ReactJS Interview Questions and Answers [2024-25]

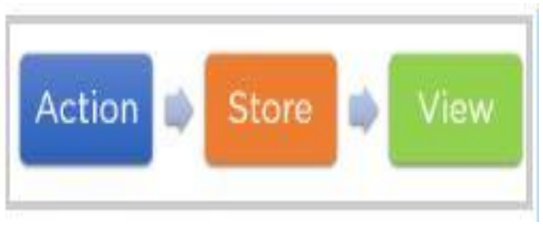
Q1: What is React?

Ans. React is an open-source JavaScript library used to develop complex and interactive web and mobile UI.React follows the component-based approach to create reusable and complex UIs from small and isolated pieces of code called components.
ReactJS is a JavaScript library used to build reusable components for the view layer in MVC architecture. It is highly efficient and uses a virtual DOM to render components. It works on the client side and is written in JSX.
Important Features of React:

- Virtual DOM: React uses a virtual DOM to efficiently update and render components, ensuring fast performance by minimizing direct DOM manipulations.
- Component-Based Architecture: React builds UI using reusable, isolated components, making code more modular, maintainable, and scalable.
- Hooks: React hooks allow functional components to manage state and side effects, making them powerful and more flexible.

Q2: What are the features of React?

	JSX: JSX is a syntax extension to JavaScript. It is used with React to describe the user interface's appearance. Using JSX, we can write HTML structures in the same file

	that contains JavaScript code.
Hypervisor Management software Deployment software Network Cloud server Cloud storage	Components: Components 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.
	Virtual DOM: React keeps a lightweight representation of the real DOM in memory, known as the virtual DOM. When an object's state changes, the virtual DOM changes only that object in the real DOM rather than updating all the objects.
	One-way data-binding: React’s one-way data binding keeps everything modular and fast. A unidirectional data flow means that when

	designing a React app, you often nest child components within parent components.
	High performance: React updates only those components that have changed rather than all the components simultaneously, resulting in faster web applications.

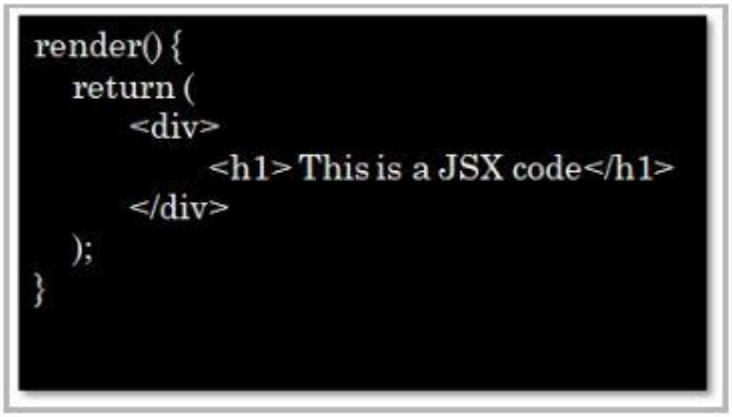
Q3: What is JSX?

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

```
const name = "Learner";

const element = (

  <h1>

    Hello,

    {name}.Welcome to Cybrom.

  </h1>
```

);

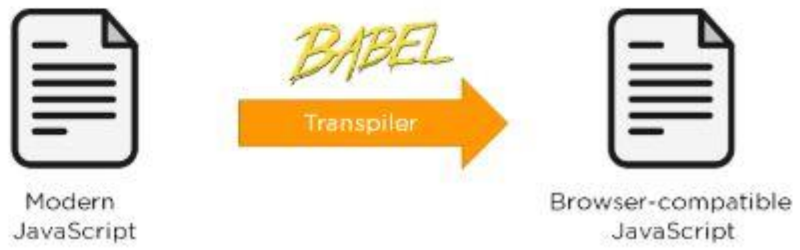
JSX is a syntax extension of JavaScript. It is used with React to describe the user interface's appearance. Using JSX, we can write HTML structures in the same file that contains JavaScript code.



```
render() {
   return (
      <div>
            <h1> This is a JSX code</h1>
      </div>
   );
}
```

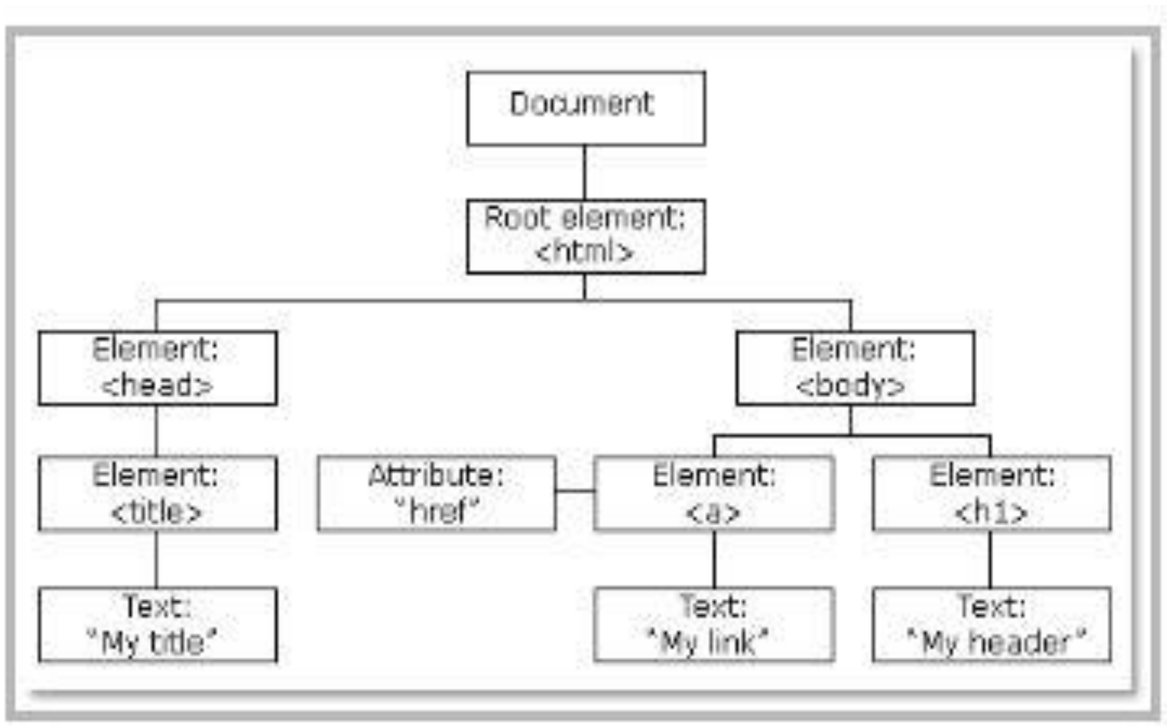
Q4: Can web browsers read JSX directly?

- Web browsers cannot read JSX directly. They are built only to read regular JS objects, and JSX is not a regular JavaScript object.
- A web browser can only read a JSX file if transformed into a regular JavaScript object. We use Babel for this.

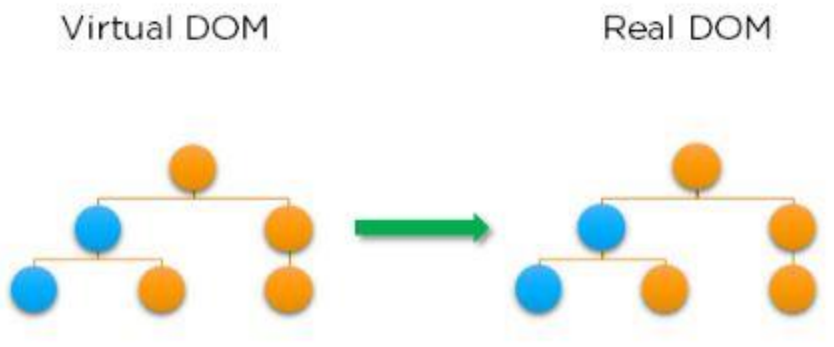


Q5: What is the virtual DOM?

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



React keeps a lightweight representation of the real DOM in memory, known as the virtual DOM. When an object's state 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.



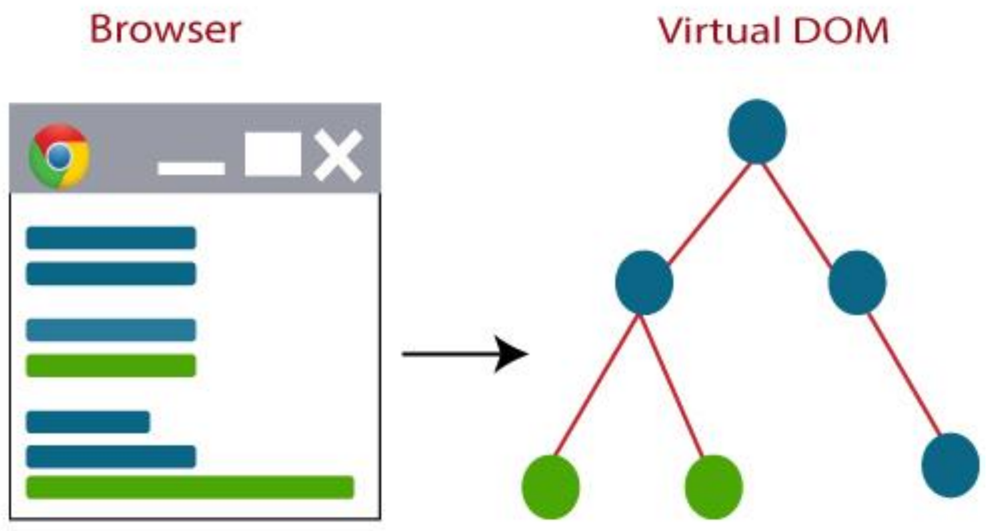
Q6. What do you understand by Virtual DOM?

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.

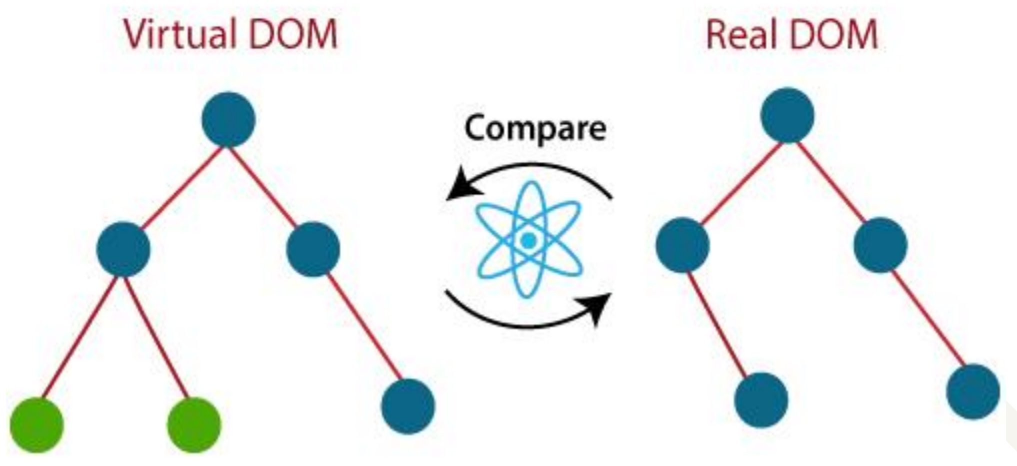
Working of Virtual DOM.

