Terminology

useEffect A function that adds the ability to perform side-effects from a function component.

mount when a component is first added to a page

unmount when a component is removed from the page

useEffect arguments

first argument The first argument to useEffect should be a function. Whatever happens in that function will get run just **after** the component renders.

the return value of the first argument The first argument to useEffect can also, optionally, **return** a function. If it does, the returned function will get run just **before** the component renders

the second argument (optional) By default, useEffect's functions get run every time the component renders. If a second argument is provided, it should be an array. When an array is passed, useEffect's functions get run only on the renders when at least one of the array's values have changed

passing an empty array Because useEffect's functions get run only on the renders when at least one of the array's values have changed, passing an empty array, `[]`, as the second argument is a way of saying "don't run these functions on ***any*** render except the first and last

useEffect (post-render examples)

```
// Show the alert after every render
useEffect(() => {
  alert("This component just rendered!")
})

// Show the alert after the first render only
useEffect(() => {
  alert("This component just mounted!")
}, [])

// After first render AND after `someValue` changes
useEffect(() => {
  alert("someValue just changed!")
}, [someValue])
```

useEffect (pre-render examples)

```
// Before every render AND before component unmounts
useEffect(() => {
  return () => alert("About to render!")
})

// Show the alert ONLY just before unmounting
useEffect(() => {
  return () => alert("About to unmount!")
}, [])

// Show the alert just before the component unmounts
// AND before every render where `someValue` changes
useEffect(() => {
  return () => alert("someValue about to change!")
}, [someValue])
```

useEffect (putting them together)

```
// Run the "just changed" alert after someValue
// changes AND after the component mounts
// Run the "about to change" alert before someValue
// changes AND before the component unmounts
useEffect(() => {
  alert("someValue just changed!")

  return () => alert("someValue about to change!")
}, [someValue])

// Run the "just changed" alert after valueOne or
// valueTwo changes AND after the component mounts
// Run the "about to change" alert before valueOne
// or valueTwo changes AND before the component
// unmounts
useEffect(() => {
  alert("Either valueOne or valueTwo just changed!")

  return () => {
    alert("valueOne or valueTwo is about to change!")
  }
}, [valueOne, valueTwo])
```

Common Gotcha

```
// The gotcha: If you're not careful, calling a setter
// function inside useEffect can cause an infinite loop
const [count, setCount] = useState(0)

useEffect(() => {
  fetch("https://some.site/where/we/saved/the/count")
    .then(response => response.json())
    .then(savedCount => {
      // Calling setCount re-renders the component
      // which means useEffect will get called
      // again, which means setCount will get
      // called again, aaaaand INFINITE LOOP
      setCount(savedCount)
    })
})

// the fix: use that second useEffect argument to
// prevent unnecessary useEffect calls
const [count, setCount] = useState(0)

useEffect(() => {
  fetch("https://some.site/where/we/saved/the/count")
    .then(response => response.json())
    .then(savedCount => {
      setCount(savedCount)
    })
// This little array argument right here is the fix!
// Now that fetch will only get called after the
// very first render
}, [])
```