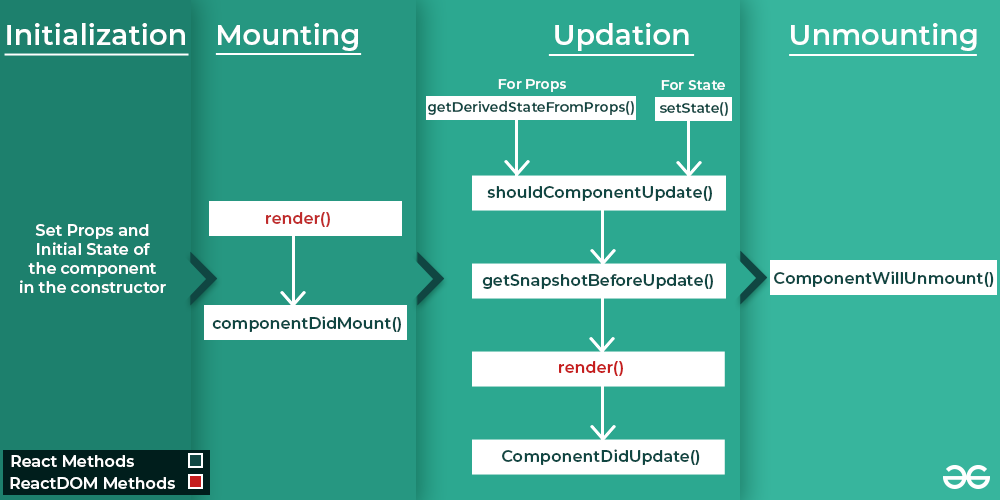
Component Life Cycle Method

**Lifecycle of Components**

Every React Component has a lifecycle of its own, lifecycle of a component can be defined as the series of methods that are invoked in different stages of the component’s existence. The definition is pretty straightforward but what do we mean by different stages? A React Component can go through four stages of its life as follows.

* **Initialization:** This is the stage where the component is constructed with the given Props and default state. This is done in the constructor of a Component Class.
* **Mounting:** Mounting is the stage of rendering the JSX returned by the render method itself.
* **Updating:** Updating is the stage when the state of a component is updated and the application is repainted.
* **Unmounting:** As the name suggests Unmounting is the final step of the component lifecycle where the component is removed from the page.

React provides the developers with a set of predefined functions that if present are invoked around specific events in the lifetime of the component. Developers are supposed to override the functions with desired logic to execute accordingly. We have illustrated the gist in the following diagram.



Now let us describe each phase and its corresponding functions.

### ****Functions of each Phase of Lifecycle:****

#### **Initialization**

In this phase, the developer has to define the props and initial state of the component this is generally done in the constructor of the component. The following code snippet describes the initialization process.

class Clock extends React.Component {   
 constructor(props)   
 {   
 // Calling the constructor of   
 // Parent Class React.Component   
 super(props);   
   
 // Setting the initial state   
 this.state = { date : new Date() };   
 }   
}

#### **Mounting**

[Mounting](https://www.geeksforgeeks.org/explain-the-meaning-of-mounting-and-demounting/) is the phase of the component lifecycle when the initialization of the component is completed and the component is mounted on the [DOM](https://www.geeksforgeeks.org/dom-document-object-model/) and rendered for the first time on the webpage. Now React follows a default procedure in the Naming Conventions of these predefined functions where the functions containing “Will” represents before some specific phase and “Did” represents after the completion of that phase. The mounting phase consists of two such predefined functions as described below.

