React Interview Questions and Answers

Last Updated : 24 Jun, 2025

React is an efficient, flexible, and open-source **JavaScript library** that allows developers to create simple, fast, and scalable web applications. Jordan Walke, a software engineer who was working for Facebook, created React. Developers with a JavaScript background can easily develop web applications with React.

In this Top React Interview Questions article, we've covered the **70+ Interview Questions of React** that cover everything from basic to advanced React concepts such as **Virtual DOM, Components, State and Props, JSX, Hooks, Routing**, and more. Whether you are a **fresher** or an **experienced professional with 2 - 10 years of experience**, these React Interview Questions give you all the confidence you need to ace your next technical interview.

React Interview Questions For Freshers

Let's discuss some common questions that you should prepare for the interviews. These questions will help clear the interviews, especially for the front-end development role.

1. What is ReactJS?

ReactJS is a JavaScript library used to build reusable components for the view layer in the MVC architecture. It is used to build the Single Page Application (SPA) due to its component-based architecture, efficient re-rendering with the Virtual DOM, and ability to manage dynamic content without needing full page reloads. It is written in JSX.

Important Features of React:

- **Virtual DOM:** React uses a virtual DOM to efficiently update and render components, ensuring fast performance by minimizing direct DOM manipulations.
- **Component-Based Architecture:** React builds UI using reusable, isolated components, making code more modular, maintainable, and scalable.
- **Hooks**: React Hooks allow functional components to manage state and side effects, making them powerful and more flexible.
- **Server-Side Rendering (SSR)**: React can be used for server-side rendering, where HTML content is generated on the server and sent to the client. This improves the app's performance, especially for SEO.
- **React Router:** React Router enables navigation in a React application. It allows you to define different routes for different views in a single-page application (SPA).

2. What is the latest version of the React?

The latest stable version of React is **v19.1.0**, released on **March 28, 2025** . This version builds upon the major updates introduced in **React v19.0.0**, which was officially released on **December 5, 2024**.

3. Explain the MVC architecture.

The Model-View-Controller (MVC) framework is an architectural/design pattern that separates an application into three main logical components: Model, View, and Controller. Each architectural component is built to handle specific development aspects of an application. It isolates the business, logic, and presentation layers from each other

4. Explain the building blocks of React.

The five main building blocks of React are

- **Components:** These are reusable blocks of code that return HTML.
- **JSX:** It stands for JavaScript and XML and allows you to write HTML in React.
- **Props and State:** props are like function parameters and State is similar to variables.
- **Context:** This allows data to be passed through components as props in a hierarchy.
- **Virtual DOM:** It is a lightweight copy of the actual DOM which makes DOM manipulation easier.

5. Explain props and state in React with differences

Props are used to pass data from one component to another. The state is local data storage that is local to the component only and cannot be passed to other components.

Here is the difference table of props and state In react

- **Virtual DOM:** It is a lightweight copy of the actual DOM which makes DOM manipulation easier.

5. Explain props and state in React with differences

Props are used to pass data from one component to another. The state is local data storage that is local to the component only and cannot be passed to other components.

Here is the difference table of props and state In react

PROPS	STATE
The Data is passed from one component to another.	The Data is passed within the component only.
It is Immutable (cannot be modified).	It is Mutable (can be modified).
Props can be used with state and functional components.	The state can be used only with the state components/class component (Before 16.0).
Props are read-only.	The state is both read and write.

6. What is virtual DOM in React?

The Virtual DOM in React is an in-memory representation of the actual DOM. It helps React efficiently update and render the user interface by comparing the current and previous virtual DOM states using a process called **diffing**.

How Virtual DOM Works

- **Efficient Rendering**: The Virtual DOM is an in-memory representation of the actual DOM that React uses to optimize the process of updating and rendering UI changes.
- **Diffing Algorithm**: React compares the current and previous versions of the Virtual DOM using a diffing algorithm, identifying the minimal set of changes required to update the real DOM.
- **Batch Updates**: Instead of updating the real DOM immediately, React batches multiple changes to reduce unnecessary re-renders, improving performance.
- **Faster Updates**: Since updating the real DOM is slow, React minimizes direct DOM manipulations by only making updates where necessary after comparing the Virtual DOM.
- **Declarative UI**: With the Virtual DOM, React allows developers to write code in a declarative style, letting React handle when and how to efficiently update the UI.

7. Differentiate between Real DOM and virtual DOM?

Real DOM	Virtual DOM
The actual DOM, a tree-like structure representing the UI elements.	A lightweight copy of the Real DOM used to optimize updates.
Slower as it requires direct updates to the actual DOM.	Faster because it minimizes direct manipulation of the Real DOM.
Directly manipulates the Real DOM, causing re-rendering of the entire UI.	Updates are made to the Virtual DOM first, then changes are batched and only the necessary changes are reflected in the Real DOM.
Entire UI might need to be re-rendered when changes occur.	Only the necessary components are re-rendered, reducing unnecessary re-renders.
Less efficient due to repeated direct updates to the Real DOM.	More efficient by minimizing direct DOM manipulation and batch updates.

8. What is JSX?

JSX is basically a syntax extension of regular JavaScript and is used to create React elements. These elements are then rendered to the React DOM. All the React components are written in JSX. To embed any JavaScript expression in a piece of code written in JSX we will have to wrap that expression in curly braces {}.

Example of JSX: The name written in curly braces { } signifies JSX

```
const name = "Learner";
const element = (
    <h1>
        Hello,
```

updates to the Real DOM. batch updates.

8. What is JSX?

JSX is basically a syntax extension of regular JavaScript and is used to create React elements. These elements are then rendered to the React DOM. All the React components are written in JSX. To embed any JavaScript expression in a piece of code written in JSX we will have to wrap that expression in curly braces {}.

Example of JSX: The name written in curly braces { } signifies JSX

```
const name = "Learner";
const element = (
    <h1>
        Hello,
        {name}.Welcome to GeeksforGeeks.
    </h1>
);
```

9. What are components and their type in React?

A Component is one of the core building blocks of React. In other words, we can say that every application you will develop in React will be made up of pieces called components. Components make the task of building UIs much easier.

In React, we mainly have two types of components:

- **Functional Components:** Functional components are simply JavaScript functions. Initially, they were limited in terms of features like state and lifecycle methods. However, with the introduction of Hooks, functional components can now use state, manage side effects, and access other features that were once exclusive to class components.
- **Class Components:** Class components are more complex than functional components. They are able to manage state, handle lifecycle methods, and can also interact with other components. Class components can pass data between each other via props, similar to functional components.

10. How do browsers read JSX?

In general, browsers are not capable of reading JSX and only can read pure JavaScript. The web browsers read JSX with the help of a transpiler. Transpilers are used to convert JSX into JavaScript. The transpiler used is called Babel.

11. Explain the steps to create a react application and print Hello World?

To install React, first, make sure Node is installed on your computer. After installing Node. Open the terminal and type the following command.

```
npx create-react-app <<Application_Name>>
```

Note: The above method is now deprecated, so use the below given method for creating the React application.

```
npm create vite@latest
```

Navigate to the folder.

```
cd <<Application_Name>>
```

This is the first code of ReactJS Hello World!

```
import React from "react";
import "./App.css";
function App() {
    return <div className="App">Hello World !</div>;
}
export default App;
```

Output:

Type the following command to run the application

```
npm start
```

```
import React from "react";
import "./App.css";
function App() {
    return <div className="App">Hello World !</div>;
}
export default App;
```

Output:

Type the following command to run the application

```
npm start
```

React app

12. How to create an event in React?

To create an event in React, attach an event handler like onClick, onChange, etc., to a JSX element. Define the handler function to specify the action when the event is triggered, such as updating state or executing logic.

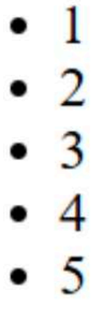
```
function Component() {
    doSomething(e);
    {
        e.preventDefault();
        // Some more response to the event
    }
    return <button onEvent={doSomething}></button>;
}
```

13. Explain the creation of a List in React?

Lists are very useful when it comes to developing the UI of any website. Lists are mainly used for displaying menus on a website. To create a list in React use the map method of array as follows.

```
import React from "react";
import ReactDOM from "react-dom/client";  // Import react-dom/client

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

const updatedNums = numbers.map((number) => {
    return <li key={number}>{number}</li>;  // Add a unique "key" prop
});

const root = ReactDOM.createRoot(document.getElementById("root"));  // Create the root
root.render(<ul>{updatedNums}</ul>);  // Render the list into the root element
```

Output:



List

14. What is a key in React?

A key is a special string attribute you need to include when creating lists of elements in React. Keys are used in React to identify which items in the list are changed, updated, or deleted. In other words, we can say that keys are used to give an identity to the elements in the lists.

15. How to write a comment in React?

There are two ways to write comments in React.

- **Multi-line comment:** We can write multi-line comments in React using the asterisk format /* */

15. How to write a comment in React?

There are two ways to write comments in React.

- **Multi-line comment:** We can write multi-line comments in React using the asterisk format /* */.

```
/*
  This is a multi-line comment.
  It can span multiple lines.
*/
```

- **Single line comment:** We can write single comments in React using the double forward slash //.

```
// This is a single-line comment
```

16. Explain the difference between React and Angular?

React	Angular
React is a JavaScript library. As it indicates react js updates only the virtual DOM is present and the data flow is always in a single direction.	Angular is a framework. Angular updates the Real DOM and the data flow is ensured in the architecture in both directions.
React is more simplified as it follows MVC ie., Model View Control.	The architecture is complex as it follows MVVM models ie., Model View-ViewModel.
It is highly scalable.	It is less scalable than React JS.
It supports Uni-directional data binding which is one-way data binding.	It supports Bi-directional data binding which is two way data binding.
It has a virtual DOM.	It has regular DOM.

17. Explain the use of render method in React?

React renders HTML to the web page by using a function called render(). The purpose of the function is to display the specified HTML code inside the specified HTML element. In the render() method, we can read props and state and return our JSX code to the root component of our app.

18. What is state in React?

The state is an instance of React Component Class that can be defined as an object of a set of observable properties that control the behaviour of the component. In other words, the State of a component is an object that holds some information that may change over the lifetime of the component.

19. Explain props in React?

React allows us to pass information to a Component using something called props (which stands for properties). Props are objects which can be used inside a component

We can access any props inside from the component's class to which the props is passed. The props can be accessed as shown below:

```
this.props.propName;
```

20. What is higher-order component in React?

Higher-order components or HOC is the advanced method of reusing the component functionality logic. It simply takes the original component and returns the enhanced component. HOC are beneficial as they are easy to code and read. Also, helps to get rid of copying the same logic in every component.

21. Explain the difference between functional and class component in React?

Functional Components	Class Components
A functional component is just a plain JavaScript pure function that accepts props as an argument.	A class component requires you to extend from React.Component and create a render function

beneficial as they are easy to code and read. Also, helps to get rid of copying the same logic in every component.

21. Explain the difference between functional and class component in React?

Functional Components	Class Components
A functional component is just a plain JavaScript pure function that accepts props as an argument	A class component requires you to extend from React. Component and create a render function
No render method used	It must have the render() method returning JSX
Also known as Stateless components	Also known as Stateful components
React lifecycle methods (for example, componentDidMount) cannot be used in functional components.	React lifecycle methods can be used inside class components (for example, componentDidMount).
Constructors are not used.	Constructor is used as it needs to store state.
Uses hooks like useState for managing state.	Uses this.state and this.setState for state management.

22. Explain one way data binding in React?

ReactJS uses one-way data binding which can be Component to View or View to Component. It is also known as one-way data flow, which means the data has one, and only one way to be transferred to other parts of the application. In essence, this means child components are not able to update the data that is coming from the parent component. It is easy to debug and less prone to errors.

23. What is Context API in React?

The Context API is a way to share data (such as theme, language preference, etc.) across components without having to pass props down manually at every level. It provides a Provider component to set the value and a Consumer component or useContext() hook to access it.

24. What is the role of shouldComponentUpdate() in React?

shouldComponentUpdate() is a lifecycle method that allows you to control whether a component should re-render when it receives new props or state. If it returns false, React will skip the re-render process for that component.

25. What is the use of dangerouslySetInnerHTML in React?

dangerouslySetInnerHTML is an attribute used to set raw HTML inside a component. It is generally discouraged because it can expose the application to XSS (cross-site scripting) attacks, but it is sometimes used for rendering third-party HTML content.

26. What are Pure Components in React?

A Pure Component is a type of React component that only re-renders if the props or state it receives change. React provides React.PureComponent, which is a base class that automatically performs a shallow comparison of props and state to determine if a re-render is necessary.

27. What is the significance of setState() in React?

setState() is a method used to update the state of a component. When the state is updated, React re-renders the component and its child components to reflect the changes.

React Intermediate Interview Questions

Here, we cover all intermediate level react interview questions with answers, that recommeded for freshers as well as for experienced professionals having 1 - 2 years of experience.

28. What is conditional rendering in React?

Conditional rendering in React involves selectively rendering components based on specified conditions. By evaluating these conditions, developers can control which components are displayed, allowing for dynamic and responsive user interfaces in React applications.

Let us look at this sample code to understand conditional rendering.

setState() is a method used to update the state of a component. When the state is updated, React re-renders the component and its child components to reflect the changes.

