

Introduction to React

React is an efficient, flexible, and open-source **JavaScript framework** library that allows developers to the creation of simple, fast, and scalable web applications. Jordan Walke, a software engineer who was working for Facebook created React. It was first deployed on the news feed of Facebook in 2011 and on Instagram in 2012. Developers from the Javascript background can easily develop web applications with the help of React.

React Hooks will allow you to use the state and other features of React in which requires a class to be written by you. In simple words, we can say that, React Hooks are the functions that will connect React state with the lifecycle features from the function components. React Hooks is among the features that are implemented latest in the version React 16.8.

Scope of React: The selection of the right technology for application or web development is becoming more challenging. React has been considered to be the fastest-growing Javascript framework among all. The tools of Javascript are firming their roots slowly and steadily in the marketplace and the React certification demand is exponentially increasing. React is a clear win for front-end developers as it has a quick learning curve, clean abstraction, and reusable components. Currently, there is no end in sight for React as it keeps evolving.

React Interview Questions for Freshers

1. What is React?

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

The important features of React are:

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

2. What are the advantages of using React?

MVC is generally abbreviated as Model View Controller.

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

3. What are the limitations of React?

The few limitations of React are as given below:

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

4. What is useState() in React?

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

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

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

```
    setOtherStuffs(...);  
    ...  
};
```


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

5. What are keys in React?

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

The “React Way” to render a List

- ✓ Use Array .map
- ✓ Not a for loop
- ✓ Give each item a unique key
- ✓ Avoid using array index as the key



