React interview

---

1. What is React, and what are its main features?

Answer:
React is an open-source JavaScript library developed by Facebook for building **user interfaces**, especially single-page applications.
Main features:

- **Declarative** → You describe what the UI should look like, and React updates it automatically when data changes.
- **Component-Based** → UI is built using reusable, independent components.
- **Virtual DOM** → Improves performance by updating only what changes, not the whole DOM.
- **Unidirectional Data Flow** → Data flows from parent to child, making debugging easier.
- **JSX** → Allows writing HTML-like syntax in JavaScript.

---

2. What is the difference between the Real DOM and the Virtual DOM?

Answer:

- **Real DOM**:
    - Directly updates the HTML document.
    - Slower because updating the DOM involves re-rendering the entire UI.
    - Creates a new DOM tree each time changes happen.
- **Virtual DOM**:
    - A lightweight copy of the Real DOM kept in memory.

- Faster, because React uses a diffing algorithm to compare changes and update only the changed parts in the Real DOM.
- Improves performance and efficiency.

---

3. What are props and state in React?

Answer:

- **Props**:
  - Short for *properties*.
  - Read-only data passed from a parent component to a child component.
  - Immutable inside the child component.
- **State**:
  - A built-in React object used to store data that may change over time.
  - Managed inside the component using `useState` (in functional components) or `this.state` (in class components).
  - Changes to state cause the component to re-render.

---

4. What are React Hooks? Name some commonly used ones.

Answer:
Hooks are functions introduced in React 16.8 that let you use state and other React features in functional components without writing classes.
Common Hooks:

- **useState** → Manages state in functional components.
- **useEffect** → Handles side effects (API calls, event listeners, etc.).
- **useContext** → Accesses context values without prop drilling.

- **useRef** → Accesses or stores mutable values without triggering re-renders.
- **useReducer** → Manages complex state logic.

---

5. What is the difference between controlled and uncontrolled components in React?

Answer:

- **Controlled Components**:
    - Form input elements whose values are controlled by React state.
    - Value is updated via `onChange` and stored in `state`.

Example:

```
 const [value, setValue] = useState("");
<input value={value} onChange={(e) => setValue(e.target.value)} />
```

    -
- **Uncontrolled Components**:
    - Form elements that manage their own state internally.
    - Values are accessed using `refs`.

Example:

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

    -

---

6. What is JSX, and why is it used in React?

Answer:
JSX (JavaScript XML) is a syntax extension for JavaScript that looks like HTML.

- It allows writing UI code directly in JavaScript.
- Makes code more readable and declarative.
- React uses Babel to compile JSX into `React.createElement()` calls.
  Example:

```
const element = <h1>Hello</h1>;
// Compiled to:
const element = React.createElement('h1', null, 'Hello');
```

---

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

Answer:

- **Functional Components**:
  - Plain JavaScript functions that return JSX.
  - Use React Hooks for state and lifecycle methods.
  - Easier to write and test.
- **Class Components**:
  - ES6 classes extending `React.Component`.
  - Use `this.state` for state and lifecycle methods like `componentDidMount`.
  - More verbose; less used after Hooks introduction.

---

8. What is the purpose of the `useEffect` Hook?

Answer:

- It lets you perform **side effects** in functional components.
- Side effects include data fetching, subscriptions, DOM manipulation, and timers.
- Runs after every render by default, but you can control it using a dependency array.
  Example:

```
useEffect(() => {
  console.log("Component mounted or updated");
}, []);
```

---

9. How does React handle forms?

Answer:

- Forms in React can be **controlled** or **uncontrolled**.
- In controlled forms, React state is the single source of truth.
- You use `value` and `onChange` to sync input with state.
- Example:

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

---

10. What is React Router, and why is it used?

Answer:
React Router is a standard library for routing in React applications.

- Allows navigation between pages without refreshing the browser.
- Uses the **Single Page Application** concept.

- Example:

```
import { BrowserRouter, Route, Routes } from 'react-router-dom';
<BrowserRouter>
  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/about" element={<About />} />
  </Routes>
</BrowserRouter>
```

---

11. What is the difference between `useMemo` and `useCallback`?

Answer:

- **useMemo** → Memoizes the **result** of a computation to avoid recalculating expensive values.
- **useCallback** → Memoizes the **function itself** to avoid recreating it on every render.
  Example:

```
const memoValue = useMemo(() => computeExpensive(a, b), [a, b]);
const memoFunc = useCallback(() => doSomething(a), [a]);
```

---

12. What is Context API in React?

Answer:
The Context API allows you to share data between components **without prop drilling**.

