14.7. Studio: Unit Testing

Let's use Test Driven Development (TDD) to help us design a function that meets the following conditions:

1. When passed a number that is ONLY divisible by 2, return **`'Launch!'`**
2. When passed a number that is ONLY divisible by 3, return **`'Code!'`**
3. When passed a number that is ONLY divisible by 5, return **`'Rocks!'`**
4. When passed a number that is divisible by 2 AND 3, return **`'LaunchCode!'`**
5. When passed a number that is divisible by 3 AND 5, return **`'Code Rocks!'`**
6. When passed a number that is divisible by 2 AND 5, return **`'Launch Rocks!'`**
7. When passed a number that is divisible by 2, 3, AND 5, return **`'LaunchCode Rocks!'`**
8. When passed a number that is NOT divisible by 2, 3, or 5, return **`'Rutabagas! That doesn't work.'`**

Rather than complete the code and *then* test it, TDD flips the process:

1. Write a test first - one that checks the program for a specific outcome.
2. Run the test to make sure it fails.
3. Write a code snippet that passes the test.
4. Repeat steps 1 - 3 for the remaining features of the program.
5. Examine the code and test scripts, and refactor them to increase efficiency. Remember the DRY idea (Don't Repeat Yourself).

14.7.1. Source Code

Open this repl.it (https://repl.it/@launchcode/UnitTestingStudio01) and note the expected files:

1. **`index.js`** holds the Jasmine script. DO NOT CHANGE THIS FILE.
2. **`launchCodeRocks.js`** holds the function we want to design, which we will call **`launchOutput`**.

3. **launchCodeRocks.spec.js** holds the testing script.

Besides **index.js** the files are mostly empty. Only a framework has been provided for you.

14.7.2. Write the First Test

In **launchCodeRocks.spec.js**, complete the **describe** function by adding a test for condition #1:

> *When passed a number that is ONLY divisible by 2, **launchOutput** returns **'Launch!'***

Run the test. It should fail because there is no code inside **launchOutput** yet!

14.7.3. Write Code to Pass the First Test

In **launchCodeRocks.js**, use an **if** statement inside the **launchOutput** function to check if the parameter is evenly divisible by 2, and then return an output. (*Hint: modulus*).

Run the test script again to see if your code passes. If not, modify **launchOutput** until it does.

14.7.4. Write the Next Two Tests

In **launchCodeRocks.spec.js**, add tests for the conditions:

2. When passed a number that is ONLY divisible by 3, **launchOutput** returns **'Code!'**
3. When passed a number that is ONLY divisible by 5, **launchOutput** returns **'Rocks!'**

Run the tests (you have three now). The two new ones should fail, but the first should still pass. Modify the **it** statements as needed if you see a different result.

14.7.5. Write Code to Pass the New Tests

Add more code inside **launchOutput** to check if the parameter is evenly divisible by 2, 3, or 5, and then return an output based on the result.

Run the test script again to see if your code passes all three tests. If not, modify **`launchOutput`** until it does.

14.7.6. Hmmm, Tricky

In **`launchCodeRocks.spec.js`**, add a test for the condition:

4. When passed a number that is divisible by 2 AND 3, **`launchOutput`** returns **`'LaunchCode!'`** (not **`'Launch!Code!'`**).

Run the tests. Only the new one should fail.

Modify **`launchOutput`** until the function passes all four of the tests.

14.7.7. More Tests and Code Snippets

Continue adding ONE test at a time for the remaining conditions. After you add EACH new test, run the script to make sure it FAILS, while the previous tests still pass.

Modify **`launchOutput`** until the function passes the new test and all of the old ones.

Presto! By starting with the *testing* script, you constructed **`launchOutput`** one segment at a time. The result is complete, valid code that has already been checked for accuracy.

14.7.8. New Condition

Now that your function passes all 8 tests, let's change one of the conditions. For the case where a number is divisible by both 2 and 5, instead of returning **`'Launch Rocks!'`**, we want the function to return **`'Launch Rocks! (CRASH!!!!)'`**.

Modify the testing and function code to deal with this new condition.

14.7.9. Bonus Missions

14.7.9.1. DRYing the Code

Examine **`launchOutput`** and the **`describe`** functions. Notice that there is quite a bit of repetition in the code.

Try adding arrays, objects and/or loops to refactor the code into a more efficient structure.

14.7.9.2. What if We Already Have Code?

A teammate tried to help you out by writing the **`launchOutput`** code before class. Unfortunately, the code contains some errors.

Open the flawed code here (https://repl.it/@launchcode/UnitTestingStudio02), and cut and paste your testing script into the **`launchCodeRocks.spec.js`** file.

Run the tests and see how many fail. Use the *descriptive feedback* from your **`it`** statements to find and fix the errors in **`launchOutput`**.

Note: You could just *cheat* and compare your correct code to the flawed sample, but the point of this mission is to give you more practice interpreting and using test results. To gain the most benefit, honor the spirit of this task.

← 14.6. Exercises: Unit Testing (exercises.html) 15. Scope → (../scope/index.html)

Back to top