The most important difference between state and props is that props are passed from a parent component, but state is managed by the component itself. A component cannot change its props, but it can change its state.

For each particular piece of changing data, there should be just one component that "owns" it in its state. Don't try to synchronize states of two different components. Instead, lift it up to their closest shared ancestor, and pass it down as props to both of them.

Lifecycle Methods
Lifecycle methods are custom functionality that gets executed during the different phases of a component. There are methods available when the component gets created and inserted into the DOM (mounting), when the component updates, and when the component gets unmounted or removed from the DOM.

Controlled vs. Uncontrolled Components

React has two different approaches to dealing with form inputs.
An input form element whose value is controlled by React is called a *controlled component*. When a user enters data into a controlled component a change event handler is triggered and your code decides whether the input is valid (by re-rendering with the updated value). If you do not re-render then the form element will remain unchanged.

An *uncontrolled component* works like form elements do outside of React. When a user inputs data into a form field (an input box, dropdown, etc) the updated information is reflected without React needing to do anything. However, this also means that you can't force the field to have a certain value.

In most cases you should use controlled components.

Keys
A "key" is a special string attribute you need to include when creating arrays of elements. Keys help React identify which items have changed, are added, or are removed. Keys should be given to the elements inside an array to give the elements a stable identity.

Keys only need to be unique among sibling elements in the same array. They don't need to be unique across the whole application or even a single component.

Don’t pass something like Math.random() to keys. It is important that keys have a “stable identity” across re-renders so that React can determine when items are added, removed, or re-ordered. Ideally, keys should correspond to unique and stable identifiers coming from your data, such as post.id.

Refs
React supports a special attribute that you can attach to any component. The ref attribute can be an object created by React.createRef() function or a callback function, or a string (in legacy API). When the ref attribute is a callback function, the function receives the underlying DOM element or class instance (depending on the type of element) as its argument. This allows you to have direct access to the DOM element or component instance.

Use refs sparingly. If you find yourself often using refs to “make things happen” in your app, consider getting more familiar with top-down data flow.

Events
Handling events with React elements has some syntactic differences:

- React event handlers are named using camelCase, rather than lowercase.
- With JSX you pass a function as the event handler, rather than a string.

Reconciliation
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 Semantics

Before I enlighten you with the mechanics of React I'd first like to define a few terms so as to grease the learning process.

Below I list the most common terms, and their definitions, used when talking about React.

Babel

Babel transforms JavaScript ES* (i.e., JS 2016, 2016, 2017) to ES5. Babel is the tool of choice from the React team for writing future ES* code and transforming JSX to ES5 code.

Babel CLI

Babel comes with a CLI tool, called Babel CLI, that can be used to compile files from the command line.

Component Configuration Options (a.k.a, "Component Specifications")

The configuration arguments passed (as an object) to the `React.createClass()` function resulting in an instance of a React component.

Component Life Cycle Methods

A sub group of component events, semantically separated from the other component configuration options
(i.e., `componentWillUnmount`, `componentDidUpdate`, `componentWillUpdate`, `shouldComponentUpdate`, `componentWillReceiveProps`, `componentDidMount`, `componentWillMount`). These methods are executed at specific points in a component's existence.

Document Object Model (a.k.a., DOM)

"The Document Object Model (DOM) is a programming interface for HTML, XML and SVG documents. It provides a structured representation of the document as a tree. The DOM defines methods that allow access to the tree, so that they can change the document structure, style and content. The DOM provides a representation of the document as a structured group of nodes and objects, possessing various properties and methods. Nodes can also have event handlers attached to them, and once an event is triggered, the event handlers get executed. Essentially, it connects web pages to scripts or programming languages." - MSD

ES5

The 5th edition of the ECMAScript standard. The ECMAScript 5.1 edition was finalized in June 2011.

ES6/ES 2015

The 6th edition of the ECMAScript standard. A.k.a, JavaScript 2015 or ECMAScript 2015. The ECMAScript 6th edition was finalized in June 2015.

ECMAScript 2016 (a.k.a, ES7)

The 7th edition of the ECMAScript standard. The ECMAScript 7th edition was finalized in June 2016.

