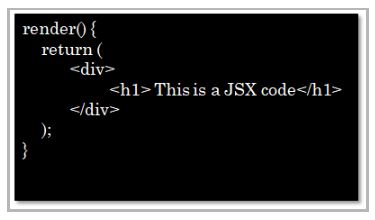
Here are some React Interview Questions on basic concepts.

### ****1. What are the features of React?****

|  |  |
| --- | --- |
| IMG_256 | **JSX:** JSX is a syntax extension to JavaScript. It is used with React to describe what the user interface should look like. By using JSX, we can write [HTML](https://www.simplilearn.com/tutorials/html-tutorial/what-is-html" \o "HTML" \t "https://www.simplilearn.com/tutorials/reactjs-tutorial/_blank) structures in the same file that contains [JavaScript](https://www.simplilearn.com/tutorials/javascript-tutorial/introduction-to-javascript" \o "JavaScript" \t "https://www.simplilearn.com/tutorials/reactjs-tutorial/_blank) code. |
| IMG_257 | **Components:**[Components](https://www.simplilearn.com/tutorials/reactjs-tutorial/reactjs-components" \o "Components" \t "https://www.simplilearn.com/tutorials/reactjs-tutorial/_blank) are the building blocks of any React application, and a single app usually consists of multiple components. It splits the user interface into independent, reusable parts that can be processed separately. |
| IMG_258 | **Virtual DOM:**React keeps a lightweight representation of the real DOM in the memory, and that is known as the virtual DOM. When the state of an object changes, virtual DOM changes only that object in the real DOM, rather than updating all the objects. |
| IMG_259 | **One-way data-binding:**React’s one-way [data binding](https://www.simplilearn.com/tutorials/angular-tutorial/angular-data-binding" \o "data binding" \t "https://www.simplilearn.com/tutorials/reactjs-tutorial/_blank) keeps everything modular and fast. A unidirectional data flow means that when designing a React app, you often nest child components within parent components. |
| IMG_260 | **High performance:**React updates only those components that have changed, rather than updating all the components at once. This results in much faster web applications. |

JSX is a syntax extension of JavaScript. It is used with React to describe what the user interface should look like. By using JSX, we can write HTML structures in the same file that contains JavaScript code.



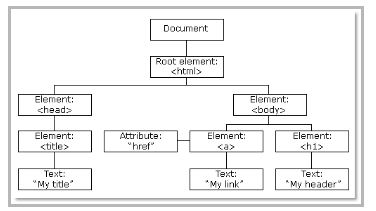
### ****3. Can web browsers read JSX directly?****

* Web browsers cannot read JSX directly. This is because they are built to only read regular JS objects and JSX is not a regular JavaScript object
* For a web browser to read a JSX file, the file needs to be transformed into a regular JavaScript object. For this, we use Babel

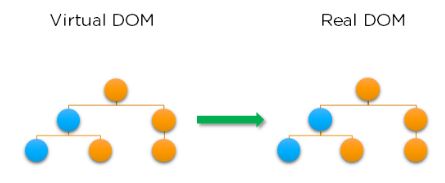


### ****4. What is the virtual DOM?****

DOM stands for Document Object Model. The DOM represents an HTML document with a logical tree structure. Each branch of the tree ends in a node, and each node contains objects.



React keeps a lightweight representation of the real DOM in the memory, and that is known as the virtual DOM. When the state of an object changes, the virtual DOM changes only that object in the real DOM, rather than updating all the objects. The following are some of the most frequently asked react interview questions.



### ****5. Why use React instead of other frameworks, like Angular?****

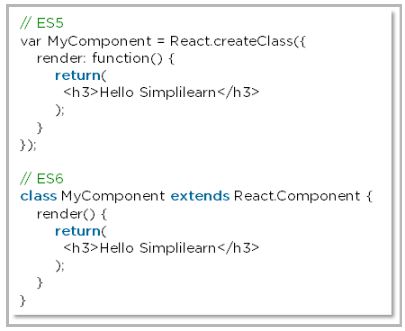
|  |  |
| --- | --- |
| IMG_267 | **Easy creation of dynamic applications:** React makes it easier to create dynamic web applications because it provides less coding and provides more functionality, whereas, with JavaScript applications, code tends to get complex very quickly. |
| IMG_268 | **Improved performance:**React uses virtual DOM, which makes web applications perform faster. Virtual DOM compares its previous state and updates only those components in the real DOM, whose states have changed, rather than updating all the components — like conventional web applications. |
| IMG_269 | **Reusable components:**Components are the building blocks of any React application, and a single app usually consists of multiple components. These components have their own logic and controls, and they can be reused through the application, which, in turn, dramatically reduces the development time of an application. |
| IMG_270 | **Unidirectional data flow:**React follows a unidirectional data flow. This means that when designing a React app, we often nest child components within parent components. And since the data flows in a single direction, it becomes easier to debug errors and know where the problem occurs in an application at the moment. |
| IMG_271 | **Dedicated tools for easy debugging:**Facebook has released a chrome extension that we can use to debug React applications. This makes the process of debugging React to web applications faster and easier. |

### 6. What is the difference between the ES6 and ES5 standards?

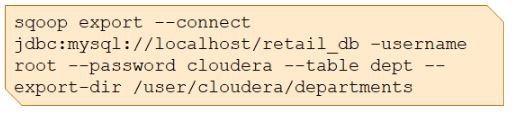
This is one of the most frequently asked react interview questions.

These are the few instances where ES6 syntax has changed from ES5 syntax:

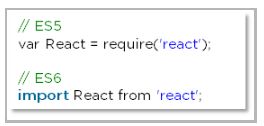
* **Components and Function**



* **exports vs export**



* **require vs import**



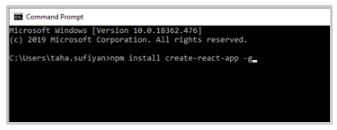
### ****7. How do you create a React app?****

These are the steps for creating a React app:

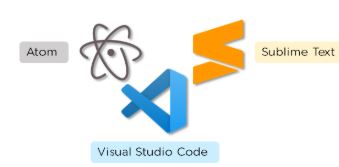
* Install [NodeJS](https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs" \o "NodeJS" \t "https://www.simplilearn.com/tutorials/reactjs-tutorial/_blank) on the computer because we need npm to install the React library. Npm is the node package manager that contains many JavaScript libraries, including React.



* Install the **create-react-app** package using the command prompt or terminal.



* Install a text editor of your choice, like VS Code or Sublime Text.



We have put together a set of [Node.js interview questions](https://www.simplilearn.com/tutorials/nodejs-tutorial/nodejs-interview-questions" \o "Node.js interview questions" \t "https://www.simplilearn.com/tutorials/reactjs-tutorial/_blank) in case you would like to explore them.Please note, This is one of the most frequently asked react interview questions.

### ****8. What is an event in React?****

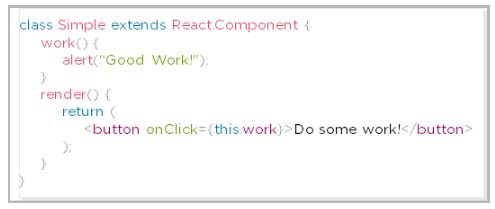
An event is an action that a user or system may trigger, such as pressing a key, a mouse click, etc.

* React events are named using camelCase, rather than lowercase in HTML.
* With JSX, you pass a function as the event handler, rather than a string in HTML.

|  |
| --- |
| <Button onPress={lightItUp} /> |

### ****9. How do you create an event in React?****

A React event can be created by doing the following:



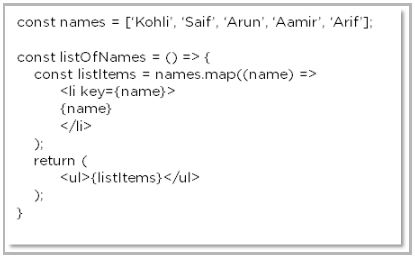
### ****10. What are synthetic events in React?****

* Synthetic events combine the response of different browser's native events into one API, ensuring that the events are consistent across different browsers.
* The application is consistent regardless of the browser it is running in. Here, **preventDefault**is a synthetic event.



### ****11. Explain how lists work in React****

* We create lists in React as we do in regular JavaScript. Lists display data in an ordered format
* The traversal of lists is done using the map() function



### ****12. Why is there a need for using keys in Lists?****

Keys are very important in lists for the following reasons:

* A key is a unique identifier and it is used to identify which items have changed, been updated or deleted from the lists
* It also helps to determine which components need to be re-rendered instead of re-rendering all the components every time. Therefore, it increases performance, as only the updated components are re-rendered

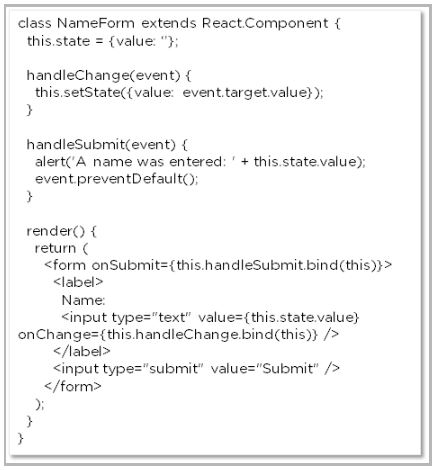
### ****13. What are forms in React?****

React employs forms to enable users to interact with web applications.

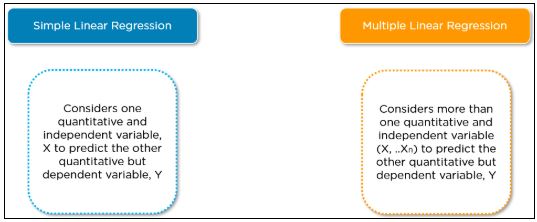
* Using forms, users can interact with the application and enter the required information whenever needed. Form contain certain elements, such as text fields, buttons, checkboxes, radio buttons, etc
* Forms are used for many different tasks such as user authentication, searching, filtering, indexing, etc

### ****14. How do you create forms in React?****

We create forms in React by doing the following:



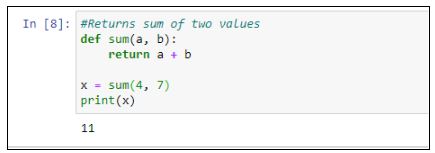
The above code will yield an input field with the label **Name**and a submit button. It will also alert the user when the submit button is pressed.



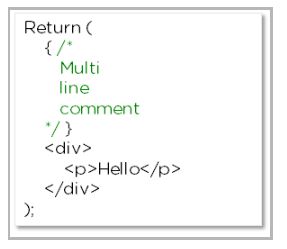
### ****15. How do you write comments in React?****

There are basically two ways in which we can write comments:

* Single-line comments

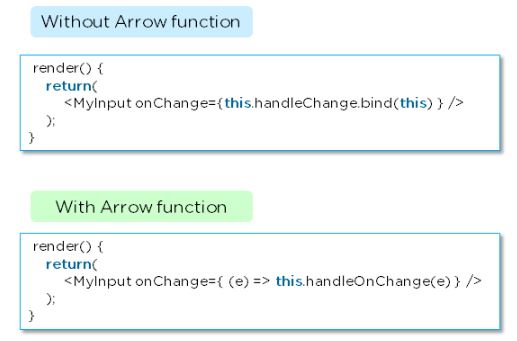


* Multi-line comments



### ****16. What is an arrow function and how is it used in React?****

* An arrow function is a short way of writing a function to React.
* It is unnecessary to bind **‘this’**inside the constructor when using an arrow function. This prevents bugs caused by the use of **‘this’**in React callbacks.



### ****17. How is React different from React Native?****

|  |  |  |
| --- | --- | --- |
|  | **React** | **React Native** |
| Release | 2013 | 2015 |
| Platform | Web | Mobile – Android, iOS |
| HTML | Yes | No |
| CSS | Yes | No |
| Prerequisites | JavaScript, HTML, CSS | React.js |

### ****18. How is React different from Angular?****

|  |  |  |
| --- | --- | --- |
|  | **Angular** | **React** |
| Author | Google | Facebook |
| Architecture | Complete MVC | View layer of MVC |
| DOM | Real DOM | Virtual DOM |
| Data-Binding | Bi-directional | Uni-directional |
| Rendering | Client-Side | Server-Side |
| Performance | Comparatively slow | Faster due to Virtual DOM |

In case you have any doubts about these Basic React interview questions and answers, please leave your questions in the comment section below.

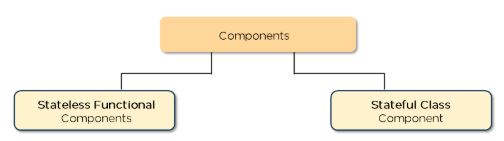
## ****ReactJS Interview Questions on Components****

Here are some React Interview Questions on components.

### ****19. What are the components in React?****

Components are the building blocks of any React application, and a single app usually consists of multiple components. A component is essentially a piece of the user interface. It splits the user interface into independent, reusable parts that can be processed separately.

There are two types of components in React:



* **Functional Components:**These types of components have no state of their own and only contain render methods, and therefore are also called **stateless components**. They may derive data from other components as props (properties).

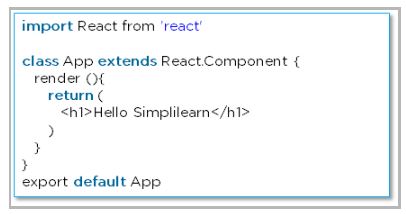
|  |
| --- |
| function Greeting(props) {    return <h1>Welcome to {props.name}</h1>;  } |

* **Class Components:**These types of components can hold and manage their own state and have a separate render method to return JSX on the screen. They are also called Stateful components as they can have a state.

|  |
| --- |
| class Greeting extends React.Component {    render() {      return <h1>Welcome to {this.props.name}</h1>;    }  } |

### ****20. What is the use of render() in React?****

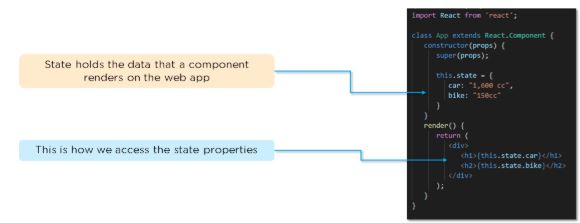
* It is required for each component to have a render() function. This function returns the HTML, which is to be displayed in the component.
* If you need to render more than one element, all of the elements must be inside one parent tag like <div>, <form>.



