**What is the purpose of render() in React.**

Each React component must have a **render()**mandatorily. It returns a single React element which is the representation of the native DOM component. If more than one HTML element needs to be rendered, then they must be grouped together inside one enclosing tag such as **<form>, <group>,<div>** etc. This function must be kept pure i.e., it must return the same result each time it is invoked.

### ****What is Props?****

Props is the shorthand for Properties in React. They are read-only components which must be kept pure i.e. immutable. They are always passed down from the parent to the child components throughout the application. A child component can never send a prop back to the parent component. This help in maintaining the unidirectional data flow and are generally used to render the dynamically generated data.

### ****What is a state in React and how is it used?****

States are the heart of React components. States are the source of data and must be kept as simple as possible. Basically, states are the objects which determine components rendering and behavior. They are mutable unlike the props and create dynamic and interactive components. They are accessed via **this.state().**

### ****Differentiate between stateful and stateless components.****

|  |  |
| --- | --- |
| **Stateful Component** | **Stateless Component** |
| 1. Stores info about component’s state change in memory | 1. Calculates the internal state of the components |
| 2. Have authority to change state | 2. Do not have the authority to change state |
| 3. Contains the knowledge of past, current and possible future changes in state | 3. Contains no knowledge of past, current and possible future state changes |
| 4. Stateless components notify them about the requirement of the state change, then they send down the props to them. | 4. They receive the props from the Stateful components and treat them as callback functions. |

### ****What are synthetic events in React?****

Synthetic events are the objects which act as a cross-browser wrapper around the browser’s native event. They combine the behavior of different browsers into one API. This is done to make sure that the events show consistent properties across different browsers.

### ****What do you know about controlled and uncontrolled components?****

|  |  |
| --- | --- |
| **Controlled Components** | **Uncontrolled Components** |
| 1. They do not maintain their own state | 1. They maintain their own state |
| 2. Data is controlled by the parent component | 2. Data is controlled by the DOM |
| 3. They take in the current values through props and then notify the changes via callbacks | 3. Refs are used to get their current values |

### ****What are Pure Components?****

*Pure*components are the simplest and fastest components which can be written. They can replace any component which only has a **render().**These components enhance the simplicity of the code and performance of the application.

### ****What are Higher Order Components(HOC)?****

Higher Order Component is an advanced way of reusing the component logic. Basically, it’s a pattern that is derived from React’s compositional nature. HOC are custom components which wrap another component within it. They can accept any dynamically provided child component but they won’t modify or copy any behavior from their input components. You can say that HOC are ‘pure’ components.

**What can you do with HOC?**

HOC can be used for many tasks like:

* Code reuse, logic and bootstrap abstraction
* Render High jacking
* State abstraction and manipulation
* Props manipulation

**How do you handle optimization in a large React application?**

* There are several techniques that can be used to optimize a large React application:
* Use the React Developer Tools to identify and fix performance bottlenecks. The React Developer Tools allow you to track the performance of individual components and identify which components are causing the most re-renders.
* Use the shouldComponentUpdate lifecycle method to prevent unnecessary re-renders. This method allows you to control when a component should update based on its props and state.
* Use PureComponent and memo instead of Components. These are more efficient alternatives to React.Component that only re-render when props or state have changed.
* Use the useEffect hook to handle side effects. This hook allows you to run side effects, such as network requests, after a component has rendered.
* Use the useMemo hook to memoize expensive calculations. This hook allows you to cache the results of expensive calculations and only recalculate them when the inputs have changed.
* Lazy loading: Lazy loading is a technique where you only load the components that are needed for the current view. This can greatly improve the performance of your application.
* Code splitting: Code splitting is a technique where you split your application into smaller chunks of code that are loaded on demand. This can greatly improve the performance of your application.
* Optimize the loading time of your application by using techniques like code minification, compression, and caching.
* It’s also important to keep in mind that performance optimization is an ongoing process and you should regularly check and optimize your application as it grows.