React Intermediate Interview Questions

Here, we cover all intermediate level react interview questions with answers, that recommeded for freshers as well as for experienced professionals having 1 - 2 years of experience.

28. What is conditional rendering in React?

Conditional rendering in React involves selectively rendering components based on specified conditions. By evaluating these conditions, developers can control which components are displayed, allowing for dynamic and responsive user interfaces in React applications.

Let us look at this sample code to understand conditional rendering.

```
{isLoggedIn == false ? <DisplayLoggedOut /> : <DisplayLoggedIn />}
```

Here if the boolean isLoggedIn is false then the DisplayLoggedOut component will be rendered otherwise DisplayLoggedIn component will be rendered.

29. What is React Router?

React Router is a standard library for routing in React. It enables the navigation among views of various components in a React Application, allows changing the browser URL, and keeps the UI in sync with the URL.

To install react router type the following command.

```
npm i react-router-dom
```

30. Explain the components of a react-router.

The main components of a react-router are:

1. **Router(usually imported as BrowserRouter):** It is the parent component that is used to store all of the other components. Everything within this will be part of the routing functionality
2. **Switch:** The switch component is used to render only the first route that matches the location rather than rendering all matching routes.
3. **Route:** This component checks the current URL and displays the component associated with that exact path. All routes are placed within the switch components.
4. **Link:** The Link component is used to create links to different routes.

31. How is React routing different from conventional routing?

React Routing	Conventional Routing
Used in Single Page Applications (SPA).	Typically used in Multi-Page Applications (MPA).
No full page reloads. React updates only the necessary parts of the UI.	Full page reload is triggered for every navigation request.
React uses a **client-side router** (e.g., React Router) to handle navigation and manage different views within the same page.	Uses **server-side routing** where the server responds with new HTML for each navigation.
Routes are managed by JavaScript, and the browser's address bar reflects the application's state.	Routes correspond to different server-side routes, and the server responds with new HTML files.

32. Explain the lifecycle methods of components

A React Component can go through four stages of its life as follows.

- **Initialization:** This is the stage where the component is constructed with the given Props and default state. This is done in the constructor of a Component Class.
- **Mounting:** Mounting is the stage of rendering the JSX returned by the render method itself.
- **Updating:** Updating is the stage when the state of a component is updated and the application is repainted.
- **Unmounting:** As the name suggests Unmounting is the final step of the component lifecycle where the component is removed from the page.

33. Explain the methods used in mounting phase of components

32. Explain the lifecycle methods of components

A React Component can go through four stages of its life as follows.

- **Initialization:** This is the stage where the component is constructed with the given Props and default state. This is done in the constructor of a Component Class.
- **Mounting:** Mounting is the stage of rendering the JSX returned by the render method itself.
- **Updating:** Updating is the stage when the state of a component is updated and the application is repainted.
- **Unmounting:** As the name suggests Unmounting is the final step of the component lifecycle where the component is removed from the page.

33. Explain the methods used in mounting phase of components

Mounting is the phase of the component lifecycle when the initialization of the component is completed and the component is mounted on the DOM and rendered for the first time on the webpage. he mounting phase consists of two such predefined functions as described below

- componentWillMount() Function: This function is invoked right before the component is mounted on the DOM.
- componentDidMount() Function: This function is invoked right after the component is mounted on the DOM.

34. What is this.setState function in React?

We use the setState() method to change the state object. It ensures that the component has been updated and calls for re-rendering of the component. The state object of a component may contain multiple attributes and React allows using setState() function to update only a subset of those attributes as well as using multiple setState() methods to update each attribute value independently.

35. What is the use of ref in React?

Refs are a function provided by React to access the DOM element and the React element that you might have created on your own. They are used in cases where we want to change the value of a child component, without making use of props and all. They have wide functionality as we can use callbacks with them.

The syntax to use ref is :

```
const node = this.myCallRef.current;
```

36. What are hooks in React?

Hooks are a new addition in React 16.8. They let developers use state and other React features without writing a class. Hooks doesn't violate any existing React concepts. Instead, Hooks provide a direct API to react concepts such as props, state, context, refs and life-cycle

37. Explain the useState hook in React?

The most used hook in React is the useState() hook. Using this hook we can declare a state variable inside a function but only one state variable can be declared using a single useState() hook. Whenever the useState() hook is used, the value of the state variable is changed and the new variable is stored in a new cell in the stack.

Syntax:

```
const [state, setState] = useState(initialState);
```

- **state**: The current state value.
- **setState:** A function used to update the state value.
- **initialState**: The initial value of the state.

38. Explain the useEffect hook in React?

The useEffect hook in React eliminates the side effect of using class based components. It is used as an alternative to componentDidUpdate() method. The useEffect hook accepts two arguments where second argument is optional.

```
useEffect(function, dependency)
```

The dependency decides when the component will be updated again after rendering.

- **initialState**: The initial value of the state.

38. Explain the useEffect hook in React?

The useEffect hook in React eliminates the side effect of using class based components. It is used as an alternative to componentDidUpdate() method. The useEffect hook accepts two arguments where second argument is optional.

```
useEffect(function, dependency)
```

The dependency decides when the component will be updated again after rendering.

39. What is React Fragments?

when we are trying to render more than one root element we have to put the entire content inside the 'div' tag which is not loved by many developers. So since React 16.2 version, Fragments were introduced, and we use them instead of the extraneous 'div' tag. The following syntax is used to create fragment in react.

```
<React.Fragment>
    <h2>Child-1</h2>
    <p> Child-2</p>
</React.Fragment>
```

40. What is a react developer tool?

React Developer Tools is a Chrome DevTools extension for the React JavaScript library. A very useful tool, if you are working on React applications. This extension adds React debugging tools to the Chrome Developer Tools. It helps you to inspect and edit the React component tree that builds the page, and for each component, one can check the props, the state, hooks, etc.

41. How to use styles in ReactJS?

CSS modules are a way to locally scope the content of your CSS file. We can create a CSS module file by naming our CSS file as App.modules.css and then it can be imported inside App.js file using the special syntax mentioned below.

Syntax:

```
import styles from './App.module.css';
```

42. Explain styled components in React?

Styled-component Module allows us to write CSS within JavaScript in a very modular and reusable way in React. Instead of having one global CSS file for a React project, we can use styled-component for enhancing the developer experience. It also removes the mapping between components and styles – using components as a low-level styling construct

The command to install styled components is

```
npm i styled-components
```

Using the below code we can custom style a button in React

```
import styled from 'styled-components'

const Button = styled.div`
    width : 100px ;
    cursor: pointer ;
    text-decoration : none;
`
export default Button;
```

43. What is prop drilling and its disadvantages?

Prop drilling is basically a situation when the same data is being sent at almost every level due to requirements in the final level. The problem with Prop Drilling is that whenever data from the Parent component will be needed, it would have to come from each level, Regardless of the fact that it is not needed there and simply needed in last.

44. What is conditional rendering in React?

Conditional rendering in React is used when you want to render different UI elements based on certain conditions. For example, rendering a login button if a user is not logged in or rendering a logout button when the user is logged in.

that it is not needed there and simply needed in last.

44. What is conditional rendering in React?

Conditional rendering in React is used when you want to render different UI elements based on certain conditions. For example, rendering a login button if a user is not logged in or rendering a logout button when the user is logged in.

```
const isLoggedIn = true;
return (
  <div>
    {isLoggedIn ?<button>Logout</button> : <button>Login</button>}
  </div>
);
```

45. What are controlled components in React?

Controlled components are React components where the form data is handled by the state within the component. The form element's value is controlled via the React state, making the form elements responsive to changes.

*For further reading, check out our dedicated article on **Intermediate ReactJS Intermediate Interview Questions**. Inside, you'll discover over 20 questions with detailed answers.*

React Interview Questions for Experienced

Here, we cover **advanced react interview questions with answers** for experienced professionals, who have over 5+ years of experience.

46. What is custom hooks in React?

Custom hooks are normal JavaScript functions whose names start with "use" and they may call other hooks. We use custom hooks to maintain the DRY concept that is Don't Repeat Yourself. It helps us to write a logic once and use it anywhere in the code.

47. How to optimize a React code?

We can improve our react code by following these practices:

- Using binding functions in constructors
- Eliminating the use of inline attributes as they slow the process of loading
- Avoiding extra tags by using React fragments
- Lazy loading

48. What is the difference between useref and createRef in React ?

Here is the difference table of useref and createRef in React

useRef	createRef
It is a hook.	It is a function.
It uses the same ref throughout the component's lifecycle.	It creates a new ref every time component re-renders.
It saves its value between re-renders in a functional component.	It creates a new ref for every re-render.
It returns a mutable ref object (i.e., can be changed).	It returns a read-only ref object (cannot be modified directly).
Used for accessing DOM elements, persisting values, or managing timers in functional components.	Used for class components, especially when referencing DOM elements.
const myRef = useRef();	const myRef = React.createRef();

49. What is react-redux?

React-redux is a state management tool which makes it easier to pass these states from one component to another irrespective of their position in the component tree and hence prevents the complexity of the application. As the number of components in our application increases it becomes difficult to pass state as props to multiple components. To overcome this situation we use react-redux

50. What are benefits of using react-redux

49. What is react-redux?

React-redux is a state management tool which makes it easier to pass these states from one component to another irrespective of their position in the component tree and hence prevents the complexity of the application. As the number of components in our application increases it becomes difficult to pass state as props to multiple components. To overcome this situation we use react-redux

50. What are benefits of using react-redux?

They are several benfits of using react-redux such as:

- It provides centralized state management i.e. a single store for whole application
- It optimizes performance as it prevents re-rendering of component
- Makes the process of debugging easier
- Since it offers persistent state management therefore storing data for long times become easier

51. Explain the core components of react-redux?

There are four fundamental concepts of redux in react which decide how the data will flow through components

- Redux Store: It is an object that holds the application state
- Action Creators: These are functions that return actions (objects).
- Actions: Actions are simple objects which conventionally have two properties- type and payload
- Reducers: Reducers are pure functions that update the state of the application in response to actions

52. How can we combine multiple reducers in React?

When working with Redux we sometimes require multiple reducers. In many cases, multiple actions are needed, resulting in the requirement of multiple reducers. However, this can become problematic when creating the Redux store. To manage the multiple reducers we have function called combineReducers in the redux. This basically helps to combine multiple reducers into a single unit and use them.

Syntax

```
import { combineReducers } from "redux";
const rootReducer = combineReducers({
    books: BooksReducer,
    activeBook: ActiveBook
});
```

53. Explain CORS in React?

In ReactJS, Cross-Origin Resource Sharing (CORS) refers to the method that allows you to make requests to the server deployed at a different domain. As a reference, if the frontend and backend are at two different domains, we need CORS there.

We can setup CORS evironment in frontend using two methods:

- axios
- fetch

54. What is axios and how to use it in React?

Axios, which is a popular library is mainly used to send asynchronous HTTP requests to REST endpoints. This library is very useful to perform CRUD operations.

- This popular library is used to communicate with the backend. Axios supports the Promise API, native to JS ES6.
- Using Axios we make API requests in our application. Once the request is made we get the data in Return, and then we use this data in our project.

To install axios package in react use the following command.

```
npm i axios
```

55. Write a program to create a counter with increment and decrement?

We can create the counter app with increment and decrement by writing the below code in the terminal:

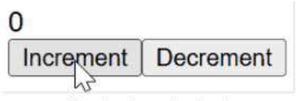
```
npm i axios
```

55. Write a program to create a counter with increment and decrement?

We can create the counter app with increment and decrement by writing the below code in the terminal:

```
import React, { useState } from "react";

const App = () => {
    const [counter, setCounter] = useState(0)
    const handleClick1 = () => {
        setCounter(counter + 1)
    }

    const handleClick2 = () => {
        setCounter(counter - 1)
    }

    return (
        <div>
            <div>
                {counter}
            </div>
            <div className="buttons">
                <button onClick={handleClick1}>
                    Increment
                </button>
                <button onClick={handleClick2}>
                    Decrement
                </button>
            </div>
        </div>
    )
}

export default App
```

Output:



Counter App using React

56. Explain why and how to update state of components using callback?

It is advised to use a callback-based approach to update the state using setState because it solves lots of bugs upfront that may occur in the future.We can use the following syntax to update state using callback

```
this.setState(st => {
    return(
        st.stateName1 = state1UpdatedValue,
        st.stateName2 = state2UpdatedValue
    )
})
```

57. What is React-Material UI?

React Material UI is an open-source React component library, offering prebuilt components for creating React applications. Developed by Google in 2014, it's compatible with JavaScript frameworks like Angular.js and Vue.js. Renowned for its quality designs and easy customization, it's favored by developers for rapid development.

58. What is flux architecture in redux?

Flux architecture in Redux is a design pattern used for managing application state in a unidirectional data flow. In this architecture, actions are dispatched to modify the store, which holds the entire application state. The store sends the updated state to the view (UI), and the cycle repeats when new actions are triggered. Redux follows this structure to ensure a predictable and maintainable state management system for large applications.

59. What are custom hooks in React?

Custom hooks are user-defined functions that use built-in hooks like useState, useEffect, etc., to reuse stateful logic across components. They allow you to extract and share common logic.

60. How can you optimize React performance?

Optimizing React performance can be achieved using:

- React.memo() for preventing unnecessary re-renders.
- Lazy loading components with React.lazy() and Suspense.
- Using useMemo() and useCallback() hooks for memoizing values and functions.

state management system for large applications.

