## Objective Answers

### 1. Explain Various Ways of Conditional Rendering

In React, conditional rendering allows the UI to change based on application state or props. It can be implemented using multiple techniques:

* if-else **statements** inside the render() method to return different JSX blocks.
* **Ternary operators (**? :**)** directly inside JSX to conditionally render components or elements inline.
* **Logical AND (**&&**) operator** to conditionally display an element only if the condition is true.
* **Element variables**, where JSX is stored in a variable and returned later based on conditions.
* **Switch-case statements**, which can be used for handling multiple rendering paths more cleanly.

These methods improve flexibility and control over the component output, making React apps more dynamic and user-responsive.

### 2. Explain How to Render Multiple Components

Rendering multiple components in React involves placing multiple component tags either next to each other or inside a parent wrapper like a <div>, <section>, or React Fragment (<>...</>). For example:

function App() {

return (

<>

<Header />

<MainContent />

<Footer />

</>

);

}

This is useful for breaking down UI into smaller, reusable blocks and rendering them together to build complete screens. We can also dynamically render components based on arrays or states using loops like .map().

### 3. Define List Component

A list component in React is any component that is responsible for displaying a collection of items, typically using the .map() function to loop over data. It renders multiple similar child components or HTML elements based on an array. For example, a BookList component may loop through an array of books and display each one in a <li> or custom component like <BookCard />.

Using list components keeps the code modular, helps apply consistent styling, and improves code readability and maintainability.

### 4. Explain About Keys in React Applications

Keys in React are special string attributes used to help React identify which items in a list have changed, been added, or removed. They are required when rendering dynamic lists to optimize performance and avoid unnecessary re-renders.

A key should be a **unique and stable identifier** for each list item, typically an ID from the data source:

books.map(book => <Book key={book.id} title={book.title} />)

Avoid using array indexes as keys unless the list is static or order never changes. Correct key usage ensures efficient reconciliation and DOM updates.

### 5. Explain How to Extract Components with Keys

When mapping over data, you can extract each item into a **separate reusable component**, passing the item’s data as props. The key is then applied to the **parent component tag** (not inside the child component). Example:

{books.map(book => <BookCard key={book.id} book={book} />)}

And inside BookCard.js:

function BookCard({ book }) {

return <li>{book.title} - {book.author}</li>;

}

This technique increases reusability and keeps rendering logic clean by isolating concerns.

### 6. Explain React Map / map() Function

The .map() function in JavaScript is used in React to iterate over arrays and render elements or components dynamically. It takes each item in the array and returns a corresponding JSX element:

const items = ['React', 'Angular', 'Vue'];

const listItems = items.map((item, index) => <li key={index}>{item}</li>);

In React, .map() is often used inside return statements to render lists like tables, menus, cards, or dashboards. It is essential to include a key for each item to ensure optimal performance.