React JS

Interview Questions & Answers

a1training167@gmail.com

www.a1training.in

8368979712

6380486914

C-167,Omicron 1,6%Abadi,Greater Noida / Earthcon Sanakriti, Noida Extension

# 75 React Interview Questions and Answers: Beginner to Advanced

## Beginner Level

1. What is React?

Answer: React is a JavaScript library for building user interfaces, particularly single-page applications. It's used for handling the view layer in web and mobile apps.

2. What are the key features of React?

Answer: Key features of React include: Virtual DOM for efficient updating, component-based architecture, unidirectional data flow, JSX syntax, and support for server-side rendering.

3. What is JSX?

Answer: JSX is a syntax extension for JavaScript, recommended for use with React. It allows you to write HTML-like code in your JavaScript files, making it easier to describe what the UI should look like.

4. What is the virtual DOM?

Answer: The virtual DOM is a lightweight copy of the actual DOM in memory. React uses it to improve performance by minimizing direct manipulation of the DOM, instead making changes to the virtual DOM and then efficiently updating the real DOM.

5. What is the difference between state and props?

Answer: Props (short for properties) are read-only and are passed from parent to child components. State is managed within the component and can be changed using setState().

6. How do you create a React component?

Answer: You can create a React component by defining a JavaScript function that returns JSX (functional component) or by creating a class that extends React.Component (class component).

7. What is the difference between a functional component and a class component?

Answer: Functional components are simpler, just JavaScript functions that return JSX. Class components are more feature-rich, allowing use of lifecycle methods and local state (prior to the introduction of Hooks).

8. How do you pass data from parent to child components?

Answer: Data is passed from parent to child components via props. The parent component sets properties on the child component when rendering it.

9. What is the purpose of `render()` in React?

Answer: The render() method is required in class components. It returns the JSX that describes what should be rendered on the screen.

10. What are React Hooks?

Answer: Hooks are functions that let you use state and other React features in functional components, without writing a class. They were introduced in React 16.8.

11. Explain the `useState` Hook.

Answer: useState is a Hook that lets you add state to functional components. It returns an array with the current state value and a function to update it.

12. What is the `useEffect` Hook used for?

Answer: useEffect is used for side effects in functional components. It serves a similar purpose to componentDidMount, componentDidUpdate, and componentWillUnmount in class components.

13. How do you handle events in React?

Answer: Events in React are handled using camelCase naming (e.g., onClick instead of onclick) and passing a function as the event handler rather than a string.

14. What is conditional rendering in React?

Answer: Conditional rendering in React allows you to render different UI elements based on certain conditions. This can be done using if statements, ternary operators, or logical && operator.

15. How do you apply CSS styles in React?

Answer: CSS can be applied in React using inline styles (as a JavaScript object), by importing CSS files, or using CSS-in-JS libraries like styled-components.

16. What is the significance of keys in React lists?

Answer: Keys help React identify which items have changed, been added, or been removed in lists. They should be given to the elements inside the array to give the elements a stable identity.

17. What is prop drilling and how can it be avoided?

Answer: Prop drilling occurs when props need to be passed through multiple levels of components. It can be avoided using Context API, Redux, or composition.

18. How do you create a controlled component?

Answer: A controlled component is one where form data is handled by the React component’s state. You create it by setting the value of the form element to a state variable and updating that state in an onChange handler.

19. What is the difference between controlled and uncontrolled components?

Answer: In controlled components, form data is handled by React state. In uncontrolled components, form data is handled by the DOM itself.

20. How do you handle forms in React?

Answer: Forms in React are typically handled using controlled components, where form inputs are controlled by React state. You use onChange handlers to update the state as the user types.

## Intermediate Level

21. What is the Context API?

Answer: The Context API provides a way to pass data through the component tree without having to pass props down manually at every level. It's designed to share data that can be considered "global" for a tree of React components.

22. Explain the concept of lifting state up.

Answer: Lifting state up is a pattern used when several components need to share the same changing data. Instead of keeping the state in child components, you move it up to their closest common ancestor.

