Q1. What are props in React.js? How are props different from state?

Ans: In React.js, props (properties) are used to pass data from a parent component to a child component. Props make components reusable and dynamic by allowing different values to be passed to the same component.

Props are read-only, which means a component cannot change its own props.

Difference Between Props and State:

Feature	Props	State
Meaning	Data passed from parent	Data managed within a component
Mutability	Read-only (cannot be changed)	Mutable (can be updated)
Ownership	Controlled by parent component	Controlled by the component itself
Usage	Used for configuration & data sharing	Used for dynamic UI behavior
Re-render	Changes cause re-render	Changes cause re-render

Q2. Explain the concept of state in React and how it is used to manage component data.

Ans: In React, state is a built-in object that is used to store and manage data that can change over time within a component. State allows a component to keep track of information such as user input, counters, toggles, or fetched data, and update the user interface whenever that data changes.

State is local to a component, which means each component can have its own state and control its own data independently.

How State Works in React

- State is defined inside a component.
- When the state changes, React automatically re-renders the component.
- State helps in creating dynamic and interactive user interfaces.

Using State in Class Components

In class components, state is defined using this.state, and it is updated using this.setState().

Example:

```
class Counter extends React.Component {
  constructor() {
    super();
    this.state = { count: 0 };
  }

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

  render() {
    return (
      <div>
        <h1>{this.state.count}</h1>
        <button onClick={this.increment}>Increase</button>
      </div>
    );
  }
}
```

Using State in Functional Components

In functional components, state is managed using React Hooks, mainly the useState hook.

Example:

```
import { useState } from "react";

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

  return (
```

```
      <div>
        <h1>{count}</h1>
        <button onClick={() => setCount(count + 1)}>Increase</button>
      </div>
    );
}
```

Q3. Why is this.setState() used in class components, and how does it work?

Ans: In React class components, this.setState() is used to update the component's state. State represents data that can change over time, and React requires setState() to update this data properly so that the user interface stays in sync with the state.

Why is this.setState() needed?

1. Triggers Re-rendering
   When this.setState() is called, React knows that the state has changed and automatically re-renders the component to reflect the updated data in the UI.
2. Ensures Proper State Management
   Directly modifying this.state does not notify React about the change. Using setState() ensures that React handles the update correctly.
3. Efficient Updates
   React may batch multiple setState() calls for better performance, which helps in updating the UI efficiently.

How does this.setState() work?

1. setState() takes an object (or a function) as an argument containing the updated state values.
2. React merges the new state with the existing state.
3. After updating the state, React re-renders the component.

Example

```
class Counter extends React.Component {
  constructor() {
    super();
    this.state = { count: 0 };
```

```
  }

  increment = () => {
    this.setState({ count: this.state.count + 1 });
  };
  render() {
    return (
      <div>
        <h1>{this.state.count}</h1>
        <button onClick={this.increment}>Increase</button>
      </div>
    );
  }
}
```