### ****21. What is a state in React?****

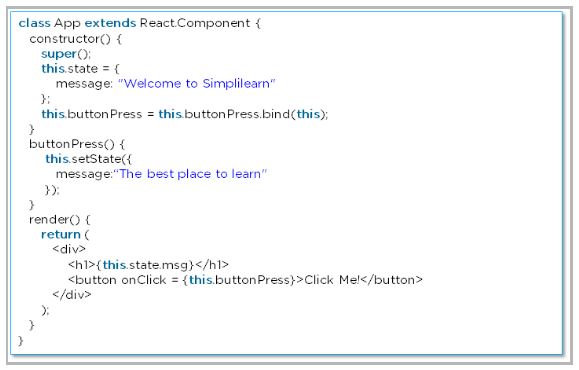
* The state is a built-in React object that is used to contain data or information about the component. The state in a component can change over time, and whenever it changes, the component re-renders.
* The change in state can happen as a response to user action or system-generated events. It determines the behavior of the component and how it will render.

### ****22. How do you implement state in React?****



### ****23. How do you update the state of a component?****

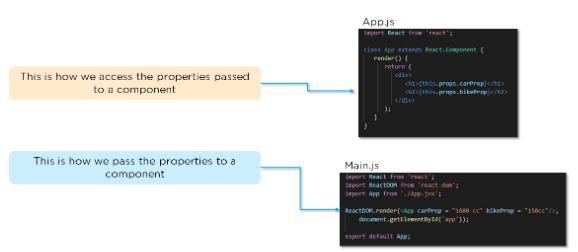
We can update the state of a component by using the built-in **‘setState()’**method:



### ****24. What are props in React?****

* [Props](https://www.simplilearn.com/tutorials/reactjs-tutorial/react-props" \o "Props" \t "https://www.simplilearn.com/tutorials/reactjs-tutorial/_blank) are short for Properties. It is a React built-in object that stores the value of attributes of a tag and works similarly to HTML attributes.
* Props provide a way to pass data from one component to another component. Props are passed to the component in the same way as arguments are passed in a function.

### ****25. How do you pass props between components?****



### ****26. What are the differences between state and props?****

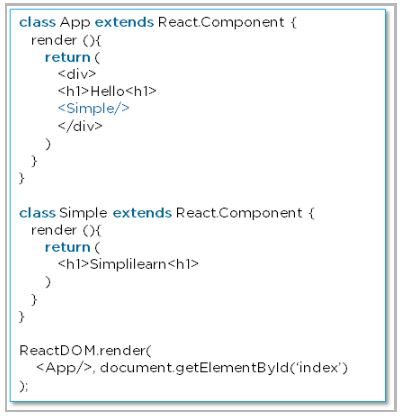
|  |  |  |
| --- | --- | --- |
|  | **State** | **Props** |
| Use | Holds information about the components | Allows to pass data from one component to other components as an argument |
| Mutability | Is mutable | Are immutable |
| Read-Only | Can be changed | Are read-only |
| Child components | Child components cannot access | Child component can access |
| Stateless components | Cannot have state | Can have props |

### ****27. What is a higher-order component in React?****

A higher-order component acts as a container for other components. This helps to keep components simple and enables re-usability. They are generally used when multiple components have to use a common logic.

### ****28. How can you embed two or more components into one?****

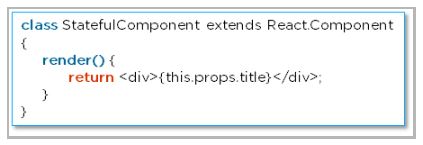
We can embed two or more components into one using this method:



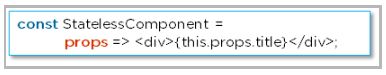
### ****29. What are the differences between class and functional components?****

|  |  |  |
| --- | --- | --- |
|  | **Class Components** | **Functional Components** |
| State | Can hold or manage state | Cannot hold or manage state |
| Simplicity | Complex as compared to the stateless component | Simple and easy to understand |
| Lifecycle methods | Can work with all lifecycle methods | Does not work with any lifecycle method |
| Reusability | Can be reused | Cannot be reused |

* **Class components example:**



* **Functional components example:**



### ****30. Explain the lifecycle methods of components.****

* **getInitialState():** This is executed before the creation of the component.
* **componentDidMount():** Is executed when the component gets rendered and placed on the DOM.
* **shouldComponentUpdate():** Is invoked when a component determines changes to the DOM and returns a “true” or “false” value based on certain conditions.
* **componentDidUpdate():** Is invoked immediately after rendering takes place.
* **componentWillUnmount():** Is invoked immediately before a component is destroyed and unmounted permanently.

So far, if you have any doubts about the above React interview questions and answers, please ask your questions in the section below.

## ****ReactJS Redux Interview Questions****

Here are some ReactJS Interview Questions on the ReactJS Redux concept.

### ****31. What is Redux?****

[Redux](https://www.simplilearn.com/tutorials/reactjs-tutorial/react-with-redux" \o "Redux" \t "https://www.simplilearn.com/tutorials/reactjs-tutorial/_blank) is an open-source, JavaScript library used to manage the application state. React uses Redux to build the user interface. It is a predictable state container for JavaScript applications and is used for the entire application’s state management.

### ****32. What are the components of Redux?****

* **Store:** Holds the state of the application.
* **Action:** The source information for the store.
* **Reducer:** Specifies how the application's state changes in response to actions sent to the store.



### ****33. What is the Flux?****

* Flux is the application architecture that Facebook uses for building web applications. It is a method of handling complex data inside a client-side application and manages how data flows in a React application.



* There is a single source of data (the store) and triggering certain actions is the only way way to update them.The actions call the dispatcher, and then the store is triggered and updated with their own data accordingly.



* When a dispatch has been triggered, and the store updates, it will emit a change event that the views can rerender accordingly.



### ****34. How is Redux different from Flux?****

|  |  |  |
| --- | --- | --- |
| **SN** | **Redux** | **Flux** |
| 1. | Redux is an open-source JavaScript library used to manage application State | Flux is an architecture and not a framework or library |
| 2. | Store’s state is immutable | Store’s state is mutable |
| 3. | Can only have a single-store | Can have multiple stores |
| 4. | Uses the concept of reducer | Uses the concept of the dispatcher |

So far, if you have any doubts about these React interview questions and answers, please leave your questions in the section below.

## ****ReactJS Router Questions****

Here are some ReactJS Interview Questions on React Router concepts.

### ****35. What is React Router?****

React Router is a routing library built on top of React, which is used to create routes in a React application. This is one of the most frequently asked react interview questions.

### ****36. Why do we need to React Router?****

* It maintains consistent structure and behavior and is used to develop single-page web applications.
* Enables multiple views in a single application by defining multiple routes in the React application.

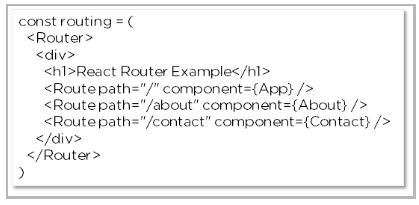
### ****37. How is React routing different from conventional routing?****

|  |  |  |
| --- | --- | --- |
| **SN** | **React Routing** | **Conventional routing** |
| 1. | Single HTML page | Each view is a new HTML file |
| 2. | The user navigates multiple views in the same file | The user navigates multiple files for each view |
| 3. | The page does not refresh since it is a single file | The page refreshes every time user navigates |
| 4. | Improved performance | Slower performance |

### ****38. How do you implement React routing?****

We can implement routing in our React application using this method:

Considering we have the components **App**, **About**, and **Contact** in our application:



Hope you have no doubts about this ReactJS interview questions article, in case of any difficulty, please leave your problems in the section below.

### **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.

**You can download a PDF version of React Interview Questions.**

### **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 ele

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:

* ****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 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.

****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** | ❌ | ✔️ | ✔️ |
| **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 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 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 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) {

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 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. 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).****

### **22. 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.

### **23. 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>

);

}

}

### **24. 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.

### **25. 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.

****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.

### **26. 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({

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),

});

}

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.

### **27. 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.

### **28. 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.

### **29. 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.

### **30. 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.

### **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. 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.

### **33. 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.

### **34. 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.

### **35. 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.

### **36. 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.

### **37. 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.jsand 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](http://localhost:4000/" \t "https://www.interviewbit.com/react-interview-questions/_blank). 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.

### **38. 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

}

### **39. 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>

}

}

### **40. 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:

### **41. 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>

}

}

}

### 1) What is React?

React is a declarative, efficient, flexible open source front-end JavaScript library developed by Facebook in 2011. It follows the component-based approach for building reusable UI components, especially for single page application. It is used for developing interactive view layer of web and mobile apps. It was created by Jordan Walke, a software engineer at Facebook. It was initially deployed on Facebook's News Feed section in 2011 and later used in its products like WhatsApp & Instagram.

For More Information, **[Click here](https://www.javatpoint.com/react-introduction)**.

### 2) What are the features of React?

React framework gaining quick popularity as the best framework among web developers. The main features of React are:

Play Videox[](https://campaign.adpushup.com/get-started/?utm_source=banner%26utm_campaign=growth_hack)

* JSX
* Components
* One-way Data Binding
* Virtual DOM
* Simplicity
* Performance

For More Information, **[Click here](https://www.javatpoint.com/react-features)**.

### 3) What are the most crucial advantages of using React?

Following is a list of the most crucial advantages of using React:

****React is easy to learn and use****

React comes with good availability of documentation, tutorials, and training resources. It is easy for any developer to switch from JavaScript background to React and easily understand and start creating web apps using React. Anyone with little knowledge of JavaScript can start building web applications using React.

****React follows the MVC architecture.****

React is the V (view part) in the MVC (Model-View-Controller) architecture model and is referred to as "one of the JavaScript frameworks." It is not fully featured but has many advantages of the open-source JavaScript User Interface (UI) library, which helps execute the task in a better manner.

****React uses Virtual DOM to improve efficiency.****

React uses virtual DOM to render the view. The 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. That's why React provides great efficiency.

****Creating dynamic web applications is easy.****

In React, creating a dynamic web application is much easier. It requires less coding and gives more functionality. It uses JSX (JavaScript Extension), which is a particular syntax letting HTML quotes and HTML tag syntax to render particular subcomponents.

****React is SEO-friendly.****

React facilitates a developer to develop an engaging user interface that can be easily navigated in various search engines. It also allows server-side rendering, which is also helpful to boost the SEO of your app.

****React allows reusable components.****

React web applications are made up of multiple components where each component has its logic and controls. These components provide a small, reusable piece of HTML code as an output that can be reused wherever you need them. The code reusability helps developers to make their apps easier to develop and maintain. It also makes the nesting of the components easy and allows developers to build complex applications of simple building blocks. The reuse of components also increases the pace of development.

****Support of handy tools****

React provides a lot of handy tools that can make the task of the developers understandable and easier. Use these tools in Chrome and Firefox dev extension, allowing us to inspect the React component hierarchies in the virtual DOM. It also allows us to select the particular components and examine and edit their current props and state.

****React has a rich set of libraries.****

React has a huge ecosystem of libraries and provides you the freedom to choose the tools, libraries, and architecture for developing the best application based on your requirement.

****Scope for testing the codes****

React web applications are easy to test. These applications provide a scope where the developer can test and debug their codes with the help of native tools.

For More Information, **[Click here](https://www.javatpoint.com/pros-and-cons-of-react)**.

### 4) What are the biggest limitations of React?

Following is the list of the biggest limitations of React:

* React is just a library. It is not a complete framework.
* It has a huge library which takes time to understand.
* It may be difficult for the new programmers to understand and code.
* React uses inline templating and JSX, which may be difficult and act as a barrier. It also makes the coding complex.

### 5) What is JSX?

JSX stands for JavaScript XML. It is a React extension which allows writing JavaScript code that looks similar to HTML. It makes HTML file easy to understand. The JSX file makes the React application robust and boosts its performance. JSX provides you to write XML-like syntax in the same file where you write JavaScript code, and then preprocessor (i.e., transpilers like Babel) transform these expressions into actual JavaScript code. Just like XML/HTML, JSX tags have a tag name, attributes, and children.

****Example****

1. **class** App **extends** React.Component {
2. render() {
3. **return**(
4. <div>
5. <h1>Hello JavaTpoint</h1>
6. </div>
7. )
8. }
9. }

In the above example, text inside <h1> tag return as JavaScript function to the render function. After compilation, the JSX expression becomes a normal JavaScript function, as shown below.

1. React.createElement("h1", **null**, "Hello JavaTpoint");

For More Information, **[Click here](https://www.javatpoint.com/react-jsx)**.

### 6) Why can't browsers read JSX?

Browsers cannot read JSX directly because they can only understand JavaScript objects, and JSX is not a regular JavaScript object. Thus, we need to transform the JSX file into a JavaScript object using transpilers like Babel and then pass it to the browser.

### 7) Why we use JSX?

* It is faster than regular JavaScript because it performs optimization while translating the code to JavaScript.
* Instead of separating technologies by putting markup and logic in separate files, React uses components that contain both.
* t is type-safe, and most of the errors can be found at compilation time.
* It makes easier to create templates.

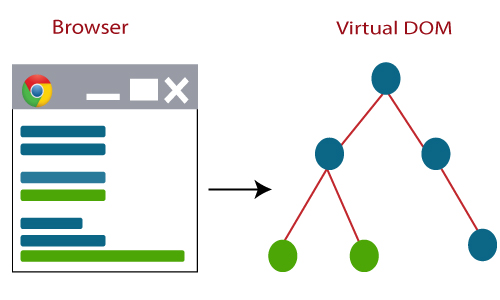
### 8) What do you understand by Virtual DOM?

A Virtual DOM is a lightweight JavaScript object which is an in-memory representation of real DOM. It is an intermediary step between the render function being called and the displaying of elements on the screen. It is similar to a node tree which lists the elements, their attributes, and content as objects and their properties. The render function creates a node tree of the React components and then updates this node tree in response to the mutations in the data model caused by various actions done by the user or by the system.

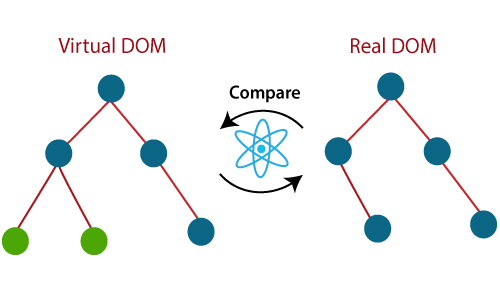
### 9) Explain the working of Virtual DOM.

Virtual DOM works in three steps:

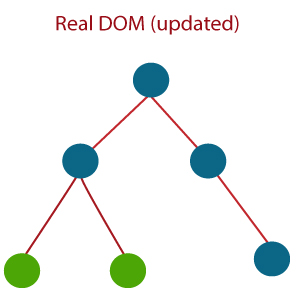
1. Whenever any data changes in the React App, the entire UI is re-rendered in Virtual DOM representation.



2. Now, the difference between the previous DOM representation and the new DOM is calculated.



3. Once the calculations are completed, the real DOM updated with only those things which are changed.



### 10) How is React different from Angular?

The React is different from Angular in the following ways.

|  |  |  |
| --- | --- | --- |
|  | **Angular** | **React** |
| ****Author**** | Google | Facebook Community |
| ****Developer**** | Misko Hevery | Jordan Walke |
| ****Initial Release**** | October 2010 | March 2013 |
| ****Language**** | JavaScript, HTML | JSX |
| ****Type**** | Open Source MVC Framework | Open Source JS Framework |
| ****Rendering**** | Client-Side | Server-Side |
| ****Data-Binding**** | Bi-directional | Uni-directional |
| ****DOM**** | Regular DOM | Virtual DOM |
| ****Testing**** | Unit and Integration Testing | Unit Testing |
| ****App Architecture**** | MVC | Flux |
| ****Performance**** | Slow | Fast, due to virtual DOM. |

For More Information, **[Click here](https://www.javatpoint.com/reactjs-vs-angularjs)**.

### 11) How React's ES6 syntax is different from ES5 syntax?

The React's ES6 syntax has changed from ES5 syntax in the following aspects.

****require vs. Import****

1. // ES5
2. var React = require('react');
4. // ES6
5. **import** React from 'react';

****exports vs. export****

1. // ES5
2. module.exports = Component;
4. // ES6
5. export **default** Component;

****component and function****

1. // ES5
2. var MyComponent = React.createClass({
3. render: function() {
4. **return**(
5. <h3>Hello JavaTpoint</h3>
6. );
7. }
8. });
10. // ES6
11. **class** MyComponent **extends** React.Component {
12. render() {
13. **return**(
14. <h3>Hello Javatpoint</h3>
15. );
16. }
17. }

****props****

1. // ES5
2. var App = React.createClass({
3. propTypes: { name: React.PropTypes.string },
4. render: function() {
5. **return**(
6. <h3>Hello, {**this**.props.name}!</h3>
7. );
8. }
9. });
11. // ES6
12. **class** App **extends** React.Component {
13. render() {
14. **return**(
15. <h3>Hello, {**this**.props.name}!</h3>
16. );
17. }
18. }

****state****

1. var App = React.createClass({
2. getInitialState: function() {
3. **return** { name: 'world' };
4. },
5. render: function() {
6. **return**(
7. <h3>Hello, {**this**.state.name}!</h3>
8. );
9. }
10. });
12. // ES6
13. **class** App **extends** React.Component {
14. constructor() {
15. **super**();
16. **this**.state = { name: 'world' };
17. }
18. render() {
19. **return**(
20. <h3>Hello, {**this**.state.name}!</h3>
21. );
22. }
23. }

### 12) What is the difference between ReactJS and React Native?

The main differences between ReactJS and React Native are given below.

|  |  |  |
| --- | --- | --- |
| **SN** | **ReactJS** | **React Native** |
| ****1.**** | Initial release in 2013. | Initial release in 2015. |
| ****2.**** | It is used for developing web applications. | It is used for developing mobile applications. |
| ****3.**** | It can be executed on all platforms. | It is not platform independent. It takes more effort to be executed on all platforms. |
| ****4.**** | It uses a JavaScript library and CSS for animations. | It comes with built-in animation libraries. |
| ****5.**** | It uses React-router for navigating web pages. | It has built-in Navigator library for navigating mobile applications. |
| ****6.**** | It uses HTML tags. | It does not use HTML tags. |
| ****7.**** | In this, the Virtual DOM renders the browser code. | In this, Native uses its API to render code for mobile applications. |

For More Information, **[Click here](https://www.javatpoint.com/reactjs-vs-reactnative)**.

### 13) What is the difference between Real DOM and Virtual DOM?

The following table specifies the key differences between the Real DOM and Virtual DOM:

The real DOM creates a new DOM if the element updates.

|  |  |
| --- | --- |
| **Real DOM** | **Virtual DOM** |
| The real DOM updates slower. | The virtual DOM updates faster. |
| The real DOM can directly update HTML. | The virtual DOM cannot directly update HTML. |
| The virtual DOM updates the JSX if the element updates. |  |
| In real DOM, DOM manipulation is very expensive. | In virtual DOM, DOM manipulation is very easy. |
| There is a lot of memory wastage in The real DOM. | There is no memory wastage in the virtual DOM. |

## React Component Interview Questions

### 14) What do you understand from "In React, everything is a component."

In React, components are the building blocks of React applications. These components divide the entire React application's UI into small, independent, and reusable pieces of code. React renders each of these components independently without affecting the rest of the application UI. Hence, we can say that, in React, everything is a component.

### 15) Explain the purpose of render() in React.

It is mandatory for each React component to have a render() function. Render function is used to return the HTML which you want to display in a component. If you need to rendered more than one HTML element, you need to grouped together inside single enclosing tag (parent tag) such as <div>, <form>, <group> etc. This function returns the same result each time it is invoked.

****Example:**** If you need to display a heading, you can do this as below.

1. **import** React from 'react'
3. **class** App **extends** React.Component {
4. render (){
5. **return** (
6. <h1>Hello World</h1>
7. )
8. }
9. }
10. export **default** App

****Points to Note:****

* Each render() function contains a return statement.
* The return statement can have only one parent HTML tag.

### 16) How can you embed two or more components into one?

You can embed two or more components into the following way:

1. **import** React from 'react'
3. **class** App **extends** React.Component {
4. render (){
5. **return** (
6. <h1>Hello World</h1>
7. )
8. }
9. }
11. **class** Example **extends** React.Component {
12. render (){
13. **return** (
14. <h1>Hello JavaTpoint</h1>
15. )
16. }
17. }
18. export **default** App

### 17) What is Props?

Props stand for "Properties" in React. They are read-only inputs to components. Props are an object which stores the value of attributes of a tag and work similar to the HTML attributes. It gives a way to pass data from the parent to the child components throughout the application.

It is similar to function arguments and passed to the component in the same way as arguments passed in a function.

Props are immutable so we cannot modify the props from inside the component. Inside the components, we can add attributes called props. These attributes are available in the component as this.props and can be used to render dynamic data in our render method.

For More Information, **[Click here](https://www.javatpoint.com/react-props)**.

### 18) What is a State in React?

The State is an updatable structure which holds the data and information about the component. It may be changed over the lifetime of the component in response to user action or system event. It is the heart of the react component which determines the behavior of the component and how it will render. It must be kept as simple as possible.

Let's create a "User" component with "message state."

1. **import** React from 'react'
3. **class** User **extends** React.Component {
4. constructor(props) {
5. **super**(props)
7. **this**.state = {
8. message: 'Welcome to JavaTpoint'
9. }
10. }
12. render() {
13. **return** (
14. <div>
15. <h1>{**this**.state.message}</h1>
16. </div>
17. )
18. }
19. }
20. export **default** User

For More Information, **[Click here](https://www.javatpoint.com/react-state)**.

### 19) Differentiate between States and Props.

The major differences between States and Props are given below.

|  |  |  |
| --- | --- | --- |
| **SN** | **Props** | **State** |
| ****1.**** | Props are read-only. | State changes can be asynchronous. |
| ****2.**** | Props are immutable. | State is mutable. |
| ****3.**** | Props allow you to pass data from one component to other components as an argument. | State holds information about the components. |
| ****4.**** | Props can be accessed by the child component. | State cannot be accessed by child components. |
| ****5.**** | Props are used to communicate between components. | States can be used for rendering dynamic changes with the component. |
| ****6.**** | The stateless component can have Props. | The stateless components cannot have State. |
| ****7.**** | Props make components reusable. | The State cannot make components reusable. |
| ****8.**** | Props are external and controlled by whatever renders the component. | The State is internal and controlled by the component itself. |

For More Information, **[Click here](https://www.javatpoint.com/react-state-vs-props)**.

### 20) How can you update the State of a component?

We can update the State of a component using this.setState() method. This method does not always replace the State immediately. Instead, it only adds changes to the original State. It is a primary method which is used to update the user interface(UI) in response to event handlers and server responses.

****Example****

1. **import** React, { Component } from 'react';
2. **import** PropTypes from 'prop-types';
4. **class** App **extends** React.Component {
5. constructor() {
6. **super**();
7. **this**.state = {
8. msg: "Welcome to JavaTpoint"
9. };
10. **this**.updateSetState = **this**.updateSetState.bind(**this**);
11. }
12. updateSetState() {
13. **this**.setState({
14. msg:"Its a best ReactJS tutorial"
15. });
16. }
17. render() {
18. **return** (
19. <div>
20. <h1>{**this**.state.msg}</h1>
21. <button onClick = {**this**.updateSetState}>SET STATE</button>
22. </div>
23. );
24. }
25. }
26. export **default** App;

For More Information, **[Click here](https://www.javatpoint.com/react-component-api)**.

### 21) Differentiate between stateless and stateful components.

The difference between stateless and stateful components are:

|  |  |  |
| --- | --- | --- |
| **SN** | **Stateless Component** | **Stateful Component** |
| ****1.**** | The stateless components do not hold or manage state. | The stateful components can hold or manage state. |
| ****2.**** | It does not contain the knowledge of past, current, and possible future state changes. | It can contain the knowledge of past, current, and possible future changes in state. |
| ****3.**** | It is also known as a functional component. | It is also known as a class component. |
| ****4.**** | It is simple and easy to understand. | It is complex as compared to the stateless component. |
| ****5.**** | It does not work with any lifecycle method of React. | It can work with all lifecycle method of React. |
| ****6.**** | The stateless components cannot be reused. | The stateful components can be reused. |

### 22) What is arrow function in React? How is it used?

The Arrow function is the new feature of the ES6 standard. If you need to use arrow functions, it is not necessary to bind any event to 'this.' Here, the scope of 'this' is global and not limited to any calling function. So If you are using Arrow Function, there is no need to bind 'this' inside the constructor. It is also called 'fat arrow '(=>) functions.

1. //General way
2. render() {
3. **return**(
4. <MyInput onChange={**this**.handleChange.bind(**this**) } />
5. );
6. }
7. //With Arrow Function
8. render() {
9. **return**(
10. <MyInput onChange={ (e) => **this**.handleOnChange(e) } />
11. );
12. }

### 23) What is an event in React?

An event is an action which triggers as a result of the user action or system generated event like a mouse click, loading of a web page, pressing a key, window resizes, etc. In React, the event handling system is very similar to handling events in DOM elements. The React event handling system is known as Synthetic Event, which is a cross-browser wrapper of the browser's native event.

Handling events with React have some syntactical differences, which are:

* React events are named as camelCase instead of lowercase.
* With JSX, a function is passed as the event handler instead of a string.

For More Information, **[Click here](https://www.javatpoint.com/react-events)**.

### 24) How do you create an event in React?

We can create an event as follows.

1. **class** Display **extends** React.Component({
2. show(msgEvent) {
3. // code
4. },
5. render() {
6. // Here, we render the div with an onClick prop
7. **return** (
8. <div onClick={**this**.show}>Click Me</div>
9. );
10. }
11. });

****Example****

1. **import** React, { Component } from 'react';
2. **class** App **extends** React.Component {
3. constructor(props) {
4. **super**(props);
5. **this**.state = {
6. companyName: ''
7. };
8. }
9. changeText(event) {
10. **this**.setState({
11. companyName: event.target.value
12. });
13. }
14. render() {
15. **return** (
16. <div>
17. <h2>Simple Event Example</h2>
18. <label htmlFor="name">Enter company name: </label>
19. <input type="text" id="companyName" onChange={**this**.changeText.bind(**this**)}/>
20. <h4>You entered: { **this**.state.companyName }</h4>
21. </div>
22. );
23. }
24. }
25. export **default** App;

For More Information, **[Click here](https://www.javatpoint.com/react-events)**.

### 25) What are synthetic events in React?

A synthetic event is an object which acts as a cross-browser wrapper around the browser's native event. It combines the behavior of different browser's native event into one API, including stopPropagation() and preventDefault().

In the given example, e is a Synthetic event.

1. function ActionLink() {
2. function handleClick(e) {
3. e.preventDefault();
4. console.log('You had clicked a Link.');
5. }
6. **return** (
7. <a href="#" onClick={handleClick}>
8. Click\_Me
9. </a>
10. );
11. }

### 26) what is the difference between controlled and uncontrolled components?

The difference between controlled and uncontrolled components are:

|  |  |  |
| --- | --- | --- |
| **SN** | **Controlled** | **Uncontrolled** |
| ****1.**** | It does not maintain its internal state. | It maintains its internal states. |
| ****2.**** | Here, data is controlled by the parent component. | Here, data is controlled by the DOM itself. |
| ****3.**** | It accepts its current value as a prop. | It uses a ref for their current values. |
| ****4.**** | It allows validation control. | It does not allow validation control. |
| ****5.**** | It has better control over the form elements and data. | It has limited control over the form elements and data. |

For More Information, **[Click here](https://www.javatpoint.com/react-controlled-vs-uncontrolled-component)**.

### 27) Explain the Lists in React.

Lists are used to display data in an ordered format. In React, Lists can be created in a similar way as we create it in JavaScript. We can traverse the elements of the list using the map() function.

****Example****

1. **import** React from 'react';
2. **import** ReactDOM from 'react-dom';
4. function NameList(props) {
5. **const** myLists = props.myLists;
6. **const** listItems = myLists.map((myList) =>
7. <li>{myList}</li>
8. );
9. **return** (
10. <div>
11. <h2>Rendering Lists inside component</h2>
12. <ul>{listItems}</ul>
13. </div>
14. );
15. }
16. **const** myLists = ['Peter', 'Sachin', 'Kevin', 'Dhoni', 'Alisa'];
17. ReactDOM.render(
18. <NameList myLists={myLists} />,
19. document.getElementById('app')
20. );
21. export **default** App;

For More Information, **[Click here](https://www.javatpoint.com/react-lists)**.

### 28) What is the significance of keys in React?

A key is a unique identifier. In React, it is used to identify which items have changed, updated, or deleted from the Lists. It is useful when we dynamically created components or when the users alter the lists. It also helps to determine which components in a collection needs to be re-rendered instead of re-rendering the entire set of components every time. It increases application performance.

For More Information, **[Click here](https://www.javatpoint.com/react-keys)**.

### 29) How are forms created in React?

Forms allow the users to interact with the application as well as gather information from the users. Forms can perform many tasks such as user authentication, adding user, searching, filtering, etc. A form can contain text fields, buttons, checkbox, radio button, etc.

React offers a stateful, reactive approach to build a form. The forms in React are similar to HTML forms. But in React, the state property of the component is only updated via setState(), and a JavaScript function handles their submission. This function has full access to the data which is entered by the user into a form.

1. **import** React, { Component } from 'react';
3. **class** App **extends** React.Component {
4. constructor(props) {
5. **super**(props);
6. **this**.state = {value: ''};
7. **this**.handleChange = **this**.handleChange.bind(**this**);
8. **this**.handleSubmit = **this**.handleSubmit.bind(**this**);
9. }
10. handleChange(event) {
11. **this**.setState({value: event.target.value});
12. }
13. handleSubmit(event) {
14. alert('You have submitted the input successfully: ' + **this**.state.value);
15. event.preventDefault();
16. }
17. render() {
18. **return** (
19. <form onSubmit={**this**.handleSubmit}>
20. <h1>Controlled Form Example</h1>
21. <label>
22. Name:
23. <input type="text" value={**this**.state.value} onChange={**this**.handleChange} />
24. </label>
25. <input type="submit" value="Submit" />
26. </form>
27. );
28. }
29. }
30. export **default** App;

For More Information, **[Click here](https://www.javatpoint.com/react-forms)**.

### 30) What are the different phases of React component's lifecycle?

The different phases of React component's lifecycle are:

****Initial Phase:**** It is the birth phase of the React lifecycle when the component starts its journey on a way to the DOM. In this phase, a component contains the default Props and initial State. These default properties are done in the constructor of a component.

****Mounting Phase:**** In this phase, the instance of a component is created and added into the DOM.

****Updating Phase:**** It is the next phase of the React lifecycle. In this phase, we get new Props and change State. This phase can potentially update and re-render only when a prop or state change occurs. The main aim of this phase is to ensure that the component is displaying the latest version of itself. This phase repeats again and again.

****Unmounting Phase:**** It is the final phase of the React lifecycle, where the component instance is destroyed and unmounted(removed) from the DOM.

For More Information, **[Click here](https://www.javatpoint.com/react-component-life-cycle)**.

### 31) Explain the lifecycle methods of React components in detail.

The important React lifecycle methods are:

* ****getInitialState():**** It is used to specify the default value of this.state. It is executed before the creation of the component.
* ****componentWillMount():**** It is executed before a component gets rendered into the DOM.
* ****componentDidMount():**** It is executed when the component gets rendered and placed on the DOM. Now, you can do any DOM querying operations.
* ****componentWillReceiveProps():**** It is invoked when a component receives new props from the parent class and before another render is called. If you want to update the State in response to prop changes, you should compare this.props and nextProps to perform State transition by using this.setState() method.
* ****shouldComponentUpdate():**** It is invoked when a component decides any changes/updation to the DOM and returns true or false value based on certain conditions. If this method returns true, the component will update. Otherwise, the component will skip the updating.
* ****componentWillUpdate():**** It is invoked before rendering takes place in the DOM. Here, you can't change the component State by invoking this.setState() method. It will not be called, if shouldComponentUpdate() returns false.
* ****componentDidUpdate():**** It is invoked immediately after rendering takes place. In this method, you can put any code inside this which you want to execute once the updating occurs.
* ****componentWillUnmount():**** It is invoked immediately before a component is destroyed and unmounted permanently. It is used to clear up the memory spaces such as invalidating timers, event listener, canceling network requests, or cleaning up DOM elements. If a component instance is unmounted, you cannot mount it again.

For More Information, **[Click here](https://www.javatpoint.com/react-component-life-cycle)**.

### 32) What are Pure Components?

Pure components introduced in React 15.3 version. The React.Component and React.PureComponent differ in the shouldComponentUpdate() React lifecycle method. This method decides the re-rendering of the component by returning a boolean value (true or false). In React.Component, shouldComponentUpdate() method returns true by default. But in React.PureComponent, it compares the changes in state or props to re-render the component. The pure component enhances the simplicity of the code and performance of the application.

### 33) What are Higher Order Components(HOC)?

In React, Higher Order Component is an advanced technique for reusing component logic. It is a function that takes a component and returns a new component. In other words, it is a function which accepts another function as an argument. According to the official website, it is not the feature(part) in React API, but a pattern that emerges from React's compositional nature.

For More Information, **[Click here](https://www.javatpoint.com/react-higher-order-components)**.

### 34) What can you do with HOC?

You can do many tasks with HOC, some of them are given below:

* Code Reusability
* Props manipulation
* State manipulation
* Render highjacking

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

The main differences between Elements and Components are:

|  |  |  |
| --- | --- | --- |
| **SN** | **Element** | **Component** |
| ****1.**** | An element is a plain JavaScript object which describes the component state and DOM node, and its desired properties. | A component is the core building block of React application. It is a class or function which accepts an input and returns a React element. |
| ****2.**** | It only holds information about the component type, its properties, and any child elements inside it. | It can contain state and props and has access to the React lifecycle methods. |
| ****3.**** | It is immutable. | It is mutable. |
| ****4.**** | We cannot apply any methods on elements. | We can apply methods on components. |
| ****5.**** | ****Example:**** const element = React.createElement( 'div', {id: 'login-btn'}, 'Login' ) | ****Example:**** function Button ({ onLogin }) { return React.createElement( 'div', {id: 'login-btn', onClick: onLogin}, 'Login' ) } |

### 36) How to write comments in React?

In React, we can write comments as we write comments in JavaScript. It can be in two ways:

****1. Single Line Comments:**** We can write comments as /\* Block Comments \*/ with curly braces:

1. {/\* Single Line comment \*/}

****2. Multiline Comments:**** If we want to comment more that one line, we can do this as

1. { /\*
2. Multi
3. line
4. comment
5. \*/ }

### 37) Why is it necessary to start component names with a capital letter?

In React, it is necessary to start component names with a capital letter. If we start the component name with lower case, it will throw an error as an unrecognized tag. It is because, in JSX, lower case tag names are considered as HTML tags.

### 38) What are fragments?

In was introduced in React 16.2 version. In React, Fragments are used for components to return multiple elements. It allows you to group a list of multiple children without adding an extra node to the DOM.

****Example****

1. render() {
2. **return** (
3. <React.Fragment>
4. <ChildA />
5. <ChildB />
6. <ChildC />
7. </React.Fragment>
8. )
9. }

There is also a shorthand syntax exists for declaring Fragments, but it's not supported in many tools:

1. render() {
2. **return** (
3. <>
4. <ChildA />
5. <ChildB />
6. <ChildC />
7. </>
8. )
9. }

For More Information, **[Click here](https://www.javatpoint.com/react-fragments)**.

### 39) Why are fragments better than container divs?

* Fragments are faster and consume less memory because it did not create an extra DOM node.
* Some CSS styling like CSS Grid and Flexbox have a special parent-child relationship and add <div> tags in the middle, which makes it hard to keep the desired layout.
* The DOM Inspector is less cluttered.

### 40) How to apply validation on props in React?

Props validation is a tool which helps the developers to avoid future bugs and problems. It makes your code more readable. React components used special property PropTypes that help you to catch bugs by validating data types of values passed through props, although it is not necessary to define components with propTypes.

We can apply validation on props using App.propTypes in React component. When some of the props are passed with an invalid type, you will get the warnings on JavaScript console. After specifying the validation patterns, you need to set the App.defaultProps.

1. **class** App **extends** React.Component {
2. render() {}
3. }
4. Component.propTypes = { /\*Definition \*/};

For More Information, **[Click here](https://www.javatpoint.com/react-props-validation)**.

### 41) What is create-react-app?

Create React App is a tool introduced by Facebook to build React applications. It provides you to create single-page React applications. The create-react-app are preconfigured, which saves you from time-consuming setup and configuration like Webpack or Babel. You need to run a single command to start the React project, which is given below.

1. $ npx create-react-app my-app

This command includes everything which we need to build a React app. Some of them are given below:

* It includes React, JSX, ES6, and Flow syntax support.
* It includes Autoprefixed CSS, so you don't need -webkit- or other prefixes.
* It includes a fast, interactive unit test runner with built-in support for coverage reporting.
* It includes a live development server that warns about common mistakes.
* It includes a build script to bundle JS, CSS, and images for production, with hashes and source maps.

For More Information, **[Click here](https://www.javatpoint.com/react-installation)**.

### 42) How can you create a component in React?

There are two possible ways to create a component in React:

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

1. function Greeting({ message }) {
2. return **<h1>**{`Hello, ${message}`}**</h1>**
3. }

****Class Components:**** The class components method facilitates you to use ES6 class to define a component. The above function component can be written as:

1. class Greeting extends React.Component {
2. render() {
3. return **<h1>**{`Hello, ${this.props.message}`}**</h1>**
4. }
5. }

### 43) When do we prefer to use a class component over a function component?

If a component needs state or lifecycle methods, we should use the class component; otherwise, use the function component. However, after React 16.8, with the addition of Hooks, you could use state, lifecycle methods, and other features that were only available in the class component right in your function component.

### 44) Is it possible for a web browser to read JSX directly?

Web browsers can't read JSX directly. This is because the web browsers are built to read the regular JS objects only, and JSX is not a regular JavaScript object.

If you want a web browser to read a JSX file, you must transform the files into a regular JavaScript object. For this purpose, Babel is used.

### 45) What do you understand by the state in React?

In react, the state of a component is an object that holds some information that may change over the component's lifetime. It would be best to try to make your state as simple as possible and minimize the number of stateful components.

****Let's see how to create a user component with message state:****

1. class User extends React.Component {
2. constructor(props) {
3. super(props)
4. this.state = {
5. message: 'Welcome to React world'
6. }
7. }
8. render() {
9. return (
10. **<div>**
11. **<h1>**{this.state.message}**</h1>**
12. **</div>**
13. )
14. }
15. }

The state is very similar to props, but it is private and fully controlled by the component. i.e., It is not accessible to any other component till the owner component decides to pass it.

### 46) What are the main changes that appear in React's ES6 syntax compared to ES5 syntax?/How different is React's ES6 syntax compared to ES5?

Following are the most visible syntax we can see while comparing ES6 and ES5:

### require vs import

****Syntax in ES5:****

1. var React = require('react');

****Syntax in ES6:****

1. import React from 'react';

### export vs exports

****Syntax in ES5:****

1. module.exports = Component;

****Syntax in ES6:****

1. export default Component;

### component and function

****Syntax in ES5:****

1. var MyComponent = React.createClass({
2. render: function() {
3. return
4. **<h3>**Hello JavaTpoint!**</h3>**
5. ;
6. }
7. });

****Syntax in ES6:****

1. class MyComponent extends React.Component {
2. render() {
3. return
4. **<h3>**Hello JavaTpoint!**</h3>**
5. ;
6. }
7. }

### props

****Syntax in ES5:****

1. var App = React.createClass({
2. propTypes: { name: React.PropTypes.string },
3. render: function() {
4. return
5. **<h3>**Hello, {this.props.name}!**</h3>**
6. ;
7. }
8. });

****Syntax in ES6:****

1. class App extends React.Component {
2. render() {
3. return
4. **<h3>**Hello, {this.props.name}!**</h3>**
5. ;
6. }
7. }

### state

****Syntax in ES5:****

1. var App = React.createClass({
2. getInitialState: function() {
3. return { name: 'world' };
4. },
5. render: function() {
6. return
7. **<h3>**Hello, {this.state.name}!**</h3>**
8. ;
9. }
10. });

****Syntax in ES6:****

1. class App extends React.Component {
2. constructor() {
3. super();
4. this.state = { name: 'world' };
5. }
6. render() {
7. return
8. **<h3>**Hello, {this.state.name}!**</h3>**
9. ;
10. }
11. }

### 47) What do you understand by props in React?

In React, the props are inputs to components. They are single values or objects containing a set of values passed to components on creation using a naming convention similar to HTML-tag attributes. They are data passed down from a parent component to a child component.

****The main purpose of props in React is to provide the following component functionality:****

* Pass custom data to your component.
* Trigger state changes.
* Use via this.props.reactProp inside component's render() method.

For example, let us create an element with reactProp property:

1. **<Element** reactProp={'1'} **/>**

This reactProp name becomes a property attached to React's native props object, which already exists on all React library components.

1. props.reactProp

### 48) What do you understand by refs in React?

Refs is the shorthand used for references in React. It is an attribute which helps to store a reference to particular DOM nodes or React elements. It provides a way to access React DOM nodes or React elements and how to interact with it. It is used when we want to change the value of a child component, without making the use of props.

For More Information, **[Click here](https://www.javatpoint.com/react-refs)**.

### 49) How to create refs?

Refs can be created by using React.createRef() and attached to React elements via the ref attribute. It is commonly assigned to an instance property when a component is created, and then can be referenced throughout the component.

1. **class** MyComponent **extends** React.Component {
2. constructor(props) {
3. **super**(props);
4. **this**.callRef = React.createRef();
5. }
6. render() {
7. **return** <div ref={**this**.callRef} />;
8. }
9. }

### 50) What are Forward Refs?

Ref forwarding is a feature which is used for passing a ref through a component to one of its child components. It can be performed by making use of the React.forwardRef() method. It is particularly useful with higher-order components and specially used in reusable component libraries.

