**14.6. Exercises: Unit Testing**

In many of your previous coding tasks, you had to verify that your code worked before moving to the next step. This often required you to add **console.log** statements to your code to check the value stored in a variable or returned from a function. This approach finds and fixes syntax, reference, or logic errors AFTER you write your code.

In this chapter, you learned how to use unit testing to solve coding errors. Even better, you learned how to PREVENT mistakes by writing test cases before completing the code. The exercises below offer practice with using tests to find bugs, and the studio asks you to implement TDD.

**14.6.1. Automatic Testing to Find Errors**

Let’s begin with the following, simple code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | **function** checkFive(num){  **let** result = '';  **if** (num < **5**){  result = num + " is less than 5.";  } **else** **if** (num === **5**){  result = num + " is equal to 5.";  } **else** {  result = num + "is greater than 5.";  }  **return** result;  } |

[repl.it](https://repl.it/@launchcode/UnitTestingExercises02)

The function checks to see if a number is greater than, less than, or equal to 5. We do not really need a function to do this, but it provides good practice for writing test cases.

Note that the repl.it contains three files:

1. **checkFive.js**, which holds the code for the function,
2. **checkFive.spec.js**, which will hold the testing code,
3. **index.js** which holds special code to make Jasmine work.

**Warning**

Do NOT change the code in **index.js**. Messing with this file will disrupt the automatic testing.

1. We need to add a few lines to **checkFive.js** and **checkFive.spec.js** to get them to talk to each other.
   1. **checkFive.spec.js** needs to access **checkFive.js**, and we also need to import the **assert** testing function. Add two **require** statements to accomplish this (review [Unit Testing in Action](https://education.launchcode.org/intro-to-professional-web-dev/chapters/unit-testing/unit-testing-in-action.html#set-up) if needed).
   2. Make the **checkFive** function available to the spec file, by using **module.exports** (review [Unit Testing in Action](https://education.launchcode.org/intro-to-professional-web-dev/chapters/unit-testing/unit-testing-in-action.html#export-set-up) if needed).
2. Set up your first test for the **checkFive** function. In the **checkFive.spec.js** file, add a **describe** function with one **it** clause:
3. **const** checkFive = require('../checkFive.js');
4. **const** assert = require('assert');
5. describe("checkFive", **function**(){
6. it("Descriptive feedback...", **function**(){
7. *//code here...*
8. });
9. });
10. Now write a test to see if **checkFive** produces the correct output when passed a number *less than 5*.
    1. First, replace **Descriptive feedback...** with a DETAILED message. This is the text that the user will see if the test *fails*. Do NOT skimp on this. Refer back to the [Specifications and Assertions](https://education.launchcode.org/intro-to-professional-web-dev/chapters/unit-testing/hello-jasmine.html#feedback) section to review best practices.
    2. Define the variable **output**, and initialize it by passing a value of **2** to **checkFive**.
    3. **const** checkFive = require('../checkFive.js');
    4. **const** assert = require('assert');
    5. describe("checkFive", **function**(){
    6. it("Descriptive feedback...", **function**(){
    7. **let** output = checkFive(**2**);
    8. });
    9. });
    10. Now use the **assert** function to check the result:
    11. **const** checkFive = require('../checkFive.js');
    12. **const** assert = require('assert');
    13. describe("checkFive", **function**(){
    14. it("Descriptive feedback...", **function**(){
    15. **let** output = checkFive(**2**);
    16. assert.strictEqual(output, "2 is less than 5.");
    17. });
    18. });
    19. Run the test script and examine the results. The test should pass and produce output similar to:
    20. Started
    21. .
    22. 1 spec, 0 failures
    23. Finished in 0.006 seconds
    24. Now change line 3 in **checkFive.js** to **if (num > 5)** and rerun the test. The output should look similar to :
    25. Started
    26. F
    27. Failures:
    28. 1) checkFive should return 'num' is less than 5 when passed a number smaller than 5.
    29. Message:
    30. AssertionError [ERR\_ASSERTION]: Input A expected to strictly equal input B:
    31. + expected - actual
    32. - '2 is greater than 5.'
    33. + '2 is less than 5.'
    34. Change line 3 back.

**Note**

We do NOT need to check every possible value that is less than 5. Testing a single example is sufficient to check that part of the function.

1. Add two more **it** clauses inside **describe**—one to test what happens when **checkFive** is passed a value greater than 5, and the other to test when the value equals 5.

**14.6.2. Try One on Your Own**

Time for Rock, Paper, Scissors! The function below takes the choices (**'rock'**, **'paper'**, or **'scissors'**) of two players as its parameters. It then decides which player won the match and returns a string.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | **function** whoWon(player1,player2){  **if** (player1 === player2){  **return** 'TIE!';  }  **if** (player1 === 'rock' && player2 === 'paper'){  **return** 'Player 2 wins!';  }  **if** (player1 === 'paper' && player2 === 'scissors'){  **return** 'Player 2 wins!';  }  **if** (player1 === 'scissors' && player2 === 'rock '){  **return** 'Player 2 wins!';  }  **return** 'Player 1 wins!';  } |

[repl.it](https://repl.it/@launchcode/UnitTestingExercises03)

1. Set up the **RPS.js** and **RPS.spec.js** files to talk to each other. If you need to review how to do this, re-read the [previous exercise](https://education.launchcode.org/intro-to-professional-web-dev/chapters/unit-testing/exercises.html#export-syntax), or check [Hello Jasmine](https://education.launchcode.org/intro-to-professional-web-dev/chapters/unit-testing/hello-jasmine.html#hello-js).
2. Write a test in **RPS.spec.js** to check if **whoWon** behaves correctly when the players tie (both choose the same option). Click “Run” and examine the output. SPOILER ALERT: The code for checking ties is correct in **whoWon**, so the test should pass. If it does not, modify your **it** statement.
3. Write tests (one at a time) for each of the remaining cases. Run the tests after each addition, and modify the code as needed. There is one mistake in **whoWon**. You might spot it on your own, but try to use automated testing to identify and fix it.

**14.6.3. Bonus Mission**

What if something OTHER than **'rock'**, **'paper'**, or **'scissors'** is passed into the **whoWon** function? Modify the code to deal with the possibility.

Don’t forget to add another **it** clause in **RPS.spec.js** to test for this case.

Index.js

//Do NOT change the code in this file!!!

var Jasmine = require('jasmine');

var jasmine = new Jasmine();

jasmine.loadConfig({

spec\_dir: 'spec',

spec\_files: [

"\*\*/\*[sS]pec.js"

],

});

jasmine.execute();

checkFive.spec.js

const checkFive = require('../checkFive');

const assert = require('assert');

describe("checkFive test", function() {

it("Testing if number is less than 5", function() {

let output = checkFive.checkFive(2);

assert.strictEqual(output, "2 is less than 5.");

});

it("Testing if number is equal to 5", function() {

let output = checkFive.checkFive(5);

assert.strictEqual(output, "5 is equal to 5.");

});

it("Testing if number is greater than 5", function() {

let output = checkFive.checkFive(8);

assert.strictEqual(output, "8 is greater than 5.");

});

});

checkFive.js

function checkFive(num){

let result = '';

if (num < 5){

result = num + " is less than 5.";

} else if (num === 5){

result = num + " is equal to 5.";

} else {

result = num + " is greater than 5.";

}

return result;

}

module.exports = {

checkFive: checkFive

}