

Module 9 - Introduction to React.js

Question 1: What is React.js? How is it different from other JavaScript frameworks and libraries?

- React.js, more commonly known as React, is a free, open-source JavaScript library. It works best to build user interfaces by combining sections of code (components) into full websites.
- Originally built by Facebook, Meta and the open-source community now maintain it. One of the good things about React is that you can use it as much or as little as you want!

JavaScript vs. ReactJS: Key Differences

1. Popularity

- **JavaScript:** JavaScript is an extremely popular and frequently used computer language for creating websites. With a large ecosystem, it is compatible with all major platforms.
- **ReactJS:** React is gaining prominence, particularly in the frontend creation sector. Enormous and active participation demonstrates how popular it is as a tool for creating contemporary user experiences.

2. Ease of Learning

- **JavaScript:** JavaScript is a comparatively low-entry language that is simple to learn for novices. However, learning its nuances, particularly in the framework of sophisticated applications, may need additional effort and expertise.
- **ReactJS:** React adds a component-based design and JSX syntax, which may need more time to master for individuals unfamiliar with the toolkit. After understanding the principles, React's modular design may improve the structure and maintainability of the code.

3. Performance

- **JavaScript:** The browser mechanism and the high standards of the code determine how well JavaScript performs. It is a single-threaded language learning, which might cause delays in performance in some situations.
- **ReactJS:** By refreshing the virtual DOM of React judiciously, it minimizes needless re-rendering and improves speed. This can lead to greater

performance than direct handling of DOM in basic JavaScript, particularly in complicated UI applications.

4. Security

- **JavaScript:** Customer-side flaws like cross-site scripting (XSS) are frequently linked to JavaScript's security. To reduce such hazards, designers must use security techniques and standard procedures.
- **ReactJS:** Although React does not completely solve server-side security problems, it does indirectly help prevent some sorts of issues by facilitating the usage of a DOM simulation and fostering a single-directional data flow.

5. Convenience

- **JavaScript:** Minor projects or those with less complex objectives might benefit from JavaScript's versatility in writing code explicitly without needing extra modules or frameworks.
- **ReactJS:** This framework makes complicated user interface management easier because of its component-based architecture and interchangeable user interface components, which also improve code adaptability and accessibility.

Question 2: Explain the core principles of React such as the virtual DOM and componentbased architecture.

The core principles of React JS is a component-based architecture, where all the UIs are properly constructed from self-contained components.

React JS Virtual DOM is an in-memory representation of the DOM. DOM refers to the Document Object Model that represents the content of XML or HTML documents as a tree structure so that the programs can be read, accessed, and changed in the document structure, style, and content.

Component-Based Architecture: The first react core principle is component-based architecture. It promotes a modular approach to building UIs via reusable components. Moreover, each set of components encapsulates its logic and state, hence making it easier to manage and maintain complex user interfaces.

1. How React Works:

At the very core of its functionality, react just maintains a DOM tree for us and shows it in the browser using various efficient algorithms and

computations. Your browser renders a DOM tree of HTML that is pretty basic of any browser to show the information on a website. React here uses a virtual DOM that keeps another copy of the browser DOM and that compares the actual DOM with a diffing algorithm that finds any changes made into the DOM tree and only saves those changes that were made. So unlike the native approach, the browser DOM does not need to be re-render as a whole. React just updates the things that have been changed in an efficient way.

This concept of virtual DOM makes react a faster and optimized option to build applications. It reduces the slow process and makes the browser efficiently render the HTML to show the users.

2. JSX:

JSX is simply a syntactic sugar for creating very specific JavaScript objects. Its a JavaScript extension. JSX is used to write HTML codes in a react component and render it to the browser. First of all, browsers don't understand JSX. This JSX have to be first converted to an understandable form of the browser. This is done by the WebPack (which uses Babel and other things) to transpile the JSX syntax into JavaScript code that could be understood by the browser.