Virtual DOM works in three steps:

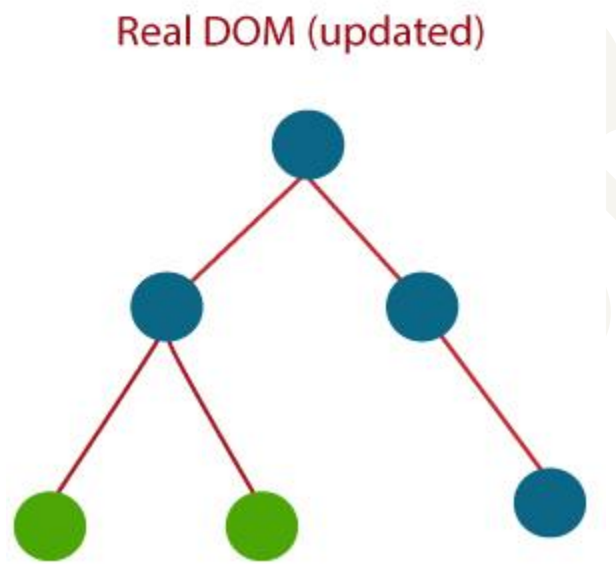
1. Whenever any data changes in the React App, the entire UI is re-rendered in Virtual DOM representation.



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.



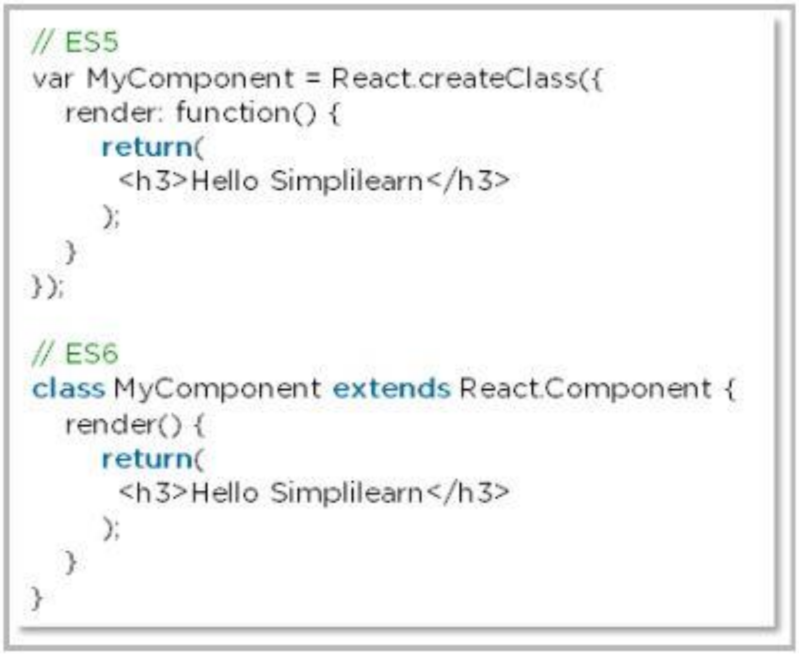
Q7: Why use React instead of other frameworks, like Angular?

- Easy creation of dynamic applications: React makes it easier to create dynamic web applications because it requires less coding and more functionality, whereas code tends to get complex very quickly with JavaScript applications.
- 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.
- Reusable components: Components are the building blocks of any React application, and a single app usually consists of multiple components. These components have logic and controls and can be reused throughout the application, dramatically reducing development time.
- Unidirectional data flow: React follows a unidirectional data flow. When designing a React app, we often nest child components within parent components. Since the data flows in a single direction, debugging errors and knowing where the problem occurs in an application become easier.
- Dedicated tools for easy debugging: Facebook has released a Chrome extension for debugging React applications. This makes the process of debugging React web applications faster and easier.

Q8: What is the difference between the ES6 and ES5 standards?

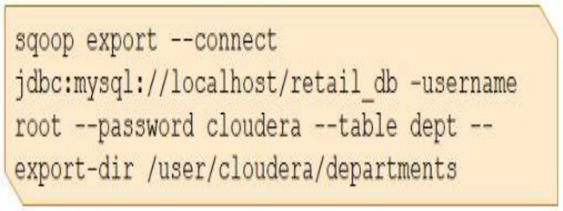
These are the few instances where ES6 syntax has changed from ES5 syntax:

- Components and Function



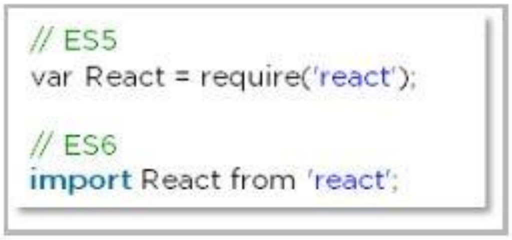
```
// ES5
var MyComponent = React.createClass({
  render: function() {
    return(
      <h3>Hello Simplilearn</h3>
    );
  }
});

// ES6
class MyComponent extends React.Component {
  render() {
    return(
      <h3>Hello Simplilearn</h3>
    );
  }
}
```

- exports vs export



```
sqoop export --connect
jdbc:mysql://localhost/retail_db -username
root --password cloudera --table dept --
export-dir /user/cloudera/departments
```

- require vs import

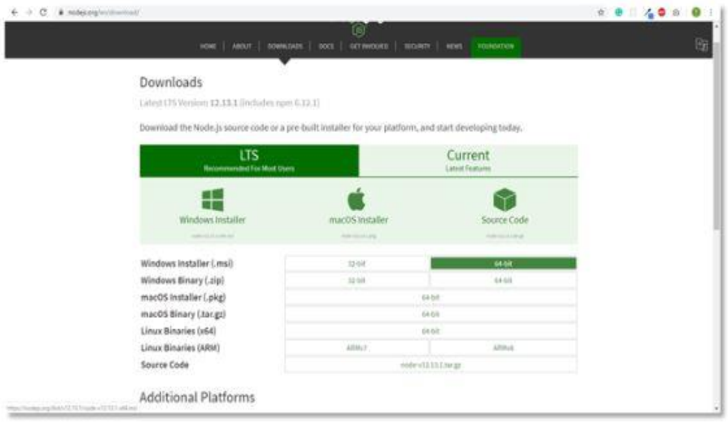


```
// ES5
var React = require('react');

// ES6
import React from 'react';
```

Q9: How do you create a React app?

These are the steps for creating a React app:

- Install NodeJS on the computer. We need NPM to install the React library. Npm is the Node package manager with many JavaScript libraries, including React.



Downloads

Latest LTS Version: 12.13.1 (includes npm 6.12.1)

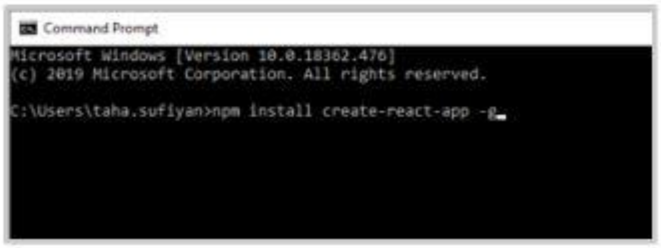
Download the Node.js source code or a pre-built installer for your platform, and start developing today.

LTS Recommended For Most Users	Current Latest Features	
Windows Installer	macOS Installer	Source Code

Windows Installer (.msi)	32-bit	64-bit
Windows Binary (.zip)	32-bit	64-bit
macOS Installer (.pkg)	64-bit	
macOS Binary (.tar.gz)	64-bit	
Linux Binaries (x64)	64-bit	
Linux Binaries (ARM)	ARMv7	ARMv8
Source Code	node-v12.13.1.tar.gz	

Additional Platforms

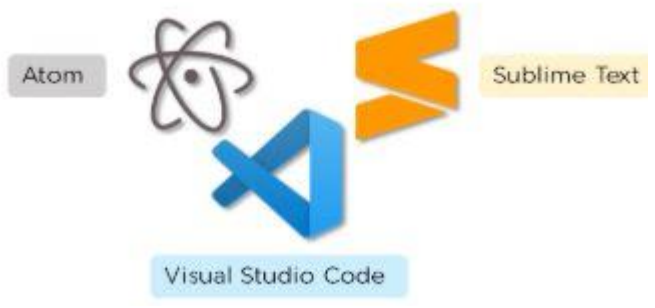
- Install the create-react-app package using the command prompt or terminal.



Command Prompt

```
Microsoft Windows [Version 10.0.18362.476]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\taha.sufiyan>npm install create-react-app -g_
```

- Install a text editor you choose, like VS Code or Sublime Text.



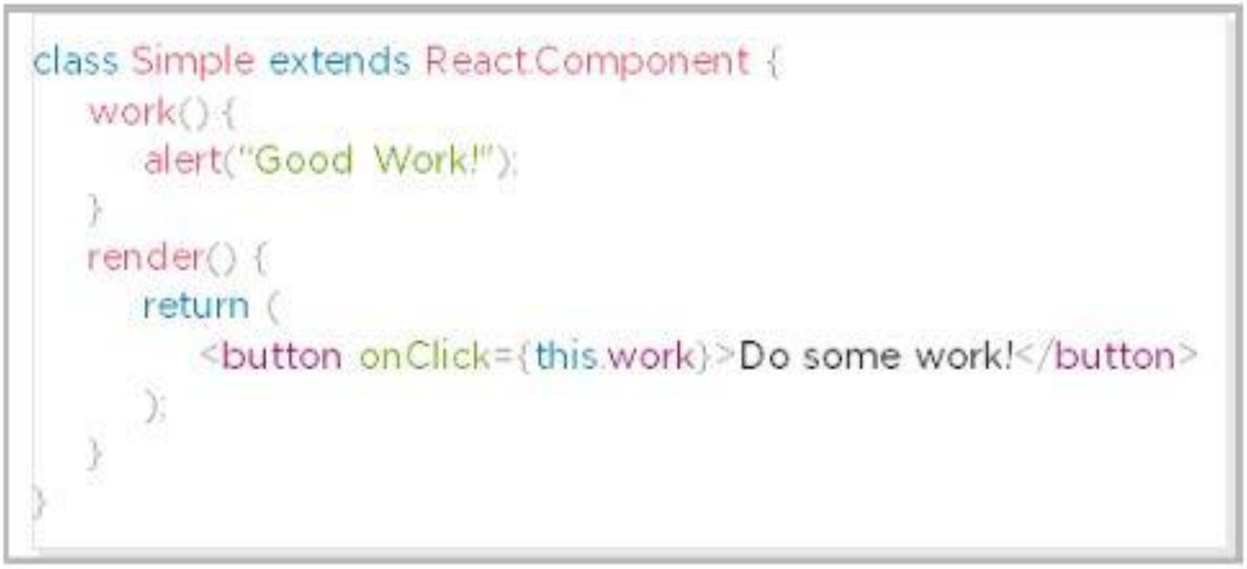
Q10: What is an event in React?