59. What are custom hooks in React?

Custom hooks are user-defined functions that use built-in hooks like useState, useEffect, etc., to reuse stateful logic across components. They allow you to extract and share common logic.

60. How can you optimize React performance?

Optimizing React performance can be achieved using:

- React.memo() for preventing unnecessary re-renders.
- Lazy loading components with React.lazy() and Suspense.
- Using useMemo() and useCallback() hooks to memoize values and functions.
- Avoiding unnecessary state updates.

61. What is the Strict Mode in React?

Strict Mode in React is a tool that helps developers find and fix problems in their app while developing. It only works in development and doesn't affect the app when it's running in production. When enabled, it checks for potential issues, such as old features that should no longer be used, and gives warnings to help avoid bugs.

We can enable the strict mode by by wrapping your application or specific components inside the **<React.StrictMode>**.

```
import React from "react";
import ReactDOM from "react-dom";

const App = () => {
  return (
    <div>
      <h1>Hello, World!</h1>
    </div>
  );
};

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById("root")
);
```

62. What is Redux and how does it work with React?

Redux is a state management library that helps manage the application state globally. It uses actions, reducers, and a central store to control state. React-Redux is used to connect Redux state to React components.

63. How does React handle concurrency?

React's Concurrent Mode is a set of features that help React apps stay responsive and gracefully adjust to the user's device capabilities and network speed. It allows React to interrupt rendering and prioritize high-priority tasks.

64. How does React handle server-side rendering (SSR)?

Server-side rendering (SSR) is the process of rendering a React application on the server and sending the fully rendered HTML to the client. This improves the initial page load performance and SEO.

65. What are forms in React?

In React, **forms** are a way to capture user input, such as text fields, checkboxes, radio buttons, and buttons. React provides controlled and uncontrolled components for handling form elements, allowing developers to manage user input and form state in an efficient and structured way.

66. How to create forms in React?

Creating forms in React involves handling user input and managing the form's state. In React, form elements like <input>, <textarea>, and <select> are controlled components, meaning their values are controlled by the React state.

- **Create the State for the Form:** Use the useState hook to manage form input values.
- **Set the Value of Form Elements:** Bind the form elements' value attribute to the corresponding state variable to make the inputs controlled.
- **Handle User Input**: Use an onChange event handler to update the state whenever the user types in the form fields.
- **Submit the Form:** Handle the form submission with an onSubmit event, and prevent the default behavior to stop the page from refreshing.

buttons. React provides controlled and uncontrolled components for handling form elements, allowing developers to manage user input and form state in an efficient and structured way.

66. How to create forms in React?

Creating forms in React involves handling user input and managing the form's state. In React, form elements like <input>, <textarea>, and <select> are controlled components, meaning their values are controlled by the React state.

- **Create the State for the Form:** Use the useState hook to manage form input values.
- **Set the Value of Form Elements:** Bind the form elements' value attribute to the corresponding state variable to make the inputs controlled.
- **Handle User Input**: Use an onChange event handler to update the state whenever the user types in the form fields.
- **Submit the Form:** Handle the form submission with an onSubmit event, and prevent the default behavior to stop the page from refreshing.

```
import React, { useState } from 'react';

function MyForm() {
    // State to store input values
    const [name, setName] = useState('');
    const [email, setEmail] = useState('');
    const [message, setMessage] = useState('');

    // Handle input change for each field
    const handleNameChange = (e) => setName(e.target.value);
    const handleEmailChange = (e) => setEmail(e.target.value);
    const handleMessageChange = (e) => setMessage(e.target.value);

    // Handle form submission
    const handleSubmit = (e) => {
        e.preventDefault();  // Prevent default form submission behavior
        console.log('Form submitted:', { name, email, message });
    };

    return (
        <form onSubmit={handleSubmit}>
            <div>
                <label>Name:</label>
                <input type="text" value={name} onChange={handleNameChange} />
            </div>
            <div>
                <label>Email:</label>
                <input type="email" value={email} onChange={handleEmailChange} />
            </div>
            <div>
                <label>Message:</label>
                <textarea value={message} onChange={handleMessageChange} />
            </div>
            <button type="submit">Submit</button>
        </form>
    );
}

export default MyForm;
```

Output:

React Form

67. What is Lazy Loading in React?

Lazy Loading in React is a technique used to load components only when they are needed, instead of loading everything at once when the app starts. This helps improve the performance of the app by reducing the initial loading time.

In React, React.lazy() is used to implement lazy loading for components, which allows you to split your code into smaller bundles and load them only when required.

68. How is React different from React Native?

React	React Native
A JavaScript library for building user interfaces, primarily for web applications.	A framework for building mobile applications using JavaScript and React.
Focuses on developing web applications.	Focuses on developing native mobile applications for iOS and Android.

In React, React.lazy() is used to implement lazy loading for components, which allows you to split your code into smaller bundles and load them only when required.

68. How is React different from React Native?

React	React Native
A JavaScript library for building user interfaces, primarily for web applications.	A framework for building mobile applications using JavaScript and React.
Focuses on developing web applications.	Focuses on developing native mobile applications for iOS and Android.
Renders HTML using the **Virtual DOM**.	Renders native mobile components using **native APIs** (e.g., `View`, `Text`, `Image`).
Uses **web technologies** like HTML, CSS, and JavaScript.	Uses **native mobile components** and styles for building mobile UIs.
Runs in the browser on a web server.	Runs on mobile devices and communicates with native code.
Can be deployed on browsers like Chrome, Firefox, etc.	Can be deployed as native apps on iOS and Android platforms.

69. What is Memoization in React?

Memoization in React is an optimization technique used to improve the performance of a component by preventing unnecessary re-renders. It involves caching the results of expensive function calls and reusing the cached result when the inputs to the function haven't changed. This is particularly useful when a component's rendering process is slow, and you want to avoid recalculating or re-rendering unnecessarily.

How Memoization Works in React:

React provides two key APIs for memoization:

1. **`React.memo()`**: This is a higher-order component (HOC) used to prevent re-renders of functional components if their props have not changed.
2. **`useMemo()`**: A hook that memorizes the result of a computation and only recomputes it when its dependencies change.

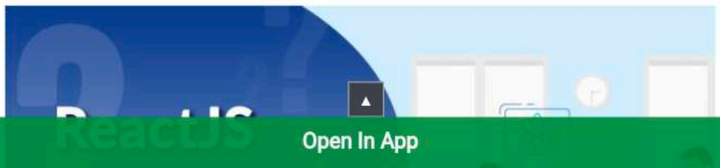
70. What is Reoncillation in React?

Reconciliation in React is the process of updating the **DOM** when a component's state or props change. React uses the **Virtual DOM** to efficiently determine what parts of the actual DOM need to be updated. Here's a simplified overview:

- **Triggering Reconciliation**: React triggers reconciliation when state or props change (e.g., through `setState()` or `useState()`).
- **Virtual DOM Diffing**: React compares the old Virtual DOM with the new one to identify changes.
- **Efficient Updates**: React calculates the minimal set of changes and applies them to the real DOM.
- **Fiber Algorithm**: React's Fiber algorithm allows the reconciliation process to be interrupted and prioritized, improving performance for complex tasks.

Top 10 Commonly Asked ReactJS Interview Questions

1. **What is React?**
2. **What are the key features of React?**
3. **What is the Virtual DOM, and how does it work?**
4. **What are React components?**
5. **What is the difference between a functional component and a class component in React?**
6. **What are React Hooks?**
7. **What is `useState` in React?**
8. **What are Custom Hooks?**
9. **What is JSX?**
10. **What is the difference between `props` and `state` in React?**



React Interview Questions for Freshers

1. What is React?

React 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 mobile and web applications as it follows the component-based approach.

The important features of React are:

- It supports server-side rendering.
- It will make use of the virtual DOM rather than real DOM (Data Object Model) as RealDOM manipulations are expensive.
- It follows unidirectional data binding or data flow.
- It uses reusable or composable UI components for developing the view.

2. What are the advantages of using React?

MVC is generally abbreviated as Model View Controller.

- **Use of Virtual DOM to improve efficiency:** React uses virtual DOM to render the view. As the name suggests, virtual DOM is a virtual representation of the real DOM. Each time the data changes in a react app, a new virtual DOM gets created. Creating a virtual DOM is much faster than rendering the UI inside the browser. Therefore, with the use of virtual DOM, the efficiency of the app improves.
- **Gentle learning curve:** React has a gentle learning curve when compared to frameworks like Angular. Anyone with little knowledge of javascript can start building web applications using React.
- **SEO friendly:** React allows developers to develop engaging user interfaces that can be easily navigated in various search engines. It also allows server-side rendering, which boosts the SEO of an app.
- **Reusable components:** React uses component-based architecture for developing applications. Components are independent and reusable bits of code. These components can be shared across various applications having similar functionality. The re-use of components increases the pace of development.
- **Huge ecosystem of libraries to choose from:** React provides you with the freedom to choose the tools, libraries, and architecture for developing an application based on your requirement.

3. What are the limitations of React?

The few limitations of React are as given below:

- React is not a full-blown framework as it is only a library.
- The components of React are numerous and will take time to fully grasp the benefits of all.
- It might be difficult for beginner programmers to understand React.
- Coding might become complex as it will make use of inline templating and JSX.

4. What is useState() in React?

The useState() is a built-in React Hook that allows you for having state variables in functional components. It should be used when the DOM has something that is dynamically manipulating/controlling.

In the below-given example code, The useState(0) will return a tuple where the count is the first parameter that represents the counter's current state and the second parameter setCounter method will allow us to update the state of the counter.

```
...
const [count, setCounter] = useState(0);
const [otherStuffs, setOtherStuffs] = useState(...);
...
const setCount = () => {
   setCounter(count + 1);
   setOtherStuffs(...);
   ...
};
```

We can make use of setCounter() method for updating the state of count anywhere. In this example, we are using setCounter() inside the setCount function where various other things can also be done. The idea with the usage of hooks is that we will be able to keep our code more functional and avoid class-based components if they are not required.

5. What are keys in React?

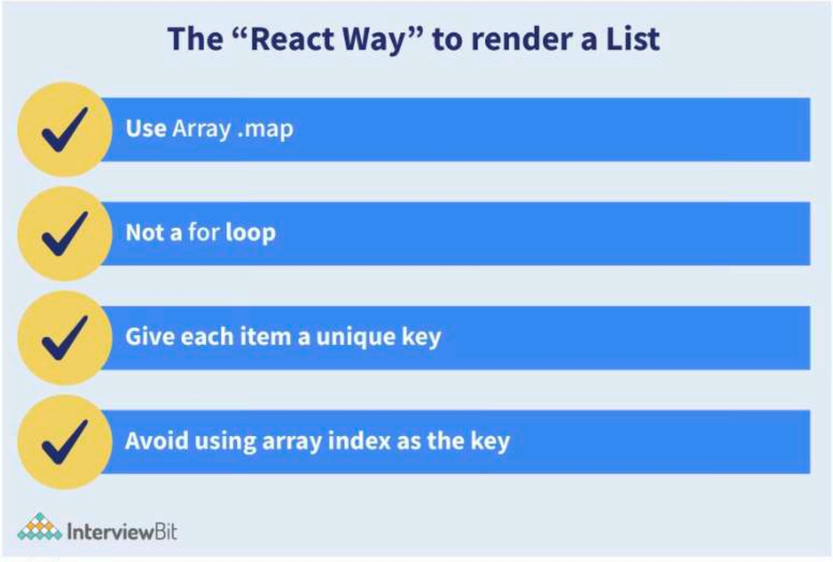
A key is a special string attribute that needs to be included when using lists of elements.

```
);
```

We can make use of setCounter() method for updating the state of count anywhere. In this example, we are using setCounter() inside the setCount function where various other things can also be done. The idea with the usage of hooks is that we will be able to keep our code more functional and avoid class-based components if they are not required.

5. What are keys in React?

A key is a special string attribute that needs to be included when using lists of elements.



Example of a list using key -

```
const ids = [1,2,3,4,5];
const listElements = ids.map((id)=>{
return(
<li key={id.toString()}>
  {id}
</li>
)
})
```

Importance of keys -

- Keys help react identify which elements were added, changed or removed.
- Keys should be given to array elements for providing a unique identity for each element.
- Without keys, React does not understand the order or uniqueness of each element.
- With keys, React has an idea of which particular element was deleted, edited, and added.
- Keys are generally used for displaying a list of data coming from an API.

***Note- Keys used within arrays should be unique among siblings. They need not be globally unique.

6. What is JSX?

JSX stands for JavaScript XML. It allows us to write HTML inside JavaScript and place them in the DOM without using functions like appendChild( ) or createElement( ).

As stated in the official docs of React, JSX provides syntactic sugar for React.createElement( ) function.

Note- We can create react applications without using JSX as well.

Let's understand **how JSX works**:

Without using JSX, we would have to create an element by the following process:

```
const text = React.createElement('p', {}, 'This is a text');
const container = React.createElement('div','{}',text );
ReactDOM.render(container,rootElement);
```

Using JSX, the above code can be simplified:

```
const container = (
<div>
  <p>This is a text</p>
</div>
);
ReactDOM.render(container,rootElement);
```

As one can see in the code above, we are directly using HTML inside JavaScript.

