Review: Variables Lab

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

(https://github.com/learn-co-curriculum/phase-1-javascript-variables-lab) (https://github.com/learn-co-curriculum/phase-1-javascript-variables-lab/issues/new)

Learning Goals

- Practice using `const` and `let` to declare variables in JavaScript

Instructions

In this lab we'll practice declaring and assigning values to variables. We'll also go over how to read the test document. Understanding how to read the tests can be a valuable tool in figuring out exactly what you'll need to do to complete the lab.

Tests

When we want to run an experiment, we need to develop a hypothesis and we need to test it. In programming, we run tests to verify that programs behave the way we think they do. Tests help us identify bugs and judge how healthy our applications are.

We use tests to describe the program's behavior, just as you would in a professional coding environment, and we also use them as teaching tools. You are in charge of getting the tests to pass.

Structure

The structure of this lab — where its files and folders are located — looks roughly like the following:

```
├── CONTRIBUTING.md
├── LICENSE.md
├── README.md
├── index.js
├── node_modules/
├── package.json
└── test
    └── indexTest.js
```

All labs will more or less have the same structure. (And non-lab lessons, for that matter, will still have CONTRIBUTING.md, LICENSE.md, and README.md files.)

Code Along

This lesson is set up as a code-along, so you'll first need to **fork and clone** it to your local environment.

Quick Review:

1. Click the **Octocat** icon in the upper right of this page. This will bring you to GitHub. Click the **Fork** button. Verify that your GitHub username is showing in the **Owner** dropdown, then click the **Create fork** button.

2. Once your fork is created, click the **Code** button in GitHub, make sure **SSH** is selected, and copy the provided git URL info.

3. Make sure you're in `Development/code/phase-1` (or wherever you're storing your code for the course) and clone the repo to your local machine with `git clone` followed by the git URL you copied.

```
$ git clone git@github.com:your-github-username/phase-1-javascript-variables-lab.git
```

4. The previous command will create a folder in the `phase-1` folder containing your fork of this lab's repository. `cd` into the repository that you just cloned down in the terminal, then run `code .` to open the files in Visual Studio Code.

```
$ cd phase-1-javascript-variables-lab
$ code .
```

Open up `index.js` in your code editor; you should see, well, nothing. We'll fix that soon.

Now open up `test/indexTest.js` . Hey, there's something! What's all of this stuff doing?

Note: The `test/indexTest.js` has great info that we want to look at, but do not edit this file otherwise you may have extra difficulty passing this lab.

A few lines down in the `test/indexTest.js` file you will see:

```
describe('index.js', function () {
  // there's stuff in here, too
});
```

`describe` is a function provided by our test library, Mocha, and it's used to hold our tests. After the word `describe` is information about our tests. Tests are used as a way to document the behavior of a function to developers. For example, the next word `describe` is followed by the word `companyName`. Here the test is telling us that the tests that come afterwards will be about `companyName`. Then comes the word `it`, where you see the following:

```
it('is set as Scuber', function () {
  // tests are here
});
```

This is telling us that the `companyName` should be set to `Scuber`. Finally, filling in the missing part of the `it` code, we see:

```
it('is set as Scuber', function () {
  expect(companyName).to.equal('Scuber');
});
```

This example shows that the test expects `companyName` to equal `Scuber`. That `expect` and `to.equal` are essentially doing the same thing as `companyName == 'Scuber'`. In other words, `expect(companyName).to.equal('Scuber')` is running code that will have this first test pass if `companyName` equals `Scuber` and fail if it does not.

Don't worry too much yet if it's hard to understand what is happening inside of the `test/indexTest.js` file. But it's a good idea to open up the file, and gather the information that you can. We will also provide instructions in the `README.md` file that will allow you to complete the lab.

Running the Tests

Start by installing the test dependencies by running `npm install` in the terminal, then run `npm test` to run the tests. You should now see the current status of the tests in the terminal. For the moment, all of the tests fail. Let's figure out how to get one of them passing! (The rest will be up to you.)

To get our first test to pass, we can open up our `index.js` file, and write the following:

```
let companyName = 'Scuber';
```

If you run `npm test` again, you'll see that our first test is now passing. However, the second test, which is also about `companyName`, is not yet passing. It's not passing because it expects `companyName` to be declared using a different keyword than the `let` keyword — it needs a keyword that is used for variables that can't be changed...

