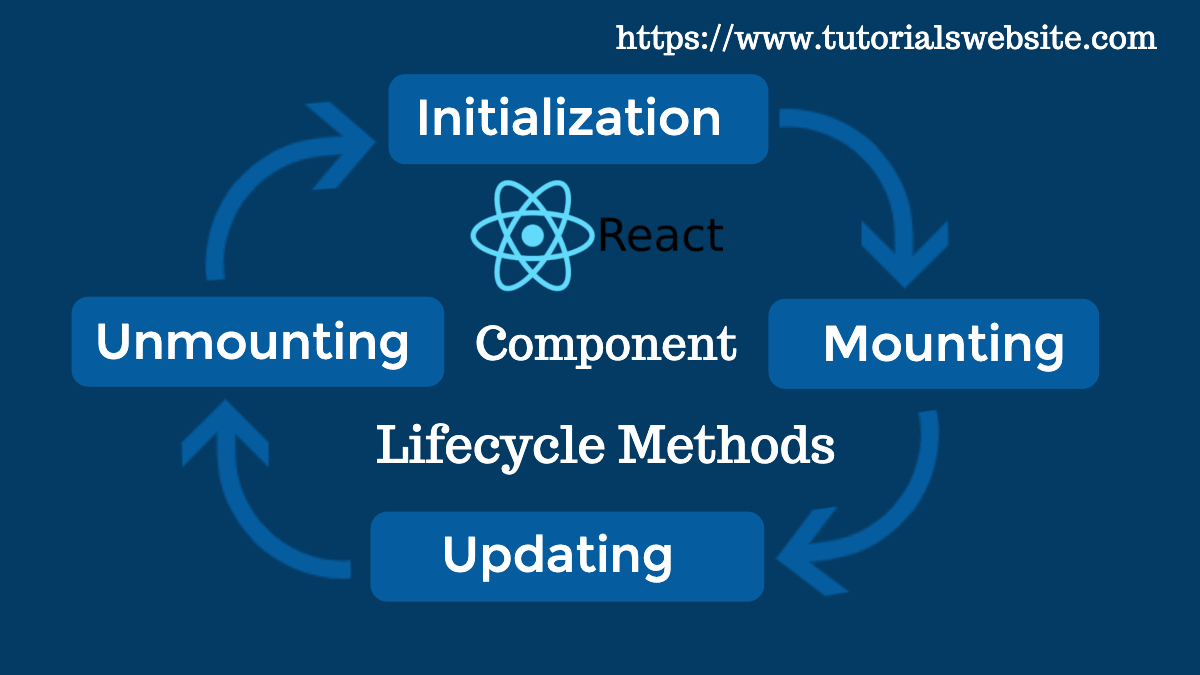
**Uday Rajput [ List and Hooks ]**

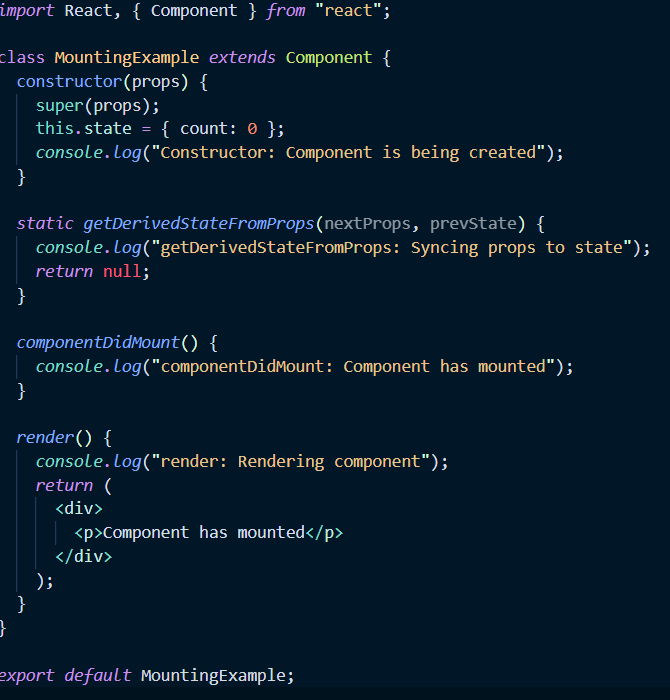
