**What is ReactJS?**

**ReactJS**is an open-source JavaScript library used to create user interfaces in a declarative and efficient way. It is a component-based front-end library responsible only for the view layer of a **Model View Controller(MVC)** architecture. React is used to create modular user interfaces and promotes the development of reusable UI components that display dynamic data.

**Concept of ReactJS**

Let’s say one of your friends posted a photo on Facebook. If you like the photo and then you started checking out the comments too. Now as you browsing over comments you can see that the likes count has increased by 100 since you liked the picture, even without reloading the page. This magical change in count is due to ReactJS.

## History of ReactJS

While building client-side application, a team of Facebook developers found that DOM is slow. Document Object Model (DOM) is an application programming interface(API) for HTML and XML documents. It defines the logical structure of documents and how a document is accessed and manipulated). To make it faster, React implements a virtual DOM which is basically a DOM tree representation in JavaScript, and React was invented.

## How does ReactJS work?

React creates a virtual DOM in memory to update the browser’s DOM. The virtual DOM will try to find the most efficient way to update the browser’s DOM.

Unlike browser DOM elements, React elements are simple objects and are cheap to create. React DOM takes care of updating the DOM to match the React elements. The reason for this is that JavaScript is very fast and it’s worth keeping a DOM tree in it to speed up its processing.

Although React was designed to be used in the browser, because of its design allows it to be used on the server with Node.js as well.

# ReactJS JSX Introduction

React JS JSX is a syntax extension of JavaScript for writing React Code in a simple way. Using JSX it is easier to create reusable UI components with fewer lines of code in a template-type language with the power of JavaScript.

## What is React JSX ?

**JSX** stands for **JavaScript XML**. JSX is basically a syntax extension of JavaScript. It 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.

JSX creates an element in React that gets rendered in the UI. It is transformed into JavaScript functions by the compiler at runtime. Error handling and warnings become easier to handle when using JSX

### **Sample JSX code:**

const ele = <h1>This is sample JSX</h1>;

The above code snippet somewhat looks like HTML and it also uses a JavaScript-like variable but is neither HTML nor JavaScript, it is JSX(JavaScript XML). With the help of JSX, we have directly written the HTML syntax in JavaScript

## Why JSX ?

* It is faster than normal JavaScript as it performs optimizations while translating to regular JavaScript.
* It makes it easier for us to create templates.
* Instead of separating the markup and logic in separate files, React uses components for this purpose. We will learn about components in detail in further articles.
* As JSX is an expression, we can use it inside of if statements and for loops, assign it to variables, accept it as arguments, or return it from functions.

## ****Expressions in JSX****

In React we are allowed to use normal JavaScript expressions with JSX. To embed any JavaScript expression in a piece of code written in JSX we will have to wrap that expression in curly braces {}. The below example specifies a basic use of JavaScript Expression in React.

### **Syntax:**

const example = "JSX"  
const ele = <div>This component uses {example} </div>

// Filename - App.js

import React from "react";

const name = "Learner";

const element = (

<h1>

Hello,

{name}.Welcome to GeeksforGeeks.

</h1>

);

ReactDOM.render(element, document.getElementById("root"));

**Output:**

A black and blue text

Description automatically generated

*ReactJS JSX*

In the above program, we have embedded the javascript expression *const name = “Learner”;* in our JSX code. We can use conditional statements instead of if-else statements in JSX.

**Example:**In this example where conditional expression is embedded in JSX:

import React from "react";

let i = 1;

const element = (

    <h1>{i == 1 ? "Hello World!" : "False!"} </h1>

);

ReactDOM.render(element, document.getElementById("root"));

In the above example, the variable ‘i’ is checked if for the value 1. As it equals 1 so the string ‘Hello World!’ is returned to the JSX code. If we modify the value of the variable ‘i’ then the string ‘False!’ will be returned.

**Attributes in JSX**

JSX allows us to use attributes with the HTML elements just like we do with normal HTML. But instead of the normal naming convention of HTML, JSX uses the camelcase convention for attributes.&

* **The change of class attribute to className:**The*class* in HTML becomes *className* in JSX. The main reason behind this is that some attribute names in HTML like ‘*class*‘ are reserved keywords in JavaScript. So, in order to avoid this problem, JSX uses the camel case naming convention for attributes.
* **Creation of custom attributes:**We can also use custom attributes in JSX. For custom attributes, the names of such attributes should be prefixed by **data-\***attribute.

