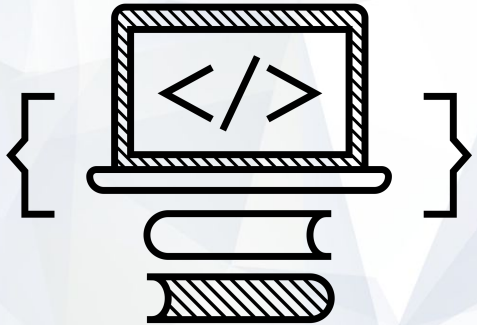
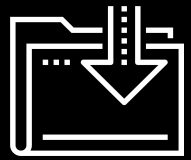


useEffect and Custom Hooks

Skills Bootcamp in Front-End Web Development

Lesson 14.1



WELCOME

Learning Objectives

By the end of class, you will be able to:

- Articulate the term “effect” in the broader sense of programming.
- Utilize React’s most common built-in Hooks: `useState` and `useEffect`.
- Create a custom reusable Hook that follows the two rules of Hooks.

Hooks & State Management

React Hooks

Review:
What are Hooks?

Hooks are functions that allow us to extend the capabilities of our components.

With Hooks we are able to add state and use other React features inside of our components.

Review:

What are the two rules of Hooks?

Two rules of React Hooks

Rule #1

- Hooks should **only** be called from the top level of a component.
- Never call Hooks from within loops, conditional or nested functions.

Rule #2

- Hooks should only be called from within components or custom hooks.
- They should **never** be called from regular Javascript functions.

If we vary from these rules, our components will not work properly.

These rules apply to <u>all</u> Hooks.

React State Management

Review:

What is state, and how does it relate to the useState Hook?

“State” is a specialized property that is used for component specific data storage.

State and the useState Hook

- Component state values differ from regular variables since updating the state triggers automatic browser updates within the React application.
- A component can set and update its own state, whereas its props are always received from up above and considered immutable (can't/shouldn't be changed).
- The `useState` Hook offers a way for components to manage state directly, eliminating the need for data retrieval through prop drilling.
- When called, the useState hook accepts an initial value as an argument. Afterwards, it returns an array containing a stateful value and a function that can be used to update it.

Instructor Demonstration

useState

Questions?

Activity: `useState`

In this activity, you will practice using the `useState` Hook in React.

Suggested Time:

15 Minutes

Time’s Up! Let’s Review.

Why might using `value` with `onChange` be a good idea?

Review: useState Activity

It all comes down to controlled input vs. uncontrolled input.

01 Controlled input

Accepts its current value as a prop. The input's value is set to match the state variable:

`value={formData.username}`,

Has a callback that allows you to change its value. Whenever `onChange` updates its value, it's essentially the input controlling itself.

02 Uncontrolled input

Uncontrolled inputs are like traditional HTML form inputs.

An initial value may be provided through the value prop but the browser is still responsible for maintaining the state of the input.

useEffect Hook

In programming, what exactly do we mean by ‘effects’?

Effects, also commonly referred to as a *side effects*, are bits of code that are responsible for managing external interactions.

useEffect Hook

Most common effects (or side effects):

useEffect Hook

- The useEffect hook provides a way to perform side effects.
- useEffect is a method that takes two arguments. The first is a callback function that will run after the component is rendered. The second argument is an array of dependencies.
- When the component re-renders, useEffect re-runs its callback function if any dependency values change. If not, React will skip the effect for that particular render.
- The useEffect hook makes it possible to run side effects independently of rendering. This is valuable because not all side effects need to occur with every component render.
- By specifying dependencies, we can control when the hook runs, avoiding unnecessary executions.

useEffect Syntax

Dependency array:

If `props.name` is provided, `useEffect` runs once on initial page render and subsequently whenever props.name changes.

```
  useEffect(() => {
// callback logic
}, [props.name]);
```

If there are no changes, React will skip the effect for that particular render.

Empty Array:

When an empty array is provided, `useEffect` will only run the **first time** the component renders.

```
  useEffect(() => {
// callback logic
}, []);
```

Instructor Demonstration

useEffect

Questions?

Activity: `useEffect`

In this activity, you will practice using the `useEffect` and `useState` Hooks by updating an incomplete component.

