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

A. In React, components are the building blocks of a React application's user interface. A component is a reusable piece of code that represents a part of the UI. Components can be thought of as JavaScript functions or classes that return HTML (or JSX) to render on the screen.

Components can be classified into:

* **Functional Components**
* **Class Components**

**Functional Components**

Functional components are JavaScript functions that accept props (optional input data) as arguments and return React elements (JSX).

Key Features:

Simple and Concise: They are plain JavaScript functions, making them easy to write and understand.

Stateless (Initially): Before React 16.8, functional components were stateless because they couldn't manage state or lifecycle methods. However, with the introduction of Hooks, functional components can now handle state and lifecycle features.

Performance: They are often more efficient and simpler than class components.

Syntax:

javascript

Copy code

const Greeting = (props) => {

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

};

Usage of Hooks:

Hooks like useState and useEffect allow functional components to use state and side effects.

**Class Components:**

Class components are ES6 classes that extend the React.Component class. They include a render() method that returns JSX to render.

Key Features:

Stateful: Class components have built-in support for managing state and lifecycle methods like componentDidMount, componentDidUpdate, and componentWillUnmount.

More Verbose: They require more boilerplate code compared to functional components.

Syntax:

import React, { Component } from 'react';

class Greeting extends Component {

render() {

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

}

}

**Lifecycle Methods:**

Class components have lifecycle methods that allow developers to run code at specific points in a component's lifecycle.

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

**Passing Data to a Component Using Props**

In React, **props** (short for "properties") are used to pass data from a parent component to a child component. Props allow components to be reusable and dynamic by providing them with specific data or configuration.

**Steps to Pass Data Using Props**

**1. Define the Data in the Parent Component**

The parent component holds the data you want to pass to the child.

**2. Pass the Data as an Attribute**

Add an attribute to the child component’s JSX tag in the parent component. The attribute name will be the prop name, and its value can be any valid JavaScript expression.

**3. Access the Data in the Child Component**

Inside the child component, access the prop using props.propName (for class components) or by directly destructuring props (for functional components).

**Key Points About Props**

1. **Read-Only:** Props are immutable, meaning a child component cannot modify them directly.
2. **Dynamic Values:** You can pass variables, strings, numbers, arrays, objects, or even functions as props.
3. **Default Props:** You can define default values for props using defaultProps if no value is provided.

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

In React, the render() method is a required lifecycle method in class components. Its primary role is to describe what should be displayed on the screen by returning React elements (JSX). The render() method ensures that the UI updates based on the component's state and props.

**Rules for Using render()**

1. **Pure Functionality:** The render() method must only focus on returning the UI representation. Avoid modifying state or props.
2. **Single Root Element:** The returned JSX must be wrapped in a single root element (e.g., <div>, <React.Fragment>).

render() {

// Correct

return (

<div>

<h1>Header</h1>

<p>Content</p>

</div>

);

// Incorrect

return (

<h1>Header</h1>

<p>Content</p>

);

}

1. **Do Not Interact with the DOM Directly:** Any direct DOM manipulation should be avoided inside render(); React's reconciliation handles that for you.

**When is render() Called?**

1. **Initially:** When the component is first rendered.
2. **On State or Props Update:** When this.setState() or new props are passed to the component, React triggers a re-render.
3. **Parent Component Updates:** If a parent component re-renders, child components will also re-render.

In summary, the render() method is essential for defining what a React class component should display. It is reactive, declarative, and forms the foundation for updating the UI in response to changes in state and props.

**Props and State**

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

Key Differences Between Props and State

| **Aspect** | Props | State |
| --- | --- | --- |
| **Definition** | Data passed from parent to child. | Data managed within a component. |
| **Mutability** | Immutable (read-only). | Mutable (can be updated). |
| **Who Controls It?** | Controlled by the parent component. | Controlled by the component itself. |
| **Purpose** | Used to pass data and configure a component. | Used to manage dynamic data within a component. |
| **Re-rendering** | Changing props in the parent triggers a re-render of the child. | Changing state triggers a re-render of the component itself. |
| **Scope** | Available in child components. | Local to the component where it's defined. |

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

In React, state is an object used to store dynamic data and manage the behaviour of a component. Unlike props, which are passed to a component and are read-only, state is managed within the component itself and can change over time.

When the state changes, react automatically re-renders the component to reflect the updated state, making it an essential concept for creating interactive and dynamic user interfaces.

**Key Rules for Using State**

1. Do Not Modify State Directly:
   * Wrong: this.state.count = 5;
   * Correct: this.setState({ count: 5 }); or setCount(5);
2. State Updates Are Asynchronous: React batches state updates for performance, so relying on the current state value directly may lead to unexpected results.

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

1. Avoid Excessive State: Keep your state minimal and focus only on the data that is essential for rendering.

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

In React class components, this.setState() is the method used to update the component's state. Directly modifying the state (e.g., this.state.someValue = newValue) does not trigger a re-render, which means the UI will not update to reflect the new state. Instead, this.setState() ensures that the state change is processed by React, causing the component to re-render and display the updated data.

How this.setState() Works

1. Merges State Updates:
   * this.setState() merges the provided object with the existing state, updating only the specified properties without affecting others.
   * For example:

this.setState({ count: 5 });

If the state object has other properties (e.g., { count: 0, name: 'John' }), only count is updated, and name remains unchanged.

1. Triggers Re-Rendering:
   * When you call this.setState(), React schedules a re-render of the component. During the re-render, the component reflects the updated state in the UI.
2. Asynchronous Behavior:
   * React batches multiple setState() calls for performance optimization. Therefore, updates to the state may not happen immediately but are queued and processed together.
   * To ensure the latest state is used, you can pass a function to this.setState() instead of an object. This function receives the previous state and props as arguments.