---

*ES**

Used to represent the current version of JavaScript as well as potential future versions that can written today using tools like Babel. When you see "ES*" it more than likely means you'll find uses of ES5, ES6, and ES7 together.

JSX

JSX is an optional XML-like syntax extension to ECMAScript that can be used to define an HTML-like tree structure in a JavaScript file. The JSX expressions in a JavaScript file must be transformed to JavaScript syntax before a JavaScript engine can parse the file. Babel is typically used and recommended for transforming JSX expressions.

Node.js

Node.js is an open-source, cross-platform runtime environment for writing JavaScript. The runtime environment interprets JavaScript using Google's V8 JavaScript engine.

---

npm

npm is the package manager for JavaScript born from the Node.js community.

React Attributes/Props

In one sense you can think of props as the configuration options for React nodes and in another sense you can think of them as HTML attributes.

Props take on several roles:

1. Props can become HTML attributes. If a prop matches a known HTML attribute then it will be added to the final HTML element in the DOM.
2. Props passed to `createElement()` become values stored in a `prop` object as an instance property of `React.createElement()` instances (i.e., `[INSTANCE].props.[NAME OF PROP]`). Props by and large are used to input values into components.
3. A few special props have side effects (e.g., `key`, `ref`, and `dangerouslySetInnerHTML`)

React

React is an open source JavaScript library for writing declarative, efficient, and flexible user interfaces.

React Component

A React component is created by calling `React.createClass()` (or, `React.Component` if using ES6 classes). This function takes an object of options that is used to configure and create a React component. The most common configuration option is the `render` function which returns React nodes. Thus, you can think of a React component as an abstraction containing one or more React nodes/components.

React Element Nodes (a.k.a., `ReactElement`)
An HTML or custom HTML element node representation in the Virtual DOM created using `React.createElement();`.

React Nodes

React nodes (i.e., element and text nodes) are the primary object type in React and can be created using `React.createElement('div');`. In other words React nodes are objects that represent DOM nodes and children DOM nodes. They are a light, stateless, immutable, virtual representation of a DOM node.

React Node Factories

A function that generates a React element node with a particular type property.

React Stateless Function Component

When a component is purely a result of props alone, no state, the component can be written as a pure function avoiding the need to create a React component instance.

```
var MyComponent = function(props){
    return <div>Hello {props.name}</div>;
};

ReactDOM.render(<MyComponent name="doug" />, app);
```

React Text Nodes (a.k.a., `ReactText`)
A text node representation in the Virtual DOM created using `React.createElement('div',null,'a text node');`.

Virtual DOM

An in-memory JavaScript tree of React elements/components that is used for efficient re-rendering (i.e., diffing via JavaScript) of the browser DOM.

Webpack

Webpack is a module loader and bundler that takes modules (.js, .css, .txt, etc.) with dependencies and generates static assets representing these modules.

Need to take a few steps back and figure out what is react javascript and how you can use it?

#1 Library

You have probably seen a hundred comparing articles of the best Javascript frontend frameworks. And React is always there. But that is technically incorrect since when talking about it, we're dealing with a library.

A library is simply a collection of class definitions, helping its user to accomplish a specific task. In the case of React, to define UIs.

#2 Components

React components are very similar to functions. They accept input, in the form of props, and then produce some output. This output defines what should appear on the screen. Components are great to build UIs, because they let you define independent pieces of it, and you can use and handle them however you want.

A component can be either a class component, meaning it is basically a class which will return something to render on screen

```
class Greetings extends React.Component {
  render() {
    return <h1>Hello world!</h1>
  }
}
```

Or similarly, a function. Which has the same purpose of describing the UI, but with some difference when it comes to other factors such as lifecycle methods or hooks.

```
function greetings(props) {
  return <h1>Hello world</h1>
}
```

#2 Props

Nothing is truly fun if not customizable a little right? Well just like how you pass data to a `add(val1, val2)` function to add numbers, you pass them to React components.

```
class Greetings extends React.Component {
 render() {
   return <h1>Hello {this.props.name}</h1>
 }
}
```

Props are just that, meaningful input you pass to your React components.