Suggested Time:

20 Minutes

Time’s Up! Let’s Review.

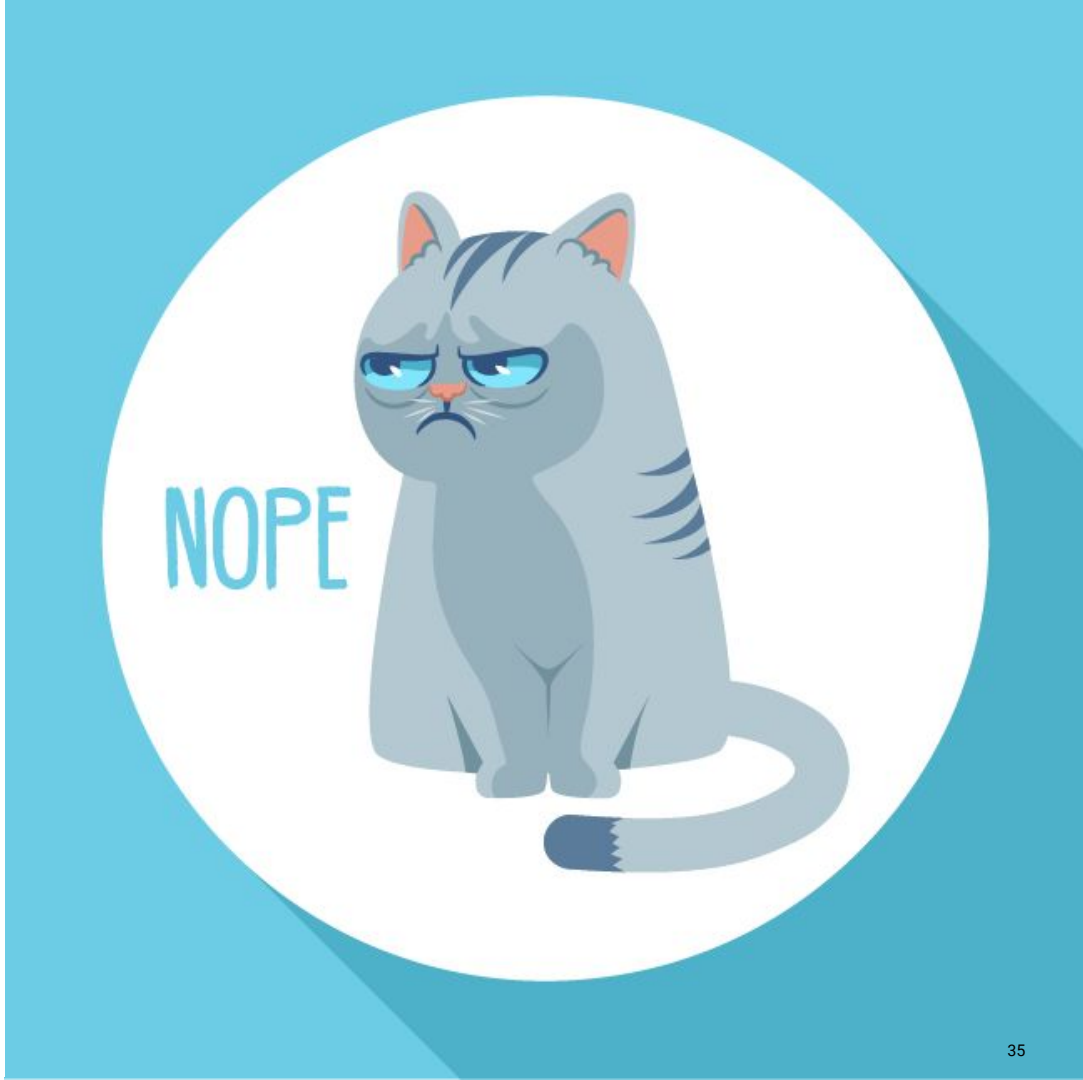
Why are Hooks useful?

Hooks make components much cleaner.

Using Hooks allows us to write fewer lines of code and manage state in a less complex way.

If we use Hooks, can state be used by other components?

The state used within a single component cannot be used by different components.



Can you think of a concept that would allow us to share state across components?