7. What are the differences between functional and class components?

Before the introduction of Hooks in React, functional components were called stateless components and were behind class components on a feature basis. After the introduction of Hooks, functional components are equivalent to class components.

Although functional components are the new trend, the react team insists on keeping class components in React. Therefore, it is important to know how these components differ.

On the following basis let's compare functional and class components:

```
ReactDOM.render(container,rootElement);
```

As one can see in the code above, we are directly using HTML inside JavaScript.

7. What are the differences between functional and class components?

Before the introduction of Hooks in React, functional components were called stateless components and were behind class components on a feature basis. After the introduction of Hooks, functional components are equivalent to class components.

Although functional components are the new trend, the react team insists on keeping class components in React. Therefore, it is important to know how these components differ.

On the following basis let's compare functional and class components:

- **Declaration**

Functional components are nothing but JavaScript functions and therefore can be declared using an arrow function or the function keyword:

```
function card(props){
 return(
    <div className="main-container">
      <h2>Title of the card</h2>
    </div>
 )
}
const card = (props) =>{
 return(
    <div className="main-container">
      <h2>Title of the card</h2>
    </div>
 )
}
```

Class components, on the other hand, are declared using the ES6 class:

```
class Card extends React.Component{
 constructor(props){
   super(props);
 }
  render(){
    return(
      <div className="main-container">
        <h2>Title of the card</h2>
      </div>
    )
  }
}
```

- **Handling props**

Let's render the following component with props and analyse how functional and class components handle props:

```
<Student Info name="Vivek" rollNumber="23" />
```

In functional components, the handling of props is pretty straightforward. Any prop provided as an argument to a functional component can be directly used inside HTML elements:

```
function StudentInfo(props){
  return(
    <div className="main">
      <h2>{props.name}</h2>
      <h4>{props.rollNumber}</h4>
    </div>
  )
}
```

In the case of class components, props are handled in a different way:

```
class StudentInfo extends React.Component{
  constructor(props){
    super(props);
  }
  render(){
    return(
      <div className="main">
        <h2>{this.props.name}</h2>
        <h4>{this.props.rollNumber}</h4>
      </div>
    )
  }
}
```

As we can see in the code above, **this** keyword is used in the case of class components.

- **Handling state**

Functional components use React hooks to handle state. It uses the useState hoo
component:

```
    }
    render(){
      return(
        <div className="main">
          <h2>{this.props.name}</h2>
          <h4>{this.props.rollNumber}</h4>
        </div>
      )
    }
   }
```

As we can see in the code above, **this** keyword is used in the case of class components.

- **Handling state**

Functional components use React hooks to handle state. It uses the useState hook to set the state of a variable inside the component:

```
function ClassRoom(props){
  let [studentsCount,setStudentsCount] = useState(0);
   const addStudent = () => {
     setStudentsCount(++studentsCount);
  }
   return(
     <div>
       <p>Number of students in class room: {studentsCount}</p>
       <button onClick={addStudent}>Add Student</button>
     </div>
   )
  }
```

Since useState hook returns an array of two items, the first item contains the current state, and the second item is a function used to update the state.

In the code above, using array destructuring we have set the variable name to studentsCount with a current value of "0" and setStudentsCount is the function that is used to update the state.

For reading the state, we can see from the code above, the variable name can be directly used to read the current state of the variable.

We cannot use React Hooks inside class components, therefore state handling is done very differently in a class component:

Let's take the same above example and convert it into a class component:

```
class ClassRoom extends React.Component{
        constructor(props){
            super(props);
            this.state = {studentsCount : 0};

            this.addStudent = this.addStudent.bind(this);
         }

            addStudent(){
            this.setState((prevState)=>{
               return {studentsCount: prevState.studentsCount++}
            });
         }

            render(){
             return(
               <div>
                 <p>Number of students in class room: {this.state.studentsCount}</p>
                 <button onClick={this.addStudent}>Add Student</button>
               </div>
             )
           }
         }
```

In the code above, we see we are using **this.state** to add the variable studentsCount and setting the value to "0".

For reading the state, we are using **this.state.studentsCount**.

For updating the state, we need to first bind the addStudent function to **this**. Only then, we will be able to use the **setState** function which is used to update the state.

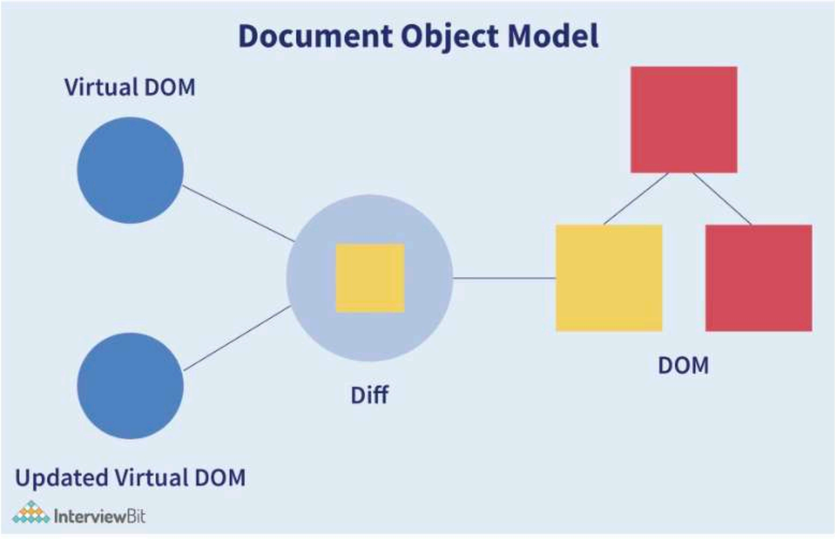
8. What is the virtual DOM? How does react use the virtual DOM to render the UI?

As stated by the react team, virtual DOM is a concept where a virtual representation of the real DOM is kept inside the memory and is synced with the real DOM by a library such as ReactDOM.

For updating the state, we need to first bind the addStudent function to **this**. Only then, we will be able to use the **setState** function which is used to update the state.

8. What is the virtual DOM? How does react use the virtual DOM to render the UI?

As stated by the react team, virtual DOM is a concept where a virtual representation of the real DOM is kept inside the memory and is synced with the real DOM by a library such as ReactDOM.



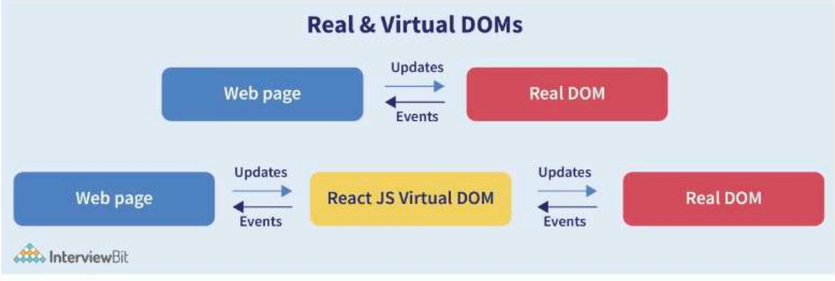
Why was virtual DOM introduced?

DOM manipulation is an integral part of any web application, but DOM manipulation is quite slow when compared to other operations in JavaScript. The efficiency of the application gets affected when several DOM manipulations are being done. Most JavaScript frameworks update the entire DOM even when a small part of the DOM changes.

For example, consider a list that is being rendered inside the DOM. If one of the items in the list changes, the entire list gets rendered again instead of just rendering the item that was changed/updated. This is called inefficient updating.

To address the problem of inefficient updating, the react team introduced the concept of virtual DOM.

How does it work?



For every DOM object, there is a corresponding virtual DOM object(copy), which has the same properties. The main difference between the real DOM object and the virtual DOM object is that any changes in the virtual DOM object will not reflect on the screen directly. Consider a virtual DOM object as a blueprint of the real DOM object. Whenever a JSX element gets rendered, every virtual DOM object gets updated.

> **Note- One may think updating every virtual DOM object might be inefficient, but that's not the case. Updating the virtual DOM is much faster than updating the real DOM since we are just updating the blueprint of the real DOM.

React uses two virtual DOMs to render the user interface. One of them is used to store the current state of the objects and the other to store the previous state of the objects. Whenever the virtual DOM gets updated, react compares the two virtual DOMs and gets to know about which virtual DOM objects were updated. After knowing which objects were updated, react renders only those objects inside the real DOM instead of rendering the complete real DOM. This way, with the use of virtual DOM, react solves the problem of inefficient updating.

9. What are the differences between controlled and uncontrolled components?

Controlled and uncontrolled components are just different approaches to handling input from elements in react.

Feature	Uncontrolled	Controlled	Name attrs
One-time value retrieval (e.g. on submit)	✓	✓	✓
Validating on submit	✓	✓	✓
Field-level Validation	✗	✓	✓
Conditionally disabling submit button	✗	✓	✓

know about which virtual DOM objects were updated. After knowing which objects were updated, react renders only those objects inside the real DOM instead of rendering the complete real DOM. This way, with the use of virtual DOM, react solves the problem of inefficient updating.

9. What are the differences between controlled and uncontrolled components?

Controlled and uncontrolled components are just different approaches to handling input from elements in react.

Feature	Uncontrolled	Controlled	Name attrs
One-time value retrieval (e.g. on submit)	✔	✔	✔
Validating on submit	✔	✔	✔
Field-level Validation	✘	✔	✔
Conditionally disabling submit button	✘	✔	✔
Enforcing input format	✘	✔	✔
several inputs for one piece of data	✘	✔	✔
dynamic inputs	✘	✔	🤔

- **Controlled component:** In a controlled component, the value of the input element is controlled by React. We store the state of the input element inside the code, and by using event-based callbacks, any changes made to the input element will be reflected in the code as well.

When a user enters data inside the input element of a controlled component, onChange function gets triggered and inside the code, we check whether the value entered is valid or invalid. If the value is valid, we change the state and re-render the input element with the new value.

Example of a controlled component:

```
function FormValidation(props) {
let [inputValue, setInputValue] = useState("");
let updateInput = e => {
  setInputValue(e.target.value);
};
return (
  <div>
    <form>
      <input type="text" value={inputValue} onChange={updateInput} />
    </form>
  </div>
);
}
```

As one can see in the code above, the value of the input element is determined by the state of the **inputValue** variable. Any changes made to the input element is handled by the **updateInput** function.

- **Uncontrolled component:** In an uncontrolled component, the value of the input element is handled by the DOM itself. Input elements inside uncontrolled components work just like normal HTML input form elements.

The state of the input element is handled by the DOM. Whenever the value of the input element is changed, event-based callbacks are not called. Basically, react does not perform any action when there are changes made to the input element.

Whenever use enters data inside the input field, the updated data is shown directly. To access the value of the input element, we can use **ref**.

Example of an uncontrolled component:

```
function FormValidation(props) {
let inputValue = React.createRef();
let handleSubmit = e => {
  alert(`Input value: ${inputValue.current.value}`);
  e.preventDefault();
};
return (
  <div>
    <form onSubmit={handleSubmit}>
      <input type="text" ref={inputValue} />
      <button type="submit">Submit</button>
    </form>
  </div>
);
}
```

As one can see in the code above, we are **not** using **onChange** function to govern the changes made to the input element. Instead, we are using **ref** to access the value of the input element.

10. What are props in React?

The props in React are the inputs to a component of React. They can be single-valued or objects having a set of values that will be passed to components of React during creation by using a naming convention that almost looks similar to HTML-tag attributes. We can say that props are the data passed from a parent component into a child component.

The main purpose of props is to provide different component functionalities such

- Passing custom data to the React component.

As one can see in the code above, we are **not** using **onChange** function to govern the changes made to the input element. Instead, we are using **ref** to access the value of the input element.

10. What are props in React?

The props in React are the inputs to a component of React. They can be single-valued or objects having a set of values that will be passed to components of React during creation by using a naming convention that almost looks similar to HTML-tag attributes. We can say that props are the data passed from a parent component into a child component.

The main purpose of props is to provide different component functionalities such as:

- Passing custom data to the React component.
- Using through `this.props.reactProp` inside render() method of the component.
- Triggering state changes.

For example, consider we are creating an element with reactProp property as given below: `<Element reactProp = "1" />` This reactProp name will be considered as a property attached to the native props object of React which already exists on each component created with the help of React library: `props.reactProp;`.

11. Explain React state and props.

Props	State
Immutable	Owned by its component
Has better performance	Locally scoped
Can be passed to child components	Writeable/Mutable
	has setState() method to modify properties
	Changes to state can be asynchronous
	can only be passed as props

- **React State**
  Every component in react has a built-in state object, which contains all the property values that belong to that component. In other words, the state object controls the behaviour of a component. Any change in the property values of the state object leads to the re-rendering of the component.

  Note- State object is not available in functional components but, we can use React Hooks to add state to a functional component.

How to declare a state object?

Example:

```
class Car extends React.Component{
constructor(props){
  super(props);
  this.state = {
    brand: "BMW",
    color: "black"
  }
}
}
```

How to use and update the state object?

```
class Car extends React.Component {
constructor(props) {
  super(props);
  this.state = {
    brand: "BMW",
    color: "Black"
  };
}
changeColor() {
  this.setState(prevState => {
    return { color: "Red" };
  });
}
render() {
  return (
    <div>
      <button onClick={() => this.changeColor()}>Change Color</button>
      <p>{this.state.color}</p>
    </div>
  );
}
}
```

As one can see in the code above, we can use the state by calling **this.state.propertyName** and we can change the state object property using **setState** method.