If we write this code in a functional component of react,

<Button color="blue">Click Me</Button>

then the transpiler would convert this JSX into JavaScript to be understandable to the browser

React.createElement(Button, {color="blue"}, Click Me)

Then this code will be rendered to the browser using the technologies that react uses under the hood.

3. Components:

Components are the main building block of react. React is known as a component based library as other frameworks like angular also uses components. Every part of a web application is a component. Components are made in a small scale and those small components are gathered together to show a piece of the application as a whole. React uses two types of components. Functional components and class based components. Now a days functional components are getting more attraction to developers because of their usability and less strain to work with.

4. Hooks:

Hooks are a new addition in React 16.8. Hooks are basically a JavaScript function that does certain things. Hooks in react has made the use of functional component more reliable. Hooks are used extensively in a react application and to have a proper understanding of react, one must have the knowledge of hooks. There are different hooks to work with in react such as useState, useEffect, useContext, useHistory, and a whole lot more.

5. State:

State is an instance of react component and can be defined as an observable property of the behavior of the components. When a property changes such as a count when button is clicked, this property change is stored in react in a most efficient way manipulated with react useState(). React state change is a very important topic to be familiar with. When a state is changed in a react component the DOM is re-rendered to save the changes made.

6. Props:

Props are another important concept in react. Sometimes it is required to pass data from one component to another. This can be done using props. Props stands for properties. Props are read-only data and in can be send in an unidirectional flow. values like number, strings, objects even functions can be send to another component from one component by using props. Props is a convenient way to make data interchangeable in react components.

7. Default Props:

React components share their data or information with another component by using props. But sometimes a component might not have to send props to another component. So the child component can initialize default props if it does not get any props from its parent components. The component will try to find the values from props and if the props is never sent to the component then it can use the default values or the default props inside it.

8. The Top-Level Component:

When a component is about the share its data to its sibling components, it might not be able to share the data without any state management system. The first thing it can do is to lift the state to their parent component and then manipulate the state from the sibling components as the parent will share the data by using props. This concept of Top-Level Component is much easier when we want to share the data between sibling components although for more

complex work, we have to use state management systems for a more efficient way of handling.

9. Optimizing performance:

Although react uses the most efficient way to render the virtual DOM to the browser and reduces many costly operations to update the UI on state changes, there can be a slight improvement more than this of your react application. For example, use the production build. It's expected that you use the development mode when working on your app and use the production mode when deploying the app to the users. And there are a lot more ways to fasten your app's performance.

10. Conditional Rendering:

It's possible that the developer would want to show some data to the user on a certain condition. If the condition is fulfilled then the user can see the data or they might not see. This concept of conditional rendering is a must-known concept for react developers. You can use the ternary operator of conditions to make the data render if the condition is fulfilled. You must remember that you can't use the basic if-else statement normally. You have to use the ternary operator.

Question 3: What are the advantages of using React.js in web development?

1 Effortless Maintenance

Creating large applications with multiple integrations is a complex task; however, maintaining it with changing business requirements can be more complicated and expensive.