For now, we can use props, but in the future, you will learn a better way.

Questions?

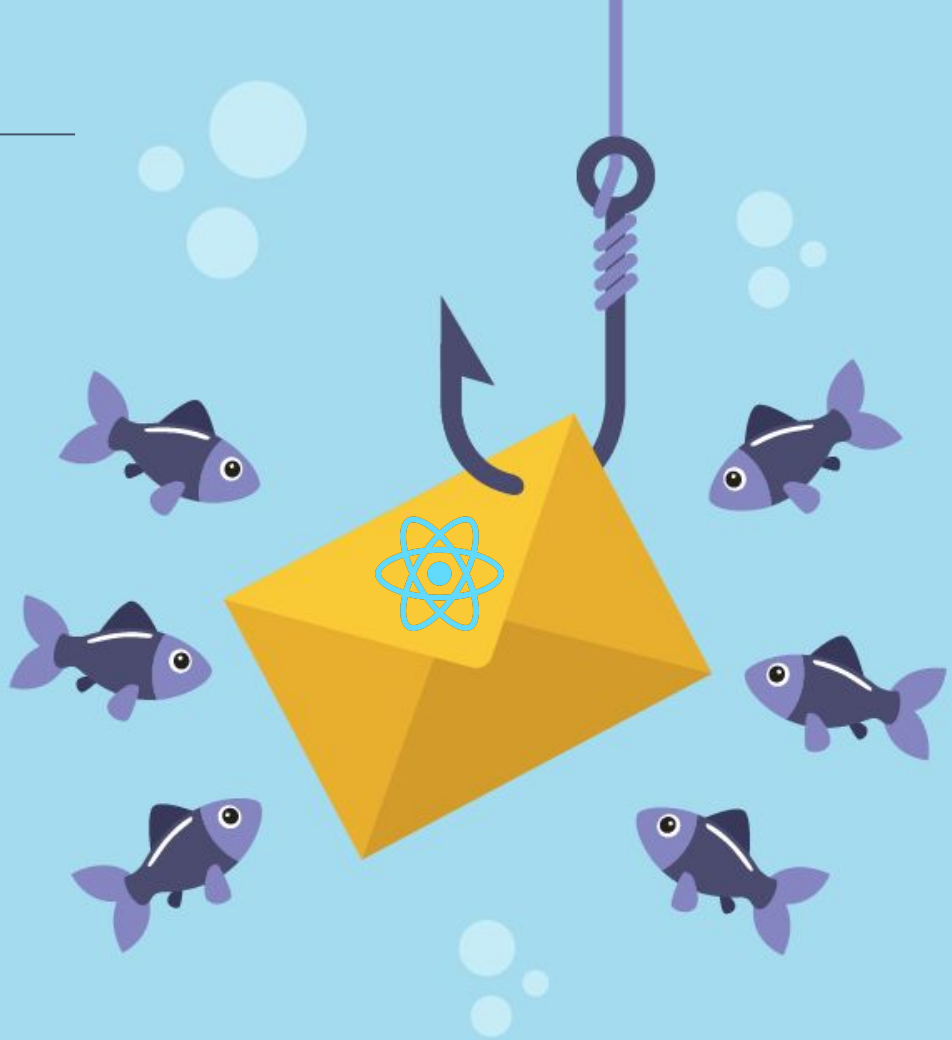

Break

Custom Hooks

Today we covered the built-in Hooks `useState` and `useEffect`, but there are many more Hooks.

The strong developer community around React has created a plethora of different [custom Hooks]() that you can plug into your applications.

This makes it so that all stateful logic is easy for the developer to find.



Instructor Demonstration

Custom Hooks

Custom Hooks

Only call Hooks from top-level components.

