Reading Tests with Mocha and Chai

This article explains how to read tests that are written with the Mocha and Chai JavaScript frameworks.

Reading Tests with Mocha and Chai

This article is a practical guide to reading tests written with the Mocha and Chai frameworks. In order to write code to pass test cases (often called 'specs'), you need to familiarize yourself with how to read tests!

Example Tests

Let's look through an abbreviated example test from the Rock, Paper, Scissors project to get some practice reading tests. You don't need to read through this project to understand this article, as we will focus on the tests themselves.

```
describe('setPlayerMoves() - Main Functionality', function() { // this is a `describe` block, everