Assignment #3: Mars Rover

This task is going to put your unit tests, modules, and exceptions knowledge to use by writing tests and classes for the Mars rover named Curiosity.

Selfie of Curiosity on Mars.

Requirements

1. Fork the Mars rover starter repl.it.
2. Write a unit test for each item in the Test List shown below.
    a. Some tests have been created for you as examples.
3. Write classes and methods for each required class and method shown below.
4. Each class should be defined in it's own file and exported and imported using modules.

Test List

Focus on one test at a time. Write the test and *then* the code to make it pass. Only write the minimum amount of code needed to make the test pass. There are some constraints on how you can implement these features. A list of required classes and methods is below.

Each numbered item describes a test. *You should use these exact phrases as the test description*. You will have 11 tests (12, if you do the bonus) at the end of this assignment.

Message Tests

To be written in `spec/message.spec.js`. Remember to use the given phrase as the test description.

1. For the test description use the text, “Throws error if name NOT passed into constructor as first parameter”.
    a. This test is provided in the starter code. The code to make it pass is also included.

    Note

    So far you have only used methods on `assert` to check for equality. Using `assert.throws` to verify if a specific error is thrown is a new concept. To learn how to use this new ability of `assert`, look at the constructor in `message.js` and look at the test named “throws error if name NOT passed into constructor” in `message.spec.js`. You can also look at the official Node.js assert.throws documentation.

    b. Click “Run” to verify that the test passes. Next, comment out line 5 in `message.js`. Click “Run” again to verify that the test fails (the expected error is not thrown when the `Message` class is called).
    c. Restore line 5 to `throw Error("Name required");`.
    d. Change line 12 in `message.spec.js` to `message: 'Oops'`. Click “Run” again to verify that the test fails (the error message did not match `"Name required"`).
    e. Restore line 12 to `message: "Name required"`.
2. For this test, use “constructor sets name” as the description. The test confirms that the `constructor` in the `Message` class correctly sets the `name` property to a new message object.
3. “contains commands passed into constructor as 2nd argument”. This test confirms that the `commands` property of a new message object contains the data passed in from the `Message(name, commands)` call.

Command Tests

Write the following test in `spec/command.spec.js`.

4. “throws error if type is NOT passed into constructor as first parameter”
    a. Look at the constructor in `message.js` and at the test named “throws error if name NOT passed into constructor” in `message.spec.js` for examples of how to complete this task.
    b. When you click “Run”, the test should fail, since you have not created the `Command` class yet.
    c. Add a `command.js` file in your project. Code the `Command` class such that your test passes. Refer to the Command Class description below for more details.

Rover Tests

To be written in `spec/rover.spec.js`.

5. “constructor sets position and default values for mode and generatorWatts”
6. “response returned by receiveMessage contains name of message”
7. “response returned by receiveMessage includes two results, if two commands are sent in message”
8. “responds correctly to status check”

    a. For the `STATUS_CHECK` command, `receiveMessage(message)` returns an object with 4 properties—`completed`, `mode`, `generatorWatts`, and `position`. The test should check each of these for accuracy.
    b. See the Rover Command Types table for more details.
9. "responds with correct status after MODE_CHANGE". The test should check the `completed` property and rover mode for accuracy.
10. "responds with false completed value, if attempt to move while in LOW_POWER mode". The test should check the `completed` property for accuracy and confirm that the rover position did not change.
11. "responds with position for move command".

Required Classes and Methods

The `Message` class is already provided for you in `message.js`. You will need to create a `command.js` file for the `Command` class and a `rover.js` file for the `Rover` class. The `Command` and `Rover` classes will need to be exported from the files they are declared in and imported into the test files.

Note

For help using `require` to import a `class`, notice in `message.js` that the `Message` class is exported using `module.exports = Message;`. In `spec/rover.spec.js` the `Message` class is imported with this statement `const Message = require('../message.js');`.

Message Class

1. This class builds an object with two properties. `constructor(name, commands)`
    a. `name` is a string that is the name of the message.
    b. `commands` is an array of `Command` objects.

Example

```
let commands = [new Command('MODE_CHANGE', 'LOW_POWER'), new Command('STATUS_CHECK')];
let message = new Message('e1', commands);
```

Command Class

1. This class builds an object with two properties. `constructor(commandType, value)`
    a. `commandType` is a string that represents the type of command (see Command Types table for possible values)
    b. `value` is a value related to the type of command.

Example

`'MODE_CHANGE'` and `MOVE` are passed in as the `commandType`

`'LOW_POWER'` and 12000 are passed in as the `value`. For a list of all modes, see Rover Modes table.

```
let modeCommand = new Command('MODE_CHANGE', 'LOW_POWER');
let moveCommand = new Command('MOVE', 12000);
```

Rover Class

This class builds a rover object with one property, but it also contains several functions outside of `constructor`.

1. `constructor(position)`
    a. `position` is a number representing the rover's position.
    b. Sets `this.position` to `position`
    c. Sets `this.mode` to `'NORMAL'`
    d. Sets default value for `generatorWatts` to 110
2. `receiveMessage(message)`
    a. `message` is a `Message` object
    b. Returns an object containing two properties—the original message and an array of *results*. Each element in the array is an object that corresponds to one `Command` in `message.commands`.
    c. Specific details about how to respond to different commands are in the Test List.

Example

```
let commands = [new Command('MODE_CHANGE', 'LOW_POWER'), new Command('STATUS_CHECK')];
let message = new Message('e1', commands);
let rover = new Rover(98382);
let response = rover.receiveMessage(message);
```

Rover Command Types

Command	Value sent with command	Result returned from `receiveMessage`
MOVE	Number representing the position the rover should move to.	`{completed: true, position: 88929237}`
STATUS_CHECK	No values sent with this command.	`{completed: true, mode: 'NORMAL', generatorWatts: 110, position: 87382098}` Values for `mode`, `generatorWatts`, `position` will depend on current state of rover.
MODE_CHANGE	String representing rover mode (see modes)	`{completed: true}`

Note

The response value for `completed` will be `false` if the command could NOT be completed.

Rover Modes

Mode	Restrictions
LOW_POWER	Can’t be moved in this state.
NORMAL	None

Bonus Mission

Add the following test that checks for unknown commands in `spec/rover.spec.js`.

12. Responds with completed false and a message for an unknown command

Submitting Your Work

In Canvas, open the Mars Rover assignment and click the “Submit” button. An input box will appear.

Copy the URL for your repl.it project and paste it into the box, then click “Submit” again.