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>
    )
  }
}
```

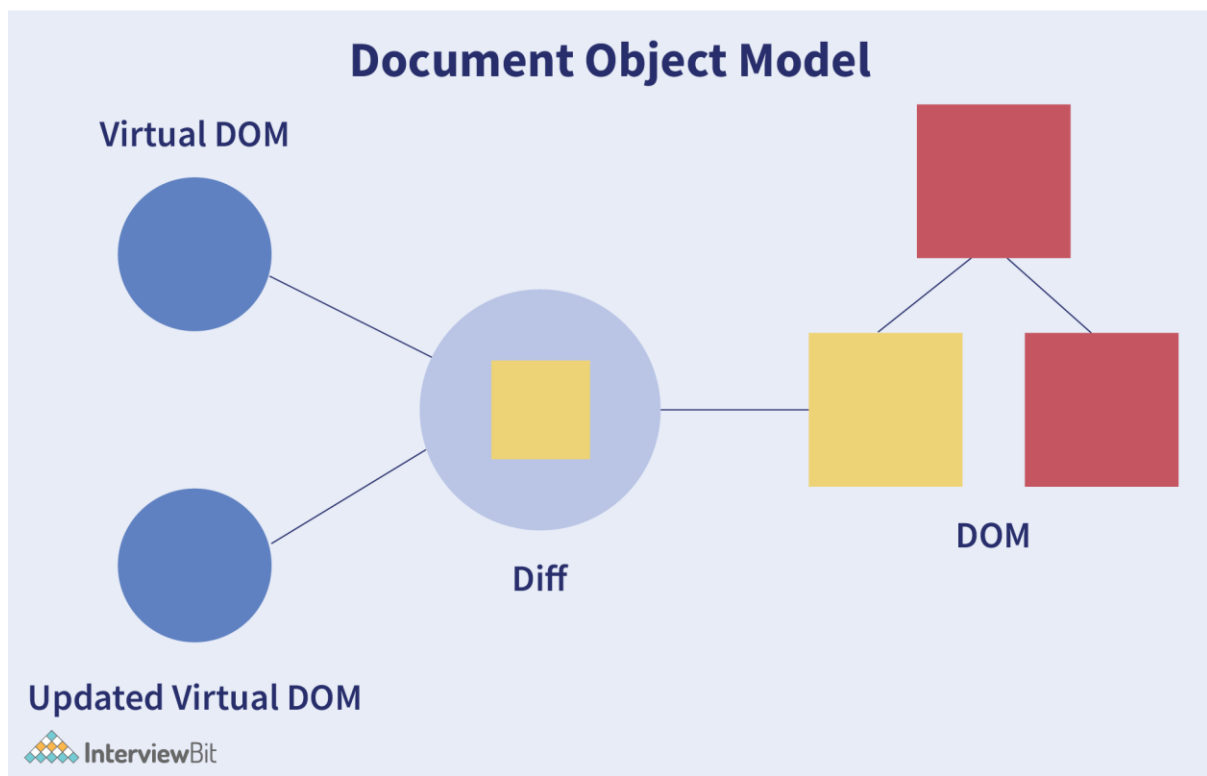
In the code above, we see we are using **this.state** to add the variable `studentsCount` and setting the value to "0".

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.



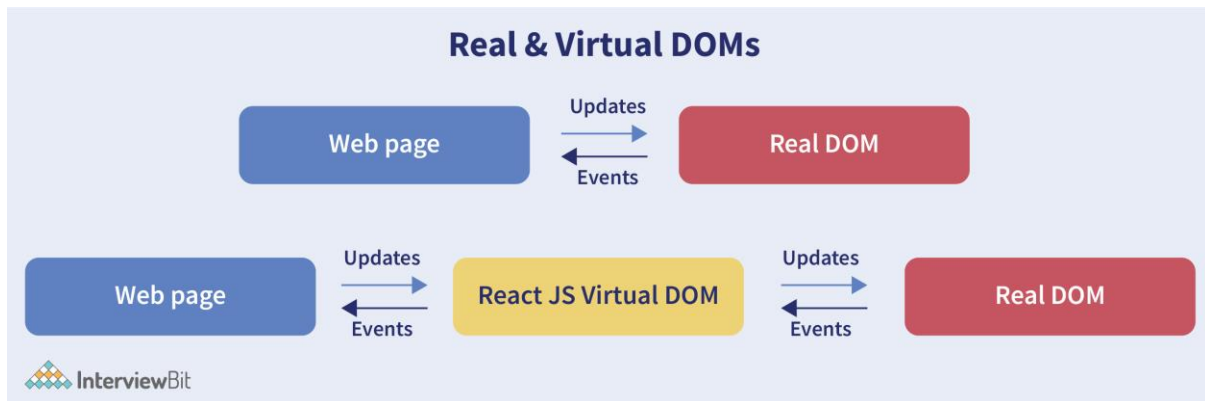
Why was virtual DOM introduced?

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	✗	✓	✓

Feature	Uncontrolled	Controlled	Name attrs
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 user 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 <code>setState()</code> method to modify properties
	Changes to state can be asynchronous

Props	State
	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.

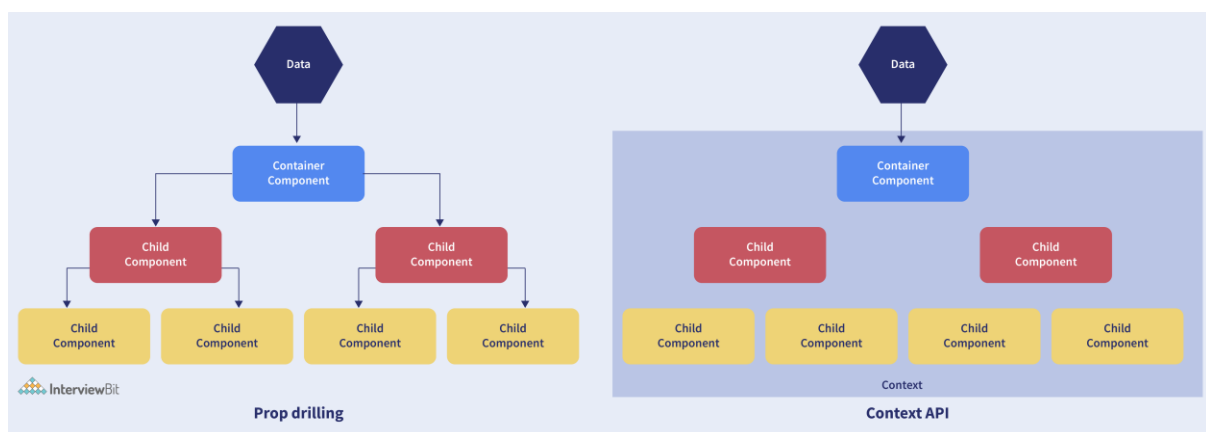
12. Explain about types of side effects in React component.

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.