React.js interview questions answers

Basic React.js Questions

1. What is React.js?

Answer:
React.js is an open-source JavaScript library for building user interfaces, primarily for single-page applications. It allows developers to create reusable UI components and manage the view layer efficiently.

2. What are the key features of React?

Answer:

- **JSX** – JavaScript XML, used to write HTML in React.
- **Components** – Reusable and independent UI building blocks.
- **Virtual DOM** – Enhances performance by updating only changed elements.
- **One-way Data Binding** – Unidirectional data flow for better control.
- **State and Props** – Manages component data dynamically.
- **Hooks** – Enable state management and lifecycle methods in functional components.

3. What is JSX?

Answer:
JSX (JavaScript XML) is a syntax extension for JavaScript that allows writing HTML elements in JavaScript and placing them in the DOM.

4. What is the difference between functional and class components?

Answer:

Feature	Functional Component	Class Component
Syntax	Function-based	Class-based (ES6)
State Management	Uses `useState` hook	Uses `this.state`
Lifecycle Methods	Uses hooks like `useEffect`	Uses `componentDidMount`, etc.
Performance	Faster, less complex	Slightly heavier

5. What are props in React?

Answer:
Props (short for "properties") are read-only inputs passed from a parent component to a child component.

6. What is the use of state in React?

Answer:
State is a built-in object in a React component that holds dynamic data and controls how the component behaves.

7. How do you update the state in a class component?

Answer:
Using `this.setState()` method:

```
this.setState({ count: this.state.count + 1 });
```

8. What is the virtual DOM?

Answer:
The Virtual DOM is a lightweight JavaScript representation of the actual DOM. React updates the Virtual DOM first and then efficiently updates only the changed parts of the real DOM.

9. What are React hooks?

Answer:
Hooks are functions that allow functional components to use state and lifecycle features. Example:

- `useState` – for state management.
- `useEffect` – for side effects (e.g., fetching data).

10. What is the difference between state and props?

Answer:

Feature	Props	State
Mutability	Immutable (read-only)	Mutable (can be updated)
Access	Passed from parent to child	Defined within the component
Usage	Used to pass data	Used to manage component data

Intermediate React.js Questions

11. How do you pass data between components in React?

Answer:

- **Parent to Child:** Using `props`
- **Child to Parent:** Using callback functions
- **Global State:** Using React Context or Redux

12. What is React Context API?

Answer:
React Context API allows sharing global data across components without prop drilling.

```
const MyContext = React.createContext();
<MyContext.Provider value={value}><Child /></MyContext.Provider>
```

13. How does React handle forms?

Answer:
Forms in React are controlled components where form elements have state management.

```
const [name, setName] = useState('');
<input type="text" value={name} onChange={(e) => setName(e.target.value)} />
```

14. What is React Router?

Answer:
React Router is a library for handling navigation in React applications.

```
import { BrowserRouter, Route } from 'react-router-dom';
<BrowserRouter>
  <Route path="/home" component={Home} />
</BrowserRouter>
```

15. What is the difference between useEffect and useLayoutEffect?

Answer:

- `useEffect` runs **after** the render.
- `useLayoutEffect` runs **before** the browser paints the screen.

16. What are controlled and uncontrolled components?

Answer:

- **Controlled:** State is controlled by React.
- **Uncontrolled:** Uses native DOM elements and refs.

Advanced React.js Questions

17. What is memoization in React?

Answer:
Memoization optimizes performance by caching results:

const MemoizedComponent = React.memo(MyComponent);

18. How do you handle performance optimization in React?

Answer:

- **Using `React.memo()`**
- **Using `useMemo()` and `useCallback()`**
- **Code splitting with React.lazy()**

19. What is server-side rendering (SSR) in React?

Answer:
SSR renders React components on the server instead of the browser.

20. What are higher-order components (HOC)?

Answer:
HOCs are functions that take a component and return an enhanced component.

```
const withLogger = (Component) => (props) => {
  console.log("Component rendered");
  return <Component {...props} />;
};
```

React.js Coding Questions

21. Create a simple React component that displays "Hello, World!".

Answer:

```
const HelloWorld = () => <h1>Hello, World!</h1>;
export default HelloWorld;
```

22. Implement a counter using React hooks.

```
import { useState } from 'react';

const Counter = () => {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
};
export default Counter;
```

23. How do you fetch data in React using `useEffect`?

```
import { useEffect, useState } from 'react';

const FetchData = () => {
  const [data, setData] = useState([]);

  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/posts")
      .then(response => response.json())
      .then(data => setData(data));
  }, []);

  return (
    <ul>{data.map(item => <li key={item.id}>{item.title}</li>)}</ul>
  );
};
export default FetchData;
```