- **React Props**

Every React component accepts a single object argument called props (which sta
to a component using HTML attributes and the component accepts these props a

```
render() {
  return (
    <div>
      <button onClick={() => this.changeColor()}>Change Color</button>
      <p>{this.state.color}</p>
    </div>
  );
}
}
```

As one can see in the code above, we can use the state by calling **this.state.propertyName** and we can change the state object property using **setState** method.

- **React Props**

Every React component accepts a single object argument called props (which stands for "properties"). These props can be passed to a component using HTML attributes and the component accepts these props as an argument.

Using props, we can pass data from one component to another.

Passing props to a component:

While rendering a component, we can pass the props as an HTML attribute:

```
<Car brand="Mercedes"/>
```

The component receives the props:

In Class component:

```
class Car extends React.Component {
constructor(props) {
  super(props);
  this.state = {
    brand: this.props.brand,
    color: "Black"
  };
}
}
```

In Functional component:

```
function Car(props) {
let [brand, setBrand] = useState(props.brand);
}
```

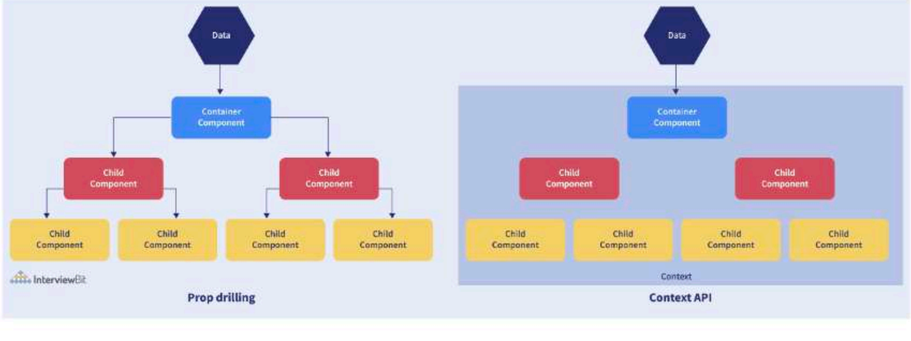
Note- Props are read-only. They cannot be manipulated or changed inside a component.

12. Explain about types of side effects in React component.

There are two types of side effects in React component. They are:

- **Effects without Cleanup:** This side effect will be used in useEffect which does not restrict the browser from screen update. It also improves the responsiveness of an application. A few common examples are network requests, Logging, manual DOM mutations, etc.
- **Effects with Cleanup:** Some of the Hook effects will require the cleanup after updating of DOM is done. For example, if you want to set up an external data source subscription, it requires cleaning up the memory else there might be a problem of memory leak. It is a known fact that React will carry out the cleanup of memory when the unmounting of components happens. But the effects will run for each render() method rather than for any specific method. Thus we can say that, before execution of the effects succeeding time the React will also cleanup effects from the preceding render.

13. What is prop drilling in React?

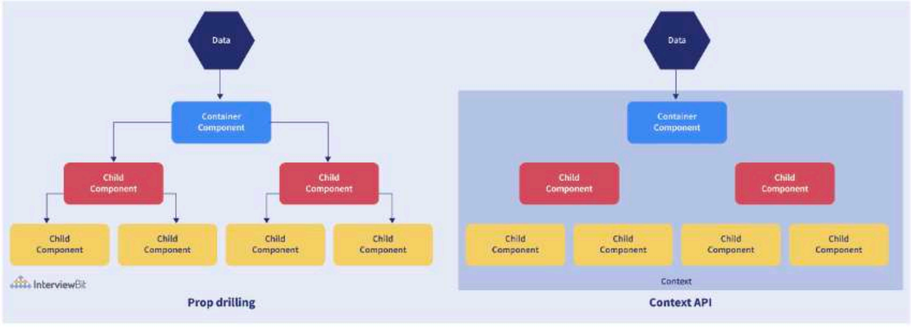


Sometimes while developing React applications, there is a need to pass data from a component that is higher in the hierarchy to a component that is deeply nested. To pass data between such components, we pass props from a source component and keep passing the prop to the next component in the hierarchy till we reach the deeply nested component.

The **disadvantage** of using prop drilling is that the components that should other

leak. It is a known fact that React will carry out the cleanup of memory when the unmounting of components happens. But the effects will run for each render() method rather than for any specific method. Thus we can say that, before execution of the effects succeeding time the React will also cleanup effects from the preceding render.

13. What is prop drilling in React?



Sometimes while developing React applications, there is a need to pass data from a component that is higher in the hierarchy to a component that is deeply nested. To pass data between such components, we pass props from a source component and keep passing the prop to the next component in the hierarchy till we reach the deeply nested component.

The **disadvantage** of using prop drilling is that the components that should otherwise be not aware of the data have access to the data.

14. What are error boundaries?

Introduced in version 16 of React, Error boundaries provide a way for us to catch errors that occur in the render phase.

- **What is an error boundary?**

Any component which uses one of the following lifecycle methods is considered an error boundary.
In what places can an error boundary detect an error?

1. Render phase
2. Inside a lifecycle method
3. Inside the constructor

Without using error boundaries:

```
class CounterComponent extends React.Component{
constructor(props){
  super(props);
  this.state = {
    counterValue: 0
  }
  this.incrementCounter = this.incrementCounter.bind(this);
}
incrementCounter(){
  this.setState(prevState => counterValue = prevState+1);
}
render(){
  if(this.state.counter === 2){
    throw new Error('Crashed');
  }
  return(
    <div>
      <button onClick={this.incrementCounter}>Increment Value</button>
      <p>Value of counter: {this.state.counterValue}</p>
    </div>
  )
}
}
```

In the code above, when the counterValue equals 2, we throw an error inside the render method.

When we are not using the error boundary, instead of seeing an error, we see a blank page. Since any error inside the render method leads to unmounting of the component. To display an error that occurs inside the render method, we use error boundaries.

With error boundaries: As mentioned above, error boundary is a component using one or both of the following methods: **static getDerivedStateFromError and componentDidCatch.**

Let's create an error boundary to handle errors in the render phase:

```
class ErrorBoundary extends React.Component {
constructor(props) {
  super(props);
  this.state = { hasError: false };
}
static getDerivedStateFromError(error) {
```

```
    </div>
  )
}
}
```

In the code above, when the counterValue equals 2, we throw an error inside the render method.

When we are not using the error boundary, instead of seeing an error, we see a blank page. Since any error inside the render method leads to unmounting of the component. To display an error that occurs inside the render method, we use error boundaries.

With error boundaries: As mentioned above, error boundary is a component using one or both of the following methods: **static getDerivedStateFromError and componentDidCatch.**

Let's create an error boundary to handle errors in the render phase:

```
class ErrorBoundary extends React.Component {
constructor(props) {
  super(props);
  this.state = { hasError: false };
}
static getDerivedStateFromError(error) {
  return { hasError: true };
}
 componentDidCatch(error, errorInfo) {
  logErrorToMyService(error, errorInfo);
}
render() {
  if (this.state.hasError) {
    return <h4>Something went wrong</h4>
  }
  return this.props.children;
}
}
```

In the code above, **getDerivedStateFromError** function renders the fallback UI interface when the render method has an error.

componentDidCatch logs the error information to an error tracking service.

Now with the error boundary, we can render the CounterComponent in the following way:

```
<ErrorBoundary>
 <CounterComponent/>
</ErrorBoundary>
```

15. What is React Hooks?

React Hooks are the built-in functions that permit developers for using the state and lifecycle methods within React components. These are newly added features made available in React 16.8 version. Each lifecycle of a component is having 3 phases which include mount, unmount, and update. Along with that, components have properties and states. Hooks will allow using these methods by developers for improving the reuse of code with higher flexibility navigating the component tree.

Using Hook, all features of React can be used without writing class components. ***For example***, before React version 16.8, it required a class component for managing the state of a component. But now using the useState hook, we can keep the state in a functional component.

16. Explain React Hooks.

What are Hooks? Hooks are functions that let us "hook into" React state and lifecycle features from a **functional component.**

React Hooks **cannot** be used in class components. They let us write components without class.

Why were Hooks introduced in React?

React hooks were introduced in the 16.8 version of React. Previously, functional components were called stateless components. Only class components were used for state management and lifecycle methods. The need to change a functional component to a class component, whenever state management or lifecycle methods were to be used, led to the development of Hooks.

Example of a hook: **useState hook:**

In functional components, the useState hook lets us define a state for a component:

```
function Person(props) {
// We are declaring a state variable called name.
// setName is a function to update/change the value of name
let [name, setName] = useState('');
}
```

The state variable "name" can be directly used inside the HTML.

17. What are the rules that must be followed while using React Hooks?

There are 2 rules which must be followed while you code with Hooks:

- React Hooks must be called only at the top level. It is not allowed to call them inside the nested functions, loops, or conditions.
- It is allowed to call the Hooks only from the React Function Components.

18. What is the use of useEffect React Hooks?

The useEffect React Hook is used for performing the side effects in functional co

```
let [name, setName] = useState('');
}
```

The state variable "name" can be directly used inside the HTML.

17. What are the rules that must be followed while using React Hooks?

There are 2 rules which must be followed while you code with Hooks:

- React Hooks must be called only at the top level. It is not allowed to call them inside the nested functions, loops, or conditions.
- It is allowed to call the Hooks only from the React Function Components.

18. What is the use of useEffect React Hooks?

The useEffect React Hook is used for performing the side effects in functional components. With the help of useEffect, you will inform React that your component requires something to be done after rendering the component or after a state change. The function you have passed(can be referred to as "effect") will be remembered by React and call afterwards the performance of DOM updates is over. Using this, we can perform various calculations such as data fetching, setting up document title, manipulating DOM directly, etc, that don't target the output value. The useEffect hook will run by default after the first render and also after each update of the component. React will guarantee that the DOM will be updated by the time when the effect has run by it.

The useEffect React Hook will accept 2 arguments: `useEffect(callback,[dependencies]);`

Where the first argument callback represents the function having the logic of side-effect and it will be immediately executed after changes were being pushed to DOM. The second argument dependencies represent an optional array of dependencies. The useEffect() will execute the callback only if there is a change in dependencies in between renderings.

Example:

```
import { useEffect } from 'react';
function WelcomeGreetings({ name }) {
 const msg = `Hi, ${name}!`;     // Calculates output
 useEffect(() => {
   document.title = `Welcome to you ${name}`;    // Side-effect!
 }, [name]);
 return <div>{msg}</div>;         // Calculates output
}
```

The above code will update the document title which is considered to be a side-effect as it will not calculate the component output directly. That is why updating of document title has been placed in a callback and provided to useEffect().

Consider you don't want to execute document title update each time on rendering of WelcomeGreetings component and you want it to be executed only when the name prop changes then you need to supply name as a dependency to `useEffect(callback, [name])`.

19. Why do React Hooks make use of refs?

Earlier, refs were only limited to class components but now it can also be accessible in function components through the useRef Hook in React.

The refs are used for:

- Managing focus, media playback, or text selection.
- Integrating with DOM libraries by third-party.
- Triggering the imperative animations.

20. What are Custom Hooks?

A Custom Hook is a function in Javascript whose name begins with 'use' and which calls other hooks. It is a part of React v16.8 hook update and permits you for reusing the stateful logic without any need for component hierarchy restructuring.

In almost all of the cases, custom hooks are considered to be sufficient for replacing render props and HoCs (Higher-Order components) and reducing the amount of nesting required. Custom Hooks will allow you for avoiding multiple layers of abstraction or wrapper hell that might come along with Render Props and HoCs.

The **disadvantage** of Custom Hooks is it cannot be used inside of the classes.

React Interview Questions for Experienced

21. How to perform automatic redirect after login?

The react-router package will provide the component `<Redirect>` in React Router. Rendering of a `<Redirect>` component will navigate to a newer location. In the history stack, the current location will be overridden by the new location just like the server-side redirects.

```
import React, { Component } from 'react'
import { Redirect } from 'react-router'
export default class LoginDemoComponent extends Component {
 render() {
   if (this.state.isLoggedIn === true) {
     return <Redirect to="/your/redirect/page" />
   } else {
     return <div>{'Please complete login'}</div>
   }
 }
}
```

or wrapper hell that might come along with Render Props and HoCs.

The **disadvantage** of Custom Hooks is it cannot be used inside of the classes.

React Interview Questions for Experienced

21. How to perform automatic redirect after login?

The react-router package will provide the component `<Redirect>` in React Router. Rendering of a `<Redirect>` component will navigate to a newer location. In the history stack, the current location will be overridden by the new location just like the server-side redirects.

```
import React, { Component } from 'react'
import { Redirect } from 'react-router'
export default class LoginDemoComponent extends Component {
 render() {
   if (this.state.isLoggedIn === true) {
     return <Redirect to="/your/redirect/page" />
   } else {
     return <div>{'Please complete login'}</div>
   }
 }
}
```

Conclusion

React has got more popularity among the top IT companies like Facebook, PayPal, Instagram, Uber, etc., around the world especially in India. Hooks is becoming a trend in the React community as it removes the state management complexities.

