1. Tell us the differences between uncontrolled and controlled components.?

Controlled components are managed by a framework or library, have a defined interface, state, and lifecycle. Examples include React, Angular, and Vue components. They offer a higher level of control but can only be used within the framework's context.
Uncontrolled components are not managed by a framework or library and have their own state and lifecycle. Examples include native HTML form elements. They are more flexible but offer less control and can be used in any context.

2. How to validate React props using PropTypes?

To validate React props using PropTypes, we can import the PropTypes library from the 'prop-types' package and define the propTypes object within the component. The propTypes object specifies the expected data type and any other validation requirements for each prop. For example, to validate that a prop called 'name' is a string and is required, we can define it as 'name: PropTypes.string.isRequired'. Once defined, React will check that the props passed to the component conform to the propTypes specification and will throw an error if the validation fails. This helps to catch errors early and ensure that the component receives the correct data.

3. Tell us the difference between nodejs and express js?

Node.js is a JavaScript runtime that allows developers to run JavaScript code on the server-side. It provides a set of built-in modules for handling common tasks such as file system operations, networking, and more.
Express.js, on the other hand, is a web application framework built on top of Node.js. It provides a set of abstractions and features for building web applications, such as routing, middleware, and template rendering.
In summary, Node.js is the underlying runtime environment, while Express.js is a framework that simplifies building web applications with Node.js by providing additional features and abstractions.

4. What is a custom hook, and why will you create a custom hook?

A custom hook in React is a reusable function that contains some stateful logic and can be shared across multiple components. Custom hooks allow us to abstract away complex logic and state management into reusable functions, making code more modular and easier to read and maintain. Custom hooks can be used to encapsulate behavior that is used across multiple components or to abstract away complex logic that doesn't belong in a component. They can also help to keep components focused on their primary responsibilities and improve code reuse.
In summary, creating a custom hook allows us to encapsulate complex logic and state management into reusable functions that can be shared across multiple components, resulting in more modular and maintainable code.