****Example****

1. **import** React, { Component } from 'react';
2. **import** { render } from 'react-dom';
4. **const** TextInput = React.forwardRef((props, ref) => (
5. <input type="text" placeholder="Hello World" ref={ref} />
6. ));
8. **const** inputRef = React.createRef();
10. **class** CustomTextInput **extends** React.Component {
11. handleSubmit = e => {
12. e.preventDefault();
13. console.log(inputRef.current.value);
14. };
15. render() {
16. **return** (
17. <div>
18. <form onSubmit={e => **this**.handleSubmit(e)}>
19. <TextInput ref={inputRef} />
20. <button>Submit</button>
21. </form>
22. </div>
23. );
24. }
25. }
26. export **default** App;

For More Information, **[Click here](https://www.javatpoint.com/react-refs)**.

### 51) Which is the preferred option callback refs or findDOMNode()?

The preferred option is to use callback refs over findDOMNode() API. Because callback refs give better control when the refs are set and unset whereas findDOMNode() prevents certain improvements in React in the future.

1. **class** MyComponent **extends** Component {
2. componentDidMount() {
3. findDOMNode(**this**).scrollIntoView()
4. }
5. render() {
6. **return** <div />
7. }
8. }

The recommended approach is:

1. **class** MyComponent **extends** Component {
2. componentDidMount() {
3. **this**.node.scrollIntoView()
4. }
5. render() {
6. **return** <div ref={node => **this**.node = node} />
7. }
8. }
9. **class** MyComponent **extends** Component {
10. componentDidMount() {
11. **this**.node.scrollIntoView()
12. }
13. render() {
14. **return** <div ref={node => **this**.node = node} />
15. }
16. }

### 52) What is the use of Refs?

The Ref in React is used in the following cases:

* It is used to return a reference to the element.
* It is used when we need DOM measurements such as managing focus, text selection, or media playback.
* It is used in triggering imperative animations.
* It is used when integrating with third-party DOM libraries.
* It can also use as in callbacks.

For More Information, **[Click here](https://www.javatpoint.com/react-refs)**.

### 53) What is React Router?

React Router is a standard routing library system built on top of the React. It is used to create Routing in the React application using React Router Package. It helps you to define multiple routes in the app. It provides the synchronous URL on the browser with data that will be displayed on the web page. It maintains the standard structure and behavior of the application and mainly used for developing single page web applications.

For More Information, **[Click here](https://www.javatpoint.com/react-router)**.

### 54) Why do we need a Router in React?

React Router plays an important role to display multiple views in a single page application. It is used to define multiple routes in the app. When a user types a specific URL into the browser, and if this URL path matches any 'route' inside the router file, the user will be redirected to that particular Route. So, we need to add a Router library to the React app, which allows creating multiple routes with each leading to us a unique view.

1. <**switch**>
2. <h1>React Router Example</h1>
3. <Route path="/" component={Home} />
4. <Route path="/about" component={About} />
5. <Route path="/contact" component={Contact} />
6. </**switch**>

### 55) List down the advantages of React Router.

The important advantages of React Router are given below:

* In this, it is not necessary to set the browser history manually.
* Link uses to navigate the internal links in the application. It is similar to the anchor tag.
* It uses Switch feature for rendering.
* The Router needs only a Single Child element.
* In this, every component is specified in <Route>.
* The packages are split into three packages, which are Web, Native, and Core. It supports the compact size of the React application.

### 56) How is React Router different from Conventional Routing?

The difference between React Routing and Conventional Routing are:

|  |  |  |
| --- | --- | --- |
| **SN** | **Conventional Routing** | **React Routing** |
| ****1.**** | In Conventional Routing, each view contains a new file. | In React Routing, there is only a single HTML page involved. |
| ****2.**** | The HTTP request is sent to a server to receive the corresponding HTML page. | Only the History attribute <BrowserRouter> is changed. |
| ****3.**** | In this, the user navigates across different pages for each view. | In this, the user is thinking he is navigating across different pages, but its an illusion only. |

### 57) Why you get "Router may have only one child element" warning?

It is because you have not to wrap your Route's in a <Switch> block or <div> block which renders a route exclusively.

****Example****

1. render((
2. <Router>
3. <Route {/\* ... \*/} />
4. <Route {/\* ... \*/} />
5. </Router>
6. )

should be

1. render(
2. <Router>
3. <Switch>
4. <Route {/\* ... \*/} />
5. <Route {/\* ... \*/} />
6. </Switch>
7. </Router>
8. )

### 58) Why switch keyword used in React Router v4?

The 'switch' keyword is used to display only a single Route to rendered amongst the several defined Routes. The <Switch> component is used to render components only when the path will be matched. Otherwise, it returns to the not found component.

### 59) How to use styles in React?

We can use style attribute for styling in React applications, which adds dynamically-computed styles at render time. It accepts a JavaScript object in camelCased properties rather than a CSS string. The style attribute is consistent with accessing the properties on DOM nodes in JavaScript.

****Example****

1. **const** divStyle = {
2. color: 'blue',
3. backgroundImage: 'url(' + imgUrl + ')'
4. };
6. function HelloWorldComponent() {
7. **return** <div style={divStyle}>Hello World!</div>
8. }

### 60) How many ways can we style the React Component?

We can style React Component in mainly four ways, which are given below:

* Inline Styling
* CSS Stylesheet
* CSS Module
* Styled Components

For More Information, **[Click here](https://www.javatpoint.com/react-css)**.

### 61) Explain CSS Module styling in React.

CSS Module is a CSS file where all class names and animation names are scoped locally by default. It is available only for the component which imports it, and without your permission, it cannot be applied to any other Components. You can create CSS Module file with the .module.css extension.

For More Information, **[Click here](https://www.javatpoint.com/react-css)**.

### 62) What are Styled Components?

Styled-Components is a library for React. It is the successor of CSS Modules. It uses enhance CSS for styling React component systems in your application, which is written with a mixture of JavaScript and CSS. It is scoped to a single component and cannot leak to any other element in the page.

The styled-components provides:

* Automatic critical CSS
* No class name bugs
* Easier deletion of CSS
* Simple dynamic styling
* Painless maintenance

For More Information, **[Click here](https://www.javatpoint.com/react-css)**.

### 63) What are hooks in React?

Hooks are the new feature introduced in React 16.8 version that facilitates us to use state and other React features without writing a class.

****See the following example of useState hook:****

1. **import** { useState } from 'react';
2. function Example() {
3. // Declare a new state variable, which we'll call "count"
4. **const** [count, setCount] = useState(0);
5. **return** (
6. <div>
7. <p>You clicked {count} times</p>
8. <button onClick={() => setCount(count + 1)}>
9. Click on **this** button
10. </button>
11. </div>
12. );
13. }

### 64) What are the rules you should follow for the hooks in React?

We have to follow the following two rules to use hooks in React:

* You should call hooks only at the top level of your React functions and not inside the loops, conditions, or nested functions. This is used to ensure that hooks are called in the same order each time a component renders, and it also preserves the state of hooks between multiple useState and useEffect calls.
* You should call hooks from React functions only. Don't call hooks from regular JavaScript functions.

### 65) What are forms in React?

In React, forms are used to enable users to interact with web applications. Following is a list of the most common usage of forms in React:

* Forms facilitate users to interact with the application. By using forms, the users can communicate with the application and enter the required information whenever required.
* Forms contain certain elements, such as text fields, buttons, checkboxes, radio buttons, etc., that can make the application more interactive and beautiful.
* Forms are the best possible way to take inputs from the users.
* Forms are used for many different tasks such as user authentication, searching, filtering, indexing, etc.

### 66) What is an error boundary or error boundaries?

An error boundary is a concept introduced in version 16 of React. Error boundaries provide a way to find out the errors that occur in the render phase. Any component which uses one of the following lifecycle methods is considered an error boundary. Let's see the places where an error boundary can detect an error:

* Render phase
* Inside a lifecycle method
* Inside the constructor

****Let's see an example to understand it better:****

****Without using error boundaries:****

1. **class** CounterComponent **extends** React.Component{
2. constructor(props){
3. **super**(props);
4. **this**.state = {
5. counterValue: 0
6. }
7. **this**.incrementCounter = **this**.incrementCounter.bind(**this**);
8. }
9. incrementCounter(){
10. **this**.setState(prevState => counterValue = prevState+1);
11. }
12. render(){
13. **if**(**this**.state.counter === 2){
14. **throw** **new** Error('Crashed');
15. }
16. **return**(
17. <div>
18. <button onClick={**this**.incrementCounter}>Increment Value</button>
19. <p>Value of counter: {**this**.state.counterValue}</p>
20. </div>
21. )
22. }
23. }

In the above code, you can see that when the counterValue equals 2, it throws an error inside the render method. We know that any error inside the render method leads to unmounting of the component so, to display an error that occurs inside the render method, we use error boundaries. When we are not using the error boundary, we see a blank page instead of seeing an error.

With error boundaries:

We have specified earlier that error boundary is a component using one or both of the following methods:

* static getDerivedStateFromError
* componentDidCatch

****See the following code where we create an error boundary to handle errors in render phase:****

1. **class** ErrorBoundary **extends** React.Component {
2. constructor(props) {
3. **super**(props);
4. **this**.state = { hasError: **false** };
5. }
6. **static** getDerivedStateFromError(error) {
7. **return** { hasError: **true** };
8. }
9. componentDidCatch(error, errorInfo) {
10. logErrorToMyService(error, errorInfo);
11. }
12. render() {
13. **if** (**this**.state.hasError) {
14. **return** <h4>Something went wrong</h4>
15. }
16. **return** **this**.props.children;
17. }
18. }

You can see in the above code the getDerivedStateFromError function renders the fallback UI interface when the render method has an error.

The componentDidCatch logs the error information to an error tracking service.

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

1. **<ErrorBoundary>**
2. **<CounterComponent/>**
3. **</ErrorBoundary>**

### 67) In which cases do error boundaries not catch errors?

Following are some cases in which error boundaries don't catch errors:

* Error boundaries don't catch errors inside the event handlers.
* During the server-side rendering.
* In the case when errors are thrown in the error boundary code itself.
* Asynchronous code using setTimeout or requestAnimationFrame callbacks.

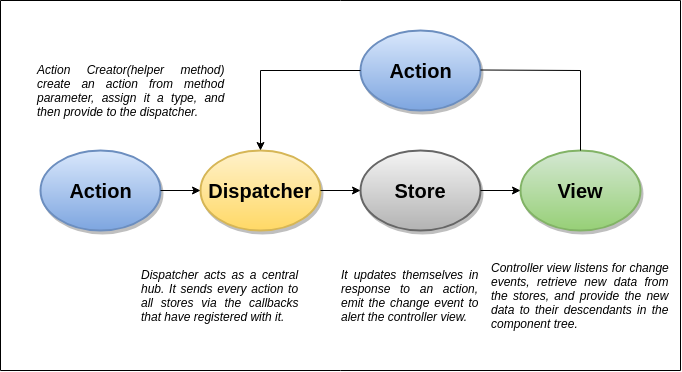
### 68) What were the major problems with MVC framework?

The major problems with the MVC framework are:

* DOM manipulation was very expensive.
* It makes the application slow and inefficient.
* There was a huge memory wastage.
* It makes the application debugging hard.

### 69) Explain the Flux concept.

Flux is an application architecture that Facebook uses internally for building the client-side web application with React. It is neither a library nor a framework. It is a kind of architecture that complements React as view and follows the concept of Unidirectional Data Flow model. It is useful when the project has dynamic data, and we need to keep the data updated in an effective manner.



For More Information, **[Click here](https://www.javatpoint.com/react-flux-concept)**.

### 70) What is Redux?

Redux is an open-source JavaScript library used to manage application state. React uses Redux for building the user interface. The Redux application is easy to test and can run in different environments showing consistent behavior. It was first introduced by Dan Abramov and Andrew Clark in 2015.

React Redux is the official React binding for Redux. It allows React components to read data from a Redux Store, and dispatch Actions to the Store to update data. Redux helps apps to scale by providing a sensible way to manage state through a unidirectional data flow model. React Redux is conceptually simple. It subscribes to the Redux store, checks to see if the data which your component wants have changed, and re-renders your component.

For More Information, **[Click here](https://www.javatpoint.com/react-redux)**.

### 71) What are the three principles that Redux follows?

The three principles that redux follows are:

1. ****Single source of truth:**** The State of your entire application is stored in an object/state tree inside a single Store. The single State tree makes it easier to keep changes over time. It also makes it easier to debug or inspect the application.
2. ****The State is read-only:**** There is only one way to change the State is to emit an action, an object describing what happened. This principle ensures that neither the views nor the network callbacks can write directly to the State.
3. ****Changes are made with pure functions:**** To specify how actions transform the state tree, you need to write reducers (pure functions). Pure functions take the previous State and Action as a parameter and return a new State.

### 72) List down the components of Redux.

The components of Redux are given below.

* ****STORE:**** A Store is a place where the entire State of your application lists. It is like a brain responsible for all moving parts in Redux.
* ****ACTION:**** It is an object which describes what happened.
* ****REDUCER:**** It determines how the State will change.

For More Information, **[Click here](https://www.javatpoint.com/react-redux)**.

### 73) Explain the role of Reducer.

Reducers read the payloads from the actions and then updates the Store via the State accordingly. It is a pure function which returns a new state from the initial State. It returns the previous State as it is if no work needs to be done.

### 74) What is the significance of Store in Redux?

A Store is an object which holds the application's State and provides methods to access the State, dispatch Actions and register listeners via subscribe(listener). The entire State tree of an application is saved in a single Store which makes the Redux simple and predictable. We can pass middleware to the Store which handles the processing of data as well as keep a log of various actions that change the Store's State. All the Actions return a new state via reducers.