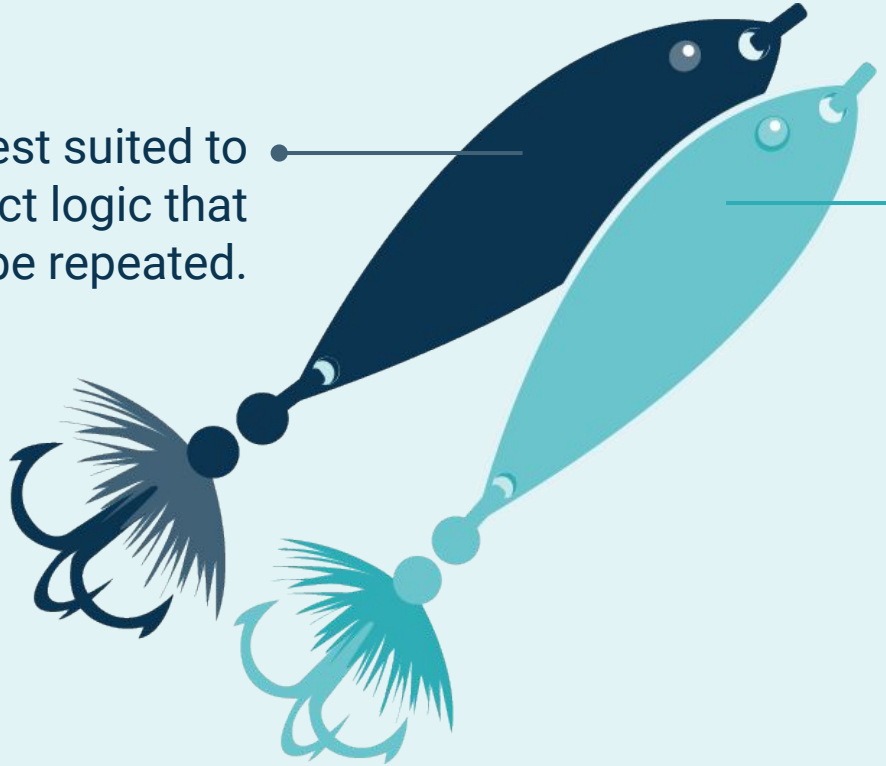
Only call Hooks from React components.

Never call Hooks from within loops, conditionals, or nested functions.

Never call a Hook from a regular JavaScript function.

Custom Hooks Can Be Practically Anything!

Custom Hooks are:



Best suited to extract logic that may be repeated.

A great way to keep your React functions pure.

Custom Hooks

As with the two rules of Hooks...

Custom Hooks must start with the word `use` so that React can ensure that your code is adhering to the two rules of Hooks.

As with `useState` and `useEffect`,

Different components that use the same custom Hook **do not** share the same state.

Activity: Custom Hooks

In this activity, you will practice using custom Hooks by creating a `useDebounce` Hook that will delay the invoking of a function for a given number of milliseconds.

Suggested Time:

15 Minutes

Time’s Up! Let’s Review.

Questions?

Activity: Third-Party Hooks

In this activity, you will practice using third-party Hooks. Specifically, you will be creating a survey form using the `react-hanger` package on npm.

