useEffect hook

`useState` isn't the only hook in React

- `useEffect` is another (and there are more)

`useEffect()` is passed a callback

- callback runs *after* the component renders

Basic example

```
import  { useEffect } from 'react';

function App() {
  useEffect( () => console.log('in effect') );
  console.log('in app');
  return (
    <div className="app">
    </div>
  );
}
```

```
in app
in effect
```

useEffect callback called on every rerender

```
import  { useState, useEffect } from 'react';

function App() {
  const [ count, setCount ] = useState(0);
  useEffect( () => console.log('in effect') );
  console.log('in app');
  return (
    <div className="app">
      <button onClick={ () => setCount(count+1) }>
        {count}
      </button>
    </div>
  );
}
```

Each `in app` followed by an `in effect`

Why "Effect"?

`useState` gives us a state

What does `useEffect` give us?

- A "side effect" of rendering
- "side effects" are something to minimize
- but can be useful

useEffect dependency array

useEffect callback doesn't have to run on ALL renders

- can be passed a second argument
- the "dependency array"
- lists values to watch
- a change in a value triggers callback to run
    - only checked on render though

Dependency Array Demonstration

```
function App() {
  const [ count, setCount ] = useState(0);
  const [ watched, setWatched ] = useState(0);
  useEffect(
    () => console.log('in effect'),
    [ watched ],
  );
  console.log('in app');
  return (
    <div className="app">
      <button onClick={ () => setCount(count+1) }>
        Unwatched: {count}
      </button>
      <button onClick={ () => setWatched(watched+1) }>
        Watched: { watched }
      </button>
    </div>
  );
}
```

Simple Results

- Whenever the `watched` value changed
    - useEffect callback was called
- When an unwatched value changed
    - useEffect callback NOT called

Tricky Dependency

When does useEffect callback get called?

```
function App() {
  const [ watched, setWatched ] = useState(0);
  const trigger = watched % 3 === 1;
  useEffect(
    () => console.log('in effect'),
    [ trigger ],
  );
  console.log('in app');
  return (
    <div className="app">
      <button onClick={ () => setWatched(watched+1) }>
        Watched: { watched }
      </button>
    </div>
  );
}
```

Tricky results

```
const trigger = watched % 3 === 1;
useEffect(
  () => console.log('in effect'),
  [ trigger ],
);
```

Called on 0, 1, 2, 4, 5, 7, 8, 10, etc

- 0: because first render, everything is "changed"
- 1, 4, 7: because trigger changes from `false` to `true`
- 2, 5, 8: because trigger changes from `true` to `false`
- 3, 6, 9: no change because trigger remains `false`

What if empty deps array?

What if:

```
useEffect(
  () => console.log('in effect'),
  [],
);
```

Empty dependency array results

- useEffect callback runs on first render
    - Not on any later renders
- If component is removed from page and reapplied
    - callback once again runs on first render
- If multiple instances of component
    - callback runs on first render of each instance

Common uses of useEffect

Load data from service

- call on first render only (`[]`)
- OR call when input to load data changes

Linter will give warning if useEffect callback

- uses values that COULD change
- but are not listed in the dependency array

This includes state and state setter functions

- generally a good idea to add these to dependency array
- some (setter function) may not change, that's fine

Infinite Loop

If you change a state that is in the dependency array

```
const [state, setState] = useState(0);
useEffect(
  () => setState(state+1),
  [state, setState],
);
```

- Infinite Loop!

Conclusion:

- Either useEffect callback should NOT change state it depends on
- OR useEffect callback only conditionally changes the state

useEffect callback can return a function

This function is called when:

- component removed from page
- this useEffect called again

This function is used for "cleanup"

Example: if your effect created timeouts or intervals

- remove them because component and component state won't be there to update

useEffect cleanup function

```
useEffect(
  () => {
    console.log('in effect', count);
    return () => {
      console.log('cleanup', count);
    };
  },
  [],
);
```

Summary - useEffect

A hook that takes a callback

- callback runs after component renders
- usually used to load data
    - now you know you need it

Summary - Dependency Array

Second param to useEffect is a dependency array

- if not present
    - callback runs every render
- if present but empty
    - callback runs after first render only
- if present with values
    - callback runs if any values change

Dep. array should include any values callback uses

- Make changing any of these values conditional!
- an array with only setters == first render only