**What is React?**

React (aka React.js or ReactJS) is a front-end and open-source JavaScript library which is useful in developing user interfaces specifically for applications with a single page. It is helpful in building complex and reusable user interface (UI) components of web applications as it follows the component-based approach.

React was created by [Jordan Walke](https://github.com/jordwalke), a software engineer working for Facebook. React was first deployed on Facebook's News Feed in 2011 and on Instagram in 2012.

**What are the major features of React?**

The major features of React are:

* Uses **JSX** syntax, a syntax extension of JS that allows developers to write HTML in their JS code.
* It uses **Virtual DOM** instead of Real DOM considering that Real DOM manipulations are expensive.
* Supports **server-side rendering** which is useful for Search Engine Optimizations(SEO).
* Follows **Unidirectional or one-way** data flow or data binding.
* Uses **reusable/composable** UI components to develop the view.

**Mention some limitations of React?**

1. React is just a view library, not a full-blown framework
2. There is a learning curve for beginners who are new to web development.
3. Integrating React.js into a traditional MVC framework requires some additional configuration
4. The code complexity increases with inline templating and JSX.
5. Too many smaller components lead to over-engineering or boilerplate

**How does React work?**

React creates a virtual DOM. When state changes in a component it firstly runs a "diffing" algorithm, which identifies what has changed in the virtual DOM. The second step is reconciliation, where it updates the DOM with the results of the difference.

**What is JSX?**

*JSX* stands for *JavaScript XML* and it is a syntax extension to JavaScript.

React JSX helps us to write HTML in JavaScript and forms the basis of React Development. Using JSX is not compulsory but it is highly recommended for programming in React as it makes the development process easier as the code becomes easy to write and read.

In the example below, the text inside <h1> tag is returned as JavaScript function to the render function.

JSX Example:

export default function App() {

return (

<h1 className="greeting">Hello, this is a JSX Code!</h1>

);

}

If you don't use JSX syntax then the respective JavaScript code should be written as below.

import { createElement } from 'react';

export default function App() {

return createElement(

'h1',

{className: 'greeting'},

'Hello, this is a JSX Code!'

);

}

### What is the difference between Element and Component?

An Element is a plain object describing what you want to appear on the screen in terms of the DOM nodes. Elements can contain other Elements in their props. Creating a React element is cheap. Once an element is created, it cannot be mutated.

The JavaScript representation(Without JSX) of React Element would be as follows:

const element = React.createElement("div", { id: "login-btn" }, "Login");

and this element can be simiplified using JSX

<div id="login-btn">Login</div>

The above React.createElement() function returns an object as below:

{

type: 'div',

props: {

children: 'Login',

id: 'login-btn'

}

}

Finally, this element renders to the DOM using ReactDOM.render().

Whereas a **component** can be declared in several different ways. It can be a class with a render() method or it can be defined as a function. In either case, it takes props as an input, and returns a JSX tree as the output:

const Button = ({ handleLogin }) => (

<div id={"login-btn"} onClick={handleLogin}>

Login

</div>

);

Then JSX gets transpiled to a React.createElement() function tree:

const Button = ({ handleLogin }) =>

React.createElement(

"div",

{ id: "login-btn", onClick: handleLogin },

"Login"

);

### How to create components in React?

Components are the building blocks of creating User Interfaces(UI) in React. There are two possible ways to create a component.

1. **Function Components:** This is the simplest way to create a component. Those are pure JavaScript functions that accept props object as the first parameter and return React elements to render the output:

function Greeting({ message }) {

return <h1>{`Hello, ${message}`}</h1>;

}

1. **Class Components:** You can also use ES6 class to define a component. The above function component can be written as a class component:

class Greeting extends React.Component {

render() {

return <h1>{`Hello, ${this.props.message}`}</h1>;