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

- 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.

Q11: How do you create an event in React?

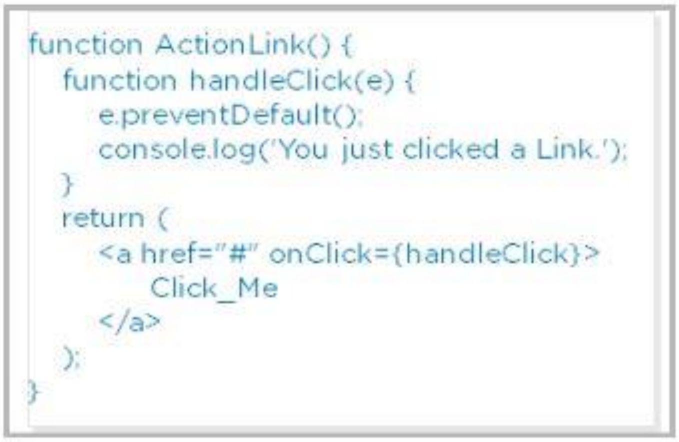
A React event can be created by doing the following:



```
class Simple extends React.Component {
    work() {
        alert("Good Work!");
    }
    render() {
        return (
            <button onClick={this.work}>Do some work!</button>
        );
    }
}
```

Q12: What are synthetic events in React?

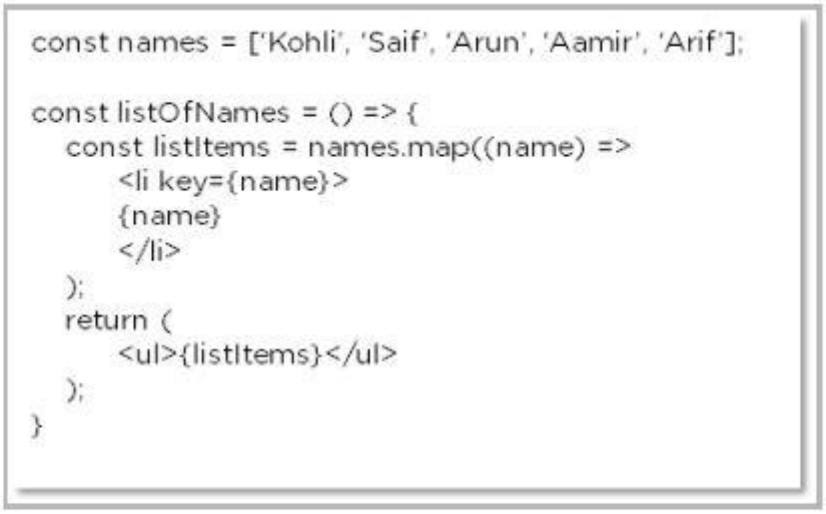
- Synthetic events combine the response of different browsers' native events into one API, ensuring consistency across browsers.
- The application is consistent regardless of the browser in which it is running. Here, preventDefault is a synthetic event.



```
function ActionLink() {
  function handleClick(e) {
    e.preventDefault();
    console.log('You just clicked a Link.');
  }
  return (
    <a href="#" onClick={handleClick}>
       Click_Me
    </a>
  );
}
```

Q13: 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.



```
const names = ['Kohli', 'Saif', 'Arun', 'Aamir', 'Arif'];

const listOfNames = () => {
  const listItems = names.map((name) =>
       <li key={name}>
       {name}
       </li>
  );
  return (
       <ul>{listItems}</ul>
  );
}
```

Q14: Why is there a need to use keys in Lists?

Keys are vital 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

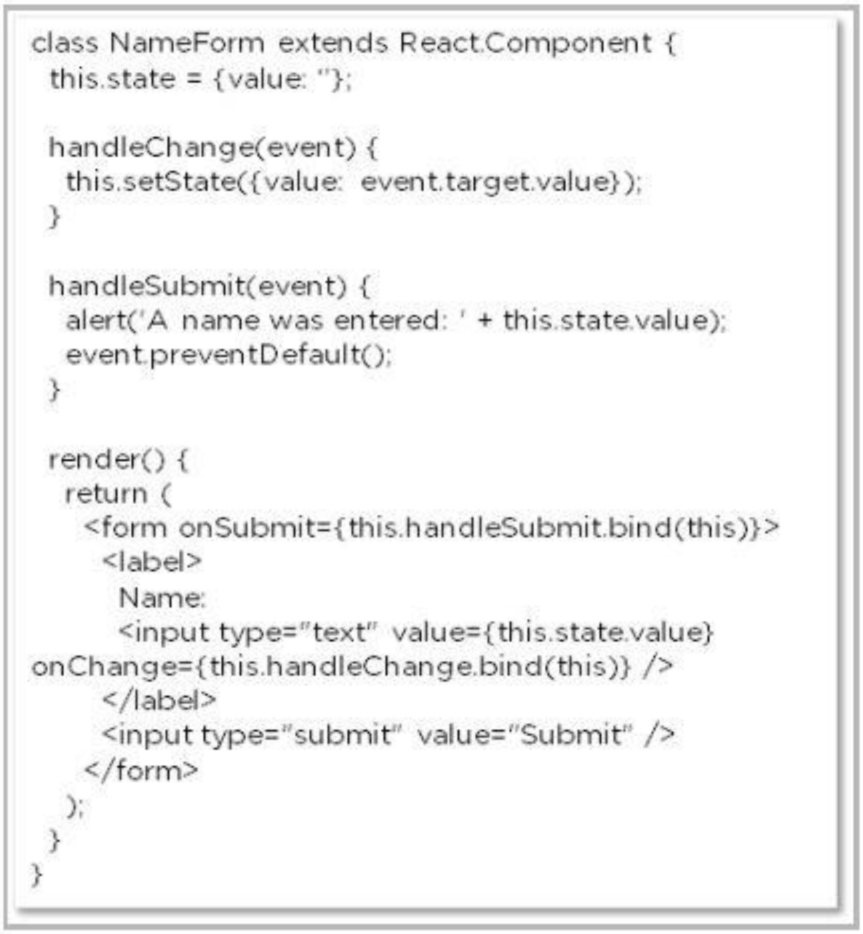
Q15: What are forms in React?

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 contains certain elements, such as text fields, buttons, checkboxes, radio buttons, etc.
- Forms are used for many tasks, such as user authentication, searching, filtering, indexing, etc.

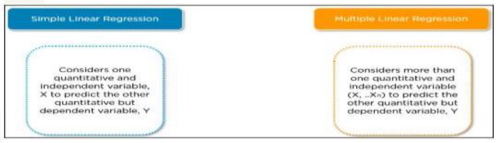
Q16. How do you create forms in React?

We create forms in React by doing the following:



```
class NameForm extends React.Component {
 this.state = {value: ''};

 handleChange(event) {
  this.setState({value:  event.target.value});
 }

 handleSubmit(event) {
  alert('A name was entered: ' + this.state.value);
  event.preventDefault();
 }

 render() {
  return (
   <form onSubmit={this.handleSubmit.bind(this)}>
    <label>
     Name:
     <input type="text"  value={this.state.value}
onChange={this.handleChange.bind(this)} />
    </label>
    <input type="submit"  value="Submit" />
   </form>
  );
 }
}
```

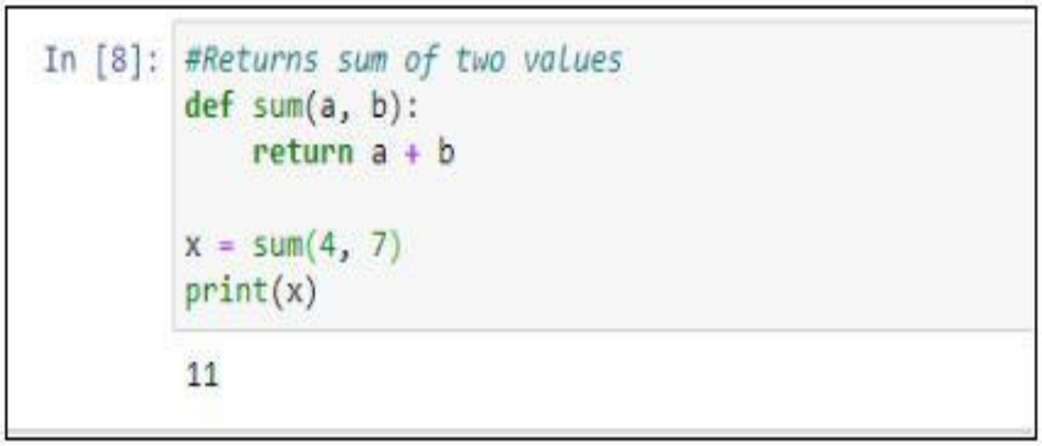
The above code will yield an input field with the label Name and a submit button. It will also alert the user when the submit button is pressed.


Simple Linear Regression
Multiple Linear Regression
Considers one quantitative and independent variable, X to predict the other quantitative but dependent variable, Y
Considers more than one quantitative and independent variable (X, ..Xn) to predict the other quantitative but dependent variable, Y

Q17: How do you write comments in React?

