## Terminology

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

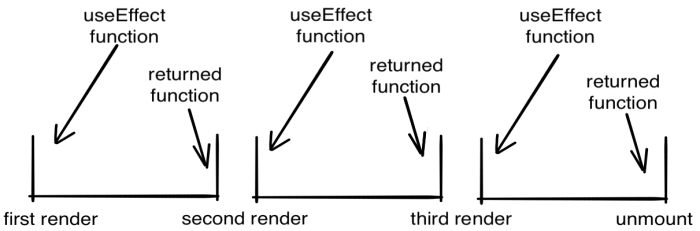
**mount** when a component is first added to a page

**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 when the component mounts and after each render.

**the return value of the first argument** The first argument to useEffect can also, optionally, **return** a "cleanup" function. If it does, the returned function will get run at the end of the render cycle, just before the next render (or unmount) happens.




useEffect lifecycle

**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 "only run these functions on the first render and the unmount

## useEffect 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 Examples (with return)

```
// Let's say we have a chat app with chatrooms.
// This code sets up a listener to display chat
// messages we receive. With this code, we will
// setup and then remove the listener on *every* render
useEffect(() => {
  listenForMessagesFrom(chatroomId, addToMessageList);

  return () => {
    stopListeningForMessagesFrom(chatroomId);
  };
});

// Same as above, except we're passing an empty array as
// the second argument to useEffect, so we only add the
// listener on the very first render, and only remove it
// right before unmounting. Much more efficient!
useEffect(() => {
  listenForMessagesFrom(chatroomId, addToMessageList);

  return () => {
    stopListeningForMessagesFrom(chatroomId);
  };
}, []); // we added the empty array on this line

// What if we change chatrooms, though? We don't want
// to display messages from our old chatroom after
// changing to a new one. So let's set it up to ALSO
// rerun the listener cleanup and setup code
// **whenever chatroomId changes**
useEffect(() => {
  listenForMessagesFrom(chatroomId, addToMessageList);

  return () => {
    stopListeningForMessagesFrom(chatroomId);
  };
}, [chatroomId]); // we added chatroomId on this line
```

## 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(() => {
  const savedCount = localStorage.get("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(() => {
  const savedCount = localStorage.get("savedCount");
  setCount(savedCount);
// This little array argument right here is the fix!
// Now this code will only run after the first render
}, []);
```