This article includes the most frequently asked ReactJS and React Hooks interview questions and answers that will help you in interview preparations. Also, remember that your success during the interview is not all about your technical skills, it will also be based on your state of mind and the good impression that you will make at first. All the best!!

Useful References and Resources:

- "Beginning React with Hooks " book by Greg Lim
- “Learn React Hooks” book by Daniel Bugl
- Node.js vs React.js
- React Native Interview Questions
- Angular Interview Questions and Answers

22. How to pass data between sibling components using React router?

Passing data between sibling components of React is possible using React Router with the help of `history.push` and `match.params`.

In the code given below, we have a Parent component `AppDemo.js` and have two Child Components `HomePage` and `AboutPage`. Everything is kept inside a Router by using React-router Route. It is also having a route for `/about/{params}` where we will pass the data.

```
import React, { Component } from ‘react’;
class AppDemo extends Component {
render() {
  return (
    <Router>
      <div className="AppDemo">
      <ul>
        <li>
          <NavLink to="/"  activeStyle={{ color:'blue' }}>Home</NavLink>
        </li>
        <li>
          <NavLink to="/about"  activeStyle={{ color:'blue' }}>About
 </NavLink>
        </li>
 </ul>
            <Route path="/about/:aboutId" component={AboutPage} />
            <Route path="/about" component={AboutPage} />
            <Route path="/" component={HomePage} />
      </div>
    </Router>
  );
}
}
export default AppDemo;
```

The HomePage is a functional component with a button. On button click, we are using `props.history.push(‘/about/’ + data)` to programmatically navigate into `/about/data`.

```
export default function HomePage(props) {
 const handleClick = (data) => {
  props.history.push('/about/' + data);
 }
return (
  <div>
    <button onClick={() => handleClick('DemoButton')}>To Abo
```

- Angular Interview Questions and Answers

22. How to pass data between sibling components using React router?

Passing data between sibling components of React is possible using React Router with the help of `history.push` and `match.params`.

In the code given below, we have a Parent component `AppDemo.js` and have two Child Components `HomePage` and `AboutPage`. Everything is kept inside a Router by using React-router Route. It is also having a route for `/about/{params}` where we will pass the data.

```
import React, { Component } from 'react';
class AppDemo extends Component {
render() {
  return (
    <Router>
      <div className="AppDemo">
      <ul>
        <li>
          <NavLink to="/"  activeStyle={{ color:'blue' }}>Home</NavLink>
        </li>
        <li>
          <NavLink to="/about"  activeStyle={{ color:'blue' }}>About
 </NavLink>
        </li>
 </ul>
            <Route path="/about/:aboutId" component={AboutPage} />
            <Route path="/about" component={AboutPage} />
            <Route path="/" component={HomePage} />
      </div>
    </Router>
  );
}
}
export default AppDemo;
```

The HomePage is a functional component with a button. On button click, we are using `props.history.push('/about/' + data)` to programmatically navigate into `/about/data`.

```
export default function HomePage(props) {
 const handleClick = (data) => {
  props.history.push('/about/' + data);
 }
return (
  <div>
    <button onClick={() => handleClick('DemoButton')}>To About</button>
  </div>
)
}
```

Also, the functional component AboutPage will obtain the data passed by `props.match.params.aboutId`.

```
export default function AboutPage(props) {
if(!props.match.params.aboutId) {
    return <div>No Data Yet</div>
}
return (
  <div>
    {`Data obtained from HomePage is ${props.match.params.aboutId}`}
  </div>
)
}
```

After button click in the HomePage the page will look like below:

23. How to re-render the view when the browser is resized?

23. How to re-render the view when the browser is resized?

It is possible to listen to the resize event in **componentDidMount()** and then update the width and height dimensions. It requires the removal of the event listener in the **componentWillUnmount()** method.

Using the below-given code, we can render the view when the browser is resized.

```
class WindowSizeDimensions extends React.Component {
 constructor(props){
   super(props);
   this.updateDimension = this.updateDimension.bind(this);
 }

 componentWillMount() {
   this.updateDimension()
 }
 componentDidMount() {
   window.addEventListener('resize', this.updateDimension)
 }
 componentWillUnmount() {
   window.removeEventListener('resize', this.updateDimension)
 }
 updateDimension() {
   this.setState({width: window.innerWidth, height: window.innerHeight})
 }
 render() {
   return <span>{this.state.width} x {this.state.height}</span>
 }
}
```

24. How to create a switching component for displaying different pages?

A switching component refers to a component that will render one of the multiple components. We should use an object for mapping prop values to components.

A below-given example will show you how to display different pages based on page prop using switching component:

```
import HomePage from './HomePage'
import AboutPage from './AboutPage'
import FacilitiesPage from './FacilitiesPage'
import ContactPage from './ContactPage'
import HelpPage from './HelpPage'
const PAGES = {
 home: HomePage,
 about: AboutPage,
 facilitiess: FacilitiesPage,
 contact: ContactPage
 help: HelpPage
}
const Page = (props) => {
 const Handler = PAGES[props.page] || HelpPage
 return <Handler {...props} />
}
// The PAGES object keys can be used in the prop types for catching errors during dev-time.
Page.propTypes = {
 page: PropTypes.oneOf(Object.keys(PAGES)).isRequired
}
```

25. Explain how to create a simple React Hooks example program.

I will assume that you are having some coding knowledge about JavaScript and have installed Node on your system for creating a below given React Hook program. An installation of Node comes along with the command-line tools: npm and npx, where npm is useful to install the packages into a project and npx is useful in running commands of Node from the command line. The npx looks in the current project folder for checking whether a command has been installed there. When the command is not available on your computer, the npx will look in the npmjs.com repository, then the latest version of the command script will be loaded and will run without locally installing it. This feature is useful in creating a skeleton React application within a few key presses.

Open the Terminal inside the folder of your choice, and run the following command:

```
npx create-react-app react-items-with-hooks
```

Here, the `create-react-app` is an app initializer created by Facebook, to help with the easy and quick creation of React application, providing options to customize it while creating the application? The above command will create a new folder named react-items-with-hooks and it will be initialized with a basic React application. Now, you will be able to open the project in your favourite IDE. You can see an src folder inside the project along with the main application component `App.js`. This file is having a single function `App()` which will return an element and it will make use of an extended JavaScript syntax(JSX) for defining the component.

JSX will permit you for writing HTML-style template syntax directly into the JavaScript file. This mixture of JavaScript and HTML will be converted by React toolchain into pure JavaScript that will render the HTML element.

It is possible to define your own React components by writing a function that will
new file `src/SearchItem.js` and put the following code into it.

25. Explain how to create a simple React Hooks example program.

I will assume that you are having some coding knowledge about JavaScript and have installed Node on your system for creating a below given React Hook program. An installation of Node comes along with the command-line tools: npm and npx, where npm is useful to install the packages into a project and npx is useful in running commands of Node from the command line. The npx looks in the current project folder for checking whether a command has been installed there. When the command is not available on your computer, the npx will look in the npmjs.com repository, then the latest version of the command script will be loaded and will run without locally installing it. This feature is useful in creating a skeleton React application within a few key presses.

Open the Terminal inside the folder of your choice, and run the following command:

```
npx create-react-app react-items-with-hooks
```

Here, the `create-react-app` is an app initializer created by Facebook, to help with the easy and quick creation of React application, providing options to customize it while creating the application? The above command will create a new folder named react-items-with-hooks and it will be initialized with a basic React application. Now, you will be able to open the project in your favourite IDE. You can see an src folder inside the project along with the main application component `App.js`. This file is having a single function `App()` which will return an element and it will make use of an extended JavaScript syntax(JSX) for defining the component.

JSX will permit you for writing HTML-style template syntax directly into the JavaScript file. This mixture of JavaScript and HTML will be converted by React toolchain into pure JavaScript that will render the HTML element.

It is possible to define your own React components by writing a function that will return a JSX element. You can try this by creating a new file `src/SearchItem.js` and put the following code into it.

```
import React from 'react';
export function SearchItem() {
 return (
   <div>
     <div className="search-input">
       <input type="text" placeholder="SearchItem"/>
     </div>
     <h1 className="h1">Search Results</h1>
     <div className="items">
       <table>
         <thead>
           <tr>
             <th className="itemname-col">Item Name</th>
             <th className="price-col">Price</th>
             <th className="quantity-col">Quantity</th>
           </tr>
         </thead>
         <tbody></tbody>
       </table>
     </div>
   </div>
 );
}
```

This is all about how you can create a component. It will only display the empty table and doesn't do anything. But you will be able to use the Search component in the application. Open the file `src/App.js` and add the import statement given below to the top of the file.

```
import { SearchItem } from './SearchItem';
```

Now, from the logo.svg, import will be removed and then contents of returned value in the function `App()` will be replaced with the following code:

```
<div className="App">
 <header>
   Items with Hooks
 </header>
 <SearchItem/>
</div>
```

You can notice that the element <SearchItem/> has been used just similar to an HTML element. The JSX syntax will enable for including the components in this approach directly within the JavaScript code. Your application can be tested by running the below-given command in your terminal.

```
npm start
```

This command will compile your application and open your default browser into http://localhost:4000. This command can be kept on running when code development is in progress to make sure that the application is up-to-date, and also this browser page will be reloaded each time you modify and save the code.

This application will work finely, but it doesn't look nice as it doesn't react to any input from the user. You can make it more interactive by adding a state with React Hooks, adding authentication, etc.

26. Explain conditional rendering in React.

Conditional rendering refers to the dynamic output of user interface markups based on a condition state. It works in the same way as JavaScript conditions. Using conditional rendering, it is possible to toggle specific application functions, API data rendering, hide or show elements, decide permission levels, authentication handling, and so on.

There are different approaches for implementing conditional rendering in React. S

This command will compile your application and open your default browser into http://localhost:4000. This command can be kept on running when code development is in progress to make sure that the application is up-to-date, and also this browser page will be reloaded each time you modify and save the code.

This application will work finely, but it doesn't look nice as it doesn't react to any input from the user. You can make it more interactive by adding a state with React Hooks, adding authentication, etc.

26. Explain conditional rendering in React.

Conditional rendering refers to the dynamic output of user interface markups based on a condition state. It works in the same way as JavaScript conditions. Using conditional rendering, it is possible to toggle specific application functions, API data rendering, hide or show elements, decide permission levels, authentication handling, and so on.

There are different approaches for implementing conditional rendering in React. Some of them are:

- Using if-else conditional logic which is suitable for smaller as well as for medium-sized applications
- Using ternary operators, which takes away some amount of complication from if-else statements
- Using element variables, which will enable us to write cleaner code.

27. Can React Hook replaces Redux?

The React Hook cannot be considered as a replacement for Redux (It is an open-source, JavaScript library useful in managing the application state) when it comes to the management of the global application state tree in large complex applications, even though the React will provide a useReducer hook that manages state transitions similar to Redux. Redux is very useful at a lower level of component hierarchy to handle the pieces of a state which are dependent on each other, instead of a declaration of multiple useState hooks.

In commercial web applications which is larger, the complexity will be high, so using only React Hook may not be sufficient. Few developers will try to tackle the challenge with the help of React Hooks and others will combine React Hooks with the Redux.

28. What is React Router?

React Router refers to the standard library used for routing in React. It permits us for building a single-page web application in React with navigation without even refreshing the page when the user navigates. It also allows to change the browser URL and will keep the user interface in sync with the URL. React Router will make use of the component structure for calling the components, using which appropriate information can be shown. Since React is a component-based framework, it's not necessary to include and use this package. Any other compatible routing library would also work with React.

The major components of React Router are given below:

- **BrowserRouter:** It is a router implementation that will make use of the HTML5 history API (pushState, popstate, and event replaceState) for keeping your UI to be in sync with the URL. It is the parent component useful in storing all other components.
- **Routes:** It is a newer component that has been introduced in the React v6 and an upgrade of the component.
- **Route:** It is considered to be a conditionally shown component and some UI will be rendered by this whenever there is a match between its path and the current URL.
- **Link:** It is useful in creating links to various routes and implementing navigation all over the application. It works similarly to the anchor tag in HTML.

29. Do Hooks cover all the functionalities provided by the classes?

Our goal is for Hooks to cover all the functionalities for classes at its earliest. There are no Hook equivalents for the following methods that are not introduced in Hooks yet:

- `getSnapshotBeforeUpdate()`
- `getDerivedStateFromError()`
- `componentDidCatch()`

Since it is an early time for Hooks, few third-party libraries may not be compatible with Hooks at present, but they will be added soon.

30. How does the performance of using Hooks will differ in comparison with the classes?

- React Hooks will avoid a lot of overheads such as the instance creation, binding of events, etc., that are present with classes.
- Hooks in React will result in smaller component trees since they will be avoiding the nesting that exists in HOCs (Higher Order Components) and will render props which result in less amount of work to be done by React.

31. Differentiate React Hooks vs Classes.

React Hooks	Classes
It is used in functional components of React.	It is used in class-based components of React.
It will not require a declaration of any kind of constructor.	It is necessary to declare the constructor inside the class component.
It does not require the use of `this` keyword in state declaration or modification.	Keyword `this` will be used in state declaration (`this.state`) and in modification (`this.setState()`).
It is easier to use because of the `useState` functionality.	No specific function is available for helping us to access the state and its corresponding setState variable.
React Hooks can be helpful in implementing Redux and context API.	Because of the long setup of state declarations, class states are generally not preferred.

