Section 1 — React DOM & Rendering Internals (12 Questions)

1. What does ReactDOM.createRoot() return in React 18?
   1. The virtual DOM object
   2. **A root object with a .render() method**
   3. The rendered HTML
   4. A DOM node reference
2. Which method is used to hydrate a server-rendered React application in React 18?
   1. hydrate()
   2. **hydrateRoot()**
   3. createHydrationRoot()
   4. renderServer()
3. What is the main purpose of React’s virtual DOM?
   1. To bypass the browser’s DOM API entirely
   2. **To improve performance by reducing direct DOM manipulations**
   3. To store CSS styles
   4. To compile JSX
4. When React re-renders a component, it primarily compares:
   1. The new real DOM with the old real DOM
   2. **The new virtual DOM with the old virtual DOM**
   3. Props with state
   4. The server DOM with client DOM
5. What happens if a React component returns null?
   1. An error is thrown
   2. **Nothing is rendered for that component**
   3. The previous render is reused
   4. React renders an empty string in the DOM
6. Why is findDOMNode() discouraged in modern React?
   1. It’s too slow
   2. **It breaks encapsulation and strict mode**
   3. It’s not compatible with ES6
   4. It’s only available in class components
7. Which DOM API does React use internally for updates?
   1. innerHTML
   2. document.write
   3. **DOM diffing and patching**
   4. Full HTML replacement
8. React batches state updates:
   1. Only inside event handlers in React 18
   2. **In both synchronous and asynchronous contexts in React 18**
   3. Only for hooks
   4. Only in class components
9. What is React’s "root" element typically in a standard CRA project?
   1. <main>
   2. <app>
   3. **<div id="root">**
   4. <section id="root">
10. In React concurrent mode, rendering is:
    1. Always synchronous
    2. **Interruptible and prioritized**
    3. Rendered twice
    4. Always delayed until idle
11. What does React’s reconciliation algorithm primarily optimize?
    1. Network requests
    2. Component logic
    3. **DOM updates by reusing existing nodes**
    4. CSS rendering speed
12. What happens if multiple root.render() calls target the same container?
    1. They merge
    2. **The last one wins, replacing previous renders**
    3. They both render in sequence
    4. React throws an error

Section 2 — Props & Component Patterns (12 Questions)

