Review: Control Flow Lab

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

[(https://github.com/learn-co-curriculum/phase-1-control-flow-lab)](https://github.com/learn-co-curriculum/phase-1-control-flow-lab) [(https://github.com/learn-co-curriculum/phase-1-control-flow-lab/issues/new)](https://github.com/learn-co-curriculum/phase-1-control-flow-lab/issues/new)

Learning Goals

- Practice writing `if...else if...else` statements.
- Practice working with the ternary operator.
- Practice writing `switch` statements.

Introduction

You have been hired as a contractor for Scuber, a burgeoning startup that helps busy parents transport their children to and from all of their activities on scooters.

Scuber's drivers charge their passengers a variable amount based on how far they need to travel. Modify the `index.js` file to make sure that Scuber's drivers are properly telling their passengers how much the ride will cost.

Getting Started

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

Next, run `npm install` to install the dependencies then run the test suite with the `npm test` command.

Read the Tests

We know that you do not have much experience with testing, so that is why it is very important for you to read the instructions in this and every lab. That being said, reading the tests can often provide important clues on how to complete a lab. Let's take a look at the first test for this lab together:

```
describe('index.js', function () {
  describe('scuberGreetingForFeet()', function () {
    it('gives customers a free sample if the ride is less than or equal to 400 feet', function () {
      expect(scuberGreetingForFeet(199)).to.equal('This one is on me!');
    });

    // tests continue...
  });
});
```

Okay, so all of the fancy `describe` words are just there to organize the requirements, and provide a description for what each function should do. By reading the text inside of the `describe` words, we can see that there is some function that should give customers a free sample, where the first 400 feet are free. Then in the next line we see a function called `scuberGreetingForFeet` being executed with `199` passed through as an argument to the function. Executing the `scuberGreetingForFeet` function with the argument should return `"This one is on me!"` .

We will tackle the details of function writing in depth in an upcoming lab. For now, briefly, a function declaration is written like so:

```
function addFive(someNumber) {
  //Everything I want my function to do I put inside these curly braces
  //In this example, let's say I want my function, addFive, to add 5 to
  //any number I pass in (someNumber), but only IF the number is greater
  //than zero:
  let result
  if (someNumber > 0) {
    result = someNumber + 5;
  }
  //at the end, if I want my function to return something, I need to state it:
  return result
}
```

```
//once our function is declared, we can call addFive, passing in values
//as arguments:

addFive(10);
//=> 15

addFive(20);
//=> 25

addFive(-5);
//=> undefined

addFive(addFive(5));
//=> 15!! In this case, the return value of addFive(5), 10, is passed in
//as the argument to the outer addFive, returning 15
```

So, looking back at our test example, `scuberGreetingForFeet(199)` is calling the function `scuberGreetingForFeet`, and passing in the value `199` as the argument. When we write this function, we need to write the logic inside the curly braces to pass our tests and return the result:

```
function scuberGreetingForFeet(someValue) {
  //this is where we can use conditionals given our argument, someValue
  //don't forget to return whatever the result is!
}
```

The big clue from reading the example test above is that the tests in the `indexTest.js` file are calling the functions that we write inside the `index.js` file. These tests pass arguments to our function. When this test passes an argument of `199` to our function, the `scuberGreetingForFeet` function should return `"This one is on me!"`. That makes sense, considering the text in the `describe` and `it` functions say that the first 400 feet should be free. That `199` must be indicating the distance in feet of the requested ride.

So reading tests is essentially like reading the instructions. It's something we may have avoided for much of our lives, but when it comes to programming, tests fill in the picture of the goal we are trying to accomplish. They run mini-experiments on our code and help us better understand our code and the problem we are solving.

Instructions

There are three functions that have been declared for you. You will need to fill in the following code:

- `scuberGreetingForFeet()` — Use `if` and `else if` statements to return the correct greeting based on the distance the passenger desires to travel.
- `ternaryCheckCity()` — Use a ternary operator to return the correct response based on the desired destination of the passenger.
- `switchOnCharmFromTip()` — Use a `switch` statement to return a different response based on the generosity of the passenger's tip.

NOTE: Beware a gotcha! In JavaScript, you cannot express the concept of 'between' in the following way:

```
2 < 5 < 4
// => true
```

It seems like that expression should evaluate to `false` because `5` is not less than `4` . However, we're forgetting about the order of operations — let's think about how the JavaScript engine evaluates that expression. First, the engine compares `2 < 5` , which evaluates to `true` . At that point, it's as though the value `true` has replaced `2 < 5` in the expression, resulting in `true < 4` . The engine sees that we're trying to compare a non-number (`true`) against a number (`4`), and under the hood it converts `true` into a number:

```
Number(true);
// => 1
```

That leaves us with `1 < 4` , which the JavaScript engine correctly evaluates to `true` . Can you figure out how to properly evaluate whether `5` is greater than `2` **AND** `5` is less than `4` using logical operators? Ponder that as you work through the assignment.

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. If you need a reminder, go back to the **Completing and Submitting Assignments with CodeGrade (https://github.com/learn-co-curriculum/phase-1-completing-assignments-with-codegrade)** lesson to review the process.

Good luck!