Explain Life cycle in Class Component and functional component with Hooks

In React, components can be created using either class-based components or functional components. Both types of components have a lifecycle that they go through during their existence. The lifecycle of a component refers to the different stages that the component goes through from initialization to unmounting.

Class Component Lifecycle:

Class components have access to a set of lifecycle methods that can be used to control the behavior of the component. These methods are called at different stages of the component's lifecycle. The basic lifecycle methods are:

1. constructor(props): This is the first method that is called when the component is initialized. It is used to set up the initial state and bind event handlers.

2. componentDidMount(): This method is called after the component has been mounted to the DOM. It is used to make any necessary API calls and set up event listeners.

3. shouldComponentUpdate(nextProps, nextState): This method is called before the component is re-rendered. It is used to determine whether or not the component needs to be re-rendered.

4. componentDidUpdate(prevProps, prevState): This method is called after the component has been updated. It is used to make any necessary updates to the DOM or state.

5. componentWillUnmount(): This method is called just before the component is unmounted from the DOM. It is used to clean up any resources that the component has created.

Functional Component with Hooks:

Functional components were previously limited to a "componentDidMount" equivalent lifecycle method using the useEffect hook, however with the introduction of new hooks such as useEffect, useState, and useContext, functional components can now fully mimic the lifecycle of class-based components.

1. useEffect(() => {}, []): This hook is used to perform side effects in a functional component. It is equivalent to the componentDidMount and componentDidUpdate methods in class components. The first parameter is a function that contains the side effect code and the second parameter is an array of dependencies. If any of the dependencies change, the effect function is re-executed.

2. useState(): This hook is used to manage state in functional components. It takes an initial value as a parameter and returns an array with two elements: the current state value and a function to update the state.

3. useContext(): This hook is used to consume values from a React context in a functional component. It takes a context object as a parameter and returns the current context value.

Overall, while the core lifecycle methods remain the same between class-based components and functional components with hooks, the implementation differs slightly. Functional components with hooks provide a more concise and easier-to-read implementation of lifecycle methods, as well as a more straightforward way of managing state and consuming context values.