**Example:**This example has a custom attribute with the <h2> tag and we are using className attribute instead of class.

* javascript

import React from "react";

import ReactDOM from "react-dom";

const element = (

    <div>

        <h1 className="hello">Hello Geek</h1>

        <h2 data-sampleAttribute="sample">

            Custom attribute

        </h2>

    </div>

);

ReactDOM.render(element, document.getElementById("root"));

## ****Comments in JSX :****

JSX allows us to use comments as it allows us to use JavaScript expressions. Comments in JSX begin with **/\*** and ends with **\*/**. We can add comments in JSX by wrapping them in curly braces {} just like we did in the case of expressions. The below example shows how to add comments in JSX:

import React from "react";

import ReactDOM from "react-dom";

const element = (

    <div>

        <h1>Hello World !{/\*This is a comment\*/}</h1>

    </div>

);

ReactDOM.render(element, document.getElementById("root"));

## ****Virtual DOM****

React uses Virtual DOM exists which is like a lightweight copy of the actual DOM(a virtual representation of the DOM). So for every object that exists in the original DOM, there is an object for that in React Virtual DOM. It is exactly the same, but it does not have the power to directly change the layout of the document.

**Manipulating DOM is slow, but manipulating Virtual DOM is fast** as nothing gets drawn on the screen. So each time there is a change in the state of our application, the virtual DOM gets updated first instead of the real DOM.

**How does virtual DOM actually make things faster?**

When anything new is added to the application, a virtual DOM is created and it is represented as a tree. Each element in the application is a node in this tree. So, whenever there is a change in the state of any element, a new Virtual DOM tree is created. This new Virtual DOM tree is then compared with the previous Virtual DOM tree and make a note of the changes. After this, it finds the best possible ways to make these changes to the real DOM. Now only the updated elements will get rendered on the page again.

**How virtual DOM Helps React?**

In React, everything is treated as a component be it a [functional component](https://www.geeksforgeeks.org/reactjs-functional-components/) or [class component.](https://www.geeksforgeeks.org/reactjs-class-based-components/) A component can contain a state. Whenever the state of any component is changed react updates its Virtual DOM tree. Though it may sound like it is ineffective the cost is not much significant as updating the virtual DOM doesn’t take much time.

React maintains two Virtual DOM at each time, one contains the updated Virtual DOM and one which is just the pre-update version of this updated Virtual DOM. Now it compares the pre-update version with the updated Virtual DOM and figures out what exactly has changed in the DOM like which components have been changed. This process of comparing the current Virtual DOM tree with the previous one is known as[**‘diffing’**](https://www.geeksforgeeks.org/explain-dom-diffing/). Once React finds out what exactly has changed then it updates those objects only, on real DOM.

React uses something called batch updates to update the real DOM. It just means that the changes to the real DOM are sent in batches instead of sending any update for a single change in the state of a component.

We have seen that the re-rendering of the UI is the most expensive part and React manages to do this most efficiently by ensuring that the Real DOM receives batch updates to re-render the UI. This entire process of transforming changes to the real DOM is called [**Reconciliation**](https://www.geeksforgeeks.org/reactjs-reconciliation/)**.**

This significantly improves the performance and is the main reason why React and its Virtual DOM are much loved by developers all around.

The diagrammatic image below briefly describes how the virtual DOM works in the real browser environment

*Real DOM to Virtual DOM*

**Virtual DOM Key Concepts :**

* Virtual DOM is the virtual representation of Real DOM
* React update the state changes in Virtual DOM first and then it syncs with Real DOM
* Virtual DOM is just like a blueprint of a machine, can do changes in the blueprint but those changes will not directly apply to the machine.
* Virtual DOM is a programming concept where a virtual representation of a UI is kept in memory synced with “Real DOM ” by a library such as ReactDOM and this process is called reconciliation
* Virtual DOM makes the performance faster, not because the processing itself is done in less time. The reason is the amount of changed information – rather than wasting time on updating the entire page, you can dissect it into small elements and interactions

**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 |

# ReactJS Lists

import React from 'react';

import ReactDOM from 'react-dom';

const numbers = [1,2,3,4,5];

const updatedNums = numbers.map((number)=>{

**return** <li>{number}</li>;

});

ReactDOM.render(

    <ul>

        {updatedNums}

    </ul>,

    document.getElementById('root')

);

## ****Rendering lists inside Components****

import React from 'react';

import ReactDOM from 'react-dom';

// Component that will return an

// unordered list

**function** Navmenu(props)

{

    const list = props.menuitems;

    const updatedList = list.map((listItems)=>{

**return** <li>{listItems}</li>;

    });

**return**(

        <ul>{updatedList}</ul>

    );

}

const menuItems = [1,2,3,4,5];

ReactDOM.render(

    <Navmenu menuitems = {menuItems} />,

    document.getElementById('root')

);

import React from 'react';

import ReactDOM from 'react-dom';

// Component that will return an

// unordered list

**function** Navmenu(props)

{

    const list = props.menuitems;

    const updatedList = list.map((listItems)=>{

**return** <li>{listItems}</li>;

    });

**return**(

        <ul>{updatedList}</ul>

    );

}

const menuItems = [1,2,3,4,5];

ReactDOM.render(

    <Navmenu menuitems = {menuItems} />,

    document.getElementById('root')

);

<https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs>

<https://www.turing.com/interview-questions/react-js>