}

}

**When to use a Class Component over a Function Component?**

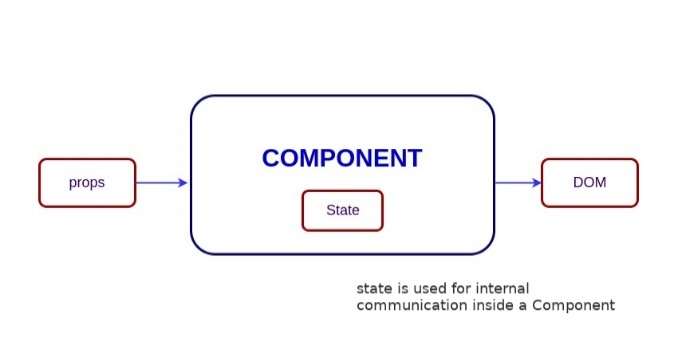
After the addition of Hooks(i.e. React 16.8 onwards) it is always recommended to use Function components over Class components in React. Because you could use state, lifecycle methods and other features that were only available in class component present in function component too.

But even there are two reasons to use Class components over Function components.

* If you need a React functionality whose Function component equivalent is not present yet, like Error Boundaries.
* In older versions, If the component needs state or lifecycle methods then you need to use class component.

**What is state in React?**

State of a component is an object that holds some information that may change over the lifetime of the component. The important point is whenever the state object changes, the component re-renders. It is always recommended to make our state as simple as possible and minimize the number of stateful components.

[](https://github.com/sudheerj/reactjs-interview-questions/blob/master/images/state.jpg)

Let's take an example of **User** component with message state. Here, **useState** hook has been used to add state to the User component and it returns an array with current state and function to update it.

import React, { useState } from "react";

function User() {

  const [message, setMessage] = useState("Welcome to React world");

  return (

    <div>

      <h1>{message}</h1>

    </div>

  );

}

**What are Pure Components?**

**Pure** **components** are a specialized type of components in React that optimize [rendering](https://www.educative.io/answers/what-is-render-in-react-javascript) performance by implementing a shallow comparison of [props](https://www.educative.io/answers/what-are-props-in-react) and state. They do not re-render when the value of state and props has been updated with the same values.

***Shallow comparison***

*When a pure component receives new props or its state is updated, React automatically performs a shallow comparison of the new and previous props and state. It checks if there are any differences between the old and new values. If there are no changes, React prevents unnecessary re-rendering of the component and its child components. This comparison is known as a shallow comparison because it checks for equality at the top level.*

The syntactic representation of memoized components looks like below,

**const MemoizedComponent = memo(SomeComponent, arePropsEqual?);**

Below is the example of how child component(i.e., EmployeeProfile) prevents re-renders for the same props passed by parent component(i.e.,EmployeeRegForm).

import { memo, useState } from 'react';

const EmployeeProfile = memo(function EmployeeProfile({ name, email }) {

  return (<>

        <p>Name:{name}</p>

        <p>Email: {email}</p>

        </>);

});

export default function EmployeeRegForm() {

  const [name, setName] = useState('');

  const [email, setEmail] = useState('');

  return (

    <>

      <label>

        Name: <input value={name} onChange={e => setName(e.target.value)} />

      </label>

      <label>

        Email: <input value={email} onChange={e => setEmail(e.target.value)} />

      </label>

      <hr/>

      <EmployeeProfile name={name} email={email}/>

    </>

  );

}

### What are props in React?

Props are inputs to components. They are single values or objects containing a set of values that are passed to components on creation similar to HTML-tag attributes. Here, the data is passed down from a parent component to a child component.

const ChildComponent = (props) => {

  return (

    <div>

      <p>{props.name}</p>

      <p>{props.age}</p>

    </div>

  );

};

const ParentComponent = () => {

  return (

    <div>

      <ChildComponent name="John" age="30" />

      <ChildComponent name="Mary" age="25" />

    </div>

  );

};

### What is the difference between state and props?

In React, both state and props are plain JavaScript objects and used to manage the data of a component, but they are used in different ways and have different characteristics.

state is managed by the component itself and can be updated using the setState() function. Unlike props, state can be modified by the component and is used to manage the internal state of the component. Changes in the state trigger a re-render of the component and its children.

props (short for "properties") are passed to a component by its parent component and are read-only, meaning that they cannot be modified by the component itself. props can be used to configure the behaviour of a component and to pass data between components.