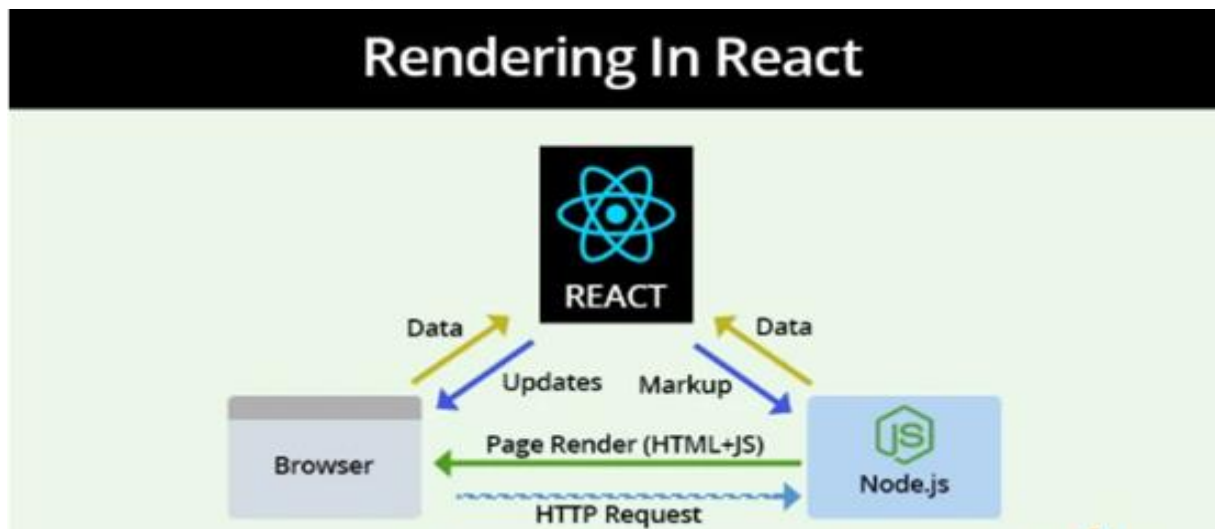
But you don't need to worry, as React JS got your back with its modular and flexible architecture, leading to modernizing each component independently without affecting others. In addition, objects defined for a particular element can be reassigned to other components to reduce the developer's effort.

As a result, less time is consumed in modernization, code is maintained, and all this is performed economically.

2 Fast Rendering

Handling high-volume user data and updating the application simultaneously is every enterprise's primary business and technical goal.

React JS comprises the in-built functionality of virtual DOM, making the rendering faster and enhancing the overall user experience. In addition, it makes the application lightweight, leading to boost performance and productivity.



Moreover, it leverages the developers to build cutting-edge applications without considering state transitions, making the development timeline shorter. Also, Virtual DOM's in-memory tree representation of real DOM makes the process sprightly accelerated between the call and display functions.

3 Search Engine Friendly

Every service-providing organization wants its website to display at the top of list of the search engines to reach the target audience. For this purpose, search engine optimization is performed.

Many websites cannot reach the top search pages, as JavaScript pages are complex to read and rank across search engines. But, the Virtual DOM feature of ReactJS renders the application more accurate and returns the web app as a typical web page.

Due to this, search bots are able to crawl through the React JS website and effortlessly analyze the relevance and rank the website on the search engine.

4 Reliable Development Tools

Having authentic tools for crafting and debugging an application is an essential requirement of software creation. Furthermore, it aids both the development team and users in integrating and utilizing forefront features.

Developers can access several developments, designing, and debugging tools freely over the internet to fabricate React JS application development. They can also seamlessly add extensions of these tools to their Google Chrome and Firefox browsers.

As a result, the development team can inspect hierarchies as they generate in Virtual DOM, and it also enables them to isolate and extend the functionality of a particular component.

5 Redux Facility

Redux helps to effortlessly store and manage the different states of large and complex applications for providing a seamless experience to the end user. Hence, it is one of the most desirable elements found in React JS.

The primary purpose of Redux is to enable other elements to access different transition states without creating a bottleneck, interacting with the child module, and utilizing a callback. Additionally, it makes code debugging and testing easy and improves server rendering.

6 Offers Mobile Application Development

React JS is probably seen as a library for building websites and web apps, but it has been upgraded to be considered for creating native mobile applications in recent years.

The phrase “learn once, write, anywhere” purely blends with the feature of React JS, as it allows to use same code structure and components for innovating mobile apps development.

The applications crafted using React JS can be installed, configured, and utilized on Android and iOS devices. In addition, its multi-platform compatibility aids organizations in building a relationship with their users diversified across different operating systems.

With this open-source library, you will be capable of extending your digital presence from desktops to mobile devices.

7 Stable and Streamline Code

Having a fault-tolerant front end is the requirement of every organization, and you can entirely depend upon React JS to accomplish this development goal.

