

MODULE: 4 (List and Hooks)

[Document subtitle]



shivani gohil

**1.Explain Life cycle in Class Component and functional component with Hooks.**

**Ans.** A [React](https://reactjs.org/?ref=retool.com) component undergoes three different phases in its lifecycle, including mounting, updating, and unmounting. Each phase has specific methods responsible for a particular stage in a component's lifecycle. These methods are technically particular to [*class-based components and not intended for functional components*](https://www.twilio.com/blog/react-choose-functional-components?ref=retool.com)*.*

However, since the concept of Hooks was released in React, you can now use abstracted versions of these lifecycle methods when you’re working with functional component state. Simply put, React Hooks are functions that allow you to [“hook into”](https://javascript.plainenglish.io/building-the-same-application-with-and-without-react-hooks-part-1-4bdcf02c9ab5?ref=retool.com) a React state and the lifecycle features within function components.

In this post, you'll learn more about the React component lifecycle and the different methods within each phase (for class-based components), focusing particularly on methods and hooks.

**Phases of a React component's lifecycle**

A React component undergoes three phases in its lifecycle: mounting, updating, and unmounting.

1. The *mounting phase* is when a new component is created and inserted into the DOM or, in other words, when the life of a component begins. This can only happen once, and is often called “initial render.”
2. The *updating phase* is when the component updates or re-renders. This reaction is triggered when the props are updated or when the state is updated. This phase can occur multiple times, which is kind of the point of React.
3. The last phase within a component's lifecycle is the *unmounting phase*, when the component is removed from the DOM.

In a class-based component, you can call different methods for each phase of the lifecycle (more on this below). These lifecycle methods are of course not applicable to functional components because they can only be written/contained within a class. However, React hooks give functional components the ability to use states.

## React lifecycle methods

The mounting phase

In the mounting phase, a component is prepared for and actually inserted into the DOM. To get through this phase, four lifecycle methods are called: constructor, static getDerivedStateFromProps, render, and componentDidMount.

### **The updating phase**

The Updating phase is triggered when component props or state change, and consists of the following methods: static getDerivedFromProps, shouldComponentUpdate, render, getSnapshotBeforeUpdate, and componentDidUpdate.

.

### **The unmounting phase**

The unmounting phase is the third and final phase of a React component. At this phase, the component is removed from the DOM. Unmounting only has one lifecycle method involved: componentWillUnmount.

## React Hooks and the component lifecycle

Versions of React before 16.8 consider two kinds of components based on statefulness: the class-based stateful component, and the stateless functional components (often referred to as a “dumb component”). But with the release of React 16.8, Hooks were introduced and empowered developers to access state from functional components, instead of writing an entire class. With this change, building components became easier and less verbose.

Hooks known as default hooks come with React, and you’re also able to create your own custom hook. A custom hook is just a function that starts with use, like useStore, or useWhatever.

The two most common default hooks are useState and useEffect. The useState hook gives state to the functional component, and useEffect allows you to add side effects within it (like after initial render), which aren’t allowed within the function's main body. You can also act upon updates on the state with useEffect.

React has released more default hooks, but useState and useEffect are the ones you should be most familiar with. Let’s take a look at how they work and compare them to the component lifecycle we covered above.

### useState

The [useState hook](https://reactjs.org/docs/hooks-state.html?ref=retool.com) is used to store state for a functional component. This hook accepts one parameter: initialState, which will be set as the initial stateful value, and returns two values: the stateful value, and the update function to update the stateful value. The update function accepts one argument, newState, which replaces the existing stateful value.