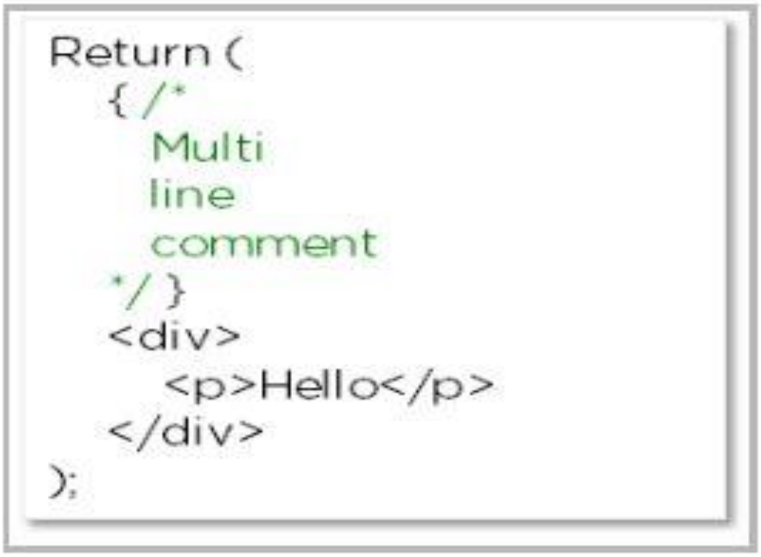
There are two ways in which we can write comments:

- Single-line comments



```
In [8]: #Returns sum of two values
        def sum(a, b):
            return a + b

        x = sum(4, 7)
        print(x)

        11
```

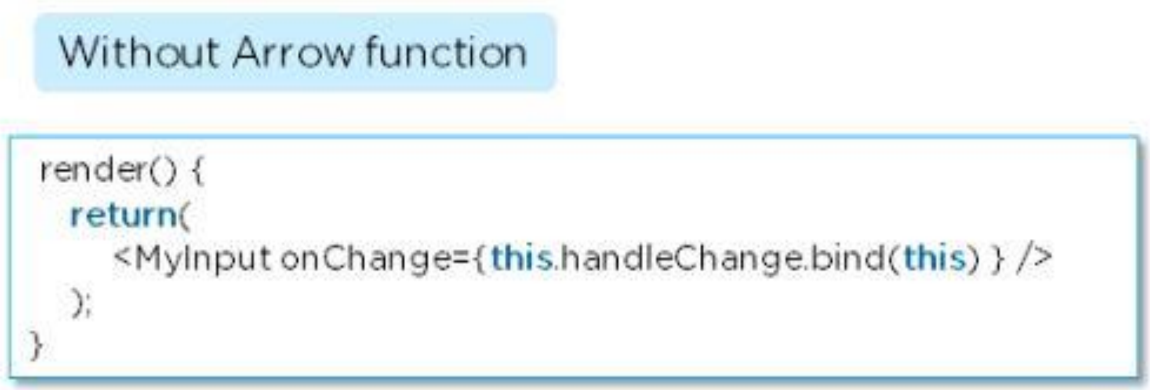
- Multi-line comments



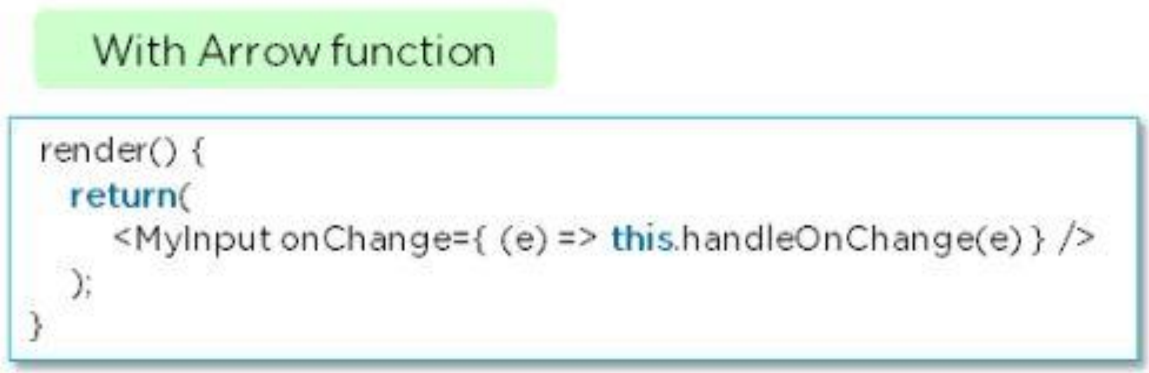
```
Return (
  {/*
    Multi
    line
    comment
  */}
  <div>
    <p>Hello</p>
  </div>
);
```

Q18: 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.



```
render() {
  return(
     <MyInput onChange={this.handleChange.bind(this) } />
  );
}
```



```
render() {
  return(
     <MyInput onChange={ (e) => this.handleOnChange(e) } />
  );
}
```

Q19: How is React different from React Native?

Aspect	**React**	**React Native**
Release	2013	2015
Platform	Web	Mobile – Android, iOS

HTML	Yes	No
CSS	Yes	No
Prerequisites	JavaScript, HTML, CSS	React.js

Q20: How is React different from Angular?

Aspect	**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

Q21: What are React Hooks?

React Hooks are functions introduced in React 16.8 that allow you to use state and lifecycle features in functional components. Hooks like useState, useEffect, and useContext make it easier to manage component logic without using classes, improving readability and reusability.

Q22: What is useState, and how does it work?

useState is a hook that allows you to add state to functional components. It takes an initial state value and returns an array with two elements: the current state and a function to update it. useState re-renders the component when the state is updated.

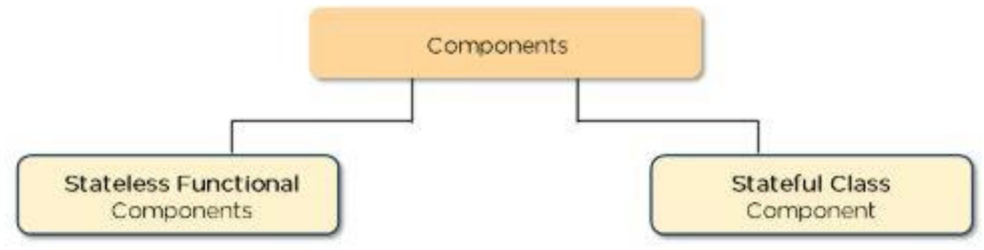
Q23: What is Memorization in React?

Memorization is an optimization technique that saves the result of expensive function calls and reuses it when the same inputs occur. React.memo and useMemo prevent unnecessary re-renders and calculations by caching results based on dependencies.

Q24: 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:

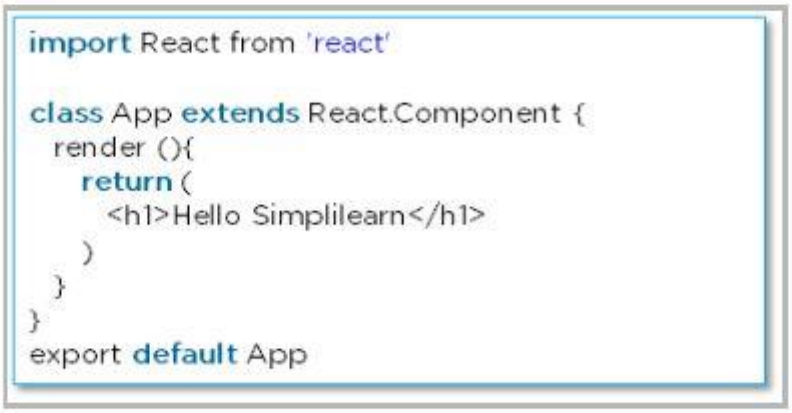


- **Functional Components**: These components have no state and only contain render methods. Therefore, they are also called stateless components. They may derive data from other components as props (properties).

- **Class Components**: These components can hold and manage their 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.

Q27. What is the use of render() in React?

render() is a lifecycle method in class components that returns the JSX to be displayed on the screen. It's called every time the component's state or props change, updating the UI accordingly.



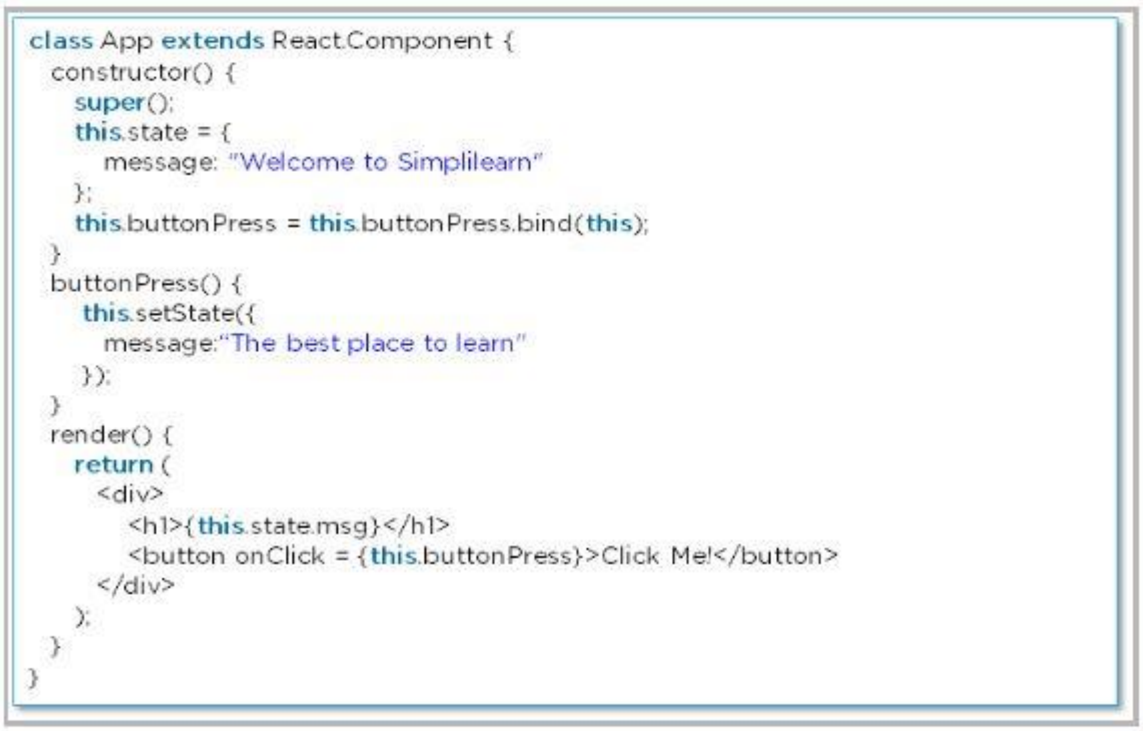
```
import React from 'react'

class App extends React.Component {
  render (){
    return (
      <h1>Hello Simplilearn</h1>
    )
  }
}
export default App
```

Q25. What is a state in React?

- The state is a built-in React object that contains data or information about the component. A component's state can change over time, and whenever it changes, the component re-renders.
- The state change can occur as a response to user action or system-generated events. It determines the component's behavior and rendering method.

Q26. How do you update the state of a component?