- Create context using `React.createContext()`.
- Provide data with `<Context.Provider>`.
- Consume with `useContext`.
  Example:

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

</MyContext.Provider>
const value = useContext(MyContext);

---

13. What is the difference between `React.Fragment` and a div wrapper?

Answer:

- **React.Fragment**:
    - Lets you group multiple elements without adding extra nodes to the DOM.
    - `<></>` is shorthand for `<React.Fragment>`.
- **div wrapper**:
    - Adds an extra HTML element to the DOM, which might affect styling or performance.

---

14. How does React's reconciliation (diffing) algorithm work?

Answer:

- React uses a **virtual DOM** to track UI changes.
- When state/props change, React:
    1. Creates a new virtual DOM tree.
    2. Compares it with the previous one (diffing).
    3. Updates only the changed nodes in the real DOM (patching).
- This process improves performance.

---

15. What is the difference between `key` and `id` in React lists?

Answer:

- **key**:
    - A special React prop used to identify elements in a list for efficient re-rendering.
    - Not visible in the DOM.
    - Should be unique **among siblings**.
- **id**:
    - A standard HTML attribute.
    - Visible in the DOM and can be accessed via CSS or JavaScript.

16. What are Higher-Order Components (HOCs) in React?

Answer:

- A Higher-Order Component is a **function that takes a component and returns a new component** with additional props or functionality.
- Used for **code reuse**, **state abstraction**, and **cross-cutting concerns** like authentication or logging.
  Example:

```
function withLogger(WrappedComponent) {
  return function(props) {
    console.log("Props:", props);
    return <WrappedComponent {...props} />;
  };
}
```

---

17. What are Error Boundaries in React?

Answer:

- Error Boundaries are React components that **catch JavaScript errors** in their child component tree and display a fallback UI instead of crashing the app.

- Implemented using **class components** with `componentDidCatch` and `getDerivedStateFromError`.
  Example:

```
class ErrorBoundary extends React.Component {
  state = { hasError: false };
  static getDerivedStateFromError() { return { hasError: true }; }
  componentDidCatch(error, info) { console.log(error, info); }
  render() {
    return this.state.hasError ? <h1>Something went wrong</h1> : this.props.children;
  }
}
```

---

18. What is React.lazy and Suspense?

Answer:

- **React.lazy** → Lets you load components **dynamically** (code-splitting) to improve performance.

- **Suspense** → Displays a fallback UI (like a loader) while the lazy component is loading.
  Example:

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

---

19. What are controlled side effects in React?

Answer:

- Controlled side effects are **synchronous** and happen during render, e.g., state updates caused by props changes (useEffect can be used to control them).

- They ensure UI stays in sync with state while avoiding unnecessary re-renders.

---

20. What is the difference between `useLayoutEffect` and `useEffect`?

Answer:

- **useEffect** → Runs **after** the render is committed to the screen (async). Good for data fetching, subscriptions, logging.
- **useLayoutEffect** → Runs **synchronously** after DOM mutations but before the browser paints. Good for DOM measurements or synchronizing layouts.

---

21. What is Prop Drilling, and how can it be avoided?

Answer:

- **Prop Drilling**: Passing data through multiple layers of components even when intermediate components don't need it.
- **Avoid it by**:
    - Using **Context API**
    - State management libraries (Redux, Zustand)
    - Composition patterns

---

22. What is the difference between `memo` and `useMemo`?

Answer:

- **React.memo** → Higher-order component that memoizes a **component** to prevent unnecessary re-renders.

- **useMemo** → Hook that memoizes a **calculation result**.
  Example:

```
const MyComp = React.memo(function MyComp(props) { ... });
const value = useMemo(() => expensiveCalc(a, b), [a, b]);
```

---

23. What are custom hooks in React?

Answer:

- A **custom hook** is a JavaScript function that uses built-in hooks and contains reusable logic.

- Naming convention: Must start with `use`.
  Example:

```
function useWindowWidth() {
  const [width, setWidth] = useState(window.innerWidth);
  useEffect(() => {
    const handleResize = () => setWidth(window.innerWidth);
    window.addEventListener("resize", handleResize);
    return () => window.removeEventListener("resize", handleResize);
  }, []);
  return width;
}
```

---

24. What is React's Strict Mode?

Answer:

- A wrapper `<React.StrictMode>` that helps detect potential problems in an application.

- Runs extra checks in development (e.g., double-invokes some lifecycle methods, warns about unsafe APIs).

- Does not affect production build.

---

25. How can you optimize performance in a React application?

Answer:

- Use **React.memo** to avoid unnecessary renders.
- Use **useCallback** and **useMemo** to memoize functions and values.
- Implement **code-splitting** with React.lazy and Suspense.
- Avoid **inline functions/objects** in frequently re-rendered components.
- Use the **React Profiler** to identify bottlenecks.

