Components (Functional & Class Components)

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

Ans.

What are Components in React?

In React, components are the basic building blocks of a user interface.
A component is a reusable, independent piece of UI that defines how a part of the screen should look and behave.

Examples of components:

- Navbar
- Button
- Login Form
- Product Card

Each component can:

- Receive **data (props)**
- Manage its own **state**
- Return **JSX** to describe the UI

Types of Components in React

There are two main types of components:

1. Functional Components
2. Class Components

1. Functional Components
   Functional components are JavaScript functions that return JSX.
   Example:

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

   Key Features:

- Simple and easy to write
- Use React Hooks (useState, useEffect, etc.)
- No this keyword
- Preferred in modern React development

With State (using Hooks) :

```
import { useState } from "react";
function Counter() {
  const [count, setCount] = useState(0);

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

Class Components :

Class components are ES6 classes that extend React.Component.

Example:

```
import React from "react";
class Welcome extends React.Component {
  render() {
    return <h1>Hello, React!</h1>;
  }
}
```

Key Features:

- Use this keyword
- State is managed using this.state
- Lifecycle methods (componentDidMount, etc.)
- More verbose and less used today

With State:

```
class Counter extends React.Component {
  state = { count: 0 };

  render() {
    return (
```

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

Difference Between Functional & Class Components :

Feature	**Functional Component**	**Class Component**
Syntax	Function	ES6 Class
State	useState Hook	this.state
Lifecycle	useEffect	Lifecycle methods
this keyword	No	Yes
Code Length	Short & clean	More verbose
Performance	Slightly better	Slightly slower

Feature	Functional Component	Class Component
Usage Today	Recommended	Mostly legacy

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

Ans.

How do you pass data to a component using props in React ? :

In React, props (short for properties) are used to pass data from a parent component to a child component. Props make components dynamic, reusable, and configurable.

Basic Idea of Props

- Props are passed like HTML attributes
- Props are read-only (child components cannot modify them)
- Data flows one-way (Parent → Child)

Passing Props from Parent to Child :-

Parent Component :

```
function App() {
  return (
    <div>
      <Welcome name="Uday" age={22} />
    </div>
  );
}
```

Child Component (Receiving Props) :

```
function Welcome(props) {
  return (
    <h1>
      Hello, {props.name}! You are {props.age} years old.
    </h1>
  );
}
```

Here:

- name and age are props
- props is an object containing all passed values

Using Destructuring for Cleaner Code (Recommended) :-

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

Passing Different Types of Data via Props :-

String :

```
<Component title="React" />
```

Number / Boolean (use {}) :

```
<Component count={10} isLoggedIn={true} />
```

Array :

```
<Component skills={["HTML", "CSS", "React"]} />
```

Object :

```
<Component user={{ name: "Uday", role: "Frontend Developer" }} />
```

Passing Functions as Props (Very Important) :-

Used for child → parent communication.

Parent :

```
function App() {
  const handleClick = () => {
    alert("Button clicked!");
  };

  return <Button onClick={handleClick} />;
```

```
}
```

Child :

```
function Button({ onClick }) {
  return <button onClick={onClick}>Click Me</button>;
}
```

Props in Class Components :-

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Usage :

```
<Welcome name="Uday" />
```

Default Props :-

```
function Welcome({ name = "Guest" }) {
  return <h1>Hello, {name}</h1>;
}
```

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

Ans.

In React class components, the render() method is mandatory and plays a central role in defining the UI.

What does render() do? :

The render() method:

- **Returns JSX** (or null)
- Describes **what should appear on the screen**
- Converts component **state and props into UI**

React calls render() **automatically**:

- When the component **mounts**
- When **state or props change**

1. Basic Example of render()

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, React!</h1>;
  }
}
```

Without render(), a class component cannot display anything.

2. Using Props Inside render()

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

3. Using State Inside render()

```
class Counter extends React.Component {
  state = { count: 0 };

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={() => this.setState({ count: this.state.count + 1
})}>
          Increment
        </button>
      </div>
    );
  }
}
```

Whenever setState() is called, render() runs again.

4. Important Rules of render()

What you should NOT do in render():

- Do not update state (setState)
- Do not make API calls
- Do not cause side effects

render() must be a **pure function**:

- Same props + state → same UI output

5. What can render() return ?
6. X
7. `null` (renders nothing)
8. A single parent element (or Fragment)

Example:

```
render() {
  return null;
}
```
