## 1. Explain various ways of conditional rendering

Conditional rendering in React allows us to dynamically decide which content to show based on application state or props. One common method is using the if statement. This is useful when you want to conditionally render a block of code before the return() statement inside a component. It offers good readability and control for complex logic but cannot be directly used inside JSX.

Another way is the ternary operator (condition ? expr1 : expr2), which is often used inside JSX. This method is concise and helps when you want to render one of two elements based on a condition. It is best suited for simple binary choices, such as displaying a login form or a welcome message based on user authentication.

The logical AND operator (&&) is also used when you want to render something only if a condition is true, and do nothing otherwise. This is useful for showing optional UI elements like a loading spinner or error message. Lastly, we can also use switch statements or create helper functions to manage more complex rendering logic, especially when dealing with multiple condition outcomes.

## 2. Explain how to render multiple components

Rendering multiple components in React is simple and central to building modular UIs. We typically import each component and include them inside a parent component’s JSX return statement. These components can be placed inside <div> tags, fragments (<> </>), or even rendered conditionally based on props or state.

You can also loop through an array and render a list of components dynamically using JavaScript’s map() function. This approach is powerful for building UIs based on dynamic data, such as product listings, blog posts, or user comments. Each component receives its own props, making it highly reusable and maintainable.

Another approach is using a layout structure where one component wraps multiple subcomponents, like in a dashboard or form. React’s composition model makes this natural and scalable. You can also encapsulate multiple components into a container component, which handles layout and logic, making your application more organized and easier to test.

## 3. Define list component

A List component in React is a component specifically designed to render a list of items, usually from an array of data. It typically takes in an array as a prop and uses the map() function to transform each item into a JSX element. This is very common in real-world apps where you need to display dynamic data fetched from a server or database.

The component can also apply styles or logic to each item conditionally, such as highlighting selected items or disabling options. By separating the list into its own component, it promotes reusability and better separation of concerns. The items can also be custom components themselves, not just plain HTML elements.

In more advanced usage, list components might include pagination, search filtering, or drag-and-drop functionality. Managing such behavior in a separate list component helps maintain cleaner code in the parent component. It also allows you to optimize performance with techniques like memoization or virtualization for large data sets.

## 4. Explain about keys in React applications

Keys are special attributes you assign to list elements in React to help identify which items have changed, been added, or removed. They are required when rendering lists using map() or loops. Keys help React optimize rendering by keeping track of elements using a unique identifier.

Without keys, React will re-render the entire list even if just one item changes. This can negatively affect performance, especially with large lists. By assigning a stable key to each element, React compares current and previous lists efficiently and updates only the necessary items. The most common practice is to use a unique ID as the key.

It is important not to use the array index as a key unless the list is static and doesn’t change. This is because reordering or deleting items can lead to unexpected UI behavior. Using stable and unique keys ensures that your UI stays consistent and performant, especially when dealing with forms or user-generated lists.

## 5. Explain how to extract components with keys

Extracting components with keys involves separating parts of your list-rendering logic into smaller, reusable child components. For example, instead of mapping over data and returning raw JSX inside the parent, you create a ListItem component and pass data as props. While doing so, each ListItem must be given a unique key prop when used in the map().

This approach improves the readability of your main component by delegating rendering logic. It also encapsulates styles, event handling, and structure for each item. The key, however, still needs to be applied to the component instance in the parent — not inside the child component — because keys only affect the element’s siblings in the virtual DOM.

Keys are never accessible inside child components; they are strictly used by React’s internal reconciliation algorithm. Hence, you should pass a separate id or item.id as a regular prop if the component needs to use it. Extracting components with keys is a best practice when building complex, interactive, or nested list UIs.

## 6. Explain React Map, map() function

The map() function in React is a JavaScript array method used to transform an array into a new array of elements — typically JSX elements. In React, it is most commonly used to render a list of components or HTML elements dynamically. It loops over each item and returns a component for each, making it ideal for rendering lists.

React doesn’t have its own version of map(); it relies on JavaScript’s native method. The key thing is how it integrates with JSX: you place it inside the return block or inside curly braces {} to output the list. Each item returned must have a unique key to ensure efficient updates and re-renders.

Using map() this way makes your components declarative and clean. It also pairs well with conditionals, allowing you to filter or transform data before rendering. Combined with props, this method makes dynamic and reusable UIs a breeze, enabling powerful patterns like list tables, grids, and carousels.

# Code Files

## App.js

import React, { useState } from "react";  
import CourseDetails from "./components/CourseDetails";  
import BookDetails from "./components/BookDetails";  
import BlogDetails from "./components/BlogDetails";  
  
function App() {  
 const [showCourses, setShowCourses] = useState(true);  
 const [showBooks, setShowBooks] = useState(true);  
 const [showBlogs, setShowBlogs] = useState(true);  
  
 return (  
 <div>  
 <div style={{ textAlign: "center", marginBottom: "20px" }}>  
 <button onClick={() => setShowCourses(!showCourses)}>Toggle Courses</button>  
 <button onClick={() => setShowBooks(!showBooks)}>Toggle Books</button>  
 <button onClick={() => setShowBlogs(!showBlogs)}>Toggle Blogs</button>  
 </div>  
 <div style={{ display: "flex", justifyContent: "space-around", padding: 40 }}>  
 {showCourses ? <CourseDetails /> : <p>No Courses to display</p>}  
 {showBooks && <BookDetails />}  
 {showBlogs ? <BlogDetails /> : null}  
 </div>  
 </div>  
 );  
}  
  
export default App;

## CourseDetails.js

import React from "react";  
  
const CourseDetails = () => {  
 const courses = [  
 { name: "Angular", date: "4/5/2021" },  
 { name: "React", date: "6/3/20201" },  
 ];  
  
 return (  
 <div>  
 <h2>Course Details</h2>  
 {courses.length > 0 ? (  
 courses.map((course, idx) => (  
 <div key={idx}>  
 <h3>{course.name}</h3>  
 <p>{course.date}</p>  
 </div>  
 ))  
 ) : (  
 <p>No course data available.</p>  
 )}  
 </div>  
 );  
};  
  
export default CourseDetails;

## BookDetails.js

import React from "react";  
  
const BookDetails = () => {  
 const books = [  
 { name: "Master React", price: 670 },  
 { name: "Deep Dive into Angular 11", price: 800 },  
 { name: "Mongo Essentials", price: 450 },  
 ];  
  
 return (  
 <div style={{ borderLeft: "3px solid green", paddingLeft: 10 }}>  
 <h2>Book Details</h2>  
 {books.map((book, idx) => (  
 <div key={idx}>  
 <h4>{book.name}</h4>  
 <p>{book.price}</p>  
 </div>  
 ))}  
 </div>  
 );  
};  
  
export default BookDetails;

## BlogDetails.js

import React from "react";  
  
const BlogDetails = () => {  
 const showInstallation = true;  
  
 return (  
 <div style={{ borderLeft: "3px solid green", paddingLeft: 10 }}>  
 <h2>Blog Details</h2>  
 <div>  
 <h3>React Learning</h3>  
 <strong>Stephen Biz</strong>  
 <p>Welcome to learning React!</p>  
 </div>  
 {showInstallation && (  
 <div>  
 <h3>Installation</h3>  
 <strong>Schwezdenier</strong>  
 <p>You can install React from npm.</p>  
 </div>  
 )}  
 </div>  
 );  
};  
  
export default BlogDetails;

## Output