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



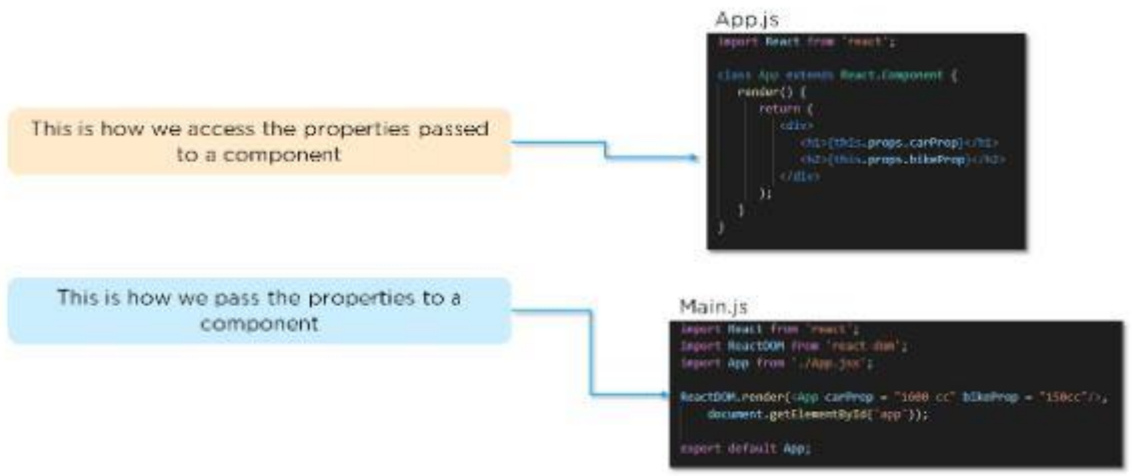
```
class App extends React.Component {
  constructor() {
    super();
    this.state = {
      message: "Welcome to Simplilearn"
    };
    this.buttonPress = this.buttonPress.bind(this);
  }
  buttonPress() {
     this.setState({
      message:"The best place to learn"
     });
  }
  render() {
    return (
      <div>
         <h1>{this.state.msg}</h1>
         <button onClick = {this.buttonPress}>Click Me!</button>
      </div>
    );
  }
}
```

Q27. What are props in React?

Props (short for properties) are inputs passed from parent to child components, allowing data and configuration to flow into the component. Props are read-only and cannot be modified within the child component.

Q28. How do you pass props between components?

Props are passed from a parent to a child component by including them as attributes in the child component's JSX tag. The child component then accesses the props through props.propertyName.


This is how we access the properties passed to a component
App.js
This is how we pass the properties to a component
Main.js

```
import React from 'react';

class App extends React.Component {
   render() {
      return (
         <div>
            <h1>{this.props.carProp}</h1>
            <h2>{this.props.bikeProp}</h2>
         </div>
      );
   }
}
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

ReactDOM.render(<App carProp = "1000 cc" bikeProp = "150cc"/>,
   document.getElementById('app'));

export default App;
```

Q29. What are the differences between state and props?

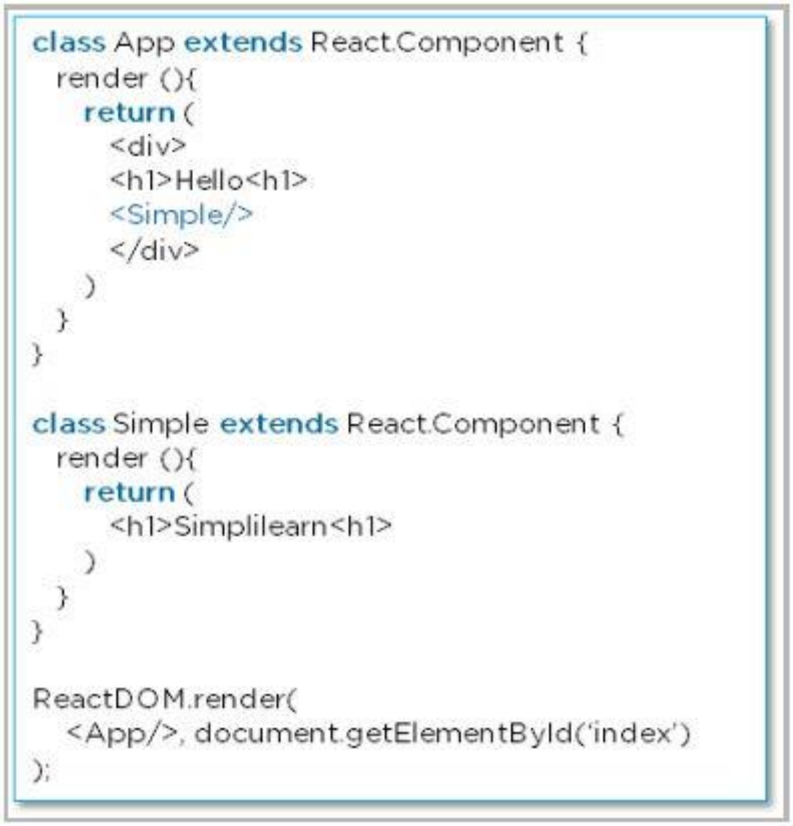
State is a local, mutable data structure managed within a component, whereas props are immutable data passed from a parent to a child component. State is used for dynamic data within the component, while props control component behavior from outside.

Q30. What is a higher-order component in React?

A higher-order component (HOC) is a function that takes a component and returns an enhanced component, adding functionality or modifying behavior. HOCs are used for cross-cutting concerns like authentication, logging, or data fetching.

Q31. How can you embed two or more components into one?

You can embed components by nesting them within a parent component or using composition to render multiple components together. You can also pass components as children or props for dynamic embedding.

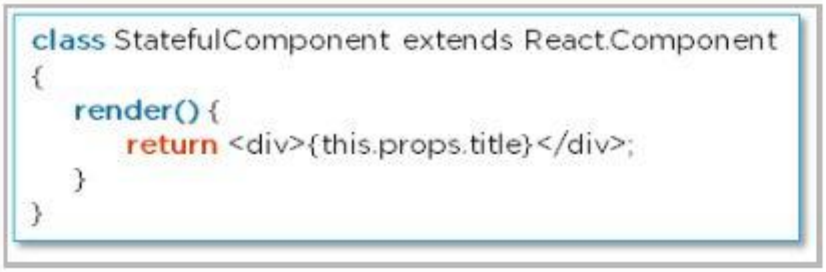


```
class App extends React.Component {
  render (){
    return (
      <div>
      <h1>Hello<h1>
      <Simple/>
      </div>
    )
  }
}

class Simple extends React.Component {
  render (){
    return (
      <h1>Simplilearn<h1>
    )
  }
}

ReactDOM.render(
   <App/>, document.getElementById('index')
);
```

Q32. What are the differences between class and functional components?

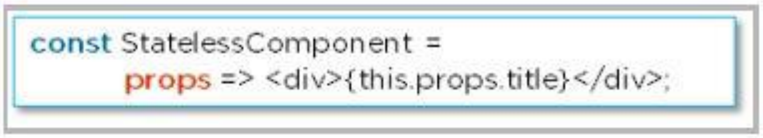
Class components use ES6 classes and provide access to lifecycle methods and state via this.state. Functional components are simpler, using functions to define components and accessing state and lifecycle features through hooks like useState and useEffect.

- Class components example:



```
class StatefulComponent extends React.Component
{
   render() {
      return <div>{this.props.title}</div>;
   }
}
```

- Functional components example:



```
const StatelessComponent =
        props => <div>{this.props.title}</div>;
```

difference between functional and class component in React:

Here we have difference table of functional and class component in React

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

Q33. Explain the lifecycle methods of components.

- getInitialState(): 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 permanently unmounted.

Q34. What is React Router?

React Router library adds navigation and routing to single-page React applications. It lets you define routes and navigate between different components without reloading the page.

Q35. Why do we need to React Router?

React Router allows seamless navigation in a single-page application (SPA) by managing URL changes without refreshing the page, improving user experience and enabling multi-view applications.

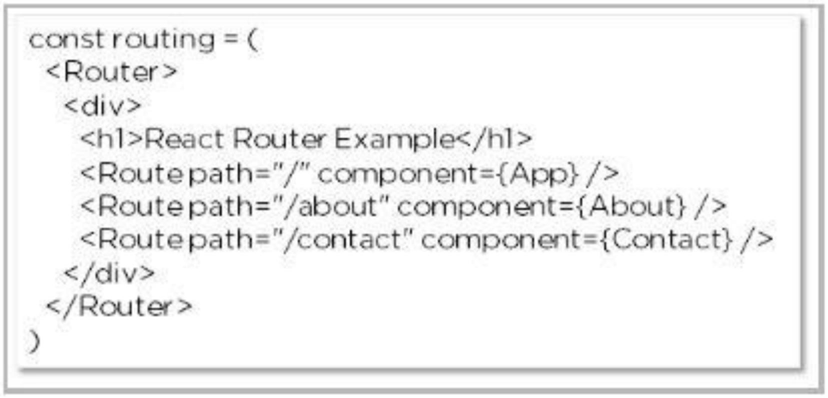
Q36. How is React routing different from conventional routing?

Traditional routing reloads the page on each navigation, while React routing dynamically loads components within the same page, creating a faster, SPA-like experience.

Q37. How do you implement React routing?

To implement React routing, install react-router-dom, define routes using Route and Switch components, and use Link or NavLink to navigate between them without reloading the page.

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

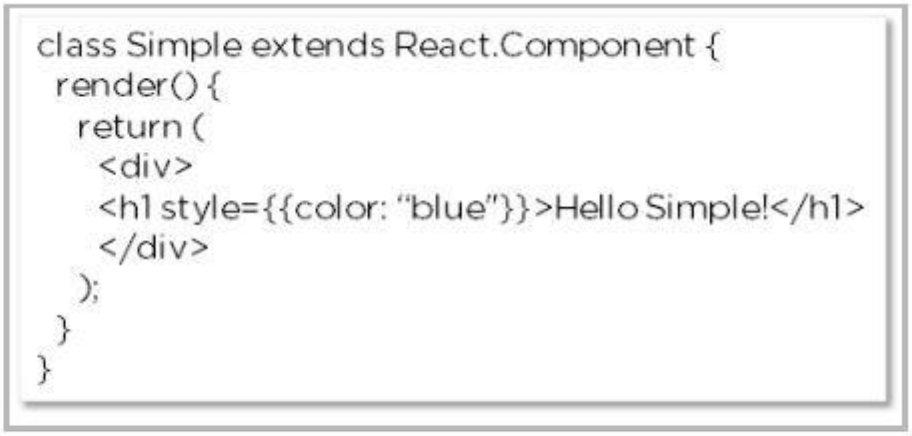


```
const routing = (
 <Router>
  <div>
   <h1>React Router Example</h1>
   <Route path="/" component={App} />
   <Route path="/about" component={About} />
   <Route path="/contact" component={Contact} />
  </div>
 </Router>
)
```

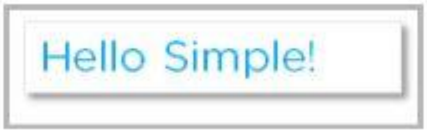
Q38. How do you style React components?

React components can be styled with CSS, inline styles, styled components, or CSS-in-JS libraries. You can also use frameworks like Material-UI or Bootstrap for pre-built styles.