The React JS application holds the code for each unit independently, aiding developers in performing unit tests on every component and precisely patching potential loopholes. Additionally, with a streamlined code structure, any software development company finds it easier to reinforce the business solution and adjoin features according to current business requirements.

Moreover, the stability of React JS refers to flawlessly integrating any change and smoothly running it on users' devices, reducing the probability of system crashes and clearing the barriers in the road to migrate the system to a newer version.

8 Comes with JSX

JSX is an optional extended feature of React JS used for creating components for the application.

The main objective of these tailored elements is to make code simple and leaner for large enterprise-graded solutions. While writing code, you will benefit from it as it displays more errors, exceptions, and warning messages, which you can handle in the initial phases and guarantee the delivery of a quality product.

Moreover, it helps the developers by providing a better insight into the working of the user interface inside the JavaScript code.

9 Component Reusability

Components are the most creative way of building an application from scratch and then upgrading it to a futuristic, cutting-edge business solution.

React JS provides the architecture and ecosystem to build business software based on components and reuse each component multiple times. Also, experts can utilize a particular module built for a web application for a mobile application.

JSX (JavaScript XML)

Question 1: What is JSX in React.js? Why is it used?

JSX stands for JavaScript XML. JSX allows us to write HTML in React. JSX makes it easier to write and add HTML in React.

JSX (JavaScript Extension), is a React extension which allows writing JavaScript code that looks like HTML. In other words, JSX is an HTML-like syntax used by React that extends ECMAScript so that HTML-like syntax can co-exist with JavaScript/React code.

Question 2: How is JSX different from regular JavaScript? Can you write JavaScript inside JSX?

Plain JS files can run in a browser as-is, but a JS file that contains JSX syntax can not and must be first transpiled into pure JavaScript before being sent to a web browser. The jsx extension lets build tools know that a pre-processing step is required for that file before bundling as pure JS.

JSX lets you write HTML-like markup inside a JavaScript file, keeping rendering logic and content in the same place. Sometimes you will want to add a little JavaScript logic or reference a dynamic property inside that markup.

JSX (JavaScript XML) is a syntax extension for JavaScript, primarily used in React to create user interface components. It differs from regular JavaScript in the following ways:

1. HTML-like Syntax:

JSX: Combines HTML-like tags with JavaScript logic. It's used to define the structure of UI components

```
const element = <h1>Hello, World!</h1>;
```

JavaScript: Uses methods like `document.createElement()` or templates to achieve similar results.

```
const element = document.createElement('h1');  
element.textContent = 'Hello, World!';
```

2.Babel Compilation Required:

JSX is not valid JavaScript and needs to be transpiled (e.g., with Babel) into standard JavaScript for browsers to understand.

JSX: `const element = <h1>Hello, World!</h1>;`

JavaScript: `const element = React.createElement('h1', null, 'Hello, World!');`

3.Attribute Naming Differences:

JSX uses camelCase for attributes instead of HTML-like attributes:

- class becomes className.
- onclick becomes onClick.

JSX: `<button className="btn" onClick={handleClick}>Click me</button>`

4.Embedding JavaScript Expressions:

JSX allows embedding JavaScript expressions within curly braces {}.

JSX: `const name = "Alice";`

`const element = <h1>Hello, {name}!</h1>;`

Regular JavaScript doesn't support this syntax; you'd need to construct strings or DOM elements manually.

5.Support for Components:

JSX makes it easy to create and reuse React components using custom tags:

JSX:

```
function Welcome() {  
  return <h1>Welcome to React!</h1>;  
}  
const app = <Welcome />;
```

In JavaScript, you'd need more boilerplate code to achieve similar functionality.

6.Syntax Rules:

JSX: requires:

Self-closing tags: `,
.`

One root element per component: Wrap multiple elements in a container like `<div>` or `<>` (React fragment).

Curly braces for expressions but not statements.

