COMPONENTS

Question 1: What are components in React? Explain the difference between functional components and class components.

Answer:
In React, **components** are the building blocks of a React application.
They allow you to split the UI into **reusable, independent pieces** that can manage their own structure and behavior.

There are **two main types** of components:

Type	Description	Example Syntax
Functional Component	A simple JavaScript function that returns JSX. It can use React Hooks like `useState` and `useEffect`.	`function Greeting() { return <h1>Hello</h1>; }`
Class Component	A component created using ES6 class syntax. It extends `React.Component` and uses a `render()` method to return JSX.	`class Greeting extends React.Component { render() { return <h1>Hello</h1>; } }`

Difference Summary:

Feature	Functional Component	Class Component
Syntax	Function-based	Class-based
State Handling	Uses Hooks (e.g. `useState`)	Uses `this.state`
Lifecycle Methods	Uses Hooks like `useEffect`	Has built-in lifecycle methods like `componentDidMount()`
Simplicity	Easier, shorter code	More complex syntax

Question 2: How do you pass data to a component using props?

Answer:
In React, you can pass data from a **parent component** to a **child component** using **props** (short for *properties*).
Props are **read-only** and help make components reusable and dynamic.

Example:

```
function Welcome(props) {
  return <h1>Hello, {props.name}!</h1>;
}
```

```
function App() {
  return <Welcome name="Ayush" />;
}
```

Here,

- `name="Ayush"` → data passed from **App** (parent)
- `props.name` → data received in **Welcome** (child)

Question 3: What is the role of render() in class components?

Answer:
In **class components**, the **`render()`** method is **required**.
It defines **what the component should display** on the screen.

Key Points:

- The `render()` method must **return JSX**.
- It gets called **every time the component's state or props change**.
- It should be **pure**, meaning it should not modify state or interact with the DOM directly.

Example:

```
class Hello extends React.Component {
  render() {
    return <h2>Hello, {this.props.name}!</h2>;
  }
}
```

Here, the `render()` method returns JSX that shows a greeting message.

Props and State

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 one component to another**, usually from **parent to child**. They make components **reusable** and **dynamic**.

Example:

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

Difference between Props and State:

Feature	Props	State
Definition	Data passed **to** a component	Data **managed inside** a component
Mutability	**Read-only**	**Can be changed** using `setState` or Hooks
Usage	Used to **pass data** between components	Used to **store and manage** component data
Controlled by	Parent Component	The Component itself

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

Answer:
In React, **state** is an object that stores **dynamic data** and controls how a component behaves and looks.
When the state changes, the component **re-renders automatically** to show the new data.

Example (Functional Component using useState):

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

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

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increase</button>
    </div>
  );
}
```

Here,

- `count` is the **state variable**
- `setCount` updates the state
- React re-renders the UI when `count` 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**.
When the state changes, React automatically **re-renders** the component with the new data.

How it works:

- It **merges** the new state with the existing one.
- It triggers **a re-render** to update the UI.
- It works **asynchronously** to improve performance.

Example:

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

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

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={this.increaseCount}>Increase</button>
      </div>
    );
  }
}
```

Here, `this.setState()` updates `count`, and the component automatically re-renders with the new value.