### 75) How is Redux different from Flux?

The Redux is different from Flux in the following manner.

|  |  |  |
| --- | --- | --- |
| **SN** | **Redux** | **Flux** |
| ****1.**** | Redux is an open-source JavaScript library used to manage application State. | Flux is neither a library nor a framework. It is a kind of architecture that complements React as view and follows the concept of Unidirectional Data Flow model. |
| ****2.**** | Store's State is immutable. | Store's State is mutable. |
| ****3.**** | In this, Store and change logic are separate. | In this, the Store contains State and change logic. |
| ****4.**** | It has only a single Store. | It can have multiple Store. |
| ****5.**** | Redux does not have Dispatcher concept. | It has single Dispatcher, and all actions pass through that Dispatcher. |

### 76) What are the advantages of Redux?

The main advantages of React Redux are:

* React Redux is the official UI bindings for react Application. It is kept up-to-date with any API changes to ensure that your React components behave as expected.
* It encourages good 'React' architecture.
* It implements many performance optimizations internally, which allows to components re-render only when it actually needs.
* It makes the code maintenance easy.
* Redux's code written as functions which are small, pure, and isolated, which makes the code testable and independent.

### 77) How to access the Redux store outside a component?

You need to export the Store from the module where it created with createStore() method. Also, you need to assure that it will not pollute the global window space.

1. store = createStore(myReducer)
2. export **default** store

### ****1.  Differentiate between Real DOM and Virtual DOM.****

|  |  |
| --- | --- |
| Real DOM vs Virtual DOM ****Real DOM**** | ****Virtual  DOM**** |
| 1. It updates slow. | 1. It updates faster. |
| 2. Can directly update HTML. | 2. Can’t directly update HTML. |
| 3. Creates a new DOM if element updates. | 3. Updates the JSX if element updates. |
| 4. DOM manipulation is very expensive. | 4. DOM manipulation is very easy. |
| 5. Too much of memory wastage. | 5. No memory wastage. |

### ****2. What is React?****

* React is a front-end JavaScript library developed by Facebook in 2011.
* It follows the component based approach which helps in building reusable UI components.
* It is used for developing complex and interactive web and mobile UI.
* Even though it was open-sourced only in 2015, it has one of the largest communities supporting it.

### ****3. What are the features of React?****

Major features of React are listed below:

[](https://www.edureka.co/reactjs-redux-certification-training?utm_source=blogbanner&utm_campaign=curriculum" \t "https://www.edureka.co/blog/interview-questions/react-interview-questions/_blank)

### **[React JS Certification Training Course](https://www.edureka.co/reactjs-redux-certification-training?utm_source=blogbanner&utm_campaign=curriculum" \t "https://www.edureka.co/blog/interview-questions/react-interview-questions/_blank)**

* *[Instructor-led Sessions](https://www.edureka.co/reactjs-redux-certification-training?utm_source=blogbanner&utm_campaign=curriculum" \t "https://www.edureka.co/blog/interview-questions/react-interview-questions/_blank)*
* *[Assessments](https://www.edureka.co/reactjs-redux-certification-training?utm_source=blogbanner&utm_campaign=curriculum" \t "https://www.edureka.co/blog/interview-questions/react-interview-questions/_blank)*
* *[Assignments](https://www.edureka.co/reactjs-redux-certification-training?utm_source=blogbanner&utm_campaign=curriculum" \t "https://www.edureka.co/blog/interview-questions/react-interview-questions/_blank)*
* *[Lifetime Access](https://www.edureka.co/reactjs-redux-certification-training?utm_source=blogbanner&utm_campaign=curriculum" \t "https://www.edureka.co/blog/interview-questions/react-interview-questions/_blank)*

[Explore Curriculum](https://www.edureka.co/reactjs-redux-certification-training?utm_source=blogbanner&utm_campaign=curriculum" \t "https://www.edureka.co/blog/interview-questions/react-interview-questions/_blank)

1. It uses the ****virtual DOM**** instead of the real DOM.
2. It uses ****server-side rendering****.
3. It follows ****uni-directional data flow**** or data binding.

### ****4. List some of the major advantages of React.****

Some of the major advantages of React are:

1. It increases the application’s performance
2. It can be conveniently used on the client as well as server side
3. Because of JSX, code’s readability increases
4. React is easy to integrate with other frameworks like Meteor, Angular, etc
5. Using React, writing UI test cases become extremely easy

### ****5. What are the limitations of React?****

Limitations of React are listed below:

1. React is just a library, not a full-blown framework
2. Its library is very large and takes time to understand
3. It can be little difficult for the novice programmers to understand
4. Coding gets complex as it uses inline templating and JSX

## ****React Interview Questions and Answers in 2023 | Edureka****

### ****6. What is JSX?****

JSX is a shorthand for JavaScript XML. This is a type of file used by React which utilizes the expressiveness of JavaScript along with HTML like template syntax. This makes the HTML file really easy to understand. This file makes applications robust and boosts its performance. Below is an example of JSX:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | render(){  **return**(    <div>    <h1> Hello World from Edureka!!</h1>             </div>        );  } |

### ****7. What do you understand by Virtual DOM? Explain its works.****

A virtual DOM is a lightweight JavaScript object which originally is just a copy of the real DOM. It is a node tree that lists the elements, their attributes and content as Objects and their properties. React’s render function creates a node tree out of the React components. It then updates this tree in response to the mutations in the data model which is caused by various actions done by the user or by the system. Check out this [Web developer course online](https://www.edureka.co/masters-program/full-stack-developer-training) to learn more about react.

This Virtual DOM works in three simple steps.

1. Whenever any underlying data changes, the entire UI is re-rendered in Virtual DOM representation.
2. Then the difference between the previous DOM representation and the new one is calculated.
3. Once the calculations are done, the real DOM will be updated with only the things that have actually changed. 

### ****8. Why can’t browsers read JSX?****

Browsers can only read JavaScript objects but JSX in not a regular JavaScript object. Thus to enable a browser to read JSX, first, we need to transform JSX file into a JavaScript object using JSX transformers like Babel and then pass it to the browser.

### ****9. How different is React’s ES6 syntax when compared to ES5?****

Syntax has changed from ES5 to ES6 in the following aspects:

1. require vs import

|  |  |
| --- | --- |
| 1  2  3  4  5 | // ES5  var React = require('react');    // ES6  **import** React from 'react'; |

1. export vs exports

|  |  |
| --- | --- |
| 1  2  3  4  5 | // ES5  module.exports = Component;    // ES6  export **default** Component; |

1. component and function

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | // ES5  var MyComponent = React.createClass({      render: function() {  **return**    <h3>Hello Edureka!</h3>  ;      }  });    // ES6  **class** MyComponent **extends** React.Component {      render() {  **return**    <h3>Hello Edureka!</h3>  ;      }  } |

1. props

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | // ES5  var App = React.createClass({      propTypes: { name: React.PropTypes.string },      render: function() {  **return**    <h3>Hello, {**this**.props.name}!</h3>  ;      }  });    // ES6  **class** App **extends** React.Component {      render() {  **return**    <h3>Hello, {**this**.props.name}!</h3>  ;      }  } |

1. state

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | // ES5  var App = React.createClass({      getInitialState: function() {  **return** { name: 'world' };      },      render: function() {  **return**    <h3>Hello, {**this**.state.name}!</h3>  ;      }  });    // ES6  **class** App **extends** React.Component {      constructor() {  **super**();  **this**.state = { name: 'world' };      }      render() {  **return**    <h3>Hello, {**this**.state.name}!</h3>  ;      }  } |

### ****10. How is React different from Angular?****

|  |  |  |
| --- | --- | --- |
| React vs Angular ****TOPIC**** | ****REACT**** | ****ANGULAR**** |
| 1. ARCHITECTURE | Only the View of MVC | Complete MVC |
| 2. RENDERING | Server-side rendering | Client-side rendering |
| 3. DOM | Uses virtual DOM | Uses real DOM |
| 4. DATA BINDING | One-way data binding | Two-way data binding |
| 5. DEBUGGING | Compile time debugging | Runtime debugging |
| 6. AUTHOR | Facebook | Google |

## ****React Components – React Interview Questions****

### ****11. “In React, everything is a component.” Explain.****

Components are the building blocks of a React application’s UI. These components split up the entire UI into small independent and reusable pieces. Then it renders each of these components independent of each other without affecting the rest of the UI.

### ****12. What is the purpose of render() in React.****

Each React component must have a ****render()****mandatorily. It returns a single React element which is the representation of the native DOM component. If more than one HTML element needs to be rendered, then they must be grouped together inside one enclosing tag such as ****<form>, <group>,<div>**** etc. This function must be kept pure i.e., it must return the same result each time it is invoked.

### ****13. How can you embed two or more components into one?****

We can embed components into one in the following way:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | **class** MyComponent **extends** React.Component{      render(){  **return**(    <div>    <h1>Hello</h1>                    <Header/>              </div>            );      }  }  **class** Header **extends** React.Component{      render(){  **return**    <h1>Header Component</h1>       };  }  ReactDOM.render(      <MyComponent/>, document.getElementById('content')  ); |

### ****14. What is Props?****

Props is the shorthand for Properties in React. They are read-only components which must be kept pure i.e. immutable. They are always passed down from the parent to the child components throughout the application. A child component can never send a prop back to the parent component. This help in maintaining the unidirectional data flow and are generally used to render the dynamically generated data.

### ****15. What is a state in React and how is it used?****

States are the heart of React components. States are the source of data and must be kept as simple as possible. Basically, states are the objects which determine components rendering and behavior. They are mutable unlike the props and create dynamic and interactive components. They are accessed via ****this.state().****

### ****16. Differentiate between states and props.****

|  |  |  |
| --- | --- | --- |
| States vs Props ****Conditions**** | ****State**** | ****Props**** |
| 1. Receive initial value from parent component | Yes | Yes |
| 2. Parent component can change value | No | Yes |
| 3. Set default values inside component | Yes | Yes |
| 4. Changes inside component | Yes | No |
| 5. Set initial value for child components | Yes | Yes |
| 6. Changes inside child components | No | Yes |

### ****17. How can you update the state of a component?****

State of a component can be updated using this.setState().

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | **class** MyComponent **extends** React.Component {      constructor() {  **super**();  **this**.state = {              name: 'Maxx',              id: '101'          }      }      render()          {              setTimeout(()=>{**this**.setState({name:'Jaeha', id:'222'})},2000)  **return** (    <div>    <h1>Hello {**this**.state.name}</h1>    <h2>Your Id is {**this**.state.id}</h2>                       </div>                );          }      }  ReactDOM.render(      <MyComponent/>, document.getElementById('content')  ); |

### ****18. What is arrow function in React? How is it used?****

Arrow functions are more of brief syntax for writing the function expression. They are also called ‘fat arrow‘ (****=>****) the functions. These functions allow to bind the context of the components properly since in ES6 auto binding is not available by default. Arrow functions are mostly useful while working with the higher order functions.

[](https://www.edureka.co/reactjs-redux-certification-training?utm_source=blogbanner&utm_campaign=batches" \t "https://www.edureka.co/blog/interview-questions/react-interview-questions/_blank)

### **[React JS Certification Training Course](https://www.edureka.co/reactjs-redux-certification-training?utm_source=blogbanner&utm_campaign=batches" \t "https://www.edureka.co/blog/interview-questions/react-interview-questions/_blank)**

[Weekday / Weekend BatchesSee Batch Details](https://www.edureka.co/reactjs-redux-certification-training?utm_source=blogbanner&utm_campaign=batches" \t "https://www.edureka.co/blog/interview-questions/react-interview-questions/_blank)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | //General way  render() {  **return**(          <MyInput onChange={**this**.handleChange.bind(**this**) } />      );  }  //With Arrow Function  render() {  **return**(          <MyInput onChange={ (e) => **this**.handleOnChange(e) } />      );  } |

### ****19. Differentiate between stateful and stateless components.****

|  |  |
| --- | --- |
| Stateful vs Stateless ****Stateful Component**** | ****Stateless Component**** |
| 1. Stores info about component’s state change in memory | 1. Calculates the internal state of the components |
| 2. Have authority to change state | 2. Do not have the authority to change state |
| 3. Contains the knowledge of past, current and possible future changes in state | 3. Contains no knowledge of past, current and possible future state changes |
| 4. Stateless components notify them about the requirement of the state change, then they send down the props to them. | 4. They receive the props from the Stateful components and treat them as callback functions. |

### ****20. What are the different phases of React component’s lifecycle?****

There are three different phases of React component’s lifecycle:

1. Initial Rendering Phase: This is the phase when the component is about to start its life journey and make its way to the DOM.
2. Updating Phase: Once the component gets added to the DOM, it can potentially update and re-render only when a prop or state change occurs. That happens only in this phase.
3. Unmounting Phase: This is the final phase of a component’s life cycle in which the component is destroyed and removed from the DOM.

Want to upskill yourself to get ahead in your career? Check out this video

## ****Top 10 Technologies to Learn in 2023| Edureka****

### ****21. Explain the lifecycle methods of React components in detail.****

Some of the most important lifecycle methods are:

1. **componentWillMount()**–Executed just before rendering takes place both on the client as well as server-side.
2. **componentDidMount()**–Executed on the client side only after the first render.
3. **componentWillReceiveProps()**– Invoked as soon as the props are received from the parent class and before another render is called.
4. **shouldComponentUpdate()**–Returns true or false value based on certain conditions. If you want your component to update, return **true** else return **false**. By default, it returns false.
5. **componentWillUpdate()**– Called just before rendering takes place in the DOM.
6. **componentDidUpdate()**–Called immediately after rendering takes place.
7. **componentWillUnmount()**– Called after the component is unmounted from the DOM. It is used to clear up the memory spaces.

### ****22. What is an event in React?****

In React, events are the triggered reactions to specific actions like mouse hover, mouse click, key press, etc. Handling these events are similar to handling events in DOM elements. But there are some syntactical differences like:

1. Events are named using camel case instead of just using the lowercase.
2. Events are passed as functions instead of strings.

The event argument contains a set of properties, which are specific to an event. Each event type contains its own properties and behavior which can be accessed via its event handler only.

### ****23. How do you create an event in React?****

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | **class** Display **extends** React.Component({      show(evt) {          // code      },      render() {          // Render the div with an onClick prop (value is a function)  **return** (    <div onClick={**this**.show}>Click Me!</div>            );      }  }); |

### ****24. What are synthetic events in React?****

Synthetic events are the objects which act as a cross-browser wrapper around the browser’s native event. They combine the behavior of different browsers into one API. This is done to make sure that the events show consistent properties across different browsers.