- Inline Styling

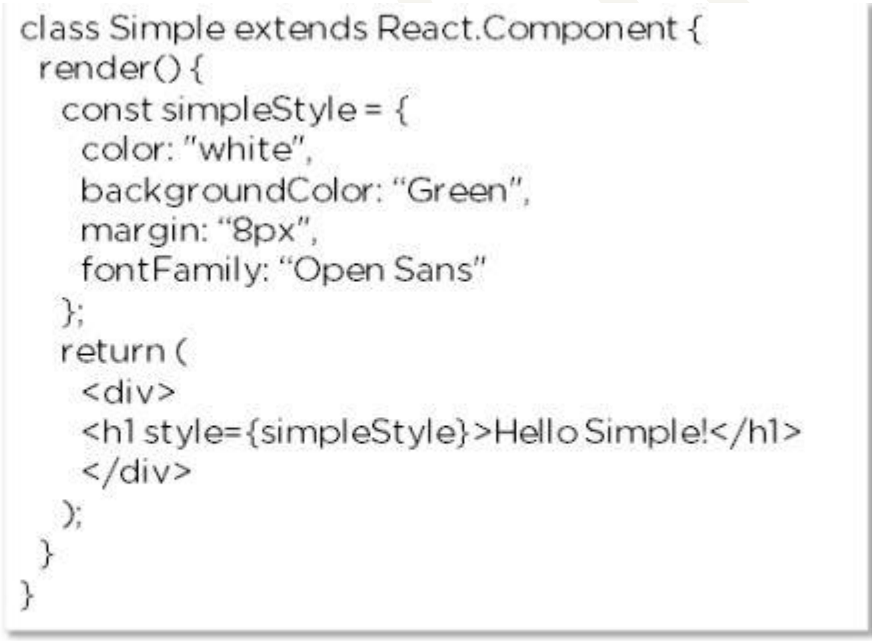


```
class Simple extends React.Component {
  render() {
    return (
      <div>
      <h1 style={{color: "blue"}}>Hello Simple!</h1>
      </div>
    );
  }
}
```

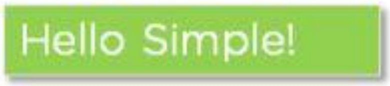


Hello Simple!

- JavaScript Object

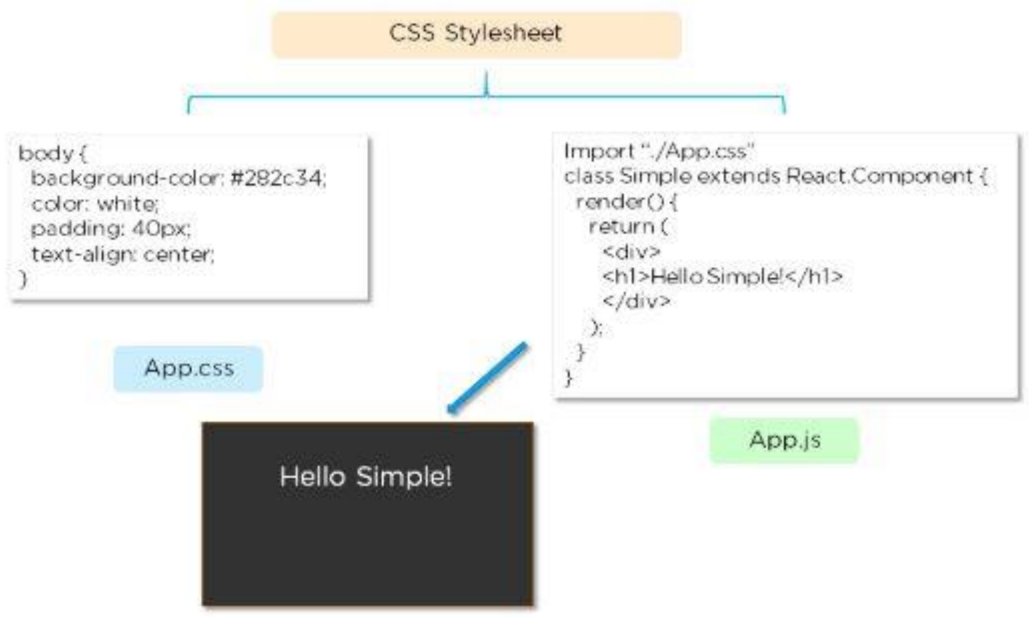


```
class Simple extends React.Component {
  render() {
    const simpleStyle = {
      color: "white",
      backgroundColor: "Green",
      margin: "8px",
      fontFamily: "Open Sans"
    };
    return (
      <div>
      <h1 style={simpleStyle}>Hello Simple!</h1>
      </div>
    );
  }
}
```



Hello Simple!

- CSS Stylesheet



```
body {
  background-color: #282c34;
  color: white;
  padding: 40px;
  text-align: center;
}
```

```
Import "./App.css"
class Simple extends React.Component {
  render() {
    return (
      <div>
      <h1>Hello Simple!</h1>
      </div>
    );
  }
}
```

Q39. Explain the MVC architecture.

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

Q40. Explain the building blocks of React.

The five main building blocks of React are:

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

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

Q41. Explain props and state in React with differences

Props are used to pass data from one component to another. The state is local data storage that is local to the component only and cannot be passed to other components.
Here is the difference table of props and state In react

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

Q42. What are components and their type in React?

A Component is one of the core building blocks of React. In other words, we can say that every application you will develop in React will

be made up of pieces called components. Components make the task of building UIs much easier.
In React, we mainly have two types of components:

- **Functional Components**: Functional components are simply javascript functions. We can create a functional component in React by writing a javascript function.
- **Class Components**: The class components are a little more complex than the functional components. The functional components are not aware of the other components in your program whereas the class components can work with each other. We can pass data from one class component to another class component.

Q43. How do browsers read JSX?

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

Q44. Explain the steps to create a react application and print Hello World?

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

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

Navigate to the folder.

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

This is the first code of ReactJS Hello World!
Javascript

```
import React from "react";

import "./App.css";

function App() {

   return <div className="App">Hello World !</div>;

}

export default App;
```

Type the following command to run the application

```
npm start
```

Q45. How to create an event in React?

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

Javascript

```
function Component() {

   doSomething(e);

   {

      e.preventDefault();

      // Some more response to the event

   }
```

```
  return <button onEvent={doSomething}></button>;
}
```

Q46. Explain the creation of a List in react?

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

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

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

const updatedNums = numbers.map((number) => {
  return <li>{number}</li>;
});

ReactDOM.render(<ul>{updatedNums}</ul>, document.getElementById("root"));
```

Q47. What is a key in React?

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

Q48. Explain the use of render method in React?

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

Q49. Explain one way data binding in React?

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

Q50. What is conditional rendering in React?

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

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

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

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

Q51. What is react router?

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

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

Q52. Explain the components of a react-router

The main components of a react-router are:

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

Q53. Explain the lifecycle methods of components

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

- Initialization: This is the stage where the component is constructed with the given Props and default state. This is done in the constructor of a Component Class.
- Mounting: Mounting is the stage of rendering the JSX returned by the render method itself.
- Updating: Updating is the stage when the state of a component is updated and the application is repainted.

- Unmounting: As the name suggests Unmounting is the final step of the component lifecycle where the component is removed from the page.

Q54. What is the use of ref in React?

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

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

Q55. What are hooks in React?

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

Q56. Explain the useState hook in React?

The most used hook in React is the useState() hook. Using this hook we can declare a state variable inside a function but only one state variable can be declared using a single useState() hook. Whenever the useState() hook is used, the value of the state variable is changed and the new variable is stored in a new cell in the stack.
When you use useState(), you declare a state variable and a function to update that state. React then manages this state internally and triggers a re-render of the component when the state changes. This allows functional components to maintain and update their internal state over time.
We have to import this hook in React using the following syntax

```
import {useState} from 'react'
```

Q57. Explain the useEffect hook in react?

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

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

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

Q58. What is React Fragments?

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

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

Q59. What is prop drilling and its disadvantages?

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

Q68. What is custom hooks in React?

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

Q60. What is the difference between useref and createRef in React ?

Here is the difference table of useref and createRef in React

useRef	**createRef**
It is a hook.	It is a function.
It uses the same ref throughout.	It creates a new ref every time.
It saves its value between re-renders in a functional component.	It creates a new ref for every re-render.
It returns a mutable ref object.	It returns a read-only ref object.
The refs created using the useRef can persist for the entire component lifetime.	The refs created using the createRef can be referenced throughout the component.
It is used in functional components.	It is used in class components.

Q61. What is context API?

Context API is used to pass global variables anywhere in the code. It helps when there is a need for sharing state between a lot of nested components. It is light in weight and easier to use, to create a context just need to call React.createContext(). It eliminates the need to install other dependencies or third-party libraries like redux for state management. It has two properties Provider and Consumer.

Q62. Explain provider and consumer in ContextAPI?

A provider is used to provide context to the whole application whereas a consumer consume the context provided by nearest provider. In other words The Provider acts as a parent it passes the state to its children whereas the Consumer uses the state that has been passed.

Q63. Explain CORS in React?

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

- axios
- fetch

Q64. What is axios and how to use it in React?

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

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

To install aixos package in react use the following command.

```
npm i axios
```

Q65. What is React-Material UI?

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

Q66. What is flux architecture in redux?

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

Q67. What are the differences between controlled and uncontrolled components?

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

- 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.

Q68. 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.

Q69. 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>
```

Q70. 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.

Q71. What is the difference between useState and useReducer?

Answer:

Feature	**useState**	**useReducer**
Usage	Manages local state in a simple form.	Manages more complex state logic.
Parameters	Initial state.	Reducer function and initial state.
Returns	State and updater function.	State and dispatch function.
Best For	Simple state changes.	Complex state changes or related updates.

Q72. What is the purpose of the dependency array in useEffect?

Answer:
The dependency array determines when the effect runs. If values in the array change, the effect re-runs.

- If omitted, the effect runs after every render.
- An empty array ([]) ensures it runs only once (on mount).
- Including specific dependencies makes it run only when those values change.

Q73. What is useMemo, and when would you use it?

Answer:
useMemo is used to optimize performance by memoizing the result of an expensive calculation so it is only recomputed when its dependencies change.

Example:

javascript

```
const memoizedValue = useMemo(() => expensiveFunction(input), [input]);
```

Use it when you want to avoid recalculating a value unnecessarily during renders.

Q74. What is the difference between useMemo and useCallback?

Answer:

Feature	useMemo	useCallback
Purpose	Memoizes a computed value.	Memoizes a function.
Returns	The computed value.	The memoized callback function.
Usage	Optimizing expensive calculations.	Preventing unnecessary re-creation of functions passed as props.

Q75. How does useRef differ from useState?

Answer:

- useRef:
  - Returns a mutable object with a .current property.
  - Changes to .current do not trigger re-renders.
  - Commonly used to reference DOM elements or persist values across renders without causing re-renders.
- useState:
  - Manages state and triggers re-renders when updated.

