React `useEffect` Hook

What is `useEffect`?

- The `useEffect` hook is a function in React that lets you perform side effects in functional components.
- Side effects include tasks such as:
    - Fetching data from an API
    - Subscribing or unsubscribing to services (e.g., WebSockets

Why Use `useEffect`?

- **Side Effects**: React components render based on state and props, but real-world applications often require actions beyond rendering, like interacting with external systems

How `useEffect` Works

`useEffect` runs after the render phase by default. It can be configured to run:

1. On every render.
2. Only when specified dependencies change.
3. Once, after the initial render.
4. During cleanup before unmounting or re-running.

Syntax

```
useEffect(() => {
  // Side effect logic here

  return () => {
    // Optional cleanup logic here
  };
}, [dependencies]);
```

Explanation

- **First Argument (`() => {}`)**: A function where you write the side effect logic. Optionally returns a cleanup function.
- **Second Argument (`[dependencies]`)**:
  - A list of values that determine when the effect should re-run.
  - **Empty array (`[]`)**: Runs only once, after the initial render.
  - **No array**: Runs after every render.
  - **Array with dependencies**: Runs only when one or more dependencies change.

Key Use Cases

1. Run Once (On Mount)

Used to fetch data or perform setup when the component is first rendered.

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

function App() {
  const [data, setData] = useState(null);

  useEffect(() => {
    fetch("https://api.example.com/data")
      .then((response) => response.json())
      .then((data) => setData(data));
  }, []); // Empty dependency array ensures this runs only once.

  return <div>{data ? JSON.stringify(data) : "Loading..."}</div>;
}
```

2. Run on Dependency Change

Re-run the effect when specific state or props change.

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

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

  useEffect(() => {
    console.log(`Count changed to: ${count}`);
  }, [count]); // Runs whenever `count` changes.

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

3. Cleanup Effect

Used to unsubscribe or clean up resources to prevent memory leaks.

```
import React, { useEffect } from "react";

function Timer() {
  useEffect(() => {
    const timer = setInterval(() => {
      console.log("Timer tick");
    }, 1000);

    return () => {
      clearInterval(timer); // Cleanup function to clear the
timer.
    };
  }, []); // Runs once, and cleanup runs when component unmounts.

  return <div>Check the console for timer ticks!</div>;
}
```

4. Multiple Effects

You can use multiple `useEffect` hooks for different tasks.

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

function App() {
  const [count, setCount] = useState(0);
  const [darkMode, setDarkMode] = useState(false);

  useEffect(() => {
    console.log("Count updated:", count);
  }, [count]);

  useEffect(() => {
    document.body.style.backgroundColor = darkMode ? "#333" :
"#FFF";
  }, [darkMode]);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count +
1)}>Increment</button>
      <button onClick={() => setDarkMode(!darkMode)}>Toggle Dark
Mode</button>
    </div>
  );
}
```

5. Fetching Data with Cleanup

Handle asynchronous data fetching with cleanup for component unmounting.

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

function FetchData() {
  const [data, setData] = useState(null);
  const [error, setError] = useState(null);

  useEffect(() => {
    let isMounted = true; // To avoid setting state on unmounted

    fetch("https://api.example.com/data")
      .then((response) => response.json())
      .then((result) => {
        if (isMounted) setData(result);
      })
      .catch((error) => {
        if (isMounted) setError(error);
      });

    return () => {
      isMounted = false; // Cleanup function
    };
  }, []);

  if (error) return <div>Error: {error.message}</div>;
  if (!data) return <div>Loading...</div>;
  return <div>{JSON.stringify(data)}</div>;
}
```