32. Explain about types of Hooks in React.

There are two types of Hooks in React. They are:

functionality.	and its corresponding setState variable.
React Hooks can be helpful in implementing Redux and context API.	Because of the long setup of state declarations, class states are generally not preferred.

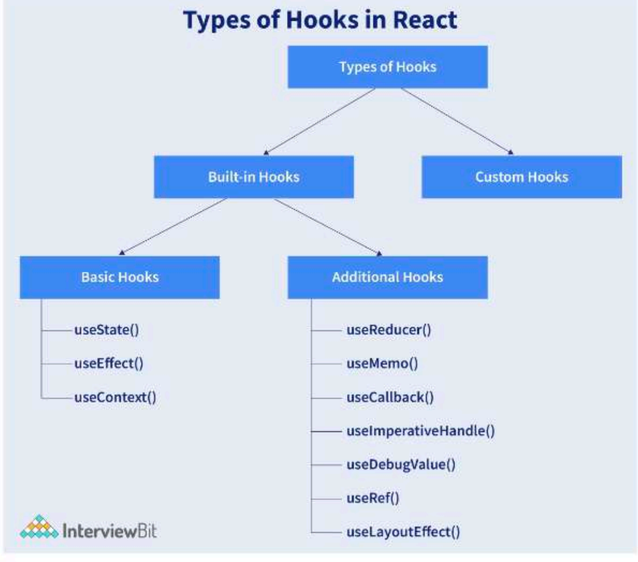
32. Explain about types of Hooks in React.

There are two types of Hooks in React. They are:

1. Built-in Hooks: The built-in Hooks are divided into 2 parts as given below:

- **Basic Hooks:**
  - `useState()`: This functional component is used to set and retrieve the state.
  - `useEffect()`: It enables for performing the side effects in the functional components.
  - `useContext()`: It is used for creating common data that is to be accessed by the components hierarchy without having to pass the props down to each level.
- **Additional Hooks:**
  - `useReducer()` : It is used when there is a complex state logic that is having several sub-values or when the upcoming state is dependent on the previous state. It will also enable you to optimization of component performance that will trigger deeper updates as it is permitted to pass the dispatch down instead of callbacks.
  - `useMemo()` : This will be used for recomputing the memoized value when there is a change in one of the dependencies. This optimization will help for avoiding expensive calculations on each render.
  - `useCallback()` : This is useful while passing callbacks into the optimized child components and depends on the equality of reference for the prevention of unneeded renders.
  - `useImperativeHandle()`: It will enable modifying the instance that will be passed with the ref object.
  - `useDebugValue()`: It is used for displaying a label for custom hooks in React DevTools.
  - `useRef()` : It will permit creating a reference to the DOM element directly within the functional component.
  - `useLayoutEffect()`: It is used for the reading layout from the DOM and re-rendering synchronously.

2. Custom Hooks: A custom Hook is basically a function of JavaScript. The Custom Hook working is similar to a regular function. The "use" at the beginning of the Custom Hook Name is required for React to understand that this is a custom Hook and also it will describe that this specific function follows the rules of Hooks. Moreover, developing custom Hooks will enable you for extracting component logic from within reusable functions.



33. Does React Hook work with static typing?

Static typing refers to the process of code check during the time of compilation for ensuring all variables will be statically typed. React Hooks are functions that are designed to make sure about all attributes must be statically typed. For enforcing stricter static typing within our code, we can make use of the React API with custom Hooks.

34. What are the lifecycle methods of React?

React lifecycle hooks will have the methods that will be automatically called at different phases in the component lifecycle and thus it provides good control over what happens at the invoked point. It provides the power to effectively control and manipulate what goes on throughout the component lifecycle.

For example, if you are developing the YouTube application, then the application will make use of a network for buffering the videos and it consumes the power of the battery (assume only these two). After playing the video if the user switches to any other application, then you should make sure that the resources like network and battery are being used most efficiently. You can stop or pause the video buffering which in turn stops the battery and network usage when the user switches to another application after video play.

So we can say that the developer will be able to produce a quality application with the help of lifecycle methods and it also helps developers to make sure to plan what and how to do it at different points of birth, growth, or death of user interfaces.

The various lifecycle methods are:

- `constructor()`: This method will be called when the component is initiated before anything has been done. It helps to set up the initial state and initial values.
- `getDerivedStateFromProps()`: This method will be called just before element(s) rendering in the DOM. It helps to set up the state object depending on the initial props. The getDerivedStateFromProps() method will have a state as an argument and it returns an object that made changes to the state. This will be the first method to be called on an updating of a component.
- `render()`: This method will output or re-render the HTML to the DOM with n… method and will be called always while the remaining methods are optional a…
- `componentDidMount()`: This method will be called after the rendering of t…

Static typing refers to the process of code check during the time of compilation for ensuring all variables will be statically typed. React Hooks are functions that are designed to make sure about all attributes must be statically typed. For enforcing stricter static typing within our code, we can make use of the React API with custom Hooks.

34. What are the lifecycle methods of React?

React lifecycle hooks will have the methods that will be automatically called at different phases in the component lifecycle and thus it provides good control over what happens at the invoked point. It provides the power to effectively control and manipulate what goes on throughout the component lifecycle.

For example, if you are developing the YouTube application, then the application will make use of a network for buffering the videos and it consumes the power of the battery (assume only these two). After playing the video if the user switches to any other application, then you should make sure that the resources like network and battery are being used most efficiently. You can stop or pause the video buffering which in turn stops the battery and network usage when the user switches to another application after video play.

So we can say that the developer will be able to produce a quality application with the help of lifecycle methods and it also helps developers to make sure to plan what and how to do it at different points of birth, growth, or death of user interfaces.

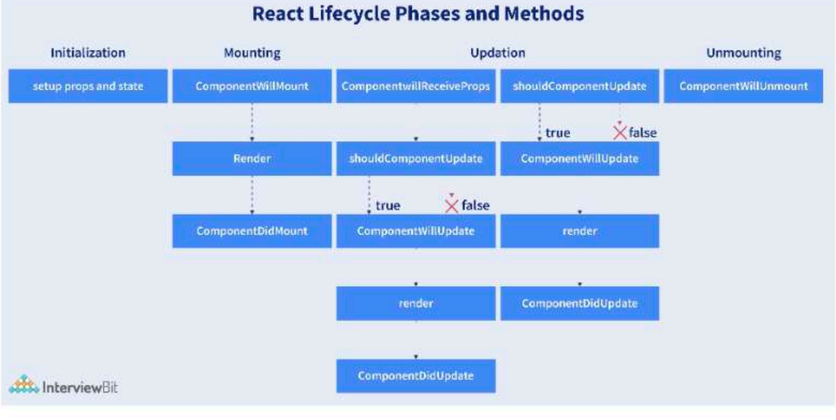
The various lifecycle methods are:

- `constructor()`: This method will be called when the component is initiated before anything has been done. It helps to set up the initial state and initial values.
- `getDerivedStateFromProps()`: This method will be called just before element(s) rendering in the DOM. It helps to set up the state object depending on the initial props. The getDerivedStateFromProps() method will have a state as an argument and it returns an object that made changes to the state. This will be the first method to be called on an updating of a component.
- `render()`: This method will output or re-render the HTML to the DOM with new changes. The render() method is an essential method and will be called always while the remaining methods are optional and will be called only if they are defined.
- `componentDidMount()`: This method will be called after the rendering of the component. Using this method, you can run statements that need the component to be already kept in the DOM.
- `shouldComponentUpdate()`: The Boolean value will be returned by this method which will specify whether React should proceed further with the rendering or not. The default value for this method will be True.
- `getSnapshotBeforeUpdate()`: This method will provide access for the props as well as for the state before the update. It is possible to check the previously present value before the update, even after the update.
- `componentDidUpdate()`: This method will be called after the component has been updated in the DOM.
- `componentWillUnmount()`: This method will be called when the component removal from the DOM is about to happen.

35. What are the different phases of the component lifecycle?

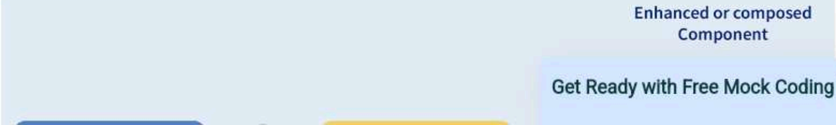
There are four different phases in the lifecycle of React component. They are:

- **Initialization:** During this phase, React component will prepare by setting up the default props and initial state for the upcoming tough journey.
- **Mounting:** Mounting refers to putting the elements into the browser DOM. Since React uses VirtualDOM, the entire browser DOM which has been currently rendered would not be refreshed. This phase includes the lifecycle methods `componentWillMount` and `componentDidMount`.
- **Updating:** In this phase, a component will be updated when there is a change in the state or props of a component. This phase will have lifecycle methods like `componentWillUpdate`, `shouldComponentUpdate`, `render`, and `componentDidUpdate`.
- **Unmounting:** In this last phase of the component lifecycle, the component will be removed from the DOM or will be unmounted from the browser DOM. This phase will have the lifecycle method named `componentWillUnmount`.



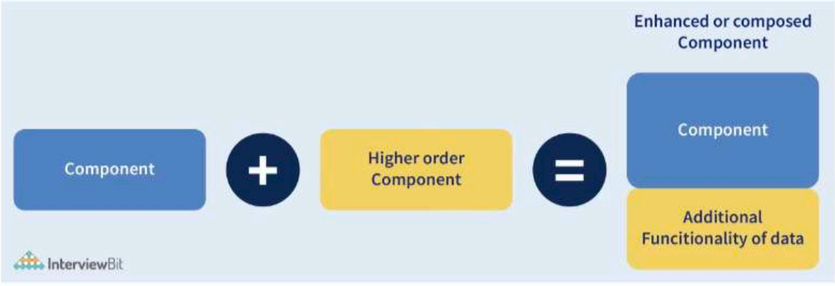
36. What are Higher Order Components?

Simply put, Higher-Order Component(HOC) is a function that takes in a component and returns a new component.



36. What are Higher Order Components?

Simply put, Higher-Order Component(HOC) is a function that takes in a component and returns a new component.



When do we need a Higher Order Component?

While developing React applications, we might develop components that are quite similar to each other with minute differences. In most cases, developing similar components might not be an issue but, while developing larger applications we need to keep our code **DRY**, therefore, we want an **abstraction** that allows us to define this logic in a single place and share it across components. HOC allows us to create that abstraction.

Example of a HOC:

Consider the following components having similar functionality. The following component displays the list of articles:

```
// "GlobalDataSource" is some global data source
class ArticlesList extends React.Component {
 constructor(props) {
   super(props);
   this.handleChange = this.handleChange.bind(this);
   this.state = {
     articles: GlobalDataSource.getArticles(),
   };
 }
 componentDidMount() {
   // Listens to the changes added
   GlobalDataSource.addChangeListener(this.handleChange);
 }
 componentWillUnmount() {
   // Listens to the changes removed
   GlobalDataSource.removeChangeListener(this.handleChange);
 }
 handleChange() {
   // States gets Update whenver data source changes
   this.setState({
     articles: GlobalDataSource.getArticles(),
   });
 }
 render() {
   return (
     <div>
       {this.state.articles.map((article) => (
         <ArticleData article={article} key={article.id} />
       ))}
     </div>
   );
 }
}
```

The following component displays the list of users:

```
// "GlobalDataSource" is some global data source
class UsersList extends React.Component {
 constructor(props) {
   super(props);
   this.handleChange = this.handleChange.bind(this);
   this.state = {
     users: GlobalDataSource.getUsers(),
   };
 }
 componentDidMount() {
   // Listens to the changes added
   GlobalDataSource.addChangeListener(this.handleChange);
```

```
 }
 componentWillUnmount() {
   // Listens to the changes removed
   GlobalDataSource.removeChangeListener(this.handleChange);
 }
 handleChange() {
   // States gets Update whenver data source changes
   this.setState({
     articles: GlobalDataSource.getArticles(),
   });
 }
 render() {
   return (
     <div>
       {this.state.articles.map((article) => (
         <ArticleData article={article} key={article.id} />
       ))}
     </div>
   );
 }
}
```

The following component displays the list of users:

```
// "GlobalDataSource" is some global data source
class UsersList extends React.Component {
 constructor(props) {
   super(props);
   this.handleChange = this.handleChange.bind(this);
   this.state = {
     users: GlobalDataSource.getUsers(),
   };
 }
 componentDidMount() {
   // Listens to the changes added
   GlobalDataSource.addChangeListener(this.handleChange);
 }
 componentWillUnmount() {
   // Listens to the changes removed
   GlobalDataSource.removeChangeListener(this.handleChange);
 }
 handleChange() {
   // States gets Update whenver data source changes
   this.setState({
     users: GlobalDataSource.getUsers(),
   });
 }
 render() {
   return (
     <div>
       {this.state.users.map((user) => (
         <UserData user={user} key={user.id} />
       ))}
     </div>
   );
 }
}
```

Notice the above components, both have similar functionality but, they are calling different methods to an API endpoint.