24. How do you implement lazy loading in React?

```
import React, { Suspense, lazy } from "react";

const LazyComponent = lazy(() => import("./MyComponent"));

const App = () => (
  <Suspense fallback={<div>Loading...</div>}>
    <LazyComponent />
  </Suspense>
);
export default App;
```

Here are **React.js interview questions 25-100**, covering **intermediate to advanced** topics with coding examples:

React.js Interview Questions (25-100)

Component Lifecycle and Hooks

25. What are the lifecycle methods in a React class component?
Answer:

- `constructor()` → Initializes state
- `componentDidMount()` → Runs after first render
- `shouldComponentUpdate()` → Determines re-rendering
- `componentDidUpdate()` → Runs after state/props change
- `componentWillUnmount()` → Cleanup before unmounting

26. What are the differences between useEffect() and componentDidMount()?
Answer:

- `useEffect()` runs **after every render** by default.
- `componentDidMount()` runs only **once after the initial render**.

```
useEffect(() => {
  console.log("Component mounted!");
}, []);  // Empty dependency array mimics componentDidMount()
```

27. What is useRef()?
Answer:
`useRef()` creates a reference to DOM elements or stores values persistently across renders.

```
const inputRef = useRef(null);
<input ref={inputRef} />;
```

28. What is useReducer()?
Answer:
`useReducer()` is an alternative to `useState()` for complex state management.

```
const reducer = (state, action) => action.type === "INCREMENT" ? state + 1 : state;
const [count, dispatch] = useReducer(reducer, 0);
```

Performance Optimization

29. What is React.memo()?
Answer:
Prevents re-rendering of a component if props don't change.

```
const MemoizedComponent = React.memo(MyComponent);
```

30. What is useMemo()?
Answer:
`useMemo()` caches the result of expensive calculations.

```
const expensiveCalculation = useMemo(() => computeValue(a, b), [a, b]);
```

31. What is useCallback()?
Answer:
Prevents function recreation unless dependencies change.

```
const memoizedCallback = useCallback(() => computeValue(a, b), [a, b]);
```

Event Handling & Forms

32. What is synthetic event in React?
Answer:
React's wrapper around native events for cross-browser compatibility.

33. How to prevent default behavior in React event handlers?
Answer:

```
const handleSubmit = (e) => {
  e.preventDefault();
};
```

34. How to bind event handlers in class components?
Answer:

```
constructor() {
  this.handleClick = this.handleClick.bind(this);
}
```

React Routing

35. What is React Router?
Answer:
A library for navigation in React applications.

36. Difference between BrowserRouter and HashRouter?
Answer:

- **BrowserRouter** uses history API (e.g., `/about`).
- **HashRouter** uses hash (`#/about`).

```
<BrowserRouter>
  <Route path="/home" component={Home} />
</BrowserRouter>
```

37. How to implement dynamic routing in React?

```
<Route path="/user/:id" component={UserDetail} />
```

State Management (Redux, Context API)

38. What is Redux?
Answer:
A state management library that follows a unidirectional data flow.

39. What are the key components of Redux?
Answer:

- **Actions** → Objects describing state changes.
- **Reducers** → Functions that modify the state.
- **Store** → Holds the entire state tree.

```
const reducer = (state = 0, action) => action.type === "INCREMENT" ? state + 1 : state;
```

40. What is the Context API?
Answer:
Provides a way to share global state without prop drilling.

Handling Side Effects

41. What are side effects in React?
Answer:
Anything affecting outside React (e.g., fetching data, subscriptions).

42. How to clean up effects in useEffect()?
Answer:

```
useEffect(() => {
  const timer = setInterval(() => console.log("Tick"), 1000);
  return () => clearInterval(timer);
}, []);
```

Advanced Topics

43. What is code splitting in React?
Answer:
Divides code into smaller bundles for faster load times.

44. How to implement lazy loading in React?

```
const LazyComponent = React.lazy(() => import("./MyComponent"));
```

45. What is reconciliation in React?
Answer:
React's algorithm to update the UI efficiently.

Testing in React

46. What is Jest?
Answer:
A JavaScript testing framework for unit testing React components.

47. What is React Testing Library?
Answer:
A library for testing React components behavior.

```
render(<MyComponent />);
expect(screen.getByText("Hello")).toBeInTheDocument();
```

React Server Components & Next.js

48. What are React Server Components?
Answer:
Components that render on the server and send HTML to the client.

49. What is Next.js?
Answer:
A React framework for server-side rendering (SSR) and static site generation (SSG).

Here are the answers to **React Advanced Interview Questions (50-100):**

---

50. What is hydration in React?

Answer:
Hydration is the process where React **attaches event listeners** and reuses the HTML sent by the server (SSR) to make it interactive.

```
import { hydrateRoot } from 'react-dom/client';

hydrateRoot(document.getElementById('root'), <App />);
```

---

51. How does React handle errors?

Answer:
React uses **Error Boundaries** (class components with `componentDidCatch()`) to catch JavaScript errors in a component tree.