* First initialize the data and the states in the constructor
* [**componentDidMount() Function**](https://www.geeksforgeeks.org/reactjs-componentdidmount-method/): This function is invoked right after the component is mounted on the DOM i.e. this function gets invoked once after the render() function is executed for the first time

#### **Updation**

React is a JS library that helps create Active web pages easily. Now active web pages are specific pages that behave according to their user. For example, let’s take the GeeksforGeeks {IDE} webpage, the webpage acts differently with each user. User A might write some code in C in the Light Theme while another User may write Python code in the Dark Theme all at the same time. This dynamic behavior that partially depends upon the user itself makes the webpage an Active webpage. Now how can this be related to Updation? Updation is the phase where the states and props of a component are updated followed by some user events such as clicking, pressing a key on the keyboard, etc. The following are the descriptions of functions that are invoked at different points of the Updation phase.

* [**getDerivedStateFromProps**](https://www.geeksforgeeks.org/react-js-static-getderivedstatefromprops/): getDerivedStateFromProps(props, state) is a static method that is called just before render() method in both mounting and updating phase in React. It takes updated props and the current state as arguments.

static getDerivedStateFromProps(props, state) {  
 if(props.name !== state.name){  
 //Change in props  
 return{  
 name: props.name  
 };  
 }  
 return null; // No change to state  
}

* [**setState() Function**](https://www.geeksforgeeks.org/reactjs-setstate/): This is not particularly a Lifecycle function and can be invoked explicitly at any instant. This function is used to update the state of a component. You may refer to [this article](https://www.geeksforgeeks.org/reactjs-state-react/) for detailed information.
* [**shouldComponentUpdate() Function**](https://www.geeksforgeeks.org/reactjs-shouldcomponentupdate-method/): By default, every state or props update re-renders the page but this may not always be the desired outcome, sometimes it is desired that updating the page will not be repainted. The shouldComponentUpdate() Function fulfills the requirement by letting React know whether the component’s output will be affected by the update or not. shouldComponentUpdate() is invoked before rendering an already mounted component when new props or states are being received. If returned false then the subsequent steps of rendering will not be carried out. This function can’t be used in the case of [forceUpdate().](https://www.geeksforgeeks.org/reactjs-forceupdate-method/) The Function takes the new Props and new State as the arguments and returns whether to re-render or not.
* [**getSnapshotBeforeUpdate() Method**](NULL)**:** The getSnapshotBeforeUpdate() method is invoked just before the DOM is being rendered. It is used to store the previous values of the state after the DOM is updated.
* [**componentDidUpdate() Function**](https://www.geeksforgeeks.org/reactjs-componentdidupdate-method/): Similarly this function is invoked after the component is rerendered i.e. this function gets invoked once after the render() function is executed after the updation of State or Props.

#### **Unmounting**

This is the final phase of the lifecycle of the component which is the phase of unmounting the component from the DOM. The following function is the sole member of this phase.

* [**componentWillUnmount() Function**](https://www.geeksforgeeks.org/reactjs-componentwillunmount-method/): This function is invoked before the component is finally unmounted from the DOM i.e. this function gets invoked once before the component is removed from the page and this denotes the end of the lifecycle.

We have so far discussed every predefined function there was in the lifecycle of the component, and we have also specified the order of execution of the function.

### ****Implementing the lifecycle methods****

Let us now see one final example to finish the article while revising what’s discussed above.

First, create a react app and edit your **index.js** file from the src folder.

**src index.js:**

* Javascript

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom/client';    class Test extends React.Component {      constructor(props) {          super(props);          this.state = { hello: "World!" };      }        componentDidMount() {          console.log("componentDidMount()");      }        changeState() {          this.setState({ hello: "Geek!" });      }        render() {          return (              <div>                  <h1>GeeksForGeeks.org, Hello{this.state.hello}</h1>                  <h2>                      <a onClick={this.changeState.bind(this)}>Press Here!</a>                  </h2>              </div>);      }        shouldComponentUpdate(nextProps, nextState) {          console.log("shouldComponentUpdate()");          return true;      }        componentDidUpdate() {          console.log("componentDidUpdate()");      }  }    const root = ReactDOM.createRoot(document.getElementById("root"));  root.render(      <Test />  ); |