Question 3: Discuss the importance of using curly braces {} in JSX expressions.

Curly braces in JSX are indispensable for integrating JavaScript logic into React components. They enable dynamic rendering, improve code readability, and enhance flexibility while keeping the declarative nature of JSX intact. Understanding their use is fundamental to mastering React development.

Curly braces `{}` in JSX are a crucial part of its syntax. They enable developers to embed JavaScript expressions directly within JSX code, providing dynamic functionality and seamless integration of logic with the UI. Here's why they are important:

1. Embedding JavaScript Expressions

Curly braces allow you to insert JavaScript expressions (not statements) directly into JSX.

Example:

```
const name = "Alice";  
const greeting = <h1>Hello, {name}!</h1>;
```

Output: Hello, Alice!

2. Dynamic Content Rendering

You can use curly braces to dynamically display data, making components more flexible and interactive.

Example:

```
const isLoggedIn = true;  
const message = <h1>{isLoggedIn ? "Welcome back!" : "Please sign  
in."}</h1>;
```

Output: Displays Welcome back! if isLoggedIn is true, otherwise Please sign in.

3. Inline Calculations

Curly braces support inline calculations or function calls, avoiding the need to precompute values outside JSX.

Example:

```
const num1 = 5, num2 = 3;  
const sumElement = <h1>The sum is: {num1 + num2}</h1>;
```

Output: The sum is: 8

4. Loops for Rendering Lists

Curly braces enable mapping through arrays to render multiple elements efficiently.

Example:

```
const items = ["Apple", "Banana", "Cherry"];  
const list = (  
  <ul>  
    {items.map(item => (  
      <li key={item}>{item}</li>  
    ))}  
  </ul>  
);
```

Output: A list of elements with Apple, Banana, and Cherry.

5. Conditional Rendering

Curly braces, combined with JavaScript expressions, facilitate conditional rendering.

Example:

```
const showElement = true;  
const element = <div>{showElement && <p>This is visible!</p>}</div>;
```

Output: Renders `<p>This is visible!</p>` only if `showElement` is true.

6. Accessing Object Properties

You can directly access object properties using curly braces.

Example:

```
const user = { name: "Alice", age: 25 };  
const element = <h1>{user.name} is {user.age} years old.</h1>;
```

Output: Alice is 25 years old.

7. Ensures Code Readability and Maintainability

Using `{}` keeps logic and markup together in one cohesive format, reducing the need for complex string concatenations or external processing.

Components (Functional & Class Components)

Question 1: What are components in React? Explain the difference between functional components and class components.

In React, components are the building blocks of the user interface. A component is a reusable piece of code that represents a part of the UI and is responsible for rendering it. Components can accept props (inputs), manage state, and handle user interactions.

There are two main types of components in React: Functional Components **and** Class Components.

1. Functional Components

Functional components are JavaScript functions that return React elements (JSX). They are simpler and primarily used for rendering UI.

Key Features of Functional Components:

- **Simplicity:** Written as plain JavaScript functions.
- **Hooks:** Can manage state and side effects using React Hooks (e.g., `useState`, `useEffect`).
- **Stateless or Stateful:** Originally designed for stateless rendering but can now handle state with Hooks.
- **Performance:** Lightweight and often faster due to simpler lifecycle management.

2. Class Components

Class components are ES6 classes that extend `React.Component`. They have more complex syntax and are typically used when working with lifecycle methods.

Key Features of Class Components:

- **Stateful:** Manage their own internal state using `this.state`.
- **Lifecycle Methods:** Support full React lifecycle methods like `componentDidMount`, `shouldComponentUpdate`, etc.
- **More Verbose:** Require the use of `this` for accessing props, state, and methods.

Question 2: How do you pass data to a component using props? •

In React, **props** (short for "properties") are used to pass data from a parent component to a child component. Props are a key mechanism for creating reusable, dynamic components by providing input values to them.

