**Question 1: What Are Props in React.js? How Are Props Different from State?**

**Props (short for "Properties")** are used to pass data from a **parent component** to a **child component** in React. Props are **read-only** and cannot be modified inside the component receiving them.

**Example of Props:**

function Greeting(props) {

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

}

<Greeting name="Desai" /> // Passing "Desai" as a prop

**Difference Between Props and State:**

| **Feature** | **Props** | **State** |
| --- | --- | --- |
| Scope | Passed from parent to child | Managed within the component |
| Mutability | Read-only (immutable) | Can be changed using setState or useState |
| Usage | Used to pass data between components | Used to store and manage dynamic data |
| Control | Controlled by parent component | Controlled by the component itself |

**Question 2: What Is State in React and How Is It Used?**

**State** is a built-in object in React that allows a component to store and manage **dynamic data**. Unlike props, state **can be changed** inside the component, and when updated, the component **re-renders** to reflect the changes.

**Using State in Functional Components (With Hooks)**

import { useState } from "react";

function Counter() {

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

return (

<div>

<p>Count: {count}</p>

<button onClick={() => setCount(count + 1)}>Increment</button>

</div>

);

}

**Using State in Class Components**

class Counter extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 }; // Declaring state

}

render() {

return (

<div>

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

<button onClick={() => this.setState({ count: this.state.count + 1 })}>

Increment

</button>

</div>

);

}

}

**Why Use State?**

1. **Manages Dynamic Data** – Helps store user input, API responses, etc.
2. **Triggers Re-Renders** – Updates UI whenever state changes.
3. **Component-Specific** – Each component manages its own state independently.

**Question 3: Why Is this.setState() Used in Class Components, and How Does It Work?**

In class components, **state should never be modified directly**. Instead, this.setState() is used to update state **safely** and trigger re-renders.

**How this.setState() Works**

1. **Merges the new state with the previous state.**
2. **Triggers re-rendering** to update the UI.
3. **Executes asynchronously** for performance optimization.

**Example of this.setState() Usage:**

class Counter extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

handleIncrement = () => {

this.setState({ count: this.state.count + 1 }); // Correct way to update state

};

render() {

return (

<div>

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

<button onClick={this.handleIncrement}>Increment</button>

</div>

);

}

}

**Why Not Modify State Directly?**

this.state.count = this.state.count + 1; // ❌ Wrong, does not trigger re-render

Direct modifications **won’t update the UI**, while setState() ensures changes reflect in the UI properly.

**Handling Previous State in setState()**

Because setState() is asynchronous, use a **callback function** when updating based on previous state:

this.setState((prevState) => ({ count: prevState.count + 1 }));

This ensures the **latest state value** is used, avoiding potential errors in rapid updates.