### ****25. What do you understand by refs in React?****

Refs is the short hand for References in React. It is an attribute which helps to store a reference to a particular React element or component, which will be returned by the components render configuration function. It is used to return references to a particular element or component returned by render(). They come in handy when we need DOM measurements or to add methods to the components.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | **class** ReferenceDemo **extends** React.Component{       display() {  **const** name = **this**.inputDemo.value;           document.getElementById('disp').innerHTML = name;       }  render() {  **return**(    <div>              Name: <input type="text" ref={input => **this**.inputDemo = input} />              <button name="Click" onClick={**this**.display}>Click</button>    <h2>Hello <span id="disp"></span> !!!</h2>          </div>      );     }   } |

### ****26. List some of the cases when you should use Refs.****

Following are the cases when refs should be used:

* When you need to manage focus, select text or media playback
* To trigger imperative animations
* Integrate with third-party DOM libraries

### ****27. How do you modularize code in React?****

We can modularize code by using the export and import properties. They help in writing the components separately in different files.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | //ChildComponent.jsx  export **default** **class** ChildComponent **extends** React.Component {      render() {  **return**(    <div>    <h1>This is a child component</h1>               </div>            );      }  }    //ParentComponent.jsx  **import** ChildComponent from './childcomponent.js';  **class** ParentComponent **extends** React.Component {      render() {  **return**(    <div>                  <App />              </div>            );      }  } |

### ****28. How**** ****are forms created in React?****

React forms are similar to HTML forms. But in React, the state is contained in the state property of the component and is only updated via setState(). Thus the elements can’t directly update their state and their submission is handled by a JavaScript function. This function has full access to the data that is entered by the user into a form.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | handleSubmit(event) {      alert('A name was submitted: ' + **this**.state.value);      event.preventDefault();  }    render() {  **return** (    <form onSubmit={**this**.handleSubmit}>              <label>                  Name:                  <input type="text" value={**this**.state.value} onChange={**this**.handleSubmit} />              </label>              <input type="submit" value="Submit" />          </form>        );  } |

### ****29. What do you know about controlled and uncontrolled components?****

|  |  |
| --- | --- |
| Controlled vs Uncontrolled Components ****Controlled Components**** | ****Uncontrolled Components**** |
| 1. They do not maintain their own state | 1. They maintain their own state |
| 2. Data is controlled by the parent component | 2. Data is controlled by the DOM |
| 3. They take in the current values through props and then notify the changes via callbacks | 3. Refs are used to get their current values |

In case you are facing any challenges with these React interview questions, please comment on your problems in the section below.

## ****React Interview Questions****

### ****30. What are Higher Order Components(HOC)?****

Higher Order Component is an advanced way of reusing the component logic. Basically, it’s a pattern that is derived from React’s compositional nature. HOC are custom components which wrap another component within it. They can accept any dynamically provided child component but they won’t modify or copy any behavior from their input components. You can say that HOC are ‘pure’ components.

### ****31. What can you do with HOC?****

HOC can be used for many tasks like:

* Code reuse, logic and bootstrap abstraction
* Render High jacking
* State abstraction and manipulation
* Props manipulation

### ****32. What are Pure Components?****

*Pure*components are the simplest and fastest components which can be written. They can replace any component which only has a **render().**These components enhance the simplicity of the code and performance of the application.

### ****33. What is the significance of keys in React?****

Keys are used for identifying unique Virtual DOM Elements with their corresponding data driving the UI. They help React to optimize the rendering by recycling all the existing elements in the DOM. These keys must be a unique number or string, using which React just reorders the elements instead of re-rendering them. This leads to increase in application’s performance.

****React Redux – React Interview Questions****

### ****34. What were the major problems with MVC framework?****

Following are some of the major problems with MVC framework:

* DOM manipulation was very expensive
* Applications were slow and inefficient
* There was huge memory wastage
* Because of circular dependencies, a complicated model was created around models and views

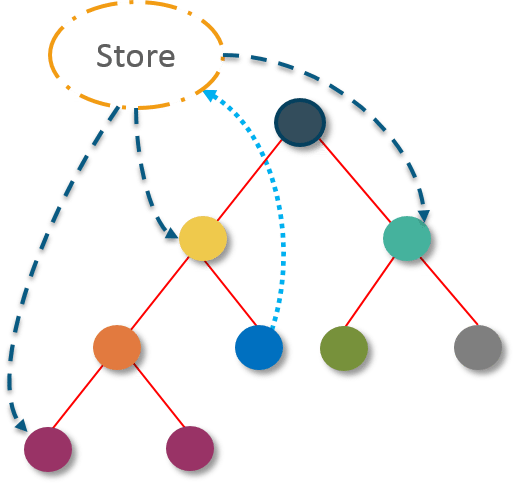
### ****35. Explain Flux.****

Flux is an architectural pattern which enforces the uni-directional data flow. It controls derived data and enables communication between multiple components using a central Store which has authority for all data. Any update in data throughout the application must occur here only. Flux provides stability to the application and reduces run-time errors.

### ****36.**** ****What is Redux?****

Redux is one of the most trending libraries for front-end development in today’s marketplace. It is a predictable state container for JavaScript applications and is used for the entire applications state management. Applications developed with Redux are easy to test and can run in different environments showing consistent behavior.

### ****37. What are the three principles that Redux follows?****

1. ****Single source of truth:****The state of the entire application is stored in an object/ state tree within a single store. The single state tree makes it easier to keep track of changes over time and debug or inspect the application.
2. **State is read-only:**The only way to change the state is to trigger an action. An action is a plain JS object describing the change. Just like state is the minimal representation of data, the action is the minimal representation of the change to that data.
3. ****Changes are made with pure functions:**** In order to specify how the state tree is transformed by actions, you need pure functions. Pure functions are those whose return value depends solely on the values of their arguments.

### ****38. What do you understand by “Single source of truth”?****

Redux uses ‘Store’ for storing the application’s entire state at one place. So all the component’s state are stored in the Store and they receive updates from the Store itself. The single state tree makes it easier to keep track of changes over time and debug or inspect the application.

### ****39. List down the components of Redux.****

Redux is composed of the following components:

1. **Action** – It’s an object that describes what happened.
2. **Reducer**–  It is a place to determine how the state will change.
3. **Store** – State/ Object tree of the entire application is saved in the Store.
4. **View** – Simply displays the data provided by the Store.

In case you are facing any challenges with these React interview questions, please comment on your problems in the section below.

### ****40. Show how the data flows through Redux?****



### ****41. How are Actions defined in Redux?****

Actions in React must have a type property that indicates the type of ACTION being performed. They must be defined as a String constant and you can add more properties to it as well. In Redux, actions are created using the functions called Action Creators. Below is an example of Action and Action Creator:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | function addTodo(text) {  **return** {                  type: ADD\_TODO,                   text      }  } |

### ****42. Explain the role of Reducer.****

Reducers are pure functions which specify how the application’s state changes in response to an ACTION. Reducers work by taking in the previous state and action, and then it returns a new state. It determines what sort of update needs to be done based on the type of the action, and then returns new values. It returns the previous state as it is, if no work needs to be done.

### ****43. What is the significance of Store in Redux?****

A store is a JavaScript object which can hold the application’s state and provide a few helper methods to access the state, dispatch actions and register listeners. The entire state/ object tree of an application is saved in a single store. As a result of this, Redux is very simple and predictable. We can pass middleware to the store to handle the processing of data as well as to keep a log of various actions that change the state of stores. All the actions return a new state via reducers.

### ****44. How is Redux different from Flux?****

|  |  |
| --- | --- |
| Flux vs Redux ****Flux**** | ****Redux**** |
| 1. The Store contains state and change logic | 1. Store and change logic are separate |
| 2. There are multiple stores | 2. There is only one store |
| 3. All the stores are disconnected and flat | 3. Single store with hierarchical reducers |
| 4. Has singleton dispatcher | 4. No concept of dispatcher |
| 5. React components subscribe to the store | 5. Container components utilize connect |
| 6. State is mutable | 6. State is immutable |

In case you are facing any challenges with these React interview questions, please comment on your problems in the section below.

## ****React Interview Questions****

### ****45. What are the advantages of Redux?****

Advantages of Redux are listed below:

* ****Predictability of outcome –****Since there is always one source of truth, i.e. the store, there is no confusion about how to sync the current state with actions and other parts of the application.
* ****Maintainability –****The code becomes easier to maintain with a predictable outcome and strict structure.
* ****Server-side rendering –**** You just need to pass the store created on the server, to the client side. This is very useful for initial render and provides a better user experience as it optimizes the application performance.
* ****Developer tools –****From actions to state changes, developers can track everything going on in the application in real time.
* ****Community and ecosystem –****Redux has a huge community behind it which makes it even more captivating to use. A large community of talented individuals contribute to the betterment of the library and develop various applications with it.
* ****Ease of testing –****Redux’s code is mostly functions which are small, pure and isolated. This makes the code testable and independent.
* ****Organization –****Redux is precise about how code should be organized, this makes the code more consistent and easier when a team works with it.

****React Router – React Interview Questions****

### ****46. What is React Router?****

React Router is a powerful routing library built on top of React, which helps in adding new screens and flows to the application. This keeps the URL in sync with data that’s being displayed on the web page. It maintains a standardized structure and behavior and is used for developing single page web applications. React Router has a simple API.

### ****47. Why**** ****is switch keyword used in React Router v4?****

Although a ****<div>**** is used to encapsulate multiple routes inside the Router. The ‘switch’ keyword is used when you want to display only a single route to be rendered amongst the several defined routes. The ****<switch>****tag when in use matches the typed URL with the defined routes in sequential order. When the first match is found, it renders the specified route. Thereby bypassing the remaining routes.

### ****48. Why do we need a Router in React?****

A Router is used to define multiple routes and when a user types a specific URL, if this URL matches the path of any ‘route’ defined inside the router, then the user is redirected to that particular route. So basically, we need to add a Router library to our app that allows creating multiple routes with each leading to us a unique view.

|  |  |
| --- | --- |
| 1  2  3  4  5 | <**switch**>      <route exact path=’/’ component={Home}/>      <route path=’/posts/:id’ component={Newpost}/>      <route path=’/posts’   component={Post}/>  </**switch**> |

### ****49. List down the advantages of React Router.****

Few advantages are:

1. Just like how React is based on components, in React Router v4, the API is ‘All About Components’. A Router can be visualized as a single root component (****<BrowserRouter>****) in which we enclose the specific child routes (****<route>****).
2. No need to manually set History value: In React Router v4, all we need to do is wrap our routes within the ****<BrowserRouter>**** component.
3. The packages are split: Three packages one each for Web, Native and Core. This supports the compact size of our application. It is easy to switch over based on a similar coding style.

### ****50. How is React Router different from conventional routing?****

|  |  |  |
| --- | --- | --- |
| Conventional Routing vs React Routing ****Topic**** | ****Conventional Routing**** | ****React Routing**** |
| ****PAGES INVOLVED**** | Each view corresponds to a new file | Only single HTML page is involved |
| ****URL CHANGES**** | A HTTP request is sent to a server and corresponding HTML page is received | Only the History attribute is changed |
| ****FEEL**** | User actually navigates across different pages for each view | User is duped thinking he is navigating across different pages |

I hope this set of [React](https://reactjs.org/" \t "https://www.edureka.co/blog/interview-questions/react-interview-questions/_blank) Interview Questions and Answers will help you in preparing for your interviews. All the best!

### ****1. What is the difference between Virtual DOM and Real DOM?****

|  |  |
| --- | --- |
| ****Virtual DOM**** | ****Real DOM**** |
| Changes can be made easily | Changes can be expensive |
| Minimal memory wastage | High demand for memory and more wastage |
| JSX element is updated if the element exists | Creates a new DOM every time an element gets updated |
| Cannot update HTML directly | Able to directly manipulate HTML |
| Faster updates | Slow updates |

### ****2. What is React?****

****React****is a widely used JavaScript library that was launched in 2011. It was created by developers at Facebook, and it is primarily used for frontend development. React uses the component-based approach, which ensures to help you build components that possess high reusability.

React is well known for developing and designing complex mobile user interfaces and web applications.

### ****3. What is the meaning of Virtual DOM?****

A virtual DOM is a simple JavaScript object that is the exact copy of the corresponding real DOM. It can be considered as a node tree that consists of elements, their attributes, and other properties. Using the render function in React, it creates a node tree and updates it based on the changes that occur in the data model. These changes are usually triggered by users or the actions caused by the system.

Next up among these React interview questions, you need to take a look at some of the important features that React offers.

### ****4. What are some of the important features of React?****

React has multiple features that are used for unique purposes. The important ones are as mentioned below:

* React makes use of a single-direction data flow model.
* It deals with complete server-side data processing and handling.
* React uses Virtual DOM that has many advantages of its own.

### ****5. What is the meaning of JSX?****

JSX is the abbreviation of JavaScript XML. It is a file that is used in React to bring out the essence of JavaScript to React and use it for its advantages.

It even includes bringing out HTML and the easy syntax of JavaScript. This ensures that the resulting HTML file will have high readability, thereby relatively increasing the performance of the application.

Consider the following example of a JSX:

render(){

return(

<div>

<h1> Hello Intellipaat learners!</h1>

</div>

);

}

### ****6. Can browsers read a JSX file?****

No, browsers cannot read JSX files directly. It can only read the objects provided by JavaScript. Now, to make a browser read a JSX file, it has to be transformed to a JavaScript object using JSX transformers, and only then it can be fed into the browser for further use in the pipeline.

窗体底端

### ****7. Why is React widely used today?****

****React**** provides users with an ample number of advantages when building an application. Some of them are as follows:

* With React, UI testing becomes very easy.
* React can integrate with [Angular](https://intellipaat.com/blog/angular-vs-react/" \t "https://intellipaat.com/blog/interview-question/react-interview-questions/_blank) and other frameworks easily.
* The high readability index ensures easy understanding.
* React can be used for both client-side and server-side requirements.
* It boosts application performance and overall efficiency.

****Enroll in our**[UI UX course](https://intellipaat.com/ui-ux-design-course/)**to learn UI UX from the IIT Guwahati faculty!****

### ****8. Are there any disadvantages to using React?****

There are some limitations when using React as mentioned below:

* Writing code is complicated as it uses JSX and inline template formatting.
* Beginners might find it tough to cope with its syntaxes and methods.
* The library contains a huge repository of information, which might be overwhelming.
* React is a simple library and not a complete framework hence calls for dependencies.

