# Passing Array Dependencies

In this lesson, we'll discuss how the useEffect function can be used for specific functionality, especially for passing arrays.

**WE'LL COVER THE FOLLOWING**

It's interesting that the effect function is invoked every time there's an update. That's great, but it's not always the desired functionality.

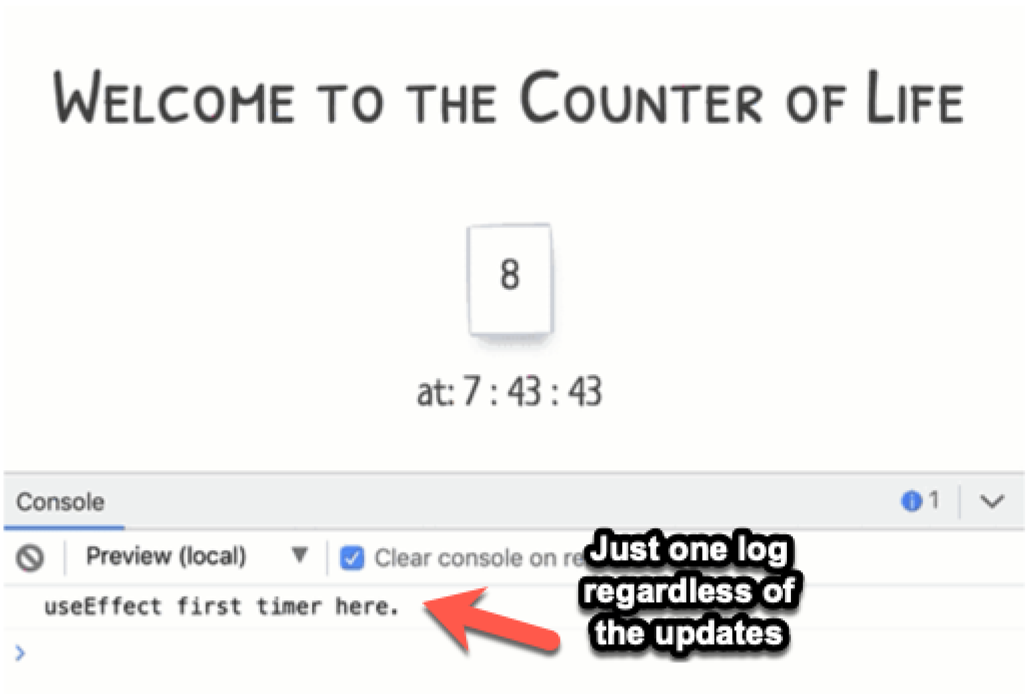
## Using Effect for Specific Functionality #

What if you want to run the effect function only when the component mounts?

That's a common use case and `useEffect` takes a second parameter, an array of dependencies to handle this.

If you pass in an empty array, the effect function is run only on the mount—subsequent re-renders don't trigger the effect function.

```
useEffect(() => {
    console.log("useEffect first timer here.")
}, [])
```


Welcome to the Counter of Life
8
at: 7 : 43 : 43
Console
Preview (local)
Clear console on re
Just one log regardless of the updates

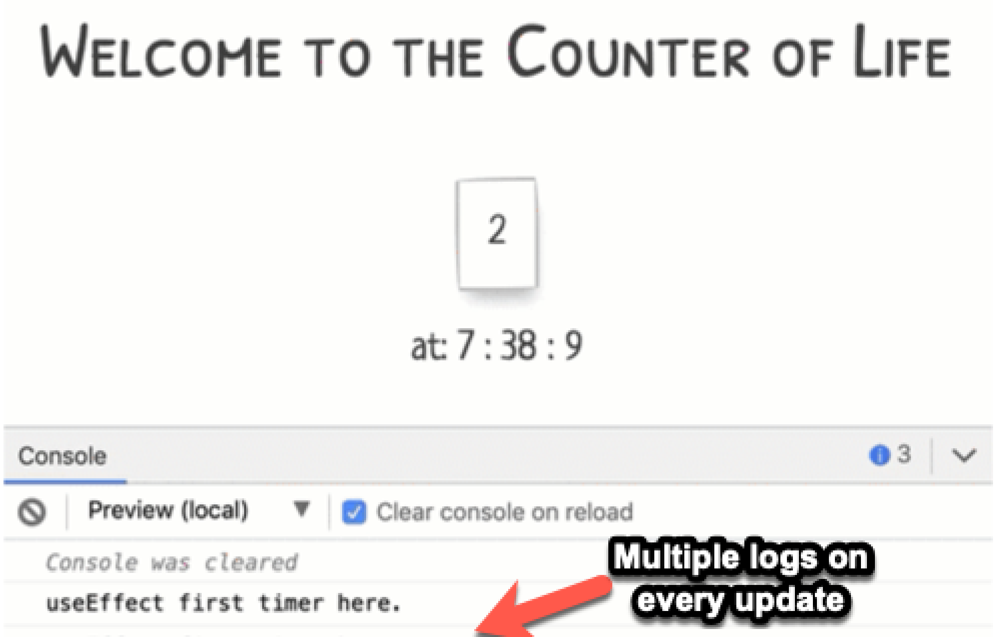
```
useEffect first timer here.
```

## Effect’s Array Argument

If you pass any values into this array, then the effect function will run on the mount, and anytime the values passed are updated, i.e., if any of the values are changed, the effect call will re-run.

```
useEffect(() => {
    console.log("useEffect first timer here.")
}, [count])
```

The effect function will run on the mount, and whenever the count variables change.



```
Console was cleared
useEffect first timer here.
```

```
useEffect first timer here.
useEffect first timer here.
```

What about subscriptions?

It’s common to subscribe and unsubscribe from certain effects in certain apps. Consider the following:

```
useEffect(() => {
  const clicked = () => console.log('window clicked');
  window.addEventListener('click', clicked);
}, [])
```

In the effect above, upon mounting, a click event listener is attached to the window.

How do we unsubscribe from this listener when the component is unmounted?

Well, `useEffect` allows for this.

If you return a function within your effect function, it will be invoked when the component unmounts. This is the perfect place to cancel subscriptions as shown below:

```
useEffect(() => {
    const clicked = () => console.log('window clicked');
    window.addEventListener('click', clicked);

    return () => {
      window.removeEventListener('click', clicked)
    }
 }, [])
```

There’s a lot more you can do with the `useEffect` hook such as making API calls.

---

In the next lesson, you’ll create your own first hook. Excited!