**What is DOM ?**

DOM stands for ‘Document Object Model’. In simple terms, it is a structured representation of the HTML elements that are present in a webpage or web app. DOM represents the entire UI of your application. The DOM is represented as a tree data structure. It contains a node for each UI element present in the web document. It is very useful as it allows web developers to modify content through JavaScript, also it being in structured format helps a lot as we can choose specific targets and all the code becomes much easier to work with.

**Disadvantages of real DOM**

Every time the DOM gets updated, the updated element and its children have to be rendered again to update the UI of our page. For this, each time there is a component update, the DOM needs to be updated and the UI components have to be re-rendered.

**What is Virtual DOM?**

The Virtual DOM (VDOM) is an in-memory representation of Real DOM. The representation of a UI is kept in memory and synced with the "real" DOM. It's a step that happens between the render function being called and the displaying of elements on the screen. This entire process is called reconciliation.

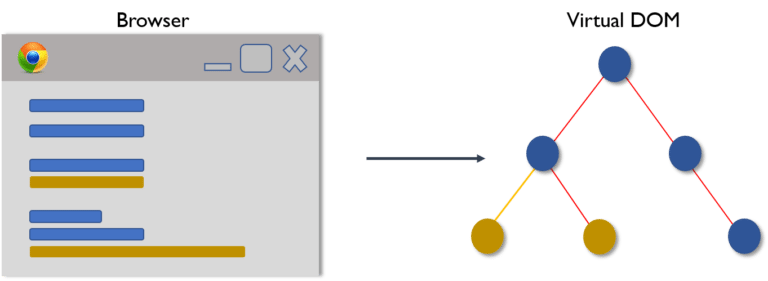
**Differences between Virtual DOM and Real DOM**

| Virtual DOM | Real DOM |
| --- | --- |
| It is a lightweight copy of the original DOM | It is a tree representation of HTML elements |
| It is maintained by JavaScript libraries | It is maintained by the browser after parsing HTML elements |
| After manipulation it only re-renders changed elements | After manipulation, it re-render the entire DOM |
| Updates are lightweight | Updates are heavyweight |
| Performance is fast and UX is optimised | Performance is slow and the UX quality is low |
| Highly efficient as it performs batch updates | Less efficient due to re-rendering of DOM after each update |

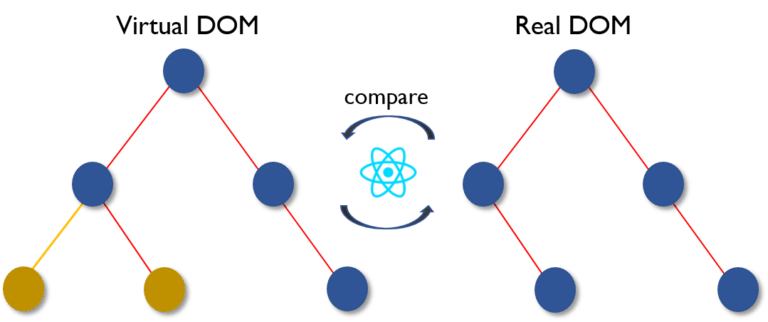
### How Virtual DOM works?

The Virtual DOM works in three simple steps.

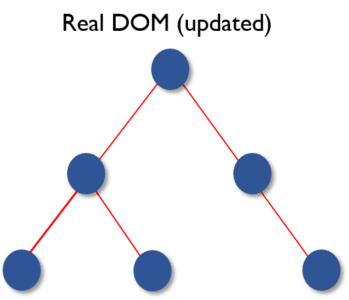
1. Whenever any underlying data changes, the entire UI is re-rendered in Virtual DOM representation.

[](https://github.com/sudheerj/reactjs-interview-questions/blob/master/images/vdom1.png)

1. Then the difference between the previous DOM representation and the new one is calculated.

[](https://github.com/sudheerj/reactjs-interview-questions/blob/master/images/vdom2.png)

1. Once the calculations are done, the real DOM will be updated with only the things that have actually changed.

[](https://github.com/sudheerj/reactjs-interview-questions/blob/master/images/vdom3.png)