Q76. What are the limitations of hooks?

Answer:

1. Cannot be used in class components.
2. Must follow the rules of hooks (top-level, only in React functions).

3. Dependency arrays in useEffect and useMemo can lead to bugs if not properly specified.
4. Overusing hooks can lead to overly complex and unreadable code.

Q77. Explain useLayoutEffect and how it differs from useEffect.

Answer:

- useEffect: Runs after the DOM has been painted. Ideal for non-blocking operations like API calls.
- useLayoutEffect: Runs synchronously after all DOM mutations but before the browser paints. Used for measuring DOM elements or applying styles that must not flicker.

Example:

```
useLayoutEffect(() => {
  const width = ref.current.offsetWidth;
  console.log(width);
}, []);
```

Q78. 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.

Q79. 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]).

Q80. 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:

Q81. 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	Classes
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.

Q82. 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.

Q83. 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).

Q84. What are Pure Components in React?

Pure components are those components that render the same output for the same state and props. They are the simplest and fastest components that can be written. They can replace any component which only has a render(). Thus, a function is said to be pure if:

- Its return value is determined only by its input values
- Its return value is always the same for the same input values

Q85. Explain the different phases of the ReactJS component lifecycle?

There are four different phases of the ReactJS components lifecycle: Initialization, Mounting, Updating, and Unmounting.

- Initialization: This is the first stage in the ReactJS lifecycle. In this stage, the component is constructed with the provided properties and a default state.
- Mounting: It is the phase that involves putting elements into the DOM. It includes componentWillMount, and componentDidMount lifecycle methods.
- Updating: This stage updates the stage of a component and repaints the application. This phase includes, shouldComponentUpdate, componentWillUpdate lifecycle methods.
- Unmounting: It is the final stage, in which the component is removed from the page.

Q86. What is the difference between stateful and stateless components?

The difference between Stateful and Stateless Components are:

Stateful (Class) Component	**Stateless (Functional) Component**
1. Stateful Component stores information about the component's state change in memory.	1. It calculates the internal state of the components.
2. It has complex UI Logic.	2. It accepts properties(props) in function and return HTML (JSX).
3. It contains the data related to past, current, and possible future changes in state.	3. It does not contain the data related to the past, current, and possible future state changes.
4. It must have the render() method returning HTML.	4. There is no render method used in the Stateless component.
5. Have the authority to change the state.	5. Do not have the authority to change the state.

Q87. What do you know about the controlled and uncontrolled components?

The differences between controlled and uncontrolled components are:

Controlled Components	**Uncontrolled Components**
They do not maintain their internal state.	They maintain their internal state.

Data is controlled by the parent component.	Data is controlled by the DOM itself.
They accept current value as a prop.	These components use ref to get their current values.
They allow validation control.	Uncontrolled components do not allow validation control.
Controlled components have good control over the form elements and data.	They have limited control over the form elements and data.

Q88. What is the purpose of useCallback Hooks?

The purpose of useCallback Hooks is used to memoize functions, and prevent unnecessary re-rendering of child components that rely on those components. The useCallback function in React is mainly used to keep a reference to a function constant across multiple re-renders. This feature becomes useful when you want to prevent the unnecessary re-creation of functions, especially when you need to pass them as dependencies to other hooks such as useMemo or useEffect.

Q89. Explain the difference between useMemo and useCallback?

Here's a difference of useMemo and useCallback int tabular form:

useMemo	useCallback

useMemo	useCallback
Memoizes a value (result of a function)	Memoizes a function
Memoized value	Memoized function reference
When you need to memoize a calculated value	When you need to memoize a function
Recalculates when any dependency changes	Recreates the function only when any dependency changes
Example: Memoizing the result of expensive computations	Example: Memoizing event handler functions to prevent re-renders

90. What is the useMemo hook and how is it used?

The useMemo hook works to memoize functions within React. Memoization is a technique for caching the result of a function call so that it does not have to be recalculated every time the function is used. This can increase the performance of your React applications.

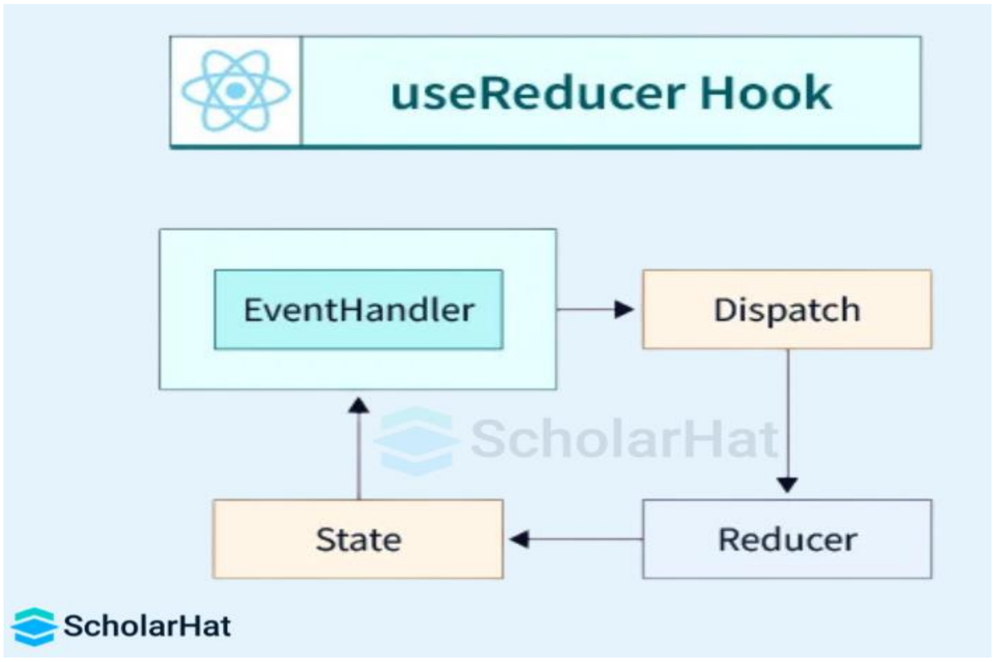
Q91. How do you handle errors in React functional components with hooks?

With hooks, there are numerous ways to handle problems in React functional components:

- The try/catch block is used.
- Making use of an error boundary component.
- Making use of a third-party library, such as react-error-boundary.

Q92. How do you use the useReducer hook for complex state management?

The useReducer hook in React is an effective tool for managing complex states in a more organized as well as predictable manner. It's especially handy when dealing with states with several sub-values or when the future state is dependent on the prior one.



Q93. How do you fetch data from an API in React with hooks?

Hooks can be used to retrieve data from an API in many ways in React:

- Making use of the fetch API.
- Using an HTTP client library like Axios or SWR.

Q94. What is the useSWR hook and how is it used?

- The useSWR hook is a React hook that provides a solution for data fetching and caching in React apps. It is a wrapper for the axios library that adds capabilities like caching and automatic revalidation.

Q95. What is the useRef hook and how is it used?

- The useRef hook creates and manages mutable references to values. This can be useful for storing non-reactive data, such as a DOM element or a subscription.

……

Redux/ Redux-Toolkit Important Questions

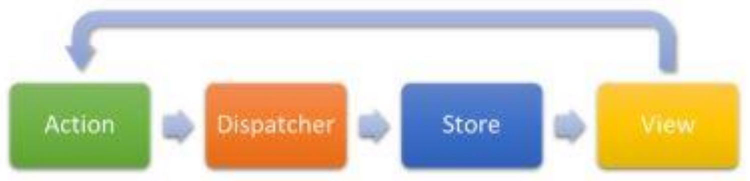
Q1. What is Redux?

Redux is a state management library for JavaScript applications, often used with React. It centralizes the application's state in a store, allowing components to access and update state through actions and reducers.

Redux is a popular open-source state management library for JavaScript applications. It provides a way to centralize the state of an application in a single store, making it easier to debug, test, and reason about the state changes in the application.

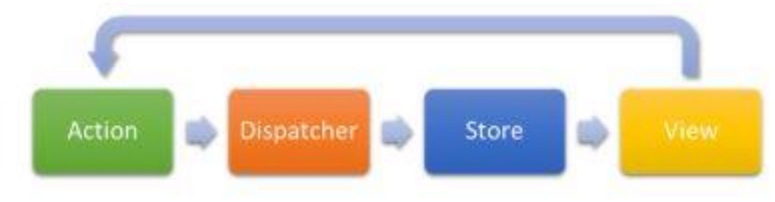
Q2. What are the components of Redux?

Redux has three main components: the store (which holds the application state), actions (which describe what to do), and reducers (which define how the state changes in response to actions).

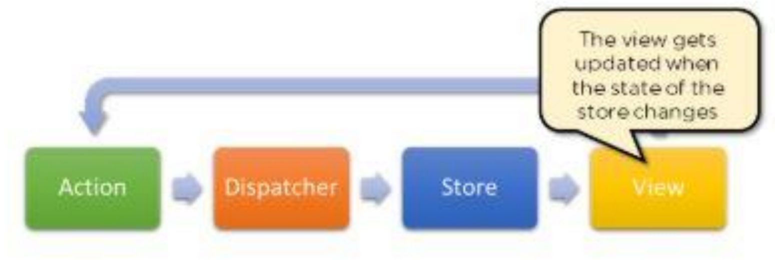


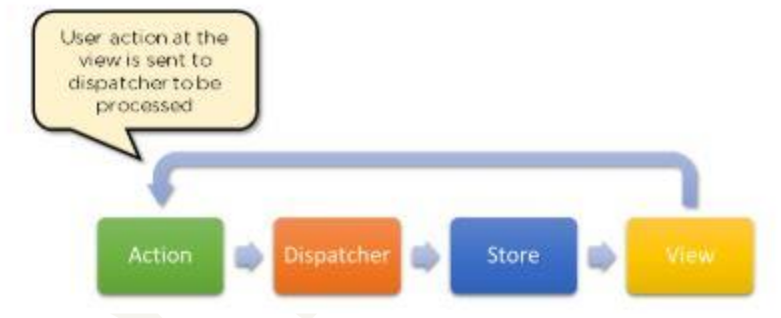
Q3. What is the Flux?

- Flux is an application architecture pattern for managing data flow. Actions are dispatched to stores, which then update views. It's commonly used with React to ensure a unidirectional data flow.

