In this lesson, we learned how to write effects that manage timers, manipulate the DOM, and fetch data from a server. In earlier versions of React, we could only have executed this type of code in the [lifecycle methods of class components](https://reactjs.org/docs/state-and-lifecycle.html#adding-lifecycle-methods-to-a-class), but with the Effect Hook, we can perform these types of actions in function components with ease!

Let’s review the main concepts from this lesson:

* useEffect() - we can import this function from the 'react' library and call it in our function components
* *effect* - refers to a function that we pass as the first argument of the useEffect() function. By default, the Effect Hook calls this effect after each render
* *cleanup function* - the function that is optionally returned by the effect. If the effect does anything that needs to be cleaned up to prevent memory leaks, then the effect returns a cleanup function, then the Effect Hook will call this cleanup function before calling the effect again as well as when the component is being unmounted
* *dependency array* - this is the optional second argument that the useEffect() function can be called with in order to prevent repeatedly calling the effect when this is not needed. This array should consist of all variables that the effect depends on.

The Effect Hook is all about scheduling when our effect’s code gets executed. We can use the dependency array to configure when our effect is called in the following ways:

| **Dependency Array** | **Effect called after first render & …** |
| --- | --- |
| undefined | every re-render |
| Empty array | no re-renders |
| Non-empty array | when any value in the dependency array changes |

Hooks gives us the flexibility to organize our code in different ways, grouping related data as well as separating concerns to keep code simple, error-free, reusable, and testable!