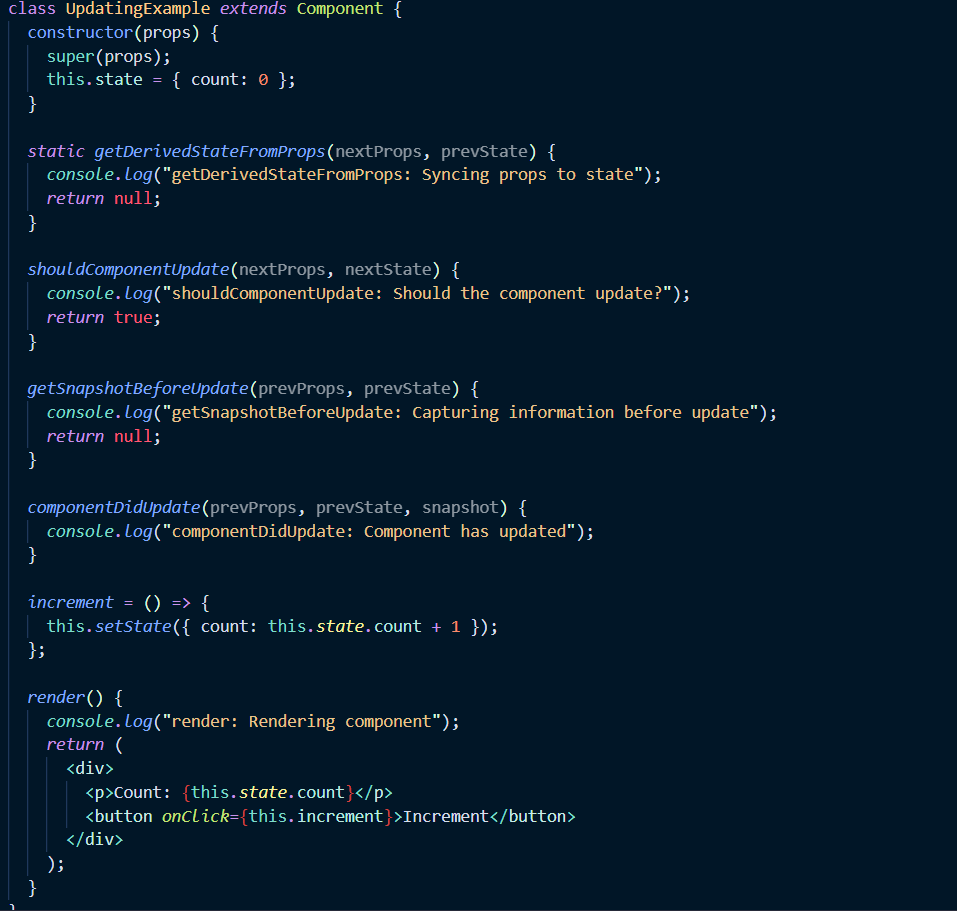
**Question :- Explain Life cycle in Class Component and functional component with Hook**