23. What are Higher-Order Components (HOCs)?

Answer: A Higher-Order Component is a function that takes a component and returns a new component with some additional functionality. It's a way to reuse component logic across multiple components.

24. What is the purpose of `React.memo()`?

Answer: React.memo() is a higher-order component that can be used to optimize functional components by memoizing the result. It prevents unnecessary re-renders if the props haven't changed.

25. How does React handle routing?

Answer: React doesn't have built-in routing. Routing in React applications is typically handled by libraries like React Router, which allows you to define routes and map them to components.

26. What is code-splitting in React?

Answer: Code-splitting is a technique to split your code into small chunks which can then be loaded on demand or in parallel. It can be used to improve performance by reducing the size of the initial bundle.

27. Explain the `useCallback` Hook.

Answer: useCallback is a hook that returns a memoized version of the callback function that only changes if one of the dependencies has changed. It's useful for optimizing performance in child components.

28. What is the purpose of the `useMemo` Hook?

Answer: useMemo is used to memoize expensive computations so that they are only re-computed when one of their dependencies changes. This can help to optimize performance.

29. How do you optimize performance in React applications?

Answer: Performance in React can be optimized by using React.memo for functional components, PureComponent for class components, code-splitting, lazy loading, using keys correctly in lists, and avoiding unnecessary re-renders.

30. What are React Portals?

Answer: Portals provide a way to render children into a DOM node that exists outside the DOM hierarchy of the parent component. They're often used for modals, tooltips, etc.

31. How do you handle error boundaries in React?

Answer: Error boundaries are React components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI instead of the component tree that crashed.

32. What is the significance of the `key` prop when rendering lists?

Answer: The key prop helps React identify which items in a list have changed, been added, or been removed. It should be a unique identifier for each item in the list.

33. How do you implement server-side rendering with React?

Answer: Server-side rendering can be implemented using frameworks like Next.js, or manually by using ReactDOMServer's renderToString method on the server and hydrating the app on the client.

34. What is the difference between `createElement` and `cloneElement`?

Answer: createElement is used to create new React elements, while cloneElement is used to clone and return a new React element using an existing element as the starting point.

35. How do you use refs in React?

Answer: Refs provide a way to access DOM nodes or React elements created in the render method. They can be created using React.createRef() and attached to React elements via the ref attribute.

36. What is the purpose of the `useRef` Hook?

Answer: useRef returns a mutable ref object whose .current property is initialized to the passed argument. It's commonly used to access DOM elements directly or to keep mutable values across renders without causing re-renders.

37. How do you implement form validation in React?

Answer: Form validation in React can be implemented using controlled components, where you validate the input in the onChange handler and store the validation state. Libraries like Formik can also be used for more complex forms.

38. What are React Fragments and when would you use them?

Answer: React Fragments let you group a list of children without adding extra nodes to the DOM. They're useful when you need to return multiple elements from a component's render method.

39. How do you handle asynchronous operations in React?

Answer: Asynchronous operations in React can be handled using useEffect for side effects, async/await syntax, or libraries like Redux-Thunk or Redux-Saga for more complex state management.

40. Explain the concept of render props.

Answer: Render props is a technique for sharing code between React components using a prop whose value is a function. It allows component logic to be used by other components.

## Advanced Level

41. What is the React Fiber architecture?

Answer: React Fiber is the new reconciliation algorithm in React 16. It allows for incremental rendering of the virtual DOM, enabling smoother UI updates and better performance for complex applications.

42. How does React's reconciliation algorithm work?

Answer: React's reconciliation algorithm compares the new virtual DOM tree with the previous one, determines the differences, and makes the minimum number of changes to the actual DOM for efficiency.

43. What are the advantages of using Suspense in React?

Answer: Suspense allows components to "wait" for something before rendering, making it easier to handle asynchronous operations. It's particularly useful for code-splitting and data fetching.

44. How do you implement code-splitting with React.lazy and Suspense?

Answer: React.lazy lets you dynamically import components. Wrap the lazy-loaded component with Suspense, which will show a fallback while the component is loading.

