React JS Questions

**What is React JS?**

React.js is a JavaScript library for building user interfaces, especially single-page applications, by creating reusable components. It uses a virtual DOM for efficient updates and employs JSX syntax for writing components. React's unidirectional data flow and hooks simplify state management in functional components.

**DOM:**

The DOM (Document Object Model) is a programming interface that represents the structure of a web page as a tree of objects, where each HTML element (like headings, paragraphs, and links) is a node in the tree. It allows programming languages, like JavaScript, to interact with, modify, or update the content, structure, and style of a webpage dynamically.

**Virtual DOM:**

The Virtual DOM is a lightweight, in-memory representation of the actual DOM (Document Object Model) used by React to track changes in the UI. Instead of directly modifying the real DOM, React updates the Virtual DOM, compares it with the previous version (a process called "reconciliation"), and only applies the necessary changes to the real DOM. This approach makes updates faster and more efficient.

**JSX (JavaScript XML):**

JSX is a syntax extension in React that allows writing HTML-like code within JavaScript. Unlike HTML, JSX can embed JavaScript expressions using curly braces {} and requires proper closing of all tags. It is transformed into JavaScript functions under the hood, making it more dynamic than regular HTML.

**React Hooks:**

React hooks are functions that allow you to use state and lifecycle features in functional components without needing class components. Some commonly used hooks are:

1. **useState**: Manages state in functional components by providing a state variable and a function to update it.
2. **useEffect**: Handles side effects like data fetching or subscribing to events, running after every render or based on dependency changes.

**Life Cycle of React**

The React component lifecycle consists of three main phases: **Mounting**, **Updating**, and **Unmounting**. During **Mounting**, components are created and inserted into the DOM, with methods like componentDidMount being called after the initial render. In the **Updating** phase, components respond to changes in props or state, with methods like componentDidUpdate handling updates, and finally, during **Unmounting**, components are removed from the DOM, invoking componentWillUnmount for cleanup tasks.

**How does React handle routing in single-page applications?**

React handles routing in single-page applications using libraries like **React Router**. React Router provides components like <BrowserRouter>, <Route>, and <Link> to manage navigation and rendering of different components based on the URL. It allows for dynamic routing by mapping URL paths to specific components, enabling a smooth, client-side navigation experience without full page reloads.

**What is the purpose of useEffect and when would you use it?**

The useEffect hook manages side effects in functional components, such as data fetching, subscriptions, or manual DOM manipulations. It runs after every render or when dependencies change, allowing you to perform operations like updating the component or cleaning up resources.

**What is the difference between props and state in React?**

**Props** are read-only values passed from a parent component to a child component, used to configure or customize the child component. **State** is a mutable data structure managed within a component that can change over time and trigger re-renders. Props are set by the parent and cannot be modified by the child, while state is managed and updated internally by the component itself.

**what is state in react js?**

In React, **state** is a mutable data structure that represents the dynamic data or UI status within a component. It allows a component to manage and update its own data, triggering re-renders when the state changes. State is typically managed using the useState hook in functional components or this.state and this.setState in class components.

**what is props?**

**Props** (short for properties) are read-only attributes passed from a parent component to a child component in React. They allow data to be shared and configured between components, enabling the child component to render based on the values provided by the parent. Unlike state, props cannot be modified by the receiving component; they are set by the parent and used for rendering or customization.