**Demonstrating differences between class and functional components**

React, a JavaScript library for building user interfaces, offers two primary ways to create components: Class Components and Functional Components. While both serve the same purpose of encapsulating reusable UI elements, they have fundamental differences in their structure, syntax, and capabilities.

**Class Components:**

1. **Syntax and Structure:**
   * Class Components are defined using ES6 classes, extending from React.Component or React.PureComponent.
   * They contain a render() method which returns the UI elements to be rendered.
   * State and lifecycle methods like componentDidMount(), componentDidUpdate(), etc., are defined within the class.
2. **State Management:**
   * Class Components have their own state, initialized in the constructor using this.state.
   * State updates are performed using this.setState(), triggering re-renders when the state changes.
   * Complex state manipulation and management are easier due to the availability of lifecycle methods.
3. **Lifecycle Methods:**
   * Class Components have a rich set of lifecycle methods such as componentDidMount(), componentDidUpdate(), componentWillUnmount(), etc.
   * These methods allow developers to hook into different stages of a component's lifecycle, enabling actions like data fetching, state updates, and cleanup.

**Functional Components:**

1. **Syntax and Structure:**
   * Functional Components are defined as plain JavaScript functions that take props as input and return React elements.
   * They are simpler and more concise compared to Class Components, reducing boilerplate code.
   * With the introduction of React Hooks, Functional Components can now manage state and side effects.
2. **State Management:**
   * Functional Components traditionally lacked state management capabilities.
   * However, with the introduction of React Hooks (useState), Functional Components can now manage state without the need for classes.
   * useState allows for local state management within Functional Components, enhancing their flexibility and usefulness.
3. **Lifecycle Methods:**
   * Prior to Hooks, Functional Components lacked lifecycle methods.
   * With Hooks, useEffect() provides functionality equivalent to componentDidMount(), componentDidUpdate(), and componentWillUnmount().
   * useEffect() enables developers to perform side effects such as data fetching, subscriptions, or manual DOM manipulations within Functional Components.