## Q1) Explain Life cycle in Class Component and functional component with Hooks?

Answer:-

In a class-based component in React, there are several life cycle methods that you can use to control the behavior of your component at different stages. Here's an overview of the main life cycle methods:

1. constructor(props): This is called when an instance of the component is being created. It's where you initialize state and bind event handlers.

2. componentDidMount(): This method is invoked after the component has been inserted into the DOM. It's often used to perform initial setup, like fetching data from an API.

3. componentDidMount(prevProps, prevState): This method is called whenever the component updates, either due to changes in props or state. You can use it to perform actions after the component re-renders.

4. componentWillUnmount(): This is called right before the component is removed from the DOM. You can use it to clean up resources, like event listeners or timers.

5. render():This is the method that actually renders your component's content. It's called whenever the component needs to re-render, typically because of changes in state or props.

Functional Component with Hooks:

In functional components, you can achieve similar behavior using hooks, which were introduced in React 16.8. Here's how you can map the class component life cycle methods to equivalent hooks in functional components:

1. useState: You can use the `useState` hook to manage component state.

2. useEffect: This hook is used to perform side effects in functional components. You can use it for tasks like data fetching, updating the DOM, or subscribing to external data sources. It combines the functionality of `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount` from class components.

Here's a basic mapping:

- componentDidMount can be replicated using `useEffect` with an empty dependency array (`[]`). Code inside this effect will run once after the initial render.

- componentDidUpdate can be replicated using `useEffect` with dependencies. Code inside this effect will run whenever the specified dependencies change.

- componentWillUnmount can be replicated using the cleanup function returned by useEffect. This function is called when the component is unmounted.

Here's an example of how you might use useState and useEffect in a functional component:

```
import React, { useState, useEffect } from 'react';

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

  useEffect(() => {
    // This code runs after every render
    document.title = `Count: ${count}`;

    // Cleanup function (equivalent to componentWillUnmount)
    return () => {
      document.title = 'React App';
    };
  }, [count]); // Dependency array specifies when this effect should run

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

export default MyComponent;
```

This functional component with hooks demonstrates how to manage state and perform side effects similar to a class component's life cycle methods.