useEffect in React.js

Definition:

The useEffect hook in React allows you to perform side effects in functional components — such as fetching data, updating the DOM, or setting timers — after the component renders.

Explanation:

React components mainly handle UI rendering, but sometimes you need to perform extra tasks (side effects). The useEffect hook lets you run such code after rendering without blocking the UI.

Syntax:

```
useEffect(() => { // Code to run after render return () => { // Optional
cleanup code }; }, [dependencies]);
```

Example 1 — Run Once on Mount:

```
import React, { useEffect } from 'react'; function Welcome() {
useEffect(() => { console.log("Component mounted!"); document.title =
"Welcome Page"; }, []); return Welcome to my page!; }
```

Runs only once after component loads because of the empty dependency array [].

Example 2 — Run When State Changes:

```
import React, { useState, useEffect } from 'react'; function Counter() {
const [count, setCount] = useState(0); useEffect(() => {
console.log("Count changed:", count); }, [count]); return ( Count:
{count} setCount(count + 1)}>Increase ); }
```

This effect runs every time 'count' changes because it is listed in the dependency array.

Example 3 — Cleanup Function:

```
import React, { useEffect } from 'react'; function Timer() {
useEffect(() => { const interval = setInterval(() => {
console.log("Running timer..."); }, 1000); return () => {
clearInterval(interval); console.log("Timer cleared!"); }; }, []);
return Timer started! Check console.; }
```

The cleanup function clears the timer when the component unmounts.

Summary Table:

Dependency Array	When useEffect Runs	Example Use
[] (empty)	Once after initial render	Fetch data, initialize

[variable]	After every change of variable	React to state changes
(none)	After every render	Rarely used (may cause loops)

Key Points:

- useEffect replaces lifecycle methods like componentDidMount and componentDidUpdate. - Always specify dependencies to control when the effect runs. - Use cleanup functions to stop intervals or listeners. - Avoid making the useEffect callback async directly — use an inner async function.