Let's create a Higher Order Component to create an abstraction:

```
// Higher Order Component which takes a component
// as input and returns another component
// "GlobalDataSource" is some global data source
function HOC(WrappedComponent, selectData) {
 return class extends React.Component {
   constructor(props) {
     super(props);
     this.handleChange = this.handleChange.bind(this);
     this.state = {
       data: selectData(GlobalDataSource, props),
     };
   }
   componentDidMount() {
     // Listens to the changes added
     GlobalDataSource.addChangeListener(this.handleChange);
   }
   componentWillUnmount() {
     // Listens to the changes removed
     GlobalDataSource.removeChangeListener(this.handleChange);
   }
   handleChange() {
     this.setState({
       data: selectData(GlobalDataSource, this.props),
     });
```

```
        <UserData user={user} key={user.id} />
      ))}
    </div>
  );
 }
}
```

Notice the above components, both have similar functionality but, they are calling different methods to an API endpoint.

Let's create a Higher Order Component to create an abstraction:

```
// Higher Order Component which takes a component
// as input and returns another component
// "GlobalDataSource" is some global data source
function HOC(WrappedComponent, selectData) {
 return class extends React.Component {
   constructor(props) {
     super(props);
     this.handleChange = this.handleChange.bind(this);
     this.state = {
       data: selectData(GlobalDataSource, props),
     };
   }
   componentDidMount() {
     // Listens to the changes added
     GlobalDataSource.addChangeListener(this.handleChange);
   }
   componentWillUnmount() {
     // Listens to the changes removed
     GlobalDataSource.removeChangeListener(this.handleChange);
   }
   handleChange() {
     this.setState({
       data: selectData(GlobalDataSource, this.props),
     });
   }
   render() {
     // Rendering the wrapped component with the latest data data
     return <WrappedComponent data={this.state.data} {...this.props} />;
   }
 };
}
```

We know HOC is a function that takes in a component and returns a component.

In the code above, we have created a function called HOC which returns a component and performs functionality that can be shared across both the **ArticlesList** component and **UsersList** Component.

The second parameter in the HOC function is the function that calls the method on the API endpoint.

We have reduced the duplicated code of the **componentDidUpdate** and **componentDidMount** functions.

Using the concept of Higher-Order Components, we can now render the **ArticlesList** and **UsersList** components in the following way:

```
const ArticlesListWithHOC = HOC(ArticlesList, (GlobalDataSource) => GlobalDataSource.getArticles()
const UsersListWithHOC = HOC(UsersList, (GlobalDataSource) => GlobalDataSource.getUsers());
```

Remember, we are not trying to change the functionality of each component, we are trying to share a single functionality across multiple components using HOC.

37. How to pass data between react components?

Parent Component to Child Component (using props)

With the help of props, we can send data from a parent to a child component.

How do we do this?

Consider the following Parent Component:

Remember, we are not trying to change the functionality of each component, we are trying to share a single functionality across multiple components using HOC.

37. How to pass data between react components?

Parent Component to Child Component (using props)

With the help of props, we can send data from a parent to a child component.

How do we do this?

Consider the following Parent Component:

```
import ChildComponent from "./Child";
   function ParentComponent(props) {
    let [counter, setCounter] = useState(0);

    let increment = () => setCounter(++counter);

    return (
      <div>
        <button onClick={increment}>Increment Counter</button>
        <ChildComponent counterValue={counter} />
      </div>
    );
   }
```

As one can see in the code above, we are rendering the child component inside the parent component, by providing a prop called counterValue. The value of the counter is being passed from the parent to the child component.

We can use the data passed by the parent component in the following way:

```
function ChildComponent(props) {
return (
  <div>
    <p>Value of counter: {props.counterValue}</p>
  </div>
);
}
```

We use the **props.counterValue** to display the data passed on by the parent component.

Child Component to Parent Component (using callbacks)

This one is a bit tricky. We follow the steps below:

- Create a callback in the parent component which takes in the data needed as a parameter.
- Pass this callback as a prop to the child component.
- Send data from the child component using the callback.

We are considering the same example above but in this case, we are going to pass the updated counterValue from child to parent.

Step1 and Step2: Create a callback in the parent component, pass this callback as a prop.

```
function ParentComponent(props) {
let [counter, setCounter] = useState(0);
let callback = valueFromChild => setCounter(valueFromChild);
return (
  <div>
    <p>Value of counter: {counter}</p>
    <ChildComponent callbackFunc={callback} counterValue={counter} />
  </div>
);
}
```

As one can see in the code above, we created a function called callback which takes in the data received from the child component as a parameter.

Next, we passed the function callback as a prop to the child component.

- Pass this callback as a prop to the child component.
- Send data from the child component using the callback.

We are considering the same example above but in this case, we are going to pass the updated counterValue from child to parent.

Step1 and Step2: Create a callback in the parent component, pass this callback as a prop.

```
function ParentComponent(props) {
let [counter, setCounter] = useState(0);
let callback = valueFromChild => setCounter(valueFromChild);
return (
  <div>
    <p>Value of counter: {counter}</p>
    <ChildComponent callbackFunc={callback} counterValue={counter} />
  </div>
);
}
```

As one can see in the code above, we created a function called callback which takes in the data received from the child component as a parameter.

Next, we passed the function callback as a prop to the child component.

Step3: Pass data from the child to the parent component.

```
function ChildComponent(props) {
let childCounterValue = props.counterValue;
return (
  <div>
    <button onClick={() => props.callbackFunc(++childCounterValue)}>
      Increment Counter
    </button>
  </div>
);
}
```

In the code above, we have used the props.counterValue and set it to a variable called childCounterValue.

Next, on button click, we pass the incremented childCounterValue to the **props.callbackFunc**.

This way, we can pass data from the child to the parent component.

38. Name a few techniques to optimize React app performance.

There are many ways through which one can optimize the performance of a React app, let's have a look at some of them:

- **Using useMemo( ) -**
  - It is a React hook that is used for caching CPU-Expensive functions.
  - Sometimes in a React app, a CPU-Expensive function gets called repeatedly due to re-renders of a component, which can lead to slow rendering.
    useMemo( ) hook can be used to cache such functions. By using useMemo( ), the CPU-Expensive function gets called only when it is needed.
- **Using React.PureComponent -**
  - It is a base component class that checks the state and props of a component to know whether the component should be updated.
  - Instead of using the simple React.Component, we can use React.PureComponent to reduce the re-renders of a component unnecessarily.
- **Maintaining State Colocation -**
  - This is a process of moving the state as close to where you need it as possible.
  - Sometimes in React app, we have a lot of unnecessary states inside the parent component which makes the code less readable and harder to maintain. Not to forget, having many states inside a single component leads to unnecessary re-renders for the component.
  - It is better to shift states which are less valuable to the parent component, to a separate component.
- **Lazy Loading -**
  - It is a technique used to reduce the load time of a React app. Lazy loading helps reduce the risk of web app performances to a minimum.

39. What are the different ways to style a React component?

There are many different ways through which one can style a React component. Some of the ways are :

- **Inline Styling:** We can directly style an element using inline style attributes. Make sure the value of style is a JavaScript object:

```
class RandomComponent extends React.Component {
 render() {
   return (
     <div>
       <h3 style={{ color: "Yellow" }}>This is a heading</h3>
       <p style={{ fontSize: "32px" }}>This is a paragraph</p>
     </div>
   );
 }
}
```

- **Using JavaScript object:** We can create a separate JavaScript object and set the desired style properties. This object can be used as the value of the inline style attribute.

- **Lazy Loading -**
  - It is a technique used to reduce the load time of a React app. Lazy loading helps reduce the risk of web app performances to a minimum.

39. What are the different ways to style a React component?

There are many different ways through which one can style a React component. Some of the ways are :

- **Inline Styling:** We can directly style an element using inline style attributes. Make sure the value of style is a JavaScript object:

```
class RandomComponent extends React.Component {
 render() {
   return (
     <div>
       <h3 style={{ color: "Yellow" }}>This is a heading</h3>
       <p style={{ fontSize: "32px" }}>This is a paragraph</p>
     </div>
   );
 }
}
```

- **Using JavaScript object:** We can create a separate JavaScript object and set the desired style properties. This object can be used as the value of the inline style attribute.

```
class RandomComponent extends React.Component {
 paragraphStyles = {
   color: "Red",
   fontSize: "32px"
 };

 headingStyles = {
   color: "blue",
   fontSize: "48px"
 };

 render() {
   return (
     <div>
       <h3 style={this.headingStyles}>This is a heading</h3>
       <p style={this.paragraphStyles}>This is a paragraph</p>
     </div>
   );
 }
}
```

- **CSS Stylesheet:** We can create a separate CSS file and write all the styles for the component inside that file. This file needs to be imported inside the component file.

```
import './RandomComponent.css';

class RandomComponent extends React.Component {
 render() {
   return (
     <div>
       <h3 className="heading">This is a heading</h3>
       <p className="paragraph">This is a paragraph</p>
     </div>
   );
 }
}
```

- **CSS Modules:** We can create a separate CSS module and import this module inside our component. Create a file with ".module.css"' extension, styles.module.css:

```
.paragraph{
 color:"red";
 border:1px solid black;
}
```

We can import this file inside the component and use it:

```
import styles from  './styles.module.css';

class RandomComponent extends React.Component {
 render() {
   return (
     <div>
       <h3 className="heading">This is a heading</h3>
       <p className={styles.paragraph} >This is a paragraph</p>
     </div>
   );
 }
}
```

40. How to prevent re-renders in React?

- **Reason for re-renders in React:**
  - Re-rendering of a component and its child components occur when prop
  - Re-rendering components that are not updated, affects the performance

```
 }
}
```

40. How to prevent re-renders in React?

- **Reason for re-renders in React:**
  - Re-rendering of a component and its child components occur when props or the state of the component has been changed.
  - Re-rendering components that are not updated, affects the performance of an application.
- **How to prevent re-rendering:**

Consider the following components:

```
class Parent extends React.Component {
state = { messageDisplayed: false };
componentDidMount() {
  this.setState({ messageDisplayed: true });
}
render() {
  console.log("Parent is getting rendered");
  return (
    <div className="App">
      <Message />
    </div>
  );
}
}
class Message extends React.Component {
constructor(props) {
  super(props);
  this.state = { message: "Hello, this is vivek" };
}
render() {
  console.log("Message is getting rendered");
  return (
    <div>
      <p>{this.state.message}</p>
    </div>
  );
}
}
```

- The **Parent** component is the parent component and the **Message** is the child component. Any change in the parent component will lead to re-rendering of the child component as well. To prevent the re-rendering of child components, we use the shouldComponentUpdate( ) method:

  **Note- Use shouldComponentUpdate( ) method only when you are sure that it's a static component.

```
class Message extends React.Component {
constructor(props) {
  super(props);
  this.state = { message: "Hello, this is vivek" };
}
shouldComponentUpdate() {
  console.log("Does not get rendered");
  return false;
}
render() {
  console.log("Message is getting rendered");
  return (
    <div>
      <p>{this.state.message}</p>
    </div>
  );
}
}
```

As one can see in the code above, we have returned **false** from the shouldComponentUpdate( ) method, which prevents the child component from re-rendering.

41. Explain Strict Mode in React.

StrictMode is a tool added in **version 16.3** of React to highlight potential problems in an application. It performs additional checks on the application.

```
function App() {
 return (
   <React.StrictMode>
     <div classname="App">
       <Header/>
       <div>
         Page Content
       </div>
       <Footer/>
     </div>
   </React.StrictMode>
 );
}
```

```
}
```

As one can see in the code above, we have returned **false** from the shouldComponentUpdate( ) method, which prevents the child component from re-rendering.

41. Explain Strict Mode in React.

StrictMode is a tool added in **version 16.3** of React to highlight potential problems in an application. It performs additional checks on the application.

```
function App() {
 return (
   <React.StrictMode>
     <div classname="App">
       <Header/>
       <div>
         Page Content
       </div>
       <Footer/>
     </div>
   </React.StrictMode>
 );
}
```

To enable StrictMode, `<React.StrictMode>` tags need to be added inside the application:

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";
const rootElement = document.getElementById("root");
ReactDOM.render(
<React.StrictMode>
  <App />
</React.StrictMode>,
rootElement
);
```

StrictMode currently helps with the following issues:

- **Identifying components with unsafe lifecycle methods:**
  - Certain lifecycle methods are unsafe to use in asynchronous react applications. With the use of third-party libraries, it becomes difficult to ensure that certain lifecycle methods are not used.
  - StrictMode helps in providing us with a warning if any of the class components use an unsafe lifecycle method.
- **Warning about the usage of legacy string API:**
  - If one is using an older version of React, **callback ref** is the recommended way to manage **refs** instead of using the **string refs**. StrictMode gives a warning if we are using **string refs** to manage refs.
- **Warning about the usage of findDOMNode:**
  - Previously, findDOMNode( ) method was used to search the tree of a DOM node. This method is deprecated in React. Hence, the StrictMode gives us a warning about the usage of this method.
- **Warning about the usage of legacy context API (because the API is error-prone).**

React MCQ Questions

1. _____ is a necessary API for every React.js component.

- ◯ renderComponent
- ◯ render
- ◯ SetinitialComponent
- ◯ All of the above

2.React is mainly used for developing _____.

- ◯ Connectivity
- ◯ Database
- ◯ User interface
- ◯ Design platform

3.The Keys given to a list of elements in React should be _____.

- ◯ Not necessarily unique
- ◯ Unique among the siblings only
- ◯ Unique in the DOM (Document Object Model)
- ◯ None of the above