---

52. What are error boundaries?

Answer:
A special React component that **catches JavaScript errors** in child components and prevents the entire app from crashing.

```
class ErrorBoundary extends React.Component {

  componentDidCatch(error, info) {

    console.log("Error caught:", error);

  }
  render() {
    return this.props.children;
  }
}
```

---

53. What is forwardRef()?

Answer:
It allows passing **a ref** to a child component, enabling parent components to access a child’s DOM node.

```
const Input = React.forwardRef((props, ref) => <input ref={ref} {...props} />);
```

---

54. What is suspense in React?

Answer:
Suspense is used for **lazy loading components** and handling async data fetching.

```
const LazyComponent = React.lazy(() => import('./Component'));
<Suspense fallback={<div>Loading...</div>}>
  <LazyComponent />
</Suspense>
```

---

55. How does React handle accessibility (a11y)?

Answer:
React provides built-in support for **ARIA attributes** and semantic elements (`<button>`, `<label>`, etc.) for accessibility.

---

56. What is React Fiber?

Answer:
Fiber is the **reconciliation engine** in React that improves rendering performance with a new diffing algorithm.

---

57. What is the difference between controlled and uncontrolled inputs?

Answer:

- **Controlled inputs** → State managed by React.
- **Uncontrolled inputs** → Use **refs** to access DOM values.

```
const [value, setValue] = useState("");  // Controlled

const inputRef = useRef(null);  // Uncontrolled
```

---

58. How to debounce input handling in React?

Answer:
Debouncing delays function execution until after a specified delay.

```
const debounce = (fn, delay) => {

  let timeout;

  return (...args) => {

    clearTimeout(timeout);

    timeout = setTimeout(() => fn(...args), delay);

  };

};
```

---

59. How to throttle events in React?

Answer:
Throttling ensures a function is executed at most **once in a given period**.

```
const throttle = (func, limit) => {

  let inThrottle;

  return (...args) => {

    if (!inThrottle) {

      func(...args);

      inThrottle = true;

      setTimeout(() => (inThrottle = false), limit);

    }

  };

};
```

---

60. What is React Concurrent Mode?

Answer:
A new mode that improves rendering performance by allowing React to **interrupt rendering** and prioritize updates.

---

61. How to handle optimistic updates?

Answer:
Optimistic UI updates **immediately update UI** before API confirmation.

```
const handleSave = (newData) => {

  setData(newData); // Optimistic update

  apiCall(newData).catch(() => setData(oldData)); // Rollback on failure

};
```

---

62. Difference between useLayoutEffect and useEffect?

Answer:

- `useEffect` runs **after** the browser paints the screen.
- `useLayoutEffect` runs **before** the browser paints.

---

63. How to create a custom React hook?

Answer:

```
const useCounter = () => {

  const [count, setCount] = useState(0);

  return { count, increment: () => setCount(count + 1) };

};
```

---

64. Best way to structure a React project?

Answer:

- `/components` – Reusable UI components
- `/pages` – Page components
- `/hooks` – Custom hooks
- `/context` – Global state

---

65. What are fragments in React?

Answer:
Fragments `<></>` let you group elements **without adding extra DOM nodes**.

---

66. What is a portal in React?

Answer:
Portals render children **outside the root DOM tree**.

ReactDOM.createPortal(<Modal />, document.getElementById('modal-root'));

---

67. What are compound components?

Answer:
A pattern for creating flexible UI components.

const Tabs = ({ children }) => children;

---

68. Difference between CSR and SSR?

Answer:

- **CSR (Client-Side Rendering)** → Renders on the browser.
- **SSR (Server-Side Rendering)** → Renders on the server.

---

69. How to implement authentication in React?

Answer:
Use `Context API`, `Redux`, or `JWT` for authentication.

---

70. What is a Progressive Web App (PWA)?

Answer:
A web app with **offline support** using a service worker.

---

71. How to use WebSockets in React?

Answer:

const ws = new WebSocket("ws://example.com");

ws.onmessage = (event) => console.log(event.data);

---

72. What is an event bus in React?

Answer:
A pub-sub pattern for communication between components.

---

73. How to optimize large lists in React?

Answer:
Use **React Virtualized** for efficient list rendering.