1. In React, props are:
   1. Mutable values that control component behavior
   2. **Immutable inputs passed from parent to child**
   3. Global variables for the component tree
   4. Used only in class components
2. Which of these is the correct way to set a default prop value in a functional component?
   1. Component.defaultProps = { propName: value }
   2. { propName = value } in function parameters
   3. **Both a and b are valid in modern React**
   4. Use setState in useEffect
3. When should you use children prop?
   1. To pass primitive values only
   2. **To pass React elements or content between component tags**
   3. To pass event handlers
   4. To manage internal state
4. What is "prop drilling"?
   1. **Passing props deeply through many layers of components unnecessarily**
   2. Overwriting default props
   3. Using props in multiple components
   4. Converting props into state
5. Which pattern helps avoid prop drilling?
   1. Higher-order components (HOC)
   2. Context API
   3. Render props
   4. **All of the above**
6. What happens if a child component changes its prop value locally?
   1. The prop value changes for the parent too
   2. **The local change won’t affect the parent or other components**
   3. The entire app re-renders
   4. React throws an error
7. Which is an example of a controlled component in React?
   1. **<input value={state} onChange={setState} />**
   2. <input /> without any value prop
   3. A class component with no state
   4. Any component receiving props
8. In React, defaultProps for functional components:
   1. Are deprecated for TypeScript projects
   2. **Still work but destructuring with default values is preferred**
   3. Cannot be used in React 17+
   4. Only work for class components
9. What is a common downside of the render props pattern?
   1. It increases bundle size
   2. **It causes "wrapper hell" with deeply nested functions**
   3. It breaks React’s lifecycle
   4. It cannot work with hooks
10. Which of the following can be passed via props?
    1. Strings
    2. Objects
    3. Functions
    4. **All of the above**
11. If you need to pass data up from a child to a parent component, you should:
    1. Use props.children
    2. **Pass a callback function from parent to child and call it in the child**
    3. Use the defaultProps property
    4. Modify the parent’s state directly from child
12. Which React pattern allows sharing logic between components without HOCs or render props?
    1. **Custom hooks**
    2. Context API
    3. Portals
    4. Prop drilling
13. In a functional component, what hook replaces this.state from class components?
    1. useEffect
    2. **useState**
    3. useReducer
    4. useMemo
14. In class components, which method is called immediately after a component is inserted into the DOM?
    1. componentWillMount
    2. **componentDidMount**
    3. componentDidUpdate
    4. shouldComponentUpdate
15. In React, state updates are:
    1. Always synchronous
    2. Always asynchronous
    3. **Sometimes batched and asynchronous**
    4. Performed directly on the state object
16. Which lifecycle method is used to clean up subscriptions or timers in a class component?
    1. **componentWillUnmount**
    2. componentDidUnmount
    3. componentWillMount
    4. componentWillClean
17. What is the primary difference between state and props?
    1. **State is mutable inside the component, props are immutable**
    2. Props are mutable, state is immutable
    3. Both are mutable
    4. Props can only store strings
18. What is the correct way to update state based on the previous state in functional components?
    1. setState(state + 1)
    2. **setState(prev => prev + 1)**
    3. state = state + 1
    4. this.state = this.state + 1
19. In React 18, state updates triggered in a setTimeout are:
    1. **Batched automatically**
    2. Not batched
    3. Only batched in strict mode
    4. Deprecated
20. In class components, which method is called after every render update?
    1. **componentDidUpdate**
    2. componentDidMount
    3. shouldComponentUpdate
    4. getDerivedStateFromProps
21. What will happen if you call setState with the same value as the current state?
    1. React will always re-render
    2. **React will skip rendering for performance optimization**
    3. An error will be thrown
    4. It will reset the component’s lifecycle
22. Which hook is most suitable for managing complex state logic with multiple sub-values?
    1. useState
    2. **useReducer**
    3. useEffect
    4. useRef
23. In class components, which lifecycle method is called before a component is removed from the DOM?
    1. **componentWillUnmount**
    2. componentDidUnmount
    3. componentWillDelete
    4. componentRemoved
24. If you need to preserve state between re-renders without causing re-renders when it changes, you should use:
    1. useState
    2. useEffect
    3. **useRef**
    4. useCallback

Section 4 — Hooks Deep Dive (18 Questions)

1. What is the primary rule of hooks in React?
   1. Hooks must only be called in class components
   2. **Hooks must be called at the top level of a functional component**
   3. Hooks can be called inside loops and conditions freely
   4. Hooks must always return JSX
2. What does the second argument to useEffect control?
   1. The order of effects
   2. **Dependencies that determine when the effect runs**
   3. Cleanup logic
   4. Component re-render count
3. Which hook would you use to memoize a computed value?
   1. **useMemo**
   2. useCallback
   3. useRef
   4. useEffect
4. Which hook is used for persisting values between renders without triggering re-renders?
   1. useMemo
   2. useState
   3. **useRef**
   4. useReducer
5. In useEffect, returning a function serves as:
   1. **A cleanup function for when the effect is re-run or the component unmounts**
   2. A new render trigger
   3. An alternative to dependencies
   4. A state reset
6. Which hook allows you to create a custom hook?
   1. **Any built-in hook can be used inside a custom hook**
   2. Only useState and useEffect
   3. useCustom
   4. useReducer only
7. What will happen if you forget to provide a dependency array to useEffect?
   1. It will run only once
   2. **It will run on every render**
   3. It will never run
   4. It will cause a syntax error
8. useLayoutEffect runs:
   1. **Before painting the screen, synchronously after DOM mutations**
   2. After painting the screen, asynchronously
   3. Only on initial mount
   4. Only when triggered manually
9. What is the main difference between useCallback and useMemo?
   1. useCallback returns a memoized value, useMemo returns a function
   2. **useCallback returns a memoized function, useMemo returns a memoized value**
   3. They are interchangeable
   4. useMemo is faster than useCallback
10. What is the initial value of a useRef() without arguments?
    1. undefined
    2. **null**
    3. 0
    4. ""
11. Which hook would you use to subscribe to an external data source?
    1. useState
    2. **useEffect**
    3. useReducer
    4. useMemo
12. How can you optimize a component that renders a list of items to avoid unnecessary re-renders?
    1. useEffect
    2. **useMemo and useCallback for handlers and derived data**
    3. useReducer only
    4. Avoid keys in list items
13. What happens if you update state inside useEffect without a dependency array?
    1. Nothing
    2. **Infinite render loop**
    3. It runs only once
    4. It throws an error
14. Which hook would you use for form input management if the form state is complex?
    1. useState
    2. **useReducer**
    3. useMemo
    4. useLayoutEffect
15. Can you use hooks inside regular JavaScript functions?
    1. Yes, anywhere
    2. **No, only inside functional components or custom hooks**
    3. Only in async functions
    4. Only in arrow functions
16. What will useEffect(() => {}, []) do?
    1. Never run
    2. Run on every render
    3. **Run only once after the component mounts**
    4. Throw an error
17. How do you prevent useEffect from running on initial render but run on updates?
    1. **Check a ref flag inside useEffect**
    2. Pass null as dependency array
    3. Use useLayoutEffect instead
    4. It’s not possible
18. Which hook is used to manually trigger a re-render without changing state?
    1. useEffect
    2. useRef with a dummy state update
    3. useReducer with a dummy dispatch
    4. **Both b and c are valid**