45. What is the purpose of the `useLayoutEffect` Hook?

Answer: useLayoutEffect is similar to useEffect, but it fires synchronously after all DOM mutations. It's useful for DOM measurements and mutations that need to be done before the browser paints.

46. How do you create a custom Hook?

Answer: Custom Hooks are created by extracting component logic into reusable functions. They can call other Hooks and should start with the word “use” to follow the Hook naming convention.

47. What are the differences between `useEffect` and `useLayoutEffect`?

Answer: useEffect runs asynchronously after render is committed to the screen, while useLayoutEffect runs synchronously immediately after React has performed all DOM mutations.

48. How do you implement a debounce function in React?

Answer: A debounce function can be implemented using useCallback and useEffect Hooks, along with setTimeout to delay the execution of a function.

49. What is the purpose of the `useImperativeHandle` Hook?

Answer: useImperativeHandle customizes the instance value that is exposed to parent components when using ref. It’s used with forwardRef to expose custom methods to parent components.

50. How do you optimize React Context for performance?

Answer: React Context can be optimized by splitting the context into smaller pieces, using useMemo to memoize the context value, and using useCallback for functions passed through context.

51. Explain the concept of render-as-you-fetch using Suspense.

Answer: Render-as-you-fetch is a pattern where you start fetching data as soon as you have the information you need to do so, potentially before rendering. Suspense then allows you to wait for this data in your render logic.

52. How do you implement infinite scrolling in React?

Answer: Infinite scrolling can be implemented using intersection observer API or libraries like react-infinite-scroll-component, along with state management to keep track of loaded items.

53. What are the best practices for state management in large React applications?

Answer: Best practices include using Redux or MobX for global state, Context API for shared state, local component state for UI state, and considering server state management tools like React Query.

54. How do you handle memory leaks in React components?

Answer: Memory leaks can be prevented by properly cleaning up subscriptions and timers in the useEffect cleanup function or componentWillUnmount lifecycle method.

55. What is the purpose of the `getDerivedStateFromProps` lifecycle method?

Answer: getDerivedStateFromProps is used to update the state of a component based on changes in props over time. It’s called every time a component is re-rendered.

56. How do you implement a compound component pattern in React?

Answer: Compound components are implemented by creating a main parent component that manages shared state and behavior, and child components that consume this shared state using Context or props.

57. What are the differences between React's `useState` and `useReducer` Hooks?

Answer: useState is simpler and good for managing independent pieces of state, while useReducer is preferable for complex state logic, especially when the next state depends on the previous one.

58. How do you implement authentication in a React application?

Answer: Authentication in React typically involves managing user state (often in Context or Redux), protecting routes, and sending authentication tokens with API requests.

59. What are the security considerations when working with React?

Answer: Security considerations include protecting against XSS by using JSX, securing API calls, implementing proper authentication and authorization, and being cautious with third-party libraries.

60. How do you handle internationalization (i18n) in React applications?

Answer: Internationalization can be handled using libraries like react-intl or react-i18next, which provide components and hooks for translating text and formatting dates, numbers, etc.

Here's the requested code block format, with all spaces between the lines removed and no JSX tags. The indentation is preserved for better readability when pasted into Medium:

Coding Examples

61. Write a simple counter component using the `useState` Hook.

```
import React, { useState } from 'react';
function Counter() {
 const [count, setCount] = useState(0);
 return (
 <div>
 <p>Count: {count}</p>
 <button onClick={() => setCount(count + 1)}>Increment</button>
 </div>
 );
}
```

62. Implement a basic form with React.

```
import React, { useState } from ‘react’;
function SimpleForm() {
 const [formData, setFormData] = useState({ name: ‘’, email: ‘’ });
 const handleChange = (e) => {
 setFormData({ ...formData, [e.target.name]: e.target.value });
 };
 const handleSubmit = (e) => {
 e.preventDefault();
 console.log(‘Form submitted:’, formData);
 };
 return (
 <form onSubmit={handleSubmit}>
 <input type=”text” name=”name” value={formData.name} onChange={handleChange}
placeholder=”Name” />
 <input type=”email” name=”email” value={formData.email} onChange={handleChange}
placeholder=”Email” />
 <button type=”submit”>Submit</button>
 </form>
 );
}
```