How to Pass Data to a Component Using Props

1. Passing Props from Parent to Child

Props are passed to a child component as attributes in the JSX of the parent component.

Example:

```
function ParentComponent() {  
  return <ChildComponent name="Priyanka" age={25} />;  
}
```

```
function ChildComponent(props) {
  return (
    <div>
      <h1>Name: {props.name}</h1>
      <h2>Age: {props.age}</h2>
    </div>
  );
}
```

Explanation:

name="priyanka" and age={25} are passed as props from ParentComponent to ChildComponent.

The child accesses these props using the props parameter.

2.Accessing Props

In **functional components**, props are accessed as an argument of the component function.

```
function ChildComponent(props) {
  return <p>Hello, {props.name}</p>;
}
```

In **class components**, props are accessed using this.props.

```
class ChildComponent extends React.Component {
  render() {
    return <p>Hello, {this.props.name}</p>;
  }
}
```

3.Default Prop Values

You can set default values for props using defaultProps.

Example:

```
function Greeting(props) {  
  return <h1>Hello, {props.name}!</h1>;  
}  
  
Greeting.defaultProps = {  
  name: "Guest",  
};
```

4.Prop Types Validation (Optional)

To enforce type checking and prevent runtime errors, you can use PropTypes.

Example:

```
import PropTypes from 'prop-types';  
  
function Greeting(props) {  
  return <h1>Hello, {props.name}!</h1>;  
}  
  
Greeting.propTypes = {  
  name: PropTypes.string.isRequired,  
};
```

5.Props Are Read-Only

Props are immutable within the child component. They cannot be modified directly by the child.

Example (Error):

```
function ChildComponent(props) {  
  props.name = "Bob"; // ✗ Error  
  return <p>{props.name}</p>;  
}
```


Key Points

- Props are passed down unidirectionally (from parent to child).
- Props are read-only in the child component.
- You can pass any type of data as props: strings, numbers, arrays, objects, or even Props are a fundamental concept in React, enabling dynamic and reusable components.

Question 3: What is the role of render() in class components?

The render() method in class components:

- Defines what should be displayed in the UI.
- Updates the UI when the component's state or props change.
- Must be a pure function that returns JSX or null.

With the rise of functional components and React Hooks, render() is used less frequently in modern React development but remains essential in understanding React's class-based architecture.

The render() method in React class components is a core part of the component lifecycle. It is responsible for defining and returning the JSX that should be rendered to the DOM. Here's an explanation of its role and functionality:

Role of render() in Class Components

1. Rendering JSX to the DOM:

The render() method outputs the UI that the component should display.

It must return a single JSX element (or null if nothing should be rendered).

2. Re-rendering on State or Prop Changes:

When the component's **state** or **props** change, React automatically calls the render() method to update the DOM with the new UI.

This ensures that the UI stays in sync with the component's data.

3. Purely Declarative:

The `render()` method should be a **pure function** of props and state. It should not modify state or cause side effects (e.g., API calls, timers).

Any side effects should be handled in lifecycle methods like `componentDidMount`.

Key Rules of `render()`

1. Must Return JSX or null:

You can only return a single JSX element, which can be wrapped in a container if needed.

Example:

```
render() {  
  return (  
    <div>  
      <h1>Title</h1>  
      <p>Content</p>  
    </div>  
  );  
}
```

2. Avoid Side Effects:

Side effects like network requests or DOM manipulation should not occur in `render()` to ensure consistent UI updates.

Incorrect:

```
render() {  
  fetch('/api/data'); // ✗ Side effect  
  return <div>Loading...</div>;  
}
```

```
}
```

3. Uses State and Props:

Access `this.props` and `this.state` to dynamically determine what to render.

Example:

```
render() {  
  return <h1>Hello, {this.props.name}</h1>;  
}
```

4. Lifecycle and the Role of render()

The `render()` method is part of the **mounting**, **updating**, and **unmounting** phases of the React component lifecycle:

Mounting: Called when the component is first rendered.