Section 5 — JSX Advanced Usage (10 Questions)

1. In JSX, which attribute should you use instead of class for CSS classes?
   1. css
   2. **className**
   3. styleClass
   4. classAttr
2. What is the correct way to insert JavaScript expressions in JSX?
   1. Using double quotes " "
   2. **Using curly braces { }**
   3. Using parentheses ( )
   4. Using backticks ` `
3. What will happen if you return multiple JSX elements without a wrapper?
   1. It will work fine
   2. **It will throw a syntax error**
   3. It will wrap them automatically in a div
   4. It will render only the first element
4. Which JSX element can be used to return multiple elements without adding extra DOM nodes?
   1. <Wrapper>
   2. <span>
   3. **<React.Fragment> or <>...</>**
   4. <Multi>
5. In JSX, how should inline styles be passed?
   1. As a string like HTML
   2. **As an object with camelCased property names**
   3. As an object with kebab-case property names
   4. Using CSS modules only
6. What will {false && <div>Hello</div>} render?
   1. <div>Hello</div>
   2. **Nothing**
   3. false
   4. An empty string
7. JSX elements must have:
   1. **A single parent element**
   2. Multiple sibling root elements
   3. A key prop always
   4. An enclosing function
8. Which statement about JSX is true?
   1. JSX is a string representation of HTML
   2. **JSX compiles to React.createElement calls**
   3. JSX is required to use React
   4. JSX cannot include JavaScript expressions
9. How can you add a comment inside JSX?
   1. <!-- comment -->
   2. // comment
   3. **{/\* comment \*/}**
   4. /\*\* comment \*/
10. In JSX, tabIndex attribute is:
    1. Written in lowercase
    2. **Written in camelCase**
    3. Not supported
    4. Only works in HTML, not JSX

Section 6 — React Performance Optimizations (12 Questions)

1. Which higher-order component can you use to prevent unnecessary re-renders of functional components?
   1. **React.memo**
   2. React.preventRender
   3. React.optimize
   4. PureComponent
2. In class components, which base class provides a shallow prop and state comparison to avoid re-renders?
   1. Component
   2. **PureComponent**
   3. StatelessComponent
   4. ShallowComponent
3. What does useCallback help with in performance optimization?
   1. **Prevents re-creation of functions unless dependencies change**
   2. Prevents state changes
   3. Reduces bundle size
   4. Avoids memory leaks
4. Which tool in React DevTools helps analyze component render performance?
   1. Network tab
   2. **Profiler**
   3. Console
   4. Lighthouse
5. What is the main purpose of useMemo in optimization?
   1. To store mutable values
   2. **To cache expensive computations**
   3. To manage component state
   4. To batch updates
6. What is code splitting in React primarily used for?
   1. **Reducing JavaScript bundle size by loading code on demand**
   2. Splitting CSS into separate files
   3. Breaking state into smaller pieces
   4. Running code in parallel
7. Which library function is often used for lazy loading components in React?
   1. **React.lazy**
   2. React.load
   3. React.defer
   4. React.split
8. When using React.memo, which scenario might still cause re-renders unnecessarily?
   1. Props being primitive values
   2. **Props being new object or array references each render**
   3. Component having no props
   4. Component having static content
9. What does shouldComponentUpdate return to skip a render?
   1. null
   2. **false**
   3. undefined
   4. 0
10. Which technique helps avoid prop drilling and improves performance in deeply nested components?
    1. **Context API**
    2. Inline styles
    3. Fragment usage
    4. JSX comments
11. What does tree shaking in React builds aim to do?
    1. **Remove unused code during bundling**
    2. Optimize component rendering
    3. Reduce API calls
    4. Cache state
12. How can you prevent a child component from re-rendering when a parent updates?
    1. **Wrap it in React.memo and ensure stable props**
    2. Always use useEffect
    3. Call stopPropagation
    4. Avoid using keys in lists