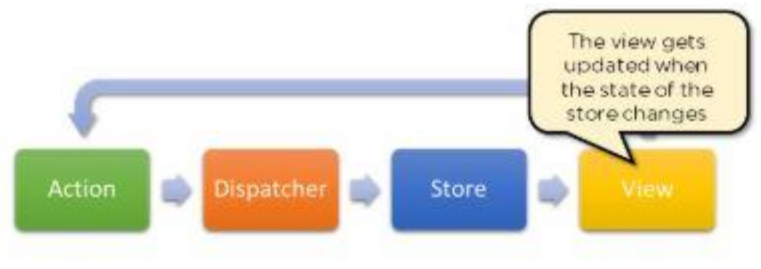


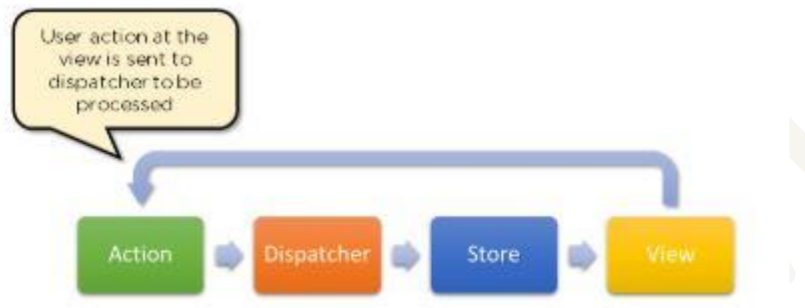
- There is a single source of data (the store), and the only way to update it is to trigger certain actions. The actions call the dispatcher, and the store is triggered and updated with its data.




User action at the view is sent to dispatcher to be processed
Action
Dispatcher
Store
View

- When a dispatch has been triggered and the store updates, it emits a change event that allows the views to re-render accordingly.




User action at the view is sent to dispatcher to be processed
Action
Dispatcher
Store
View

Q4. How is Redux different from Flux?

Both Redux and Flux are state management architectures, but Redux has a single store, while Flux uses multiple stores. Redux also enforces immutability and provides middleware like Redux Thunk for asynchronous actions.

Q5. What is the difference between Context API and Redux?

Context API is a simpler way to share state across components without prop drilling and is suitable for less complex applications. On the other hand, Redux is more robust, with middleware and a global store, making it better for complex, scalable state management.

Q6. What are the problems that Redux solves?

Redux solves many problems faced while building complex front-end web applications.

- **State Management**: Redux provides a centralized way of managing the application state.
- **Predictable State Changes**: Redux follows unidirectional data flow, which makes it easier to understand how data changes over time.
- **Debugging**: Redux includes tools like Redux DevTools, which allow you to inspect state changes through the application's state history.
- **Handling Asynchronous Logic**: While Redux is synchronous, it integrates with middleware libraries like Redux Thunk or Redux Saga, which allow developers to handle asynchronous logic.
- **Time Travel and Undo/Redo**: Redux keeps a history of state changes, so it's possible to implement features like time travel debugging, where you can move back and forth through the application's state history to understand how the state evolved.

Q7. What are the advantages of Redux in React?

- **Centralized state management system:** Redux state is stored globally in the store. All the components of the entire application can easily access the data directly.
- **Performance Optimizations:** Redux store helps in improving the performance by skipping unnecessary re-renders and ensuring that a given component re-renders only when its data has actually changed.
- **Pure reducer functions:**Redux depends on pure functions which takes a given state (object) and passes it to each reducer in a loop. In case of any data changes, a new object is returned from the reducer (re-rendering takes place).
- **Storing long-term data:** Since data stored in redux persists until page refresh, it is widely used to store long-term data.
- **Great supportive community** Since redux has a large community of users, it becomes easier to learn about best practices, get help when stuck, reuse your knowledge across different applications.

Q8. Explain the core principles of Redux.

Three principles that Redux follows are:

- **Redux is a Single Source of Truth**
- **The State is Reada only State**
- **The Modifications are Done with Pure Functions**

Q9. What is the difference between Redux and Context API.

Features	Redux	Context API
Middleware	Middlewares present.	Middlewares absent.
State management approach	Centralized	Decentralized
Data Flow	Unidirectional flow of data.	Bidirectional flow of data.
API	Actions, reducers, middleware	Context.Provider, Context.Consumer
Debugging	Dedicated Redux development tools for debugging.	No tools for debugging.

Q10. Explain some features of React Redux.

Some of the major features of Redux are as follows:

- **Reduces code complexity:** React-redux with the introduction of the store holds the state of the application's components and hence makes the code simpler as it removes the confusion of where the state should have resided.

- **Easier Maintainability:** With the help of the store, the state is now located at one position in the application which makes it easier to maintain whenever an update in the state takes place in various components.
- **Reduces Time:** React-redux refreshes the parts of a page rather than reloading the whole page and therefore saves us time.
- **Effortless Debugging:** React-redux introduces reducers which are pure functions operating on the state which makes the logic simpler and helps in achieving effortless debugging.

Q11. What are the key components of Redux architecture?

Key components of Redux architecture.

- **Actions:** Actions are a plain JavaScript object that contains information. Actions are the only source of information for the store. Actions have a type field that tells what kind of action to perform and all other fields contain information or data.
- **Reducers:** Reducers are the pure functions that take the current state and action and return the new state and tell the store how to do.
- **Store:** The store is the object which holds the state of the application.

Q12. What do you understand about Redux Toolkit?

Redux toolkit is a **npm package** that is made to simplify the creation of redux store and provide easy state management. Before the introduction of the Redux toolkit state management was complex in simple redux. The Redux toolkit acts as a wrapper around redux and encapsulates its necessary functions. Redux toolkit is flexible and provides a simple way to make a store for large applications. It follows

SOPE principle which means it is simple, Opinionated, Powerful, and Effective.

Q13. What is the purpose of the Redux store?

Store is an object which provides the state of the application. This object is accessible with help of the provider in the files of the project. The only way to change the state inside it is to dispatch an action on it.
There are three important parts of the store:

- **createStore():** To create a store object in redux.
- **dispatch(action):** To change the state of store data by using the actions.
- **getState():** For getting the current state of the store in redux.

Redux-Toolkit

Q1. What is Redux Toolkit (RTK)?

Answer:
Redux Toolkit is the official, opinionated toolset for efficient Redux development. It provides utilities to simplify common Redux tasks such as store setup, reducer logic, and middleware integration. RTK is designed to reduce boilerplate code and improve developer productivity.

Q2. What are the core features of Redux Toolkit?

Answer:
The key features include:

- configureStore: Simplifies store creation with sensible defaults.

- createSlice: Combines reducers, actions, and initial state into a single function.
- createAsyncThunk: Manages asynchronous logic like API calls with built-in loading states.
- createReducer: Allows writing reducers with "mutating" syntax using Immer.
- RTK Query: A powerful data-fetching and caching library.
- Built-in middleware like redux-thunk for async operations.

Q3. What is createSlice in Redux Toolkit?

Answer:
createSlice is a utility function that combines:

- Initial state
- Reducer functions
- Automatically generated action creators and action types
  It simplifies defining reducers and actions in Redux.

Example:

javascript
Copy code
```
import { createSlice } from '@reduxjs/toolkit';

const counterSlice = createSlice({
  name: 'counter',
  initialState: { value: 0 },
  reducers: {
    increment: (state) => { state.value += 1; },
    decrement: (state) => { state.value -= 1; },
    reset: (state) => { state.value = 0; },
  },
});

export const { increment, decrement, reset } = counterSlice.actions;
export default counterSlice.reducer;
```

Q4. What is configureStore, and how does it differ from createStore?

Answer:
configureStore is a utility provided by RTK for creating a Redux store with default configurations like:

- Automatic inclusion of Redux DevTools.
- Built-in middleware (e.g., redux-thunk).
- Simplified setup of slices and reducers.

Example:

```
import { configureStore } from '@reduxjs/toolkit';
import counterReducer from './counterSlice';

const store = configureStore({
  reducer: { counter: counterReducer },
});

export default store;
```

It differs from createStore by reducing boilerplate and automating configurations.

Q5. How does createAsyncThunk simplify async operations?

Answer:
createAsyncThunk simplifies async logic (e.g., API calls) by:

- Handling pending, fulfilled, and rejected states automatically.
- Generating action types for each step of the async process.
- Returning a promise for chaining actions.

Example:

javascript
Copy code

```
import { createAsyncThunk } from '@reduxjs/toolkit';

export const fetchUser = createAsyncThunk('user/fetch', async (userId)
=> {
  const response = await fetch(`/api/users/${userId}`);
  return response.json();
});

// In a slice:
const userSlice = createSlice({
  name: 'user',
  initialState: { data: null, status: 'idle' },
  reducers: {},
  extraReducers: (builder) => {
    builder
      .addCase(fetchUser.pending, (state) => { state.status = 'loading'; })
      .addCase(fetchUser.fulfilled, (state, action) => {
        state.status = 'succeeded';
        state.data = action.payload;
    })
    .addCase(fetchUser.rejected, (state) => { state.status = 'failed'; });
  },
});
```

Q6. What is RTK Query, and why is it useful?

Answer:
RTK Query is a library for data fetching and caching, built into Redux Toolkit. It simplifies:

- Fetching, caching, and syncing server data.
- Automatic refetching and cache invalidation.
- Managing loading, success, and error states.

Example:

```
import { createApi, fetchBaseQuery } from
'@reduxjs/toolkit/query/react';

const api = createApi({
  reducerPath: 'api',
  baseQuery: fetchBaseQuery({ baseUrl: '/api' }),
  endpoints: (builder) => ({
    getUser: builder.query({
      query: (id) => `user/${id}`,
    }),
  }),
});

export const { useGetUserQuery } = api;
```

Q7. What are the benefits of using Redux Toolkit over plain Redux?

Answer:

- Reduced Boilerplate: Combines actions, reducers, and constants in a single slice.
- Simplified Store Configuration: Pre-configured middleware and Redux DevTools.
- Improved Developer Experience: Opinionated patterns and built-in utilities.
- Immutability Made Easy: Uses Immer to enable "mutating" state safely.

Q8. How do you use multiple slices in Redux Toolkit?

Answer:
You can combine multiple slices by passing them as properties of the reducer object in configureStore.

Example:

```
import { configureStore } from '@reduxjs/toolkit';
import counterReducer from './counterSlice';
import userReducer from './userSlice';

const store = configureStore({
  reducer: {
    counter: counterReducer,
    user: userReducer,
  },
});

export default store;
```

Q9. How does Redux Toolkit handle immutability?

Answer:
Redux Toolkit uses Immer.js under the hood. It allows writing "mutating" logic in reducers, which Immer converts into immutable updates automatically. This simplifies the development process and avoids manual state copying.

Example:

javascript
Copy code

```
reducers: {
  increment: (state) => { state.value += 1; },
}
```

Here, state.value += 1 is written as a mutation, but Immer ensures the actual state is updated immutably.

Q10. Can Redux Toolkit replace Redux Thunk or Redux Saga?

Answer:
Redux Toolkit includes redux-thunk by default, so it works out of the box for most async needs. While redux-saga is not included, RTK Query

or createAsyncThunk often makes sagas unnecessary. However, if you need advanced async workflows, you can still integrate redux-saga.