63. Create a custom Hook for fetching data.

```
import { useState, useEffect } from ‘react’;
function useFetch(url) {
 const [data, setData] = useState(null);
 const [loading, setLoading] = useState(true);
 const [error, setError] = useState(null);

 useEffect(() => {
 fetch(url)
 .then((response) => response.json())
 .then((data) => {
 setData(data);
 setLoading(false);
 })
 .catch((error) => {
 setError(error);
 setLoading(false);
 });
 }, [url]);
return { data, loading, error };
}
```

64. Implement a simple React Context.

```
import React, { createContext, useContext, useState } from ‘react’;
const ThemeContext = createContext();
export function ThemeProvider({ children }) {
 const [theme, setTheme] = useState(‘light’);
 return <ThemeContext.Provider value={{ theme, setTheme
}}>{children}</ThemeContext.Provider>;
}
export function useTheme() {
 return useContext(ThemeContext);
}
```

65. Create a Higher-Order Component for loading state.

```
function withLoading(WrappedComponent) {
 return function WithLoadingComponent({ isLoading, ...props }) {
 if (isLoading) return <div>Loading...</div>;
 return <WrappedComponent {...props} />;
 };
}
// Usage
const ComponentWithLoading = withLoading(MyComponent);
```

66. Implement a basic routing setup using React Router.

```
import React from ‘react’;
import { BrowserRouter as Router, Route, Link, Switch } from ‘react-router-dom’;
function App() {
 return (
 <Router>
 <nav>
 <ul>
 <li><Link to=”/”>Home</Link></li>
 <li><Link to=”/about”>About</Link></li>
 <li><Link to=”/contact”>Contact</Link></li>
 </ul>
 </nav>
 <Switch>
 <Route exact path="/" component={Home} />
 <Route path="/about" component={About} />
 <Route path="/contact" component={Contact} />
 </Switch>
 </Router>
 );
}
```

67. Create a simple animation using React Spring.

```
import React from 'react';
import { useSpring, animated } from 'react-spring';
function AnimatedComponent() {
 const props = useSpring({ opacity: 1, from: { opacity: 0 } });
 return <animated.div style={props}>I will fade in</animated.div>;
}
```

68. Implement a basic Redux setup with React.

```
// actions.js
export const increment = () => ({ type: 'INCREMENT' });
export const decrement = () => ({ type: 'DECREMENT' });
// reducer.js
const initialState = { count: 0 };
function counterReducer(state = initialState, action) {
 switch (action.type) {
 case 'INCREMENT':
 return { ...state, count: state.count + 1 };
 case 'DECREMENT':
 return { ...state, count: state.count — 1 };
 default:
 return state;
 }
}
// Component
import React from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { increment, decrement } from './actions';
function Counter() {
 const count = useSelector((state) => state.count);
 const dispatch = useDispatch();
 return (
 <div>
 <p>Count: {count}</p>
 <button onClick={() => dispatch(increment())}>Increment</button>
 <button onClick={() => dispatch(decrement())}>Decrement</button>
 </div>
 );
}
```

69. Use the `useReducer` Hook for complex state management.

```
import React, { useReducer } from 'react';
const initialState = { count: 0 };
function reducer(state, action) {
 switch (action.type) {
 case 'increment':
 return { count: state.count + 1 };
 case 'decrement':
 return { count: state.count — 1 };
 default:
 throw new Error();
 }
}
function Counter() {
 const [state, dispatch] = useReducer(reducer, initialState);
 return (
 <>
 Count: {state.count}
 <button onClick={() => dispatch({ type: 'increment' })}>+</button>
 <button onClick={() => dispatch({ type: 'decrement' })}>-</button>
 </>
 );
}
```