# Arrays Lab

- Due No Due Date
- Points 1
- Submitting a website url

 (https://github.com/learn-co-curriculum/phase-0-intro-to-js-2-array-lab)  (https://github.com/learn-co-curriculum/phase-0-intro-to-js-2-array-lab/issues/new)

# Learning Goals

- Practice writing arrays
- Practice using *destructive* methods to manipulate arrays
- Practice using *nondestructive* methods to manipulate arrays

# Introduction

We've learned about how arrays work and about the array methods built in to JavaScript that we can use to manipulate them. Now it's time to practice what we've learned.

If you haven't already, **fork and clone** this lab into your local environment. Navigate into its directory in the terminal, then run `code .` to open the files in Visual Studio Code.

# Instructions

Open up the `test` folder and take a look at `indexTest.js`. Note that some of the names of the functions you will be writing begin with `destructively` and some don't. This is a clue as to which `Array` method you will need to use for each function.

Note also that the first test asks for an array called `cats`, set to an initial value of `["Milo", "Otis", "Garfield"]`. In your functions, you will be accessing and manipulating this array.

Near the top of `indexTest.js` you will see the following:

```
beforeEach(function () {
  cats.length = 0;

  cats.push("Milo", "Otis", "Garfield");
});
```

What this code does is *reset* the array to its original contents before each test is run. The reason we need to do this is because some of your functions will be *destructive* — they will change the original `cats` array. This is a problem because it means the input to the remaining functions will be dependent on the outcome of other functions. It also means that the expected return value of a function might change if the tests are run in a different order. This makes it more difficult both to write tests in the first place and to figure out how to get the tests to pass. Resetting the array returns us to a blank slate between tests.

This is also a good illustration of why it's generally good practice to avoid mutating a program's state whenever possible. If we use only *nondestructive* methods, we have complete control over what's going into and coming out of the function. This makes our programs more robust, easier to maintain, and less prone to bugs.

Remember the workflow:

1. Install the dependencies using `npm install`.
2. Run the tests using `npm test`.
3. Read the errors; vocalize what they're asking you to do.
4. Write code; repeat steps 2 and 3 often until a test passes.
5. Repeat as needed for the remaining tests.

After you have all the tests passing, remember to commit and push your changes up to GitHub, then submit your work to Canvas using CodeGrade.