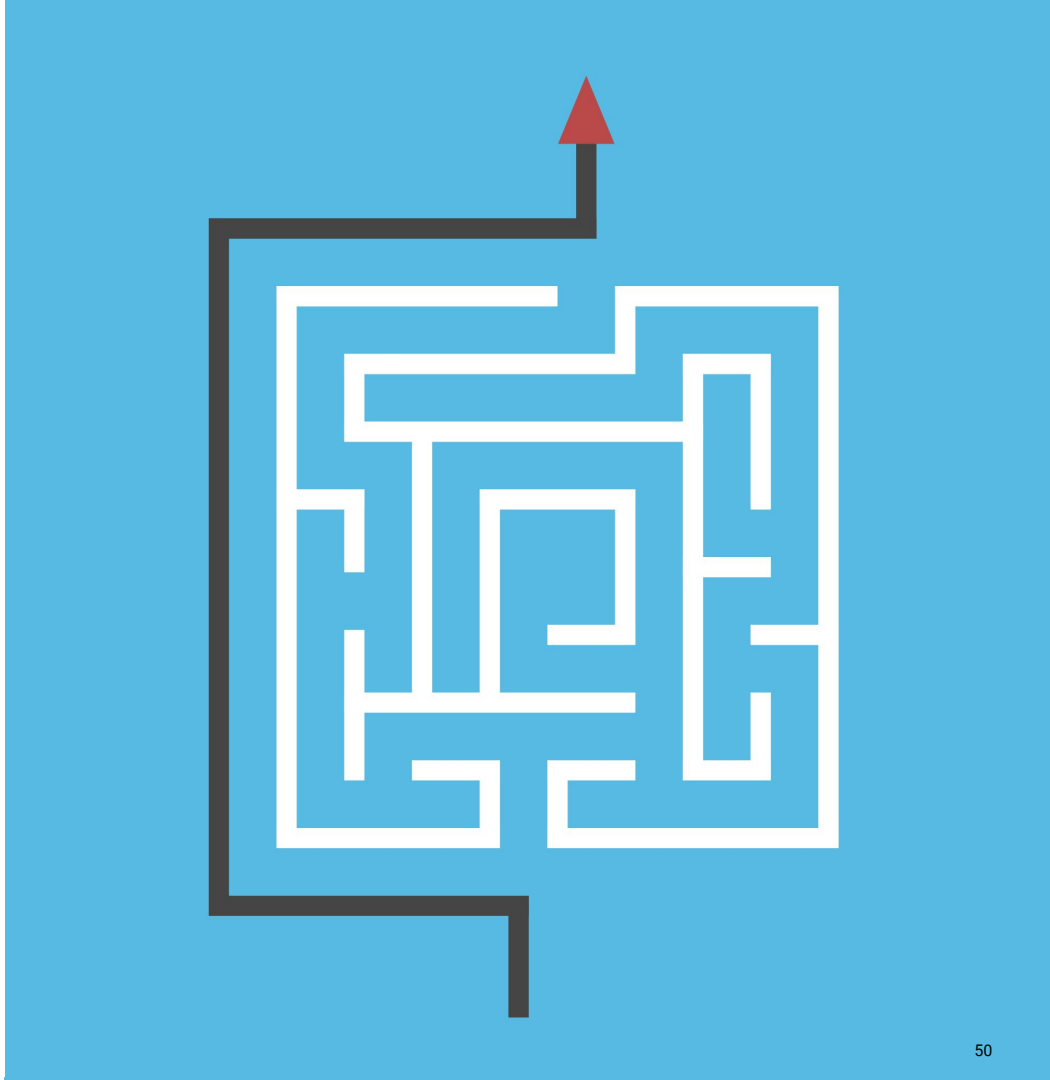
Suggested Time:

20 Minutes

`react-hanger` is one of many custom Hooks packages on

There are many ways to satisfy the requirements of this application.

It is recommended that you attempt the **most straightforward solution** first, then refactor your working app.



Time’s Up! Let’s Review.

Questions?

Next Class

Do your best to go through the following sections of the React documentation before the next class:

React Hooks Docs

React Docs Tutorial Blog Community Search v17.0.2 Languages GitHub

Introducing Hooks

Hooks are a new addition in React 16.8. They let you use state and other React features without writing a class.

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

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
```

INSTALLATION
MAIN CONCEPTS
ADVANCED GUIDES
API REFERENCE
HOOKS
1. Introducing Hooks
2. Hooks at a Glance
3. Using the State Hook
4. Using the Effect Hook
5. Rules of Hooks
6. Building Your Own Hooks
7. Hooks API Reference
8. Hooks FAQ

A Complete Guide to useEffect()

Overreacted

A Complete Guide to useEffect

March 9, 2019 · 49 min read

Translated by readers into: Español · Português do Brasil · 日本語 · 简体中文 · 繁體中文 · 한국어

You wrote a few components with Hooks. Maybe even a small app. You're mostly satisfied. You're comfortable with the API and picked up a few tricks along the way. You even made some custom Hooks to extract repetitive logic (300 lines gone!) and showed it off to your colleagues. "Great job", they said.

But sometimes when you `useEffect`, the pieces don't quite fit together. You have a nagging feeling that you're missing something. It seems similar to class lifecycles... but is it really? You find yourself asking questions like:

List of React Hooks

Collection of React Hooks

You can add your hooks by opening a pull-request at https://github.com/nikgraf/react-hooks.

filter by name

State Management (80) Sensor (29) UI (26) Web API (26) Network (25)

Found 440 entries

use-abortable-stream-fetch marconi/use-abortable-stream-fetch

```
import useAbortableStreamFetch from 'use-abortable-stream-fetch';
```

react hooks fetch ajax abort useState useEffect

The End