1)Explain about conditional rendering in React?

Conditional Rendering in React

Conditional rendering is a fundamental concept in React that allows you to display different elements or components based on certain conditions. It's the process of deciding what to render in the user interface based on the state of your application. This is crucial for creating dynamic and interactive user interfaces where the content changes in response to user actions, data updates, or other factors.

In React, you use standard JavaScript operators and constructs to implement conditional rendering. Here are the most common methods:

1. if and else Statements

This is the most straightforward approach, using a standard if statement to conditionally return different JSX. You can use this within a component's render method or a function that returns JSX.

Example:

function UserGreeting(props) {

return <h2>Welcome back!</h2>;

}

function GuestGreeting(props) {

return <h2>Please sign up.</h2>;

}

function Greeting(props) {

const isLoggedIn = props.isLoggedIn;

if (isLoggedIn) {

return <UserGreeting />;

}

return <GuestGreeting />;

}

function App() {

return <Greeting isLoggedIn={true} />;

}

2)Ternary Operator (condition ? true : false)

The ternary operator is a concise way to write an if-else statement inline within your JSX. It's very popular for simple conditional rendering.

Example:

function Greeting(props) {

const isLoggedIn = props.isLoggedIn;

return (

<div>

{isLoggedIn ? <UserGreeting /> : <GuestGreeting />}

</div>

);

}

function App() {

return <Greeting isLoggedIn={true} />;

}

3)switch Statements

While less common for simple conditions, a switch statement can be very effective when you have multiple possible conditions and you want to render different things for each case.

Example:

function getComponent(status) {

switch (status) {

case 'loading':

return <LoadingSpinner />;

case 'success':

return <SuccessMessage />;

case 'error':

return <ErrorMessage />;

default:

return null;

}

}

function App() {

const apiStatus = 'success';

return (

<div>

{getComponent(apiStatus)}

</div>

);

}

4)

4)Logical && Operator (Short-Circuit Evaluation)

The logical && operator is useful for rendering a block of JSX only if a condition is true, and rendering nothing if it's false. In JavaScript, true && expression evaluates to expression, and false && expression evaluates to false. In JSX, React treats false as "do not render."

Example:

function Mailbox(props) {

const unreadMessages = props.unreadMessages;

return (

<div>

<h1>Hello!</h1>

{unreadMessages.length > 0 &&

<h2>

You have {unreadMessages.length} unread messages.

</h2>

}

</div>

);

}

const messages = ['React', 'Re: React', 'Re: Re: React'];

ReactDOM.render(

<Mailbox unreadMessages={messages} />,

document.getElementById('root')

);

2) Define element variables?

In React, an element variable is a standard JavaScript variable that is used to hold JSX elements. Instead of writing JSX directly inside the return statement of a component, you can assign it to a variable and then use that variable within the return statement. This technique is particularly useful for implementing conditional rendering and for making your component's return statement cleaner and more readable.

Basic Example:

function MyComponent() {

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

return (

<div>

{element}

</div>

);

}

In this example, the JSX <h1>Hello, World!</h1> is assigned to the element variable. The variable element is then used inside the curly braces {} within the main return statement, and React will render the <h1> element.

3) Explain how to prevent components from rendering?

1. Using if statement inside the component

This is the most straightforward method.

function Welcome(props) {

if (!props.isLoggedIn) {

return null; // Prevents rendering

}

return <h1>Welcome back!</h1>;

}

* If isLoggedIn is false, the component returns null, so nothing gets rendered.
* Returning null from a component = no UI output.

2. Using conditional rendering with && operator

{isVisible && <MyComponent />}

* If isVisible is true, MyComponent will render.
* If isVisible is false, React skips rendering it (short-circuits).

3. Using ternary operator ? :

{showAlert ? <AlertMessage /> : null}

* If showAlert is true → render <AlertMessage />
* If false → render null (nothing)

4. Controlling rendering in parent component

can prevent rendering a child component from the parent, by not including it in JSX:

function App() {

const showComponent = false;

return (

<div>

<h1>Hello</h1>

{showComponent && <ChildComponent />} {/\* Won’t render \*/}

</div>

);

}

5. Using shouldComponentUpdate (Class components only)

For class components:

shouldComponentUpdate(nextProps, nextState) {

return false; // Prevents re-rendering

}