Continue to work through the problems below. Keep in mind the general workflow for a lab:

1. Run `npm test`.
2. Read the errors; vocalize what they're asking you to do.
3. Write code; repeat steps 1 and 2 often until a test passes.
4. Repeat as needed until all the tests are passing.

Working Through the Problems

If you open up `test/indexTest.js`, you will see the tasks in front of you:

- `companyName` — Inside the `test/indexTest.js` file, look at the `describe` function call for the `companyName` variable. The two `it` function calls inside this `describe` tell us the features of `companyName` we need to create. To review, in the first `it` function call, it says that `it` (companyName) `is set as Scuber`. In the next line, you can see that the test checks to make sure this occurs by seeing if `companyName` equals `Scuber`. So this means that you need to go to your `index.js` file and declare a variable named `companyName` and set it equal to `Scuber`.
- In the second `it` function call for `companyName`, it says it `is defined as a const`. The next line of code tests this. So you need to make sure that you are using the correct type of variable declaration such that attempting to reassign the variable throws an error.
- `mostProfitableNeighborhood` and `companyCeo` — Here we are getting more practice with declaring variables. Read the tests to see how you need to code these two variables to get the remaining tests passing.

Submitting Your Work to Canvas

Once you've got all the tests passing, it's time to push your completed code up to GitHub and submit it to Canvas using CodeGrade. We'll do a quick review of how to do that below, but you may want to review the full process in the **Completing and Submitting Assignments with CodeGrade (https://github.com/learn-co-curriculum/phase-1-completing-assignments-with-codegrade)** lesson. You'll be going through this process for every lab you do in this program!

Let's review the process. First, you need to "stage" your changes using the `git add` command:

```
$ git add index.js
```

or

```
$ git add .
```

Recall that the `.` shortcut will stage **all** files that have changes. In this case there's only one so either command will work.

Next, you need to "commit" your changes, which basically saves a record of the changes you've made. Don't forget to use the `-m` flag and include a commit message! Use the message shown below or choose your own:

```
$ git commit -m "complete lab"
```

Finally, push your changes up to your GitHub account (your fork of this lab):

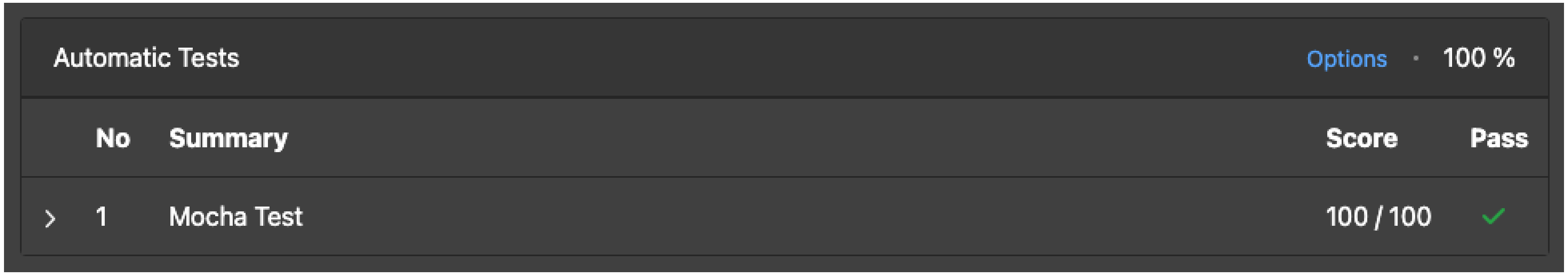
```
$ git push
```

If you go back to your repo in GitHub and refresh the page, you should now see a new commit with your commit message.

The final step is to submit your work to Canvas:

1. Scroll to the bottom of this lesson page in Canvas and click the button labeled "Load Review: Variables Lab in a new window".
2. In the CodeGrade window that opens, click "Create Submission". You should now see a list of your repositories.
3. Find the repo for this lab and click Connect.

4. When you get the message that your repo has been connected, click on the embedded link, then the "AutoTest" tab to watch your progress. Once the tests have finished running, you should see the green checkmark in the "Pass" column, indicating that you've successfully completed the lab.



No	Summary	Score	Pass
1	Mocha Test	100 / 100	✓

Resources

- [MDN: Let (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let)
- [MDN: Const (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const)