**Explain Various Ways of Conditional Rendering**

**Conditional rendering** in React allows you to render different components or elements based on certain conditions. Here are the main approaches:

**1. If-Else Statements**

function UserGreeting({ isLoggedIn }) {  
 if (isLoggedIn) {  
 return <h1>Welcome back!</h1>;  
 } else {  
 return <h1>Please sign up.</h1>;  
 }  
}

**2. Ternary Operator**

function LoginButton({ isLoggedIn }) {  
 return (  
 <div>  
 {isLoggedIn ? (  
 <button>Logout</button>  
 ) : (  
 <button>Login</button>  
 )}  
 </div>  
 );  
}

**3. Logical AND Operator**

function Notifications({ messages }) {  
 return (  
 <div>  
 <h1>Hello!</h1>  
 {messages.length > 0 &&  
 <h2>You have {messages.length} unread messages.</h2>  
 }  
 </div>  
 );  
}

**4. Element Variables**

function Welcome({ user }) {  
 let greeting;  
   
 if (user) {  
 greeting = <h1>Welcome, {user.name}!</h1>;  
 } else {  
 greeting = <h1>Welcome, Guest!</h1>;  
 }  
   
 return <div>{greeting}</div>;  
}

**5. Switch Statement (in function)**

function UserRole({ role }) {  
 const renderByRole = () => {  
 switch(role) {  
 case 'admin':  
 return <AdminPanel />;  
 case 'user':  
 return <UserPanel />;  
 case 'guest':  
 return <GuestPanel />;  
 default:  
 return <div>Unknown role</div>;  
 }  
 };  
   
 return <div>{renderByRole()}</div>;  
}

**Explain How to Render Multiple Components**

**Rendering multiple components** in React can be done in several ways:

**1. Using Array of Components**

function Dashboard() {  
 const components = [  
 <Header key="header" />,  
 <Sidebar key="sidebar" />,  
 <MainContent key="main" />,  
 <Footer key="footer" />  
 ];  
   
 return <div>{components}</div>;  
}

**2. Using React Fragment**

function MultipleComponents() {  
 return (  
 <React.Fragment>  
 <Header />  
 <Navigation />  
 <Content />  
 <Footer />  
 </React.Fragment>  
 );  
}  
  
// Short syntax  
function MultipleComponents() {  
 return (  
 <>  
 <Header />  
 <Navigation />  
 <Content />  
 <Footer />  
 </>  
 );  
}

**3. Using Map Function**

function ComponentList() {  
 const componentData = [  
 { id: 1, name: 'Header', type: 'header' },  
 { id: 2, name: 'Content', type: 'content' },  
 { id: 3, name: 'Footer', type: 'footer' }  
 ];  
   
 return (  
 <div>  
 {componentData.map(item => (  
 <div key={item.id}>  
 <h3>{item.name}</h3>  
 <p>Type: {item.type}</p>  
 </div>  
 ))}  
 </div>  
 );  
}

**4. Conditional Multiple Components**

function ConditionalComponents({ userType }) {  
 const renderAdminComponents = () => (  
 <>  
 <AdminHeader />  
 <AdminSidebar />  
 <AdminContent />  
 </>  
 );  
   
 const renderUserComponents = () => (  
 <>  
 <UserHeader />  
 <UserContent />  
 </>  
 );  
   
 return (  
 <div>  
 {userType === 'admin' ? renderAdminComponents() : renderUserComponents()}  
 </div>  
 );  
}

**Define List Component**

A **list component** is a React component that renders a collection of similar items. It typically uses the map() function to transform an array of data into an array of JSX elements.

**Basic List Component Structure**

function ListComponent({ items }) {  
 return (  
 <ul>  
 {items.map(item => (  
 <li key={item.id}>{item.name}</li>  
 ))}  
 </ul>  
 );  
}

**Complete List Component Example**

function StudentList() {  
 const students = [  
 { id: 1, name: 'John Doe', grade: 'A' },  
 { id: 2, name: 'Jane Smith', grade: 'B' },  
 { id: 3, name: 'Mike Johnson', grade: 'A' },  
 { id: 4, name: 'Sarah Wilson', grade: 'C' }  
 ];  
   
 return (  
 <div className="student-list">  
 <h2>Student List</h2>  
 <ul>  
 {students.map(student => (  
 <li key={student.id} className="student-item">  
 <strong>{student.name}</strong> - Grade: {student.grade}  
 </li>  
 ))}  
 </ul>  
 </div>  
 );  
}

**Dynamic List Component**

function ProductList({ products, onProductClick }) {  
 if (!products || products.length === 0) {  
 return <div>No products available</div>;  
 }  
   
 return (  
 <div className="product-list">  
 {products.map(product => (  
 <div   
 key={product.id}   
 className="product-item"  
 onClick={() => onProductClick(product.id)}  
 >  
 <h3>{product.name}</h3>  
 <p>Price: ${product.price}</p>  
 <p>Stock: {product.stock}</p>  
 </div>  
 ))}  
 </div>  
 );  
}

**Explain About Keys in React Applications**

**Keys** are special attributes that help React identify which items in a list have changed, been added, or been removed. They are crucial for efficient rendering and maintaining component state.