Section 7 — JavaScript ES6+ Advanced (24 Questions)

1. Which ES6 feature allows unpacking values from arrays or objects into distinct variables?
   1. Spread syntax
   2. **Destructuring**
   3. Rest parameters
   4. Template literals
2. What will console.log([...new Set([1,2,2,3])]) output?
   1. [1, 2, 2, 3]
   2. **[1, 2, 3]**
   3. {1, 2, 3}
   4. Error
3. Which statement about let and const is correct?
   1. Both are function-scoped
   2. **Both are block-scoped**
   3. let is block-scoped, const is function-scoped
   4. const cannot be used with objects
4. What will be logged? console.log(0 == '0');

console.log(0 === '0');

* 1. **true false**
  2. false true
  3. true true
  4. false false

1. Which method can be used to merge two objects in ES6?
   1. **Object.assign or spread syntax {...obj1, ...obj2}**
   2. concat
   3. merge()
   4. combine()
2. What will typeof NaN return?
   1. **"number"**
   2. "NaN"
   3. "undefined"
   4. "object"
3. Which operator allows default values in destructuring?

**a) =**

b) ||

1. ??
2. &&
3. What will be logged?

let x = [1, 2, 3]; let y = x; y.push(4); console.log(x);

* 1. [1, 2, 3]
  2. **[1, 2, 3, 4]**
  3. [4]
  4. Error

1. Which ES6 feature allows functions to have variable numbers of arguments?
   1. Spread syntax
   2. **Rest parameters**
   3. Default parameters
   4. Arrow functions
2. What will console.log([...'hello']) output?
   1. ['hello']
   2. **['h', 'e', 'l', 'l', 'o']**
   3. "hello"
   4. Error
3. Which array method returns a new array without mutating the original?
   1. push
   2. splice
   3. **slice**
   4. sort
4. What is the output? console.log([] + []);

console.log([] + {});

console.log({} + []);

* 1. **"" "[object Object]" "[object Object]"**
  2. [] {} {}
  3. undefined undefined undefined
  4. Error

1. Which keyword is used to create a class in ES6?
   1. **class**
   2. function
   3. constructor
   4. prototype
2. What is the output? let a = 10;

let b = a++;

console.log(a, b);

* 1. 10 10
  2. **11 10**
  3. 11 11
  4. 10 11

1. What will console.log('5' - 3) output?
   1. "53"
   2. **2**
   3. "2"
   4. Error
2. Which of the following is NOT a JavaScript primitive type?
   1. boolean
   2. string
   3. **object**
   4. symbol
3. What does Object.freeze() do?
   1. **Prevents object properties from being changed or added**
   2. Deletes all object properties
   3. Makes the object immutable recursively
   4. Converts object to JSON
4. Which method checks if all elements in an array pass a test?
   1. **every()**
   2. some()
   3. filter()
   4. map()
5. What is the result of null == undefined?
   1. **true**
   2. false
   3. Error
   4. undefined
6. Which statement about arrow functions is true?
   1. They have their own this binding
   2. **They inherit this from their enclosing scope**
   3. They must always return a value
   4. They can be used as constructors
7. What is the output? console.log(typeof function(){});
   1. "object"
   2. **"function"**
   3. "method"
   4. "callable"
8. What is the correct syntax for optional chaining in JavaScript?
   1. **obj?.prop**
   2. obj.?prop
   3. obj::prop
   4. **obj?.[prop]**
9. Which operator is used for nullish coalescing in JavaScript?
   1. **??**
   2. ||
   3. &&
   4. ?:
10. What is the output? console.log(1 < 2 < 3);

console.log(3 > 2 > 1);

* 1. true true
  2. **true false**
  3. false false
  4. false true