**The below Program implement the component life cycle function as follows:**

import React,{Component} from 'react'

class Footer extends Component

{

    state = {

    name:' '   /\* this is initializing the state \*/

    }

    changed =(evnt)=>{

       // console.log(evnt.target.value)

        this.setState({name:evnt.target.value})  /\*changing the stae by geting key typeved event\*/

        console.log(this.state.name)

    }

    componentDidMount()

    {

        this.setState({name:"The value from componenet Did mount"})

        console.log('componenet did mount is called');

    }

    componentWillUnmount()

    {

        console.log('comonenet will unmount is called');

    }

    render()

    {

        return (

            <div>

            <h2>{this.props.trademark}</h2>

            <input onChange={this.changed} type="text" value={this.state.name}/>

            </div>

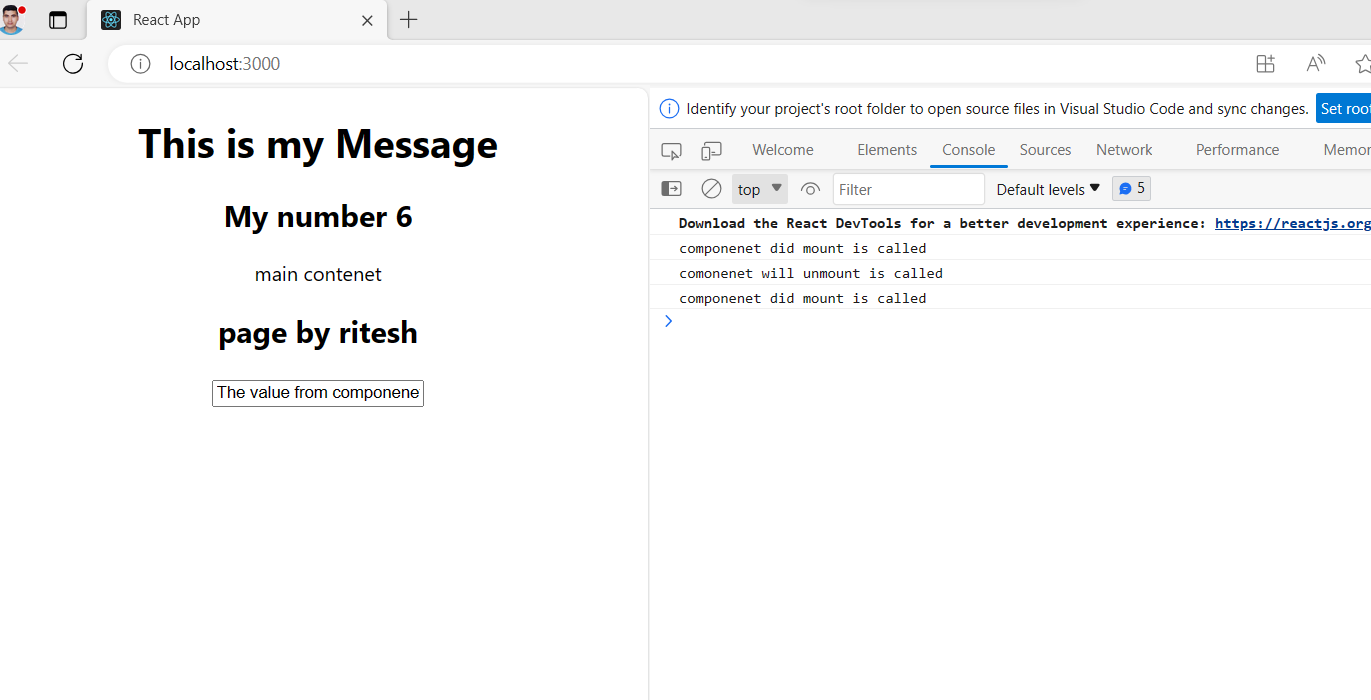
        )

    }

}

export default Footer

output:



In the above program component dismount called two times because This function is invoked right after the component is mounted on the DOM i.e. this function gets invoked once after the render () function is executed for the first time (The component is nested in side App component)