

Components (Functional & Class Components)

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

Components are the core building blocks of a React app. They are reusable pieces of UI that return JSX and can manage their own data using props and state.

Types of Components

1. Functional Components

- Defined as JavaScript functions
- Simpler and easier to write
- Use hooks like `useState` and `useEffect` for state and side effects

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

2. Class Components

- Created using ES6 classes
- Use `this.state` and lifecycle methods like `componentDidMount`

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

Functional components are cleaner and preferred in modern React, while class components are older and more verbose.

How do you pass data to a component using props?

In React, you pass data to a component using props (short for "properties"). Props allow you to make components dynamic and reusable by supplying different values from a parent component.

Passing Props

To pass props, you add attributes to the component when you use it:

```
<Profile name="Alice" age={30} />
```

Here, name and age are props being passed to the Profile component.

Receiving Props

In a functional component, props are received as a parameter:

```
function Profile(props) {  
  return <p>{props.name} is {props.age} years old.</p>;  
}
```

Or using destructuring for cleaner syntax:

```
function Profile({ name, age }) {  
  return <p>{name} is {age} years old.</p>;  
}
```

In a class component, props are accessed using this.props:

```
class Profile extends React.Component {  
  render() {  
    return <p>{this.props.name} is {this.props.age} years old.</p>;  
  }  
}
```

```
}  
}
```

Key Points

- Props are read-only and passed from parent to child.
- They help make components flexible and reusable.
- You can pass any data type: strings, numbers, arrays, functions, or even other components.

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

In React class components, the render() method is essential. It defines what the component should display on the screen. Every class component must include a render() method, which returns JSX—the structure of the UI.

Key Functions of render():

1. Returns JSX:

The main job of render() is to return JSX that describes the component's UI.

2. class Greeting extends React.Component {

3. render() {

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

5. }

6. }

7. Automatic Execution:

React automatically calls render():

- When the component first loads (mounts)
- When there are changes in state or props

8. Pure Function:

`render()` should be a pure function, meaning it shouldn't modify state or cause side effects. It should only return what the UI looks like based on the current data.

9. Single Root Element:

It must return one root element or wrap multiple elements in a React fragment (`<>...</>`).

The `render()` method is a required part of class components in React. It's responsible for returning JSX, which React uses to display the UI.

Props and State

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

Sure! Here's a medium-length explanation:

In **React.js**, **props** and **state** are two core ways to manage and pass data within components.

Props

Props (short for "properties") are **read-only** values passed from a **parent component to a child component**. They're used to customize or configure child components. For example, if you have a Greeting component that displays a name, you can pass the name as a prop:

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

// Usage

```
<Greeting name="Alice" />
```

Props cannot be modified by the receiving component — they're controlled by the parent.

State

State is a data structure that belongs to a component itself and is used to **track changes over time**. Unlike props, state is **mutable** and can be updated using hooks like `useState` in functional components:

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

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

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

When state changes, the component re-renders with the new state.

Key Differences

Feature	Props	State
Scope	Passed from parent	Managed within the component
Mutability	Immutable	Mutable
Purpose	Configures components	Handles dynamic behavior

props are for passing data *into* a component, and **state** is for managing data *within* a component.

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

In **React**, **state** is a special object used to manage **dynamic data** within a component. Unlike props, which are passed from parent to

child and are read-only, **state is local and can be changed** by the component itself.

In **functional components**, state is managed using the useState hook:

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

```
function Counter() {  
  const [count, setCount] = useState(0); // count is state, setCount  
  updates it  
  
  return (  
    <div>  
      <p>You clicked {count} times</p>  
      <button onClick={() => setCount(count + 1)}>Click me</button>  
    </div>  
  );  
}
```

In this example, count is a piece of state, and setCount is the function that updates it. When the button is clicked, the state updates, and the component **re-renders automatically** to show the new count.

State is useful for:

- Tracking user input
- Managing UI interactions (like modals, toggles)
- Storing temporary values like counters or form data

Key Points:


- State is **local** to the component.

- It is **mutable** (can be updated).
- Updating state triggers a **re-render** of the component.

State is essential for building **interactive and dynamic React apps**.

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

In React class components, `this.setState()` is used to update the component's state and trigger a re-render. You shouldn't update state directly (e.g., `this.state.count = 1`) because React won't know it needs to update the UI.

 How it works:

- It merges the new state with the existing state.
- It tells React to re-render the component with updated data.

Ex

```
this.setState({ count: this.state.count + 1 });
```

If the new state depends on the old state, use a function:

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

`this.setState()` keeps your UI in sync with your data.