**Factorial Feature**

In this project, you will use test-driven development to build a factorial method. You will use the assert module and mocha testing library to drive your development using tests that follow the phases: setup, exercise, and verification.

**How To Calculate Factorial**

You take an integer and you multiply that by all the integers that are less than it.

The factorial of an integer n is denoted by an exclamation mark n!, so 5! is equal to:

5 × 4 × 3 × 2 × 1 = 120

**Edge Case**

An *edge case* is a problem or situation that occurs only at an extreme (maximum or minimum) operating parameter — you can think of these as special cases that you need to account for. Based on the logic of factorials, you would expect that calculating 0! would look like this:

0 × 0 = 0

But mathematically this is not true. In this instance of a factorial method, there is an edge-case which is that 0 factorial (or 0!) is 1.

The steps below follow the *red-green-refactor* cycle and will guide you through the process of using test-driven development to create a factorial method. The factorial method will be inside an object named Calculate. You will be implementing your test code in the file **index\_test.js**, and your implementation code in **index.js**.

After you complete each step, run the test suite in the terminal to test the results.

If you get stuck during this project, check out the **project walkthrough video** which can be found at the bottom of the page after the final step of the project.

**Tasks**

**1/11Complete**

Mark the tasks as complete by checking them off

**Factorial Feature With TDD**

**1.**

**Get into the red part 1a:**

In your **index\_test.js** file, add an it statement, and then fill in the error message to state that it will test if the output of 5! is equal to 120.

Hint

Structure for an it statement:

it('...', () => {

});

**2.**

**Get into the red part 1b:**

Follow the steps below to write a test that checks if 120 is equal to Calculate.factorial(5).

* Write an assert.equal() statement inside your it block
* The first argument in your assert.equal() statement should be Calculate.factorial(5)
* The second argument should be 120

Remember to run npm test to see the error message.

**3.**

\*\* Get into the green part 1:\*\*

* In your **index.js** file, add a factorial method inside the Calculate object
* The factorial method should accept one input and return 120

Hint

The expected outcome of 5! is 120

**4.**

**Refactor part 1**

In your **index\_test.js** file, refactor your test code so that it follows the first three phases of a good test.

* Setup
* Exercise
* Verify

Hint

* Make descriptive variables to represent your input and expected output value
  + Ex. inputNumber or expectedResult
* Make a descriptive variable to represent the outcome of .factorial(inputNumber)
* Write an assert.equal() statement that compares your expected and actual outcomes

**5.**

\*\* Get into the red part 2:\*\*

Write another test, with an input other than 5 to put yourself back in the red. This will force you to write implementation code that covers more conditions than just .factorial(5).

* You can write a test that checks if your factorial method returns 6 when you pass 3 to .factorial()

Hint

* Add another it statement to your test
* The test will be mostly the same as your first it statement, except it will have different values for the input and expected output
* Setup the first three phases of a good test

**6.**

**Get into the green part 2:**

In your **index.js** file, replace the line of code return 120 with code that will actually calculate the factorial of the input argument passed to the .factorial() method.

Hint

The code block you could use to calculate the factorial of a number is:

for (let i = inputNumber - 1; i >= 1; i--) {

inputNumber = inputNumber \* i;

}

return inputNumber

**7.**

**Refactor part 2**

* If you haven’t already done so, make sure that all your test code uses descriptive variables instead of actual numeric values, and make sure that it follows the first three phases of a good test.
* Refactor your implementation code so that it is more concise and efficient. This could include cleaning up the existing code, or using a built-in method instead of a for loop for calculating the factorial of any input.

Hint

In the implementation code, you could replace the line:

inputNumber = inputNumber\* i

with the more concise:

inputNumber \*= i

**8.**

**Edge case: test code part 1**

In your **index\_test.js** file, write an empty it block with an error message that covers the 0! edge case.

Hint

a string that reflects best practices:

"returns 1 when you pass in 0."

**9.**

**Edge case — production code**

In your **index.js** file add implementation code that makes your edge case test pass.

Hint

The structure of a conditional statement looks like this:

if (/\* the input number is 0\*/) {

// then return 1

}

**10.**

**Edge case: test code part 2**

Finish writing your edge-case test using

* Setup
* Exercise
* Verify

Hint

* Setup: create descriptive variables for your test
* Exercise: set a variable to hold the outcome of running Calculate.factorial(0)
* Verify: use an assert.equal() statement to check if the output of Calculate.factorial(0) is equal to 1 , using your variables