React

1. Explain Various Ways of Conditional Rendering

React provides several methods for **conditional rendering**, allowing developers to dynamically render content based on state, props, or logic. The most commonly used approaches include:

a) If-Else Statement

```javascript
if (isLoggedIn) {
  return <h2>Welcome back!</h2>;
} else {
  return <h2>Please sign in.</h2>;
}
```

b) Ternary Operator

Ideal for rendering one of two elements:

<h2>{isLoggedIn ? 'Welcome back!' : 'Please log in.'}</h2>

c) Logical AND (&&) Operator

Useful for rendering one element conditionally:

{hasError && <p style={{ color: 'red' }}>An error occurred.</p>}

d) Switch Statement (within functions)

```javascript
switch (userType) {
  case 'admin':
    return <AdminDashboard />;
  case 'guest':
    return <GuestPage />;
  default:
    return <HomePage />;
}
```

---

2. Explain How to Render Multiple Components

To render **multiple components** in React, you can place them side by side within a parent element such as a , <> (Fragment), or a custom wrapper component.

Example:

```javascript
function App() {
  return (
    <div>
      <Header />
      <Navigation />
      <Content />
      <Footer />
    </div>
  );
}
```

Alternatively, using **React Fragments** to avoid extra DOM nodes:

```javascript
<>
  <ComponentOne />
  <ComponentTwo />
</>
```

Rendering multiple components allows for modular, maintainable code and promotes reusability throughout the application.

---

3. Define List Component

A **List component** in React is a component designed to render a collection of similar items, such as an array of users, tasks, or products. Lists are usually created by iterating over an array using JavaScript's map() function and returning a JSX element for each item.

Example:

```javascript
function UserList(props) {
  return (
    <ul>
      {props.users.map(user => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}
```

A list component improves UI structure and data representation by handling dynamic sets of data efficiently.

---

4. Explain About Keys in React Applications

Keys in React are unique identifiers assigned to elements in a list to help React distinguish which items have changed, been added, or removed. This improves rendering performance and prevents unexpected behavior during re-renders.

Key Rules:

- Keys must be unique **among siblings**.
- Ideally, use a unique ID from the data itself.
- Avoid using array indices as keys unless the list is static.

Example:

```javascript
{items.map(item => (
  <li key={item.id}>{item.name}</li>
))}
```

---

5. Explain How to Extract Components with Keys

When rendering lists, it's common to extract each rendered item into its own **child component**. The key prop should be passed to the **element in the list**, not inside the child component itself.

Example :

```javascript
function ListItem(props) {
  return <li>{props.value}</li>;
}

function NumberList(props) {
  return (
    <ul>
      {props.numbers.map(number => (
        <ListItem key={number.toString()} value={number} />
      ))}
    </ul>
  );
}
```

Here, key is used in the array mapping where each ListItem is created. This ensures efficient updating of individual list items.

---

6. Explain React Map, map() Function

The **map() function** in React refers to the native JavaScript Array.prototype.map() method, used for transforming arrays into arrays of React elements. It is a fundamental tool for rendering lists in JSX.

Syntax:

```javascript
array.map((item, index) => {
  // return JSX
});
```

Example in React:

```javascript
const fruits = ['Apple', 'Banana', 'Cherry'];

const fruitList = fruits.map((fruit, index) => (
  <li key={index}>{fruit}</li>
));

return <ul>{fruitList}</ul>;
```

React + map() allows:

- Dynamic generation of components
- Clean and declarative rendering of collections
- Incorporation of logic or formatting per item