**Why Keys Are Important**

* **Performance optimization**: Helps React minimize DOM manipulations
* **State preservation**: Maintains component state during re-renders
* **Avoid rendering bugs**: Prevents incorrect component updates

**Key Rules**

1. **Must be unique**: Among siblings (not globally)
2. **Should be stable**: Don't change between renders
3. **Should be predictable**: Same item should have same key

**Good Key Examples**

// Using unique ID (BEST)  
function UserList({ users }) {  
 return (  
 <ul>  
 {users.map(user => (  
 <li key={user.id}>{user.name}</li>  
 ))}  
 </ul>  
 );  
}  
  
// Using unique combination  
function OrderItems({ items, orderId }) {  
 return (  
 <div>  
 {items.map(item => (  
 <div key={`${orderId}-${item.productId}`}>  
 {item.productName}: {item.quantity}  
 </div>  
 ))}  
 </div>  
 );  
}

**Bad Key Examples**

// BAD: Using array index (when list can change)  
function BadList({ items }) {  
 return (  
 <ul>  
 {items.map((item, index) => (  
 <li key={index}>{item.name}</li> // Don't do this!  
 ))}  
 </ul>  
 );  
}  
  
// BAD: Using Math.random()  
function VeryBadList({ items }) {  
 return (  
 <ul>  
 {items.map(item => (  
 <li key={Math.random()}>{item.name}</li> // Never do this!  
 ))}  
 </ul>  
 );  
}

**When Array Index is Acceptable**

// OK: When list is static and never changes  
function StaticList() {  
 const staticItems = ['Apple', 'Banana', 'Orange'];  
   
 return (  
 <ul>  
 {staticItems.map((item, index) => (  
 <li key={index}>{item}</li> // OK for static lists  
 ))}  
 </ul>  
 );  
}

**Explain How to Extract Components with Keys**

**Extracting components with keys** means creating separate, reusable components from list items while properly handling the key attribute.

**Basic Extraction**

// Before extraction  
function BookList({ books }) {  
 return (  
 <div>  
 {books.map(book => (  
 <div key={book.id} className="book-item">  
 <h3>{book.title}</h3>  
 <p>Author: {book.author}</p>  
 <p>Price: ${book.price}</p>  
 <button>Add to Cart</button>  
 </div>  
 ))}  
 </div>  
 );  
}  
  
// After extraction  
function BookItem({ book, onAddToCart }) {  
 return (  
 <div className="book-item">  
 <h3>{book.title}</h3>  
 <p>Author: {book.author}</p>  
 <p>Price: ${book.price}</p>  
 <button onClick={() => onAddToCart(book.id)}>Add to Cart</button>  
 </div>  
 );  
}  
  
function BookList({ books, onAddToCart }) {  
 return (  
 <div>  
 {books.map(book => (  
 <BookItem   
 key={book.id} // Key stays in the parent component  
 book={book}   
 onAddToCart={onAddToCart}  
 />  
 ))}  
 </div>  
 );  
}

**Advanced Component Extraction**

// Complex list item extracted to component  
function CommentItem({ comment, onReply, onLike, onDelete, currentUser }) {  
 const isOwner = comment.userId === currentUser.id;  
   
 return (  
 <div className="comment-item">  
 <div className="comment-header">  
 <img src={comment.author.avatar} alt="Avatar" />  
 <span className="author">{comment.author.name}</span>  
 <span className="timestamp">{comment.timestamp}</span>  
 </div>  
   
 <div className="comment-content">  
 <p>{comment.text}</p>  
 </div>  
   
 <div className="comment-actions">  
 <button onClick={() => onLike(comment.id)}>  
 Like ({comment.likes})  
 </button>  
 <button onClick={() => onReply(comment.id)}>  
 Reply  
 </button>  
 {isOwner && (  
 <button onClick={() => onDelete(comment.id)}>  
 Delete  
 </button>  
 )}  
 </div>  
 </div>  
 );  
}  
  
function CommentList({ comments, currentUser, onReply, onLike, onDelete }) {  
 return (  
 <div className="comment-list">  
 {comments.map(comment => (  
 <CommentItem  
 key={comment.id}  
 comment={comment}  
 currentUser={currentUser}  
 onReply={onReply}  
 onLike={onLike}  
 onDelete={onDelete}  
 />  
 ))}  
 </div>  
 );  
}

**Key Points for Component Extraction**

1. **Key attribute** must be on the extracted component, not inside it
2. **Pass necessary data** as props to the extracted component
3. **Event handlers** should be passed down as props
4. **Keep extracted components reusable** and focused on single responsibility

**Explain React Map, map() Function**

The **map()** function is a JavaScript array method commonly used in React to transform arrays of data into arrays of JSX elements.

