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

Ans:

Class Component :

**componentDidMount()**

The mounting phase refers to the period when a component is being created and inserted into the DOM.

During this phase, several lifecycle methods are invoked by React to enable the developer to configure the component, set up any necessary state or event listeners, and perform other initialization tasks.

**componentDidMount()**

This method is called once the component has been mounted into the DOM. It is typically used to set up any necessary event listeners or timers, perform any necessary API calls or data fetching, and perform other initialization tasks that require access to the browser's DOM API.

**componentDidUpdate()**

The method is a lifecycle method in React that is called after a component has been updated and re-rendered. It is useful for performing side effects or additional operations when the component's props or state have changed.

**componentWillUnmount()**

This method is called just before the component is removed from the DOM. It allows you to perform any necessary cleanup, such as canceling timers, removing event listeners, or clearing any data structures that were set up during the mounting phase. After componentWillUnmount() is called, the component is removed from the DOM and all of its state and props are destroyed.

Function Component:

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.

useState :

The useState hook 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.

useEffect :

As with the render() method of a class-based component, the main body of a functional component should be kept pure. With the useEffect hook, you're able to create side effects while maintaining the component's purity. Within this hook, you can send network requests, make subscriptions, and change the state value.