---

74. What is rehydration in React?

Answer:
The process of attaching event listeners **after SSR**.

---

75. How to handle cookies and local storage?

Answer:
Use `localStorage.getItem()` and `document.cookie`.

---

76. What are static and dynamic imports?

Answer:

- **Static** → `import X from 'module'`
- **Dynamic** → `import('module')`

---

77. Purpose of key prop in lists?

Answer:
Helps React **identify** which items changed.

---

78. How does React handle memory leaks?

Answer:
By **cleaning up side effects** in `useEffect()`.

---

79. Difference between hydration and SSR?

Answer:
SSR generates **HTML**, hydration adds **interactivity**.

---

80. What are render props in React?

Answer:
A pattern for passing functions as props.

---

81. What is tree shaking in React?

Answer:
Removes **unused code** from the final bundle.

---

82. How does React handle animations?

Answer:
With **CSS transitions** or `react-spring`.

---

83. What is useImperativeHandle()?

Answer:
Customizes the instance returned by `useRef()`.

---

84. What is SWR in React?

Answer:
A data fetching library for **caching** API requests.

Here are the answers for **questions 85-100 on advanced React topics** 🚀

---

85. How to implement infinite scrolling?

Answer:
Use `IntersectionObserver` to detect when the user reaches the bottom and fetch more data.

```
const observer = new IntersectionObserver((entries) => {

  if (entries[0].isIntersecting) loadMoreData();

});
```

For libraries, use **react-infinite-scroll-component**.

---

86. What is a React suspense boundary?

Answer:
A wrapper around a component that **handles fallback UI** when an async component is loading.

```
<Suspense fallback={<div>Loading...</div>}>

  <LazyComponent />

</Suspense>
```

---

87. What are web workers in React?

Answer:
Web Workers allow running **background tasks** without blocking the UI.

```
const worker = new Worker('worker.js');

worker.postMessage('Start');
```

---

88. How to use React Query for data fetching?

Answer:
React Query is a data-fetching library that **caches, syncs, and updates** API calls efficiently.

```
const { data, error } = useQuery('fetchData', fetchData);
```

---

89. How to use Zustand for state management?

Answer:
Zustand is a **lightweight state management** alternative to Redux.

```
const useStore = create((set) => ({

  count: 0,

  increment: () => set((state) => ({ count: state.count + 1 })),

}));
```

---

90. What are micro-frontends in React?

Answer:
Micro-frontends **divide an app into smaller independent apps** that work together.

Example: Loading a micro-frontend inside a React app using **Module Federation**.

91. What are event bubbling and capturing?

Answer:

- **Bubbling** → Event propagates from the **child to parent**.
- **Capturing** → Event propagates from the **parent to child**.

```
<button onClick={(e) => e.stopPropagation()}>Click</button>
```

92. What are render cycles in React?

Answer:

- **Initial render** → Component mounts
- **Re-renders** → Caused by **state or props updates**
- **Unmounting** → Component is removed

93. What are React DevTools?

Answer:
A Chrome/Firefox extension to **inspect React components, props, and state**.

94. What is hydration mismatch?

Answer:
When **server-rendered HTML does not match** the client-side React tree.

To fix: Ensure **SSR and client-side state** match.

95. How to manage multiple themes in React?

Answer:
Use **CSS variables or context API**.

```
const ThemeContext = createContext('light');

<ThemeContext.Provider value="dark">

  <App />

</ThemeContext.Provider>
```

---

96. What is a singleton pattern in React?

Answer:
Ensures **only one instance** of a class or object exists.

Example: Creating a single **global store**.

```
const instance = null;

export const getInstance = () => instance || new SomeClass();
```

---

97. How to mock API calls in React tests?

Answer:
Use **Jest and MSW (Mock Service Worker)**.

```
jest.mock('./api', () => ({

  fetchData: jest.fn(() => Promise.resolve({ data: [] })),

}));
```

---

98. How to handle file uploads in React?

Answer:
Use `FormData` to send files via API.

```
const formData = new FormData();
```

formData.append("file", selectedFile);

fetch("/upload", { method: "POST", body: formData });

---

99. What is a service worker in React?

Answer:
A background script that **caches assets for offline use** in Progressive Web Apps (PWAs).

navigator.serviceWorker.register('/sw.js');

---

100. How to optimize React performance for mobile?

Answer:
✅ Use `React.memo()` to prevent unnecessary renders.
✅ Use **lazy loading** (`React.lazy()`).
✅ Optimize **images and assets**.
✅ Avoid unnecessary **re-renders** with `useCallback()` and `useMemo()`.