**Basic map() Syntax**

array.map((item, index, array) => {  
 // Return JSX element  
 return <Component key={item.id} data={item} />;  
})

**Simple map() Examples**

// Basic string array  
function FruitList() {  
 const fruits = ['Apple', 'Banana', 'Orange', 'Grape'];  
   
 return (  
 <ul>  
 {fruits.map((fruit, index) => (  
 <li key={index}>{fruit}</li>  
 ))}  
 </ul>  
 );  
}  
  
// Number array with calculations  
function NumberList() {  
 const numbers = [1, 2, 3, 4, 5];  
   
 return (  
 <ul>  
 {numbers.map(number => (  
 <li key={number}>  
 {number} squared is {number \* number}  
 </li>  
 ))}  
 </ul>  
 );  
}

**Complex Object Mapping**

function EmployeeList() {  
 const employees = [  
 { id: 1, name: 'John Doe', department: 'IT', salary: 50000 },  
 { id: 2, name: 'Jane Smith', department: 'HR', salary: 45000 },  
 { id: 3, name: 'Mike Johnson', department: 'Finance', salary: 55000 }  
 ];  
   
 return (  
 <div className="employee-list">  
 <h2>Employee Directory</h2>  
 {employees.map(employee => (  
 <div key={employee.id} className="employee-card">  
 <h3>{employee.name}</h3>  
 <p>Department: {employee.department}</p>  
 <p>Salary: ${employee.salary.toLocaleString()}</p>  
 </div>  
 ))}  
 </div>  
 );  
}

**map() with Conditional Rendering**

function TaskList({ tasks }) {  
 return (  
 <div className="task-list">  
 {tasks.map(task => (  
 <div   
 key={task.id}   
 className={`task ${task.completed ? 'completed' : 'pending'}`}  
 >  
 <h4>{task.title}</h4>  
 <p>{task.description}</p>  
 {task.priority === 'high' && (  
 <span className="priority-badge">High Priority</span>  
 )}  
 <div className="task-status">  
 Status: {task.completed ? 'Completed' : 'Pending'}  
 </div>  
 </div>  
 ))}  
 </div>  
 );  
}

**map() with Filter**

function FilteredProductList({ products, category }) {  
 return (  
 <div className="product-list">  
 {products  
 .filter(product => category === 'all' || product.category === category)  
 .map(product => (  
 <div key={product.id} className="product-card">  
 <img src={product.image} alt={product.name} />  
 <h3>{product.name}</h3>  
 <p>Category: {product.category}</p>  
 <p>Price: ${product.price}</p>  
 <p>Rating: {product.rating}/5</p>  
 </div>  
 ))}  
 </div>  
 );  
}

**Nested map() for Complex Structures**

function CategoryList({ categories }) {  
 return (  
 <div className="category-list">  
 {categories.map(category => (  
 <div key={category.id} className="category-section">  
 <h2>{category.name}</h2>  
 <div className="items-grid">  
 {category.items.map(item => (  
 <div key={item.id} className="item-card">  
 <h4>{item.name}</h4>  
 <p>{item.description}</p>  
 <span className="price">${item.price}</span>  
 </div>  
 ))}  
 </div>  
 </div>  
 ))}  
 </div>  
 );  
}

**Complete Example: Todo App with map()**

function TodoApp() {  
 const [todos, setTodos] = useState([  
 { id: 1, text: 'Learn React', completed: false, priority: 'high' },  
 { id: 2, text: 'Build a project', completed: false, priority: 'medium' },  
 { id: 3, text: 'Deploy app', completed: true, priority: 'low' }  
 ]);  
  
 const toggleTodo = (id) => {  
 setTodos(todos.map(todo =>  
 todo.id === id ? { ...todo, completed: !todo.completed } : todo  
 ));  
 };  
  
 return (  
 <div className="todo-app">  
 <h1>My Todo List</h1>  
 <div className="todo-list">  
 {todos.map(todo => (  
 <div   
 key={todo.id}   
 className={`todo-item ${todo.completed ? 'completed' : ''}`}  
 >  
 <input  
 type="checkbox"  
 checked={todo.completed}  
 onChange={() => toggleTodo(todo.id)}  
 />  
 <span className="todo-text">{todo.text}</span>  
 <span className={`priority ${todo.priority}`}>  
 {todo.priority}  
 </span>  
 </div>  
 ))}  
 </div>  
 </div>  
 );  
}