#3 State

A component needs state when it requires some way to store local data, which can change over time. Like an isChecked info in the state for a CheckBox component. Or show inside the state of an Accordion .

And remember, the biggest difference between state and props is where they come from. Props come from a parent component, while state is managed by the component you're defining itself. Props can't be changed, but the state can.

```
class usingState extends React.Component {
  state = {
    name: 'Piero Borrelli'
  }

  render() {
    return <h1>Hello {this.state.name}!</h1>
  }
}
```

#4 JSX

React components are not defined in a syntax you are probably familiar with. React uses JSX, which is a syntax extension for Javascript. Your JSX will get compiled to React.createElement() calls, which will return plain JavaScript objects called "React elements".

```
const thisIsJsx = <p>this is a combination of Js and html </p>
```

#5 Javascript Compiler

A Javascript compiler takes Javascript code, transforms it, and then return it under a different format. The most common case is when you want to use the newest ES6 features, which are still not supported by older browsers. Babel is the compiler most commonly used with React.

#6 ES6

Acronym referring to one of the latest versions of the ECMAScript Language Specification standard, which the JavaScript language is an implementation of. This version included many news such as let and const, arrow functions, and template literals.

#7 Bundler

A bundler helps you taking different JS and CSS modules, and combine them into just a few files better optimized for the browsers. The most common bundler used with React is Webpack.

#8 Elements

React elements are the building blocks of React applications. One might confuse elements with components. An element describes what you want to see on the screen. And remember, elements are immutable.

```
const element = <h1>Hello, element</h1>
```

#9 Lifecycle Methods

Components can be considered just like living creatures. They go through stages of their lives, and we can access these stages via lifecycle methods. For example, when a component is born, we call it mounting and use the `componentDidMount()` method to manage this part of its lifecycle. Or when it changes, we say it updates, and so we use the `componentDidUpdate()`.

- **Single-page Application** – An application that loads a single HTML page and all the necessary assets (such as Javascript and CSS) required for the application to run.
- **ES6,ES2015,ES2016** – Most recent versions of the ECMAScript Language Specification standard, which is the javascript language is an implementation of.
- **Compiler** – A Javascript compiler takes JavaScript code, transforms it and returns JavaScript code in a different format.

- **Bundlers** – Bundlers takes JavaScript and CSS code written as separate modules and compile them together into a few files better optimized for the browser.
- **Package Managers** – Package Managers are tools that allow you to manage dependencies in your project.
- **CDN** – CDN stands for Content Delivery Network. CDNs deliver cached, static content from a network of servers across the globe.
- **Elements** – React elements are the building blocks of React applications. An elements describes what you want to see on the screen.React elements are immutable.
- **JSX –** Allows us to write HTML like syntax which gets transformed to lightweight Javascript objects.
- **Virtual DOM –** A Javascript representation of the actual DOM.
- **React.Component** – The way in which you create a new component.
- **render method** – Describes what the UI will look like for the particular component.
- **ReactDOM.render** – Renders a React component to a DOM node.
- **State** – The internal data store (Object) of a component.
- **constructor (this.state)** – The way in which you establish the initial state of a component.
- **setState** – A helper method used for updating the state of a component and re-rendering the UI.
- **props** – The data which is passed to the child component from the parent component.
- **propTypes** – Allow you to control the presence or types of certain props passed to the child component.
- **defaultProps** – Allows you set default props for your component.
- **props.children** – Available on every component. It contains the content between the opening and closing tags of a component.
- **Component Lifecycle**
    - **componentDidMount** – Fired before the component mounted.
    - **componentWillUnmount** – Fired after the component will unmount.
    - **getDerivedStateFromProps** – This method fired when the component mounts and whenever the props change. Used to update the state of a component when its props change.
- **Events**
    - onClick
    - onSubmit
    - onChange
- **Controlled Component** – An input form element whose value is controlled by React

- **Uncontrolled Component** – An uncontrolled component works like form elements do outside of React