**Question 1: What are props in React.js? How are props different from state?**

**Answer:** Props (short for "properties") are used to pass data from a parent component to a child component in React. They are **immutable**, meaning they cannot be changed by the receiving component.

**Example:**

function Greeting(props) {

return <h1>Hello, {props.name}!</h1>;

}

function App() {

return <Greeting name="Ali" />;

}

**Difference between Props and State:**

| **Feature** | **Props** | **State** |
| --- | --- | --- |
| **Mutability** | Immutable (cannot be changed by the component receiving them) | Mutable (can be updated within the component) |
| **Scope** | Passed from parent to child | Managed within the component itself |
| **Usage** | Used to pass data and configuration | Used for managing dynamic component behavior |
| **Change Handling** | Cannot be modified inside the component | Can be modified using setState() or useState() |

**Question 2: Explain the concept of state in React and how it is used to manage component data.**

**Answer:**  
State is an object in React that stores dynamic data within a component. It allows components to react to user input, network responses, and other dynamic changes.

**Using State in Class Components:**

class Counter extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

render() {

return <h1>Count: {this.state.count}</h1>;

}

}

**Using State in Functional Components (with Hooks):**

import { useState } from "react";

function Counter() {

const [count, setCount] = useState(0);

return <h1>Count: {count}</h1>;

}

* **State is mutable**, meaning it can be updated.
* **Re-renders the component** whenever the state changes.

**Question 3: Why is this.setState() used in class components, and how does it work?**

**Answer:**  
In class components, this.setState() is used to update the component's state and trigger a re-render. Directly modifying this.state **does not** update the UI, so setState() is required.

**How it Works:**

* Merges the new state with the existing state.
* Asynchronously updates the component and triggers a re-render.

**Example:**

class Counter extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

increment = () => {

this.setState({ count: this.state.count + 1 });

};

render() {

return (

<div>

<h1>Count: {this.state.count}</h1>

<button onClick={this.increment}>Increase</button>

</div>

);

}

}

**Why not modify state directly?**

this.state.count = this.state.count + 1; // ❌ Won't re-render the UI

Instead, always use this.setState() to ensure proper state updates and UI re-renders.