**Rooster Regulation**

An experimental rooster object has been developed without any tests! It’s up to you to write them.

By the end of this project you will have a *fast*, *complete*, *reliable*, *isolated*, *maintainable*, and *expressive* test suite: it will provide confidence that the rooster is functioning correctly and it will act as a form of documentation for other developers.

You can find the rooster defined in **index.js**. You will write a test suite in **index\_test.js** within the **test** directory using the [Mocha test framework](https://mochajs.org/) and [assert methods](https://nodejs.org/api/assert.html) from the Node.js standard library. If you’re unsure about any of the methods, you can use the documentation linked.

If you get stuck during this project or would like to see an experienced developer work through it, click “**Get Help**“ to see a **project walkthrough video**.

**Tasks**

**0/12Complete**

Mark the tasks as complete by checking them off

**Construct a test suite**

**1.**

Using require(), import the assert module at the top of **index\_test.js**.

Hint

const module = require('module');

**2.**

Using require(), import the Rooster module from ../index. This allows you to access methods for testing.

Hint

const Rooster = require('../index');

**3.**

Write a describe block for Rooster.

Hint

describe('string', () => {

});

**4.**

Within the first block write another describe block for the .announceDawn method.

Hint

describe('string', () => {

describe('string', () => {

});

});

**5.**

Within the nested describe block, write an it block that tests that the method 'returns a rooster call'.

Hint

describe('string', () => {

describe('string', () => {

it('string', () => {

});

});

});

**6.**

Within the it block, write the relevant test using an assert function.

Don’t forget to use the four phases of a test: *setup*, *exercise*, and *verify* in your it block (*teardown* optional).

* In *setup* define your expected result
* In *exercise* call the function under test
* In *verify* use an assert function

// Define expected output

const expected = 'cock-a-doodle-doo!';

// Call Rooster.announceDawn and store result in variable

// Use an assert method to compare actual and expected result

**7.**

Save your changes and run the test in the command line. You’ve discovered an issue in **index.js**! Make the correction.

Hint

Enter npm test in the terminal and edit index.js based on the test output.

**8.**

Below the describe block for .announceDawn in **index\_test.js** write another block for .timeAtDawn.

**9.**

Test that .timeAtDawn 'returns its argument as a string'. Make sure to use it() and an assert method that uses strict equality.

Hint

You’ll need to call Rooster.timeAtDawn(). Pass it a Number and assert that the output is the Number converted to a String.

Review the documentation for assert.strictEqual() on the [Node.js site](https://nodejs.org/api/assert.html#assert_assert_strictequal_actual_expected_message).

**10.**

You will next test that .timeAtDawn throws an error if its argument is not a valid hour (less than 0 or greater than 23).

Test both bounds to make your test suite *complete*. Test each bound in a separate it block to make it *maintainable*.

Test that .timeAtDawn 'throws an error if passed a number less than 0'. Make sure to use it() and an assert method that checks for thrown errors.

Hint

Review the documentation for assert.throws() on the [Node.js site](https://nodejs.org/api/assert.html#assert_assert_throws_block_error_message). The first argument will be a function which calls .timeAtDawn() and the second argument will be the error you expect to be thrown.

assert.throws(

() => {

Rooster.timeAtDawn(hour);

},

RangeError

);

**11.**

In another it block, test that .timeAtDawn 'throws an error if passed a number greater than 23'.

**12.**

Your test suite is *fast*, *complete*, *isolated*, *maintainable*, and *expressive*! Run it and review the output in the terminal.

Hint

Enter npm test in the terminal.