**Ans : -**

****

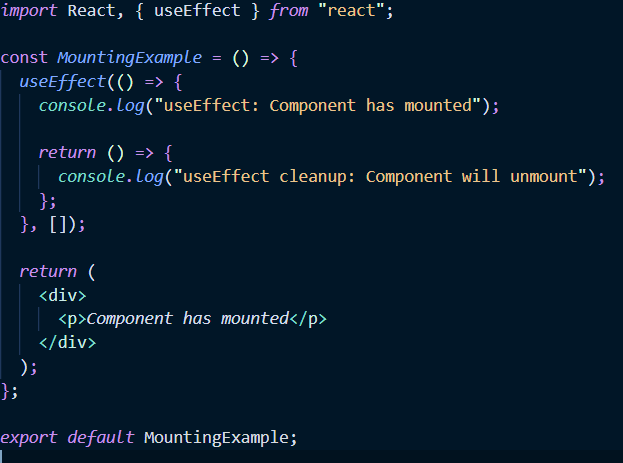
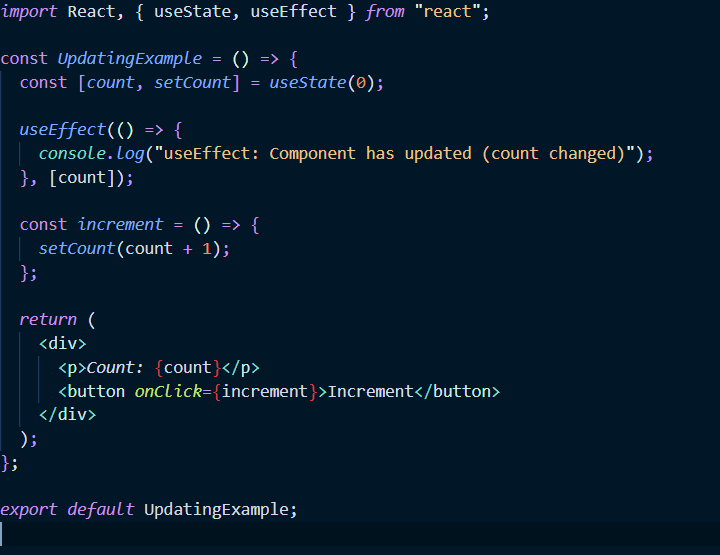
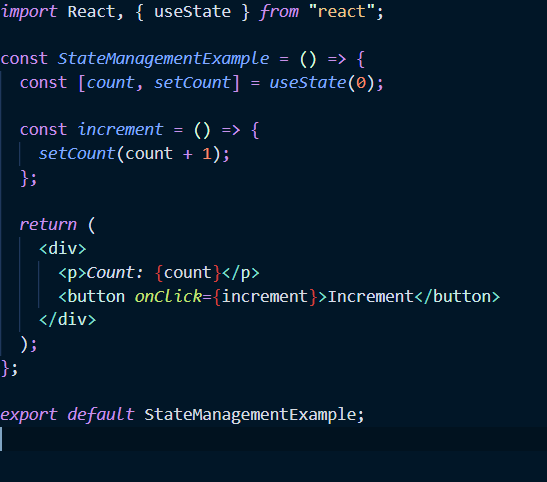
**Class Component Lifecycle : -**

Class components in React have a well-defined lifecycle that can be broken down into three main phases:

1. **Mounting**: When an instance of a component is being created and inserted into the DOM.
   * constructor(): Initializes the component's state and binds methods.
   * static getDerivedStateFromProps(): Syncs state to props if necessary.
   * render(): Returns the JSX to be rendered.
   * componentDidMount(): Runs after the component is inserted into the DOM, suitable for side-effects like API calls.
   * Example :- 
2. **Updating**: When the component's props or state change.
   * static getDerivedStateFromProps(): Syncs state to props if necessary.
   * shouldComponentUpdate(): Determines if a re-render is necessary.
   * render(): Returns the JSX to be rendered.
   * getSnapshotBeforeUpdate(): Captures some information (e.g., scroll position) from the DOM before it changes.
   * componentDidUpdate(): Runs after the component updates in the DOM, suitable for DOM manipulations or additional API calls.
   * Example :- 
3. **Unmounting**: When the component is being removed from the DOM.
   * componentWillUnmount(): Clean up side-effects, like canceling API calls or removing event listeners.
   * Example :- 

### Functional Components with Hooks :

Functional components were initially stateless, but with the introduction of hooks in React 16.8, they gained the ability to manage state and side effects. Hooks provide a more direct way to tap into the lifecy cle of a component.

1. **Mounting and Updating**: Managed through the useEffect hook.
   * useEffect(callback, dependencies): Combines the functionalities of componentDidMount, componentDidUpdate, and componentWillUnmount. The callback function runs after the render, and if the dependencies array changes, the effect runs again.
     + If dependencies is an empty array ([]), useEffect acts like componentDidMount.
     + If dependencies includes specific state or props, useEffect acts like both componentDidMount and componentDidUpdate.
     + The callback can return a cleanup function, which runs like componentWillUnmount.
     + Mounting example :- 
     + Updating example:-
     + 
2. **State Management**: Managed through the useState hook.
   * const [state, setState] = useState(initialState): Declares a state variable and a function to update it.
   * Example :- 

**Examples :-**

Class Component Example:

****

Functional component example :-

