**What is react hook?**

## React hooks are simply javascript function that can use to isolate the reusable part from a functional component. Hook Rules

There are 3 rules for hooks:

* Hooks can only be called inside React function components.
* Hooks can only be called at the top level of a component.
* Hooks cannot be conditional

**Note:** Hooks will not work in React class components.

**Types of React Hooks**

The Built-in React Hooks are:

* **State Hooks**
* **Context Hooks**
* **Ref Hooks**
* **Effect Hooks**
* **Performance Hooks**
* **Resource Hooks**
* **Other Hooks**

## What is useEffect hook in React?

The useEffect in ReactJS is used to handle the side effects such as fetching data and updating DOM. This hook runs on every render but there is also a way of using a dependency array using which we can control the effect of rendering.

## ****How does it work?****

* You call useEffect with a callback function that contains the side effect logic.
* By default, this function runs after every render of the component.
* You can optionally provide a dependency array as the second argument.
* The effect will only run again if any of the values in the dependency array change.

## Importing useEffect hook

To import the useEffect hook, write the following code at the top level of your component

import { useEffect } from "react";

## ****Structure of useEffect hook****

The useEffect hook syntax accepts two arguments where the second argument is optional

### **React useEffect Hook Syntax:**

useEffect(<FUNCTION>, <DEPENDECY>)

### React useEffect Hook ShortHand for:

* **FUNCTION:** contains the code to be executed when useEffect triggers.
* **DEPENDENCY:**is an optional parameter, useEffect triggers when the given dependency is changed.

## Controlling side effects in useEffect :

1. To run useEffect on every render do not pass any dependency

useEffect(()->{  
 // Example Code  
})

2. To run useEffect only once on the first render pass any empty array in the dependecy

useEffect(()->{  
 // Example Code  
}, [] )

3. To run useEffect on change of a particular value. Pass the state and props in the dependency array

useEffect(()->{  
 // Example Code  
}, [props, state] )

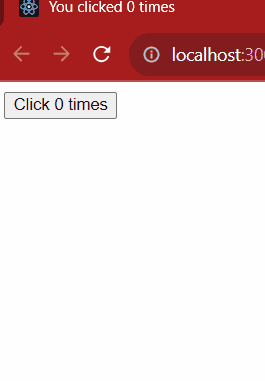
### React UseEffect Hook Example:

Let’s look at an example of how to use the useEffect hook as a feature that can mimic the life-cycle methods but in functional components. The functional component will look like the code below:

**Example:**This example demonstrates the use of useEffect Hooks to render the click counts.

|  |
| --- |
| // File name - HookCounterOne.js  // useEffect is defined here    import { useState, useEffect } from "react";    **function** HookCounterOne() {      const [count, setCount] = useState(0);        useEffect(() => {          document.title = `You clicked ${count} times`;      }, [count]);    **return** (          <div>              <button onClick={() => setCount((prevCount) => prevCount + 1)}>                  Click {count} times{" "}              </button>          </div>      );  }  export **default** HookCounterOne; |

**Output:**Initially, the document title reads “You clicked 0 times”. when you click on the button, the count value increments, and the document title is updated.



* **useEffect** triggers a function on every component render, leveraging React to execute specified tasks efficiently.
* Positioned within the component, it grants easy access to state and props without additional coding.
* For replicating lifecycle methods in functional components, copy and customize the provided code snippet according to your needs.

**Timer Example:**

import { useState, useEffect } from "react";

const Timer = () => {

    const [timer,setTimer] = useState(new Date());

    useEffect(() => {

        setInterval(() => {

            setTimer(new Date())

        },1000)

    },[])

    return (

        <div>

            <p>Timer example</p>

            <p>{timer.toLocaleTimeString()}</p>

        </div>

    );

};

export default Timer;

