Complete React Interview Guide (2025)

React and JSX Syntax and Rendering Elements

Q1: What is JSX and how is it transpiled?

Keywords: JSX, Babel, React.createElement

Explanation: JSX is a syntax extension that looks like HTML in JavaScript. It is transpiled by Babel into React.createElement() calls.

Real-life Example: Example: <h1>Hello</h1> becomes React.createElement('h1', null, 'Hello')

Q2: What is the Virtual DOM and how does React use it?

Keywords: Virtual DOM, diffing, performance

Explanation: The Virtual DOM is a lightweight copy of the real DOM. React uses it to efficiently update only changed parts of the UI.

Real-life Example: Typing into an input only updates that input, not the entire page.

Q3: Why must JSX return a single root element?

Keywords: Fragment, JSX rules

Explanation: JSX must return a single root element, so multiple siblings need to be wrapped in a div or React.Fragment.

Real-life Example: Using <> <h1/> <h2/> </> instead of two separate tags.

Q4: How are props embedded into JSX?

Keywords: JSX, props, expressions

Explanation: Props are embedded using curly braces. These allow dynamic rendering.

Real-life Example: Example: <Greeting name={userName} />

Q5: What happens during React rendering?

Keywords: Rendering, reconciliation, Virtual DOM

Explanation: React re-renders components when props/state change, updating the Virtual DOM and syncing it with the real DOM.

Real-life Example: Changing a counter value re-renders only that part of the UI.

React Components and Props

Q1: What are React components?

Keywords: Function, Class, Reusability

Explanation: Components are reusable UI elements, written as functions or classes.

Real-life Example: Example: <Navbar />, <Footer />

Q2: Difference between class and function components?

Keywords: Hooks, lifecycle, simplicity

Explanation: Function components use hooks and are preferred today. Class components use lifecycle methods.

Real-life Example: Function component: useEffect vs class: componentDidMount

Q3: How are props passed and accessed?

Keywords: Props, unidirectional data flow

Explanation: Props are passed from parent to child via attributes and accessed using 'props' inside the child.

Real-life Example: Example: <Profile name='Priya' />

Q4: Can components be nested?

Keywords: Composition, nesting

Explanation: Yes, components can be composed together to build complex UIs.

Real-life Example: Example: Dashboard has Sidebar, Header, and Content components.

Q5: How to validate props?

Keywords: PropTypes, type checking

Explanation: Use the prop-types package to define expected prop types.

Real-life Example: Example: name: PropTypes.string.isRequired

Conditional and List Rendering, and State in React

Q1: How do you conditionally render elements in React?

Keywords: Conditionals, &&, ternary operator

Explanation: Use JavaScript expressions like ternary (a ? b : c) or && to render components conditionally.

Real-life Example: Example: isLoggedIn ? <Logout /> : <Login />

Q2: How do you render lists in React?

Keywords: Array.map(), keys, dynamic rendering

Explanation: Use the .map() method to render lists of elements. Each child needs a unique key.

Real-life Example: Example: todos.map(todo => <li key={todo.id}>{todo.text}</li>)

Q3: Why are keys important in list rendering?

Keywords: Keys, identity, reconciliation

Explanation: Keys help React identify which items changed or moved, improving efficiency.

Real-life Example: Using 'key={item.id}' in a list of user cards

Q4: What is state in React?

Keywords: State, useState, local data

Explanation: State is data maintained by the component. It changes over time using useState.

Real-life Example: const [count, setCount] = useState(0)

Q5: How does state updating trigger re-renders?

Keywords: setState, triggers re-render

Explanation: When state is updated, React re-renders the component to reflect changes.

Real-life Example: Clicking a button updates the count on screen

Handling Events, React Fragments and Keys

Q1: How are events handled in React?

Keywords: Synthetic events, camelCase

Explanation: React uses synthetic events. Handlers are written in camelCase.

Real-life Example: onClick={handleClick} for a button

Q2: What is the difference between HTML and React event handling?

Keywords: Synthetic event, preventDefault

Explanation: React events use camelCase and handle event objects via SyntheticEvent.

Real-life Example: event.preventDefault() in form submission

Q3: What are React Fragments?

Keywords: Fragment, avoid extra DOM nodes

Explanation: Fragments let you group elements without adding extra nodes to the DOM.

Real-life Example: Return <> <h1/> <p/> </> instead of wrapping in a <div>

Q4: Why use keys in lists?

Keywords: Keys, efficient diffing

Explanation: Keys help React track elements across renders for better performance.

Real-life Example: Use stable IDs like post.id, not index

Q5: Can event handlers be passed as props?

Keywords: Function props, event delegation

Explanation: Yes, parent components can pass event handler functions as props to children.

Real-life Example: onDelete={handleDelete} passed from parent to a Card component

useEffect Hook, useState and useEffect

Q1: What is the useState hook?

Keywords: State hook, functional components

Explanation: useState is used to declare and manage state in function components.

Real-life Example: const [value, setValue] = useState('')

Q2: What is the useEffect hook used for?

Keywords: Side effects, lifecycle replacement

Explanation: useEffect handles side effects like data fetching, subscriptions, or DOM updates.

Real-life Example: useEffect(() => { fetchData() }, [])

Q3: When does useEffect run?

Keywords: Dependencies, lifecycle phases

Explanation: It runs after render and re-runs when its dependency array changes.

Real-life Example: [] runs once, [count] runs when 'count' changes

Q4: How to clean up in useEffect?

Keywords: Cleanup function, unmount, clearInterval

Explanation: Return a cleanup function from useEffect to run on unmount.

Real-life Example: return () => clearInterval(timerId)

Q5: How to fetch data in useEffect?

Keywords: Fetch, async call, useEffect

Explanation: Place fetch inside useEffect. Use async/await with care inside.

Real-life Example: useEffect(() => { fetchData() }, [])

useContext Hook and Custom Hook

Q1: What is the useContext hook?

Keywords: Context API, global data

Explanation: useContext allows access to context values without prop drilling.

Real-life Example: const theme = useContext(ThemeContext)

Q2: How is context different from props?

Keywords: Global vs local, prop drilling

Explanation: Props are local and passed down manually. Context provides global access.

Real-life Example: Using AuthContext to access user data in multiple components

Q3: How do you create a context?

Keywords: React.createContext, Provider

Explanation: Create with React.createContext(), and wrap components in Provider.

Real-life Example: const ThemeContext = React.createContext()

Q4: What is a custom hook?

Keywords: Reusable logic, naming convention

Explanation: A custom hook is a function that uses other hooks to encapsulate logic.

Real-life Example: function useWindowSize() { ... }

Q5: When should you create custom hooks?

Keywords: Code reuse, abstraction

Explanation: When logic is reused across components, abstract it into a custom hook.

Real-life Example: Extracting form input logic into useForm()