### ****9. Differentiate between Angular and React.****

|  |  |  |
| --- | --- | --- |
| ****Comparison Factor**** | ****Angular**** | ****React**** |
| Created by | Google | Facebook |
| DOM | Real DOM | Virtual DOM |
| Render Support | Client-side | Server-side |
| Architecture | Full MVC support | Only the view aspect of MVC |
| Data Binding | Unidirectional binding | Two-way binding |

****To learn more about React JS, Enroll now in our**[React certification course](https://intellipaat.com/react-js-certification-training-course/" \t "https://intellipaat.com/blog/interview-question/react-interview-questions/_blank)**to gain in-depth knowledge.****

### ****10. What is the meaning of the component-based architecture of React?****

In ****React****, components are foundations used to build user interfaces for applications. With the component-based system in place, all of the individual entities become completely reusable and independent of each other. This means that rendering the application is easy and not dependent on the other components of the UI.

### ****11. How does rendering work in React?****

Rendering is an important aspect of React as every single component must be rendered. This is done using the render() function. Once the function is called, it returns an element that represents a DOM component.

It is also possible to render more than one HTML element at a time by enclosing the HTML tags and passing them through the render function.

### ****12. What are states in React?****

States form to be one of the vital aspects of React. It is considered as a source of data or objects that control aspects such as component behavior and rendering. In React, states are used to easily create dynamic and interactive components.

Check out our blog on [ReactJS Tutorial](https://intellipaat.com/blog/reactjs-tutorial/" \t "https://intellipaat.com/blog/interview-question/react-interview-questions/_blank) to learn more about ReactJS.

### ****13. What are props in React?****

Props are the shorthand name given to properties in React. Props are read-only components that are immutable in nature. In an application, props follow a hierarchy that is passed down from parents to child components. However, the reverse is not supported. This is done to ensure that there is only a single directional flow in data at all times.

### ****14. What is the use of an arrow function in React?****

An arrow function is used to write an expression in React. It allows users to manually bind components easily. The functionality of arrow functions can be very useful when you are working with higher-order functions particularly.

Consider the following example:

//The usual way

render() {

return(

<MyInput onChange={this.handleChange.bind(this) } />

);

}

//Making use of the arrow function

render() {

return(

<MyInput onChange={ (e) => this.handleOnChange(e) } />

);

}

### ****15. What is a higher-order component in React?****

Higher-order components (HOCs) are a widely used technique in React for applying concepts that involve the component reusability logic. They are not a native part of the React API and allow users to easily reuse the code and bootstrap abstraction.

HOCs are also used to allow simple sharing of behaviors across all of the components in React, adding more advances to the efficiency and functioning of the application.

### ****16. What is the meaning of create-react-app in React?****

The create-react app in ****React****is a simple command-line interface (CLI) that is used in the creation of React applications, which have no build configuration.

All tools are pre-configured when using the CLI, and this allows users to focus on the code more than on dependencies to develop the application.

The following syntax is used to start a simple project in React:

Create-react-app my-app

### ****17. What are some of the advantages of using create-react-app in React?****

Making use of create-react-app is advantageous in the following way:

* Support for JSX, ES6, and flow statements
* Already built and ready auto-prefixed CSS
* Fast interactive testing components
* Live development servers that help in debugging
* Scripts to handle JSS, CSS, and other files

Next up on these React Redux interview questions, you need to understand the meaning of Redux.

### ****18. What is the meaning of Redux?****

Redux is used to store the state of the application in a single entity. This simple entity is usually a JavaScript object. Changing states can be done by pushing out actions from the application and writing corresponding objects for them that are used to modify the states.

For example:

{

first\_name: ‘John’,

last\_name : ‘Kelly’,

age: 25

}

All of the data is retained by Redux (also called a store).

### ****19. What is the difference between props and states?****

|  |  |  |
| --- | --- | --- |
| ****Condition**** | ****Props**** | ****States**** |
| Changes in child components | Yes | No |
| Parent component changing values | Yes | No |
| Changes inside components | No | Yes |

Next up on this top React interview questions and answers blog, take a look at the questions categorized as intermediate!

## **Intermediate React Interview Questions**

### ****20. What are the three phases of a component life cycle in React?****

The following are the three phases of a component life cycle:

* ****Initial rendering****: This is the phase that involves the beginning of the journey of the component to the DOM.
* ****Update****: Here, the component can be updated and rendered again if required after it gets added to the DOM.
* ****Unmounting****: The final phase involves the destruction of the component and its eventual removal from the DOM.

### ****21. What are events in React?****

Whenever there are actions performed in [React](https://reactjs.org/" \t "https://intellipaat.com/blog/interview-question/react-interview-questions/_blank), such as hovering the mouse or pressing a key on the keyboard, these actions trigger events. Events then perform set activities as a response to these triggers. Handling an event in React is very similar to that in the DOM architecture.

### ****22. How are events created in React?****

Events can be created very easily in React as shown here:

class Display extends React.Component({

show(evt) {

// Code inside

},

render() {

// Render the div with an onClick prop (value is a function)

return (

<div onClick={this.show}>Click Here</div>

);

}

});

### ****23. How is routing in React different from conventional routing?****

Differences between the conventional routing and the routing in React can be shown using the following aspects:

* ****Pages****: Each view is considered as a new file in conventional routing while it is considered as a single HTML entity in React.
* ****Navigation****: In conventional routing, users have to move across web pages for viewing. In React, the views are not refreshed as objects are re-issued to create new views.

### ****24. Differentiate between Flux and Redux in React.****

|  |  |  |
| --- | --- | --- |
| ****Comparison Factor**** | ****Flux**** | ****Redux**** |
| Components | Components connected to Flux in React | Container components directly connect |
| Dispatcher | Has a dispatcher | No dispatcher |
| Number of Stores | Single store | Multiple stores |
| State | Mutable state | Immutable state |
| Storage | Contains state and logic | State and logic are separate |

### ****25. Can AJAX be used with React?****

Yes, any AJAX library, such as Axios and jQuery AJAX, can be used with React easily. One important thing is to maintain the states of the components, and here too, the props are passed from the parents to the child components.

Child components still cannot send back props to parents, and this factor greatly increases rendering efficiency when dynamic data is considered.

**If you are looking forward to becoming proficient in Angular.js, then make sure to check out Intellipaat’s latest offerings for the [Angular JS  Course](https://intellipaat.com/angular-training/" \t "https://intellipaat.com/blog/interview-question/react-interview-questions/_blank).**

### ****26. What is the meaning of synthetic events in React?****

Synthetic events in React are objects that act as cross-browser wrappers, allowing for the use of native events. This is done to ensure that a variety of browsers can run the API and that the event contains all properties.

### ****27. What are stateful components in React?****

Stateful components are entities that store the changes that happen and place them into the memory. Here, the state can be changed, alongside storing information such as past, current, and future changes.

### ****in React?****

‘Refs’ is short for references in React. Refs are used to store a reference to a single React element or a React component. This is later returned using the render function.

They are mainly used in the following scenarios:

* To initiate imperative animations
* To join third-party DOM libraries
* To manage focus and apply media playback

### ****29. What are controlled components in React?****

Controlled components in React refer to the components that have the ability to maintain their state. The data is completely controlled by the parent component, and the current value is fetched by making use of props. This is done to notify about any change that occurs when using callbacks.

### ****30. Why is a router required in React?****

A router is very much necessary in React as it is used to manage multiple routes whenever a user types in a URL. If the route is present in the router for that corresponding URL, then the user is taken to the particular route.

To do this, the router library needs to be added in React. It can be done using the following syntax:

<switch>

<route exact path=’/’ component={Home}/>

<route path=’/posts/:id’ component={Newpost}/>

<route path=’/posts’   component={Post}/>

</switch>

### ****31. What are the components of Redux in React?****

Redux consists of four main components as shown below:

* ****Action****: An object that describes the call
* ****Reducer****: The state change storage unit
* ****Store****: the state and object tree storage
* ****View****: Displays data provided by the store

### ****32. What are the advantages of using Redux?****

There are many advantages of Redux, and some of them are as given below:

|  |  |
| --- | --- |
| Organized Approach | Redux requires code to be organized, thereby making it consistent and easy to work with |
| Testing Ability | Redux functions are small and isolated, making the code more independent and testable |
| Tools | Developers can track actions and all of the tools in React using Redux easily |
| Community | Redux has a larger community, helping users with efficient and easy-to-use libraries |

### ****33. What are the disadvantages of using MVC in React?****

Among a plethora of advantages of using MVC in React, there are minor problems as stated below:

* A lot of memory wastage occurs.
* DOM manipulation costs a lot.
* The application becomes slow.
* Lots of dependencies are created.
* The complexity of models increases.

Next up among these ReactJS interview questions, you have to understand about pure components.

### ****34. What are pure components in React?****

Pure components are singular entities that are written in React. They are fast and simple to write and have the ability to replace a component that has only the render() function. This is done to ensure that the performance of the application is good and that the code is kept simple at the same time.

Next up on this top React interview questions blog, take a look at the questions categorized as advanced!

### ****35. What are higher-order components (HOCs) used for?****

HOCs are used for a variety of tasks such as:

* Manipulation of props
* State manipulation and abstraction
* Render highjacking
* Code reusing
* Bootstrap abstraction

### ****36. What are keys in React?****

Keys are used in React to check all items and to track changes actively. They are used to directly check if an item has been added or removed from a list.

Consider the following syntax:

function List ({ todos }) {

return (

<ul>

{todos.map(({ task, id} ) => <li key={id}>{task}</li>}

</ul>

)

}

### ****37. Differentiate between a controlled component and an uncontrolled component in React.****

A controlled component, as the name suggests, is a component over which React has complete control. It is the singular point of data for the forms.

An uncontrolled component is one where the form data gets handled by DOM and not the React component. This is usually done using refs in React.

### ****38. How can you tell React to build in the production mode?****

****React**** can be coded to directly build into production by setting the process.env.NODE\_ENV variable to production.

****Note****: When React is in production, warnings and other development features are not shown.

### ****39. What is the difference between cloneElement and createElement in React?****

In React, cloneElement is primarily used to clone an element and pass it to new props directly. Whereas, createElement is the entity that JSX gets compiled into. This is also used to create elements in React.

Next up on this top React interview questions and answers blog, take a look at the use of the second argument.

### ****40. What is the use of the second argument that is passed to setState? Is it optional?****

When setState is finished, a callback function is invoked, and the components get re-rendered in React.

Yes, it is an optional argument. Since setState is asynchronous, it takes in another callback function. However, in programming practice, it is always good to use another life cycle method instead of this.

Next up on this top React interview questions and answers blog, you need to take a look at binding.

### ****41. Is there a way to avoid the requirement of binding when using React?****

Yes, there are two main ways you can use to avoid the need for binding. They are as follows:

* ****Defining the Event Handler as an Inline Arrow function:****

class SubmitButton extends React.Component {

constructor(props) {

super(props);

this.state = {

isFormSubmitted: false

};

}

render() {

return (

<button onClick={() => {

this.setState({ isFormSubmitted: true });

}}>Submit</button>

)

}

}

* ****Using a function component along with Hooks:****

const SubmitButton = () => {

const [isFormSubmitted, setIsFormSubmitted] = useState(false);

return (

<button onClick={() => {

setIsFormSubmitted(true);

}}>Submit</button>

)

};

Also, the Event Handler can be defined as an Arrow function, which is eventually assigned to a Class Field to obtain similar results.

### ****42. What is the StrictMode component used in React?****

The StrictMode component when used would benefit users immensely while creating new codebases to understand the components being used.

However, it can fit well in debugging as well because it will help solve the problem faster when it is wrapped with other components, which could be causing the problem.

Next up on these interview questions on React JS, you have to understand how to speed up rendering.

### ****43. What would you do if your React application is rendering slowly?****

The cause of slow rendering in React is mostly because of the number of re-render operations, which are sometimes unnecessary. There are two main tools provided by React to help users here:

* ****memo():**** This is used to prevent all of the unnecessary re-rendering carried out by the function components.
* ****PureComponent:**** This is used to ensure that the unnecessary re-rendering of class components is avoided.

### ****44. Can you conditionally add attributes to components in React?****

Yes, there is a way in which you can add attributes to a React component when certain conditions are met.

React has the ability to omit an attribute if the value passed to it is not true.

Consider the following example:

var condition = true;

var component = (

<div

value="foo"

{ ...( condition && { disabled: true } ) } />

);

### ****45. Why is props passed to the super() function in React?****

Props gets passed onto the super() function if a user wishes to access this.props in the constructor.

Consider the following example:

class MyComponent extends React.Component {

constructor(props) {

super(props)

console.log(this.props)

// -> { icon: 'home', … }

}

}

### ****46. What is the difference between using getInitialState and constructors in React?****

When using ES6, users must initialize the state in the constructor and the getInitialState method is defined. This is done using React.createClass as shown in the below example:

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = { /\* initial state \*/ };

}

}

This above piece of code is equivalent to the following:

var MyComponent = React.createClass({

getInitialState() {

return { /\* initial state \*/ };

},

});

Next up among these interview questions on React JS, you have to know what predefined props are.

### ****47. What are the predefined prop types present in React?****

There are five main predefined prop types in React. They are as follows:

1. PropTypes.bool
2. PropTypes.func
3. PropTypes.node
4. PropTypes.number
5. PropTypes.string

The propTypes can be defined for the user component as shown below:

import PropTypes from 'prop-types';

class User extends React.Component {

render() {

return (

<h1>Welcome, {this.props.name}</h1>

<h2>Age, {this.props.age}

);

}

}

User.propTypes = {

name: PropTypes.string.isRequired,

age: PropTypes.number.isRequired

};

### ****48. What is React Fiber?****

React Fiber is a new engine in React. It is the reimplementation core algorithm in React 16.

The main goal of React Fiber is to ensure that there are incremental rendering facilities for the virtual DOM. This increases efficiency when rendering animations, gestures, etc. and also helps in assigning priority to updates based on the requirement, thereby increasing overall efficiency.

### ****49. What are Hooks in React?****

Hooks are used to make use of the state and other features without having to explicitly write a class. Hooks were added to the React version, v16.8. The stateful logic can be extracted from a component easily, alongside testing and reusing it. All of this is done without making any changes to the component hierarchy.