REACT MEACHINE TEST

When preparing for a React.js machine test, you can expect questions that assess your ability to handle common tasks and problems in React development. Here are 20 popular machine test questions for React.js:

1. \*\*Implement a React component that fetches and displays data from an API.\*\*

- Create a component using `useEffect` and `useState` to handle data fetching.

2. \*\*Write a custom hook that handles form input management.\*\*

- Implement a hook that provides methods to update form values and handle form submission.

3. \*\*Create a React component that uses React Context to manage global state.\*\*

- Implement a context provider and consumer to share state across components.

4. \*\*Build a component that uses `React.memo` to optimize performance.\*\*

- Demonstrate how to prevent unnecessary re-renders with `React.memo`.

5. \*\*Develop a React component that performs lazy loading of a child component.\*\*

- Use `React.lazy` and `Suspense` to load a component asynchronously.

6. \*\*Implement a React component with a controlled input field.\*\*

- Use state and event handlers to control the value of an input field.

7. \*\*Create a higher-order component (HOC) that adds authentication logic.\*\*

- Write an HOC that wraps a component and checks for authentication before rendering.

8. \*\*Write a React component that uses `useReducer` for complex state management.\*\*

- Implement a reducer function and `useReducer` to handle state changes.

9. \*\*Build a component that demonstrates how to use `useCallback` to memoize callbacks.\*\*

- Show how to prevent unnecessary function recreations with `useCallback`.

10. \*\*Create a React component that handles component lifecycle events using class components.\*\*

- Implement `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount`.

11. \*\*Write a component that uses `useEffect` to synchronize with local storage.\*\*

- Store and retrieve state from local storage within a `useEffect` hook.

12. \*\*Develop a React component that implements conditional rendering based on props.\*\*

- Show how to render different UI elements based on prop values.

13. \*\*Create a component that demonstrates the use of `forwardRef` to forward refs to a child component.\*\*

- Use `React.forwardRef` to pass a ref from a parent component to a child component.

14. \*\*Implement a simple to-do list with add, remove, and toggle functionalities using functional components.\*\*

- Create a component to manage a list of to-do items with basic CRUD operations.

15. \*\*Write a component that performs server-side rendering with Next.js.\*\*

- Show how to use Next.js to fetch data and render a React component on the server.

16. \*\*Create a component that uses `useImperativeHandle` to customize the instance value exposed to parent components.\*\*

- Use `useImperativeHandle` with `forwardRef` to control what methods or properties are exposed.

17. \*\*Implement a React component with a dynamic list of items where items can be added or removed.\*\*

- Handle dynamic updates to a list with state and event handlers.

18. \*\*Write a component that uses `useLayoutEffect` instead of `useEffect`.\*\*

- Show how `useLayoutEffect` can be used to perform measurements or layout calculations.

19. \*\*Create a component that demonstrates error boundaries to catch JavaScript errors.\*\*

- Implement an error boundary class component to catch errors in child components.

20. \*\*Develop a component that uses `useContext` to consume values from a React Context.\*\*

- Show how to use the `useContext` hook to access and use context values in a functional component.

These questions cover a range of React features and best practices, including state management, performance optimization, and component lifecycle methods. They are designed to test your ability to handle common React tasks and problems in a real-world development environment.