---

26. What is Redux, and how does it work with React?

Answer:

- Redux is a **state management library** for JavaScript apps.
- Works on the principle of:
    1. **Store** → Holds application state.
    2. **Action** → Plain JS object describing what happened.
    3. **Reducer** → Pure function that updates state based on the action.
- React connects to Redux using **react-redux** hooks like `useSelector` and `useDispatch`.

---

27. Difference between Redux and Context API?

Answer:

- **Context API**:
    - Built into React.
    - Good for small-scale state sharing.
    - Simpler but no built-in middleware or dev tools.
- **Redux**:
    - External library.
    - Good for large-scale apps with complex state changes.
    - Includes middleware, dev tools, predictable flow.

---

28. What is the difference between `localStorage`, `sessionStorage`, and cookies in React?

Answer:

- **localStorage** → Stores data with no expiry (persists even after browser close).
- **sessionStorage** → Stores data only until the browser tab is closed.
- **Cookies** → Can store small data, sent with every HTTP request, can have expiry dates, useful for authentication.

---

29. What is hydration in React?

Answer:

- Hydration is the process where React **attaches event listeners** to server-rendered HTML during client-side rendering.
- Common in **Server-Side Rendering (SSR)** with frameworks like Next.js.

---

30. What are portals in React?

Answer:

- Portals allow rendering children into a DOM node **outside the parent hierarchy**.
- Useful for modals, tooltips, pop-ups.
  Example:

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

---

31. What is reconciliation in React, and when does it happen?

Answer:

- Reconciliation is React's process of comparing the new virtual DOM with the old one and updating only the changed elements in the real DOM.
- Happens when:
    - State changes.
    - Props change.
    - Parent re-renders.

---

32. What are synthetic events in React?

Answer:

- Synthetic events are React's **cross-browser wrapper** around native browser events.
- Provide a consistent API across different browsers.
- Example: `onClick`, `onChange`, `onSubmit`.

---

33. What is the difference between `useRef` and `createRef`?

Answer:

- **useRef** → Used in functional components, persists the same ref object between renders.
- **createRef** → Used in class components, creates a new ref each time the component renders.

---

34. What are render props in React?

Answer:

- A pattern where a component's **children is a function** that returns JSX.
- Allows sharing logic between components without HOCs.
  Example:

<MouseTracker render={(pos) => <h1>{pos.x}, {pos.y}</h1>} />

---

35. What is the difference between server-side rendering (SSR) and client-side rendering (CSR) in React?

Answer:

- **SSR**:
    - HTML is generated on the server and sent to the browser.
    - Faster first load, better for SEO.
    - Example: Next.js.
- **CSR**:

    - HTML is generated in the browser using JavaScript.
    - Slower first load, but faster navigation after load.
    - Example: Create React App.

---

36. Difference between SPA (Single Page Application) and MPA (Multi Page Application)

- SPA loads only one HTML file and updates content dynamically (React apps are usually SPAs).
- MPA loads a new HTML page for each navigation.
- SPA → Faster navigation, MPA → Better for large SEO-heavy sites.

---

37. Lifting State Up

- Moving state from child components to a common parent so that multiple components can share the same state.
- Common in forms and shared UI states.

---

38. React's Key Prop Gotchas

- Keys should be **unique among siblings**, not globally unique.
- Using index as a key can cause UI bugs if the list changes order.

---

39. Difference between CSR, SSR, and SSG (Static Site Generation)

- CSR → Render in browser (Create React App).

- SSR → Render on server (Next.js).
- SSG → Pre-render HTML at build time (Next.js static export).

---

40. Debouncing and Throttling in React

- **Debouncing** → Delay function execution until after a specified wait time (good for search input).
- **Throttling** → Ensure a function runs at most once every set time (good for scroll events).

---

41. React Strict Mode's Double Rendering in Dev Mode

- In development, React renders components twice to detect side effects.
- Doesn't happen in production.

---

42. React Fiber Architecture

- The internal reimplementation of React's rendering engine to enable features like concurrent rendering, Suspense, and interruption.

---

43. Controlled vs Uncontrolled File Inputs

- File inputs are special because browsers don't allow setting the file path via state (security reasons).
- Use refs for uncontrolled file inputs.

---

44. useId Hook

- Introduced in React 18 to generate unique IDs that are consistent between server and client for accessibility and forms.

---

45. Handling Memory Leaks in React

- Cancel API requests in `useEffect` cleanup.
- Remove event listeners in cleanup.
- Avoid updating state after a component is unmounted.