Updating: Called whenever state or props change.

Unmounting: The method is not called during unmounting.

Props and State

Question 1: What are props in React.js? How are props different from state?

Props (short for "properties") are used in React to pass data from a parent component to a child component. They make components reusable and dynamic by allowing them to receive inputs that can affect their rendering.

Key Features of Props

1. Read-Only:

Props are immutable within the child component. Once passed, they cannot be modified by the receiving component.

2. Passed from Parent to Child:

Props flow **unidirectionally** (from parent to child) in the component hierarchy.

3. Used for Customization:

Props allow components to be configured for specific use cases.

Example:

```
function Greeting(props) {  
  return <h1>Hello, {props.name}!</h1>;  
}  
  
function App() {  
  return <Greeting name="Alice" />;  
}
```

4. Can Be Any Data Type:

Props can be strings, numbers, arrays, objects, or even functions.

5. Accessible in Components:

Functional Components: Props are accessed via the function argument.

Class Components: Props are accessed via `this.props`.

How Are Props Different from State?

Feature	Props	State
Definition	Input data passed to a component by its parent.	Internal data managed by the component itself.
Mutability	Immutable (read-only).	Mutable (can be changed using <code>setState</code> or <code>useState</code>).
Responsibility	Controlled by the parent component.	Controlled by the component itself.

Purpose	Used to pass data and functions to child components.	Used to store and manage the component's dynamic data.
Access	Passed as an argument in functional components or accessed via this.props in class components.	Managed using React's useState Hook or this.state in class components.
Lifecycle Scope	Props are passed and exist throughout the component's lifecycle.	State can change during the component's lifecycle.
Usage	To configure a component or make it reusable.	To handle user interactions, updates, or dynamic behavior within a component.

- **Props** allow data to flow between components, making them reusable and configurable.
- **State** is used to manage dynamic data within a component. Understanding the distinction and interplay between props and state is crucial for building effective and maintainable React applications.

Question 2: Explain the concept of state in React and how it is used to manage component data.

State in React

State is a core concept in React that allows components to store, track, and manage dynamic data over time. It represents the "current situation" or "snapshot" of a component and determines how it renders and behaves.

State is a powerful feature in React for managing dynamic, component-specific data. It enables interactivity, conditional rendering, and user input handling, making it essential for building dynamic, responsive applications. By

leveraging state, developers can create rich and engaging user interfaces with React.

Key Features of State

1. **Mutable:** Unlike props, state is mutable, meaning it can change over the component's lifecycle.
2. **Local to the Component:** State is managed within the component where it is defined and cannot be directly accessed or modified by other components.
3. **Triggers Re-Rendering:** When state changes, React automatically re-renders the component to reflect the updated state in the UI.
4. **Initial State:** Components can define their initial state when they are created.

How State is Used to Manage Component Data

1. Defining State

In **functional components**, state is managed using the useState Hook.

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

function Counter() {
  const [count, setCount] = useState(0); // Define state
  return <p>Count: {count}</p>;
}
```

In **class components**, state is defined as an object within the constructor.

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 }; // Define state
  }
  render() {
    return <p>Count: {this.state.count}</p>;
  }
}
```

2. Updating State

In functional components, you use the state updater function (e.g., `setState`) returned by `useState` to update the state.

```
function Counter() {  
  const [count, setCount] = useState(0);  
  
  const increment = () => setCount(count + 1);  
  
  return (  
    <div>  
      <p>Count: {count}</p>  
      <button onClick={increment}>Increment</button>  
    </div>  
  );  
}
```

In class components, you use `this.setState()` to update the state.

```
class Counter extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { count: 0 };  
  }  
  
  increment = () => {  
    this.setState({ count: this.state.count + 1 });  
  };  
  
  render() {  
    return (  
      <div>  
        <p>Count: {this.state.count}</p>  
        <button onClick={this.increment}>Increment</button>  
      </div>  
    );  
  }  
}
```

```
}  
}
```

3. Using State for Conditional Rendering

State can control what is displayed on the UI based on its value.

```
function LoginStatus() {  
  const [isLoggedIn, setIsLoggedIn] = useState(false);  
  
  return (  
    <div>  
      <h1>{isLoggedIn ? "Welcome Back!" : "Please Log In"}</h1>  
      <button onClick={() => setIsLoggedIn(!isLoggedIn)}>  
        {isLoggedIn ? "Log Out" : "Log In"}  
      </button>  
    </div>  
  );  
}
```

4. Using State for Forms

State can manage user input in forms.

```
function Form() {  
  const [input, setInput] = useState("");  
  
  const handleChange = (e) => setInput(e.target.value);  
  
  return (  
    <div>  
      <input type="text" value={input} onChange={handleChange} />  
      <p>You typed: {input}</p>  
    </div>  
  );  
}
```


Key Points to Remember

1. React is Reactive:

State changes trigger re-renders, allowing the UI to stay in sync with the component's data.

2. State Updates are Asynchronous:

React may batch multiple `setState` calls for performance optimization.

3. Use State for Component-Local Data:

State is for managing data that changes over time within the component.

For global or shared data, consider using **props** or **state management libraries** like Redux or Context API.

4. Avoid Directly Modifying State:

State should only be updated via `setState` (class components) or the updater function (functional components).

Question 3: Why is `this.setState()` used in class components, and how does it work?

In React class components, `this.setState()` is used to update the component's state. It is the only way to modify the state property and trigger a re-render of the component, ensuring that the UI reflects the latest state.

`this.setState()` is essential for managing state in React class components. It allows you to update the component's state in a controlled and efficient manner, ensuring that the UI stays in sync with the data. By adhering to best practices, such as using the functional form when needed and avoiding direct

state mutations, you can avoid common pitfalls and build robust React applications.

How `this.setState()` Works

1. Merges State Changes:

`this.setState()` merges the new state with the existing state. You only need to provide the properties you want to update, and the rest of the state remains unchanged.

Example:

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0, name: "Counter" };
  }

  increment = () => {
    this.setState({ count: this.state.count + 1 });
  };

  render() {
    return (
      <div>
        <h1>{this.state.name}</h1>
        <p>Count: {this.state.count}</p>
        <button onClick={this.increment}>Increment</button>
      </div>
    );
  }
}
```

1. Asynchronous Updates:

`this.setState()` is asynchronous, meaning React may batch multiple state updates for performance optimization.

To ensure the updated state depends on the previous state, use a **function** instead of an object:

```
this.setState((prevState) => ({  
  count: prevState.count + 1  
}));
```

2. Triggers Re-Rendering:

When `this.setState()` is called, React schedules a re-render of the component. During the re-render, `render()` is called with the updated state to update the DOM.

3. State Updates May Be Batched:

Multiple `setState` calls in a single event handler might be batched to improve performance.

Example:

```
handleClick = () => {  
  this.setState({ count: this.state.count + 1 });  
  this.setState({ count: this.state.count + 1 });  
  console.log(this.state.count); // May log the previous state  
};
```

4. Does Not Mutate State Directly:

You should never directly modify the state object (e.g., `this.state.count = 1`). Always use `this.setState()` to ensure React knows about the change and updates the UI correctly.

Syntax of `this.setState()`

1. Passing an Object:

Use this when the new state does not depend on the previous state.
`this.setState({ property: newValue });`

2. Passing a Function:

Use this when the new state depends on the previous state.

```
this.setState((prevState, props) => ({  
  property: prevState.property + 1  
}));
```

Advantages of this.setState()

1. Ensures React is Aware of State Changes:

Direct state modification (`this.state`) does not inform React of changes, but `this.setState()` does.

2. Asynchronous and Efficient:

React batches updates to improve performance and minimize re-renders.

3. Declarative UI Updates:

By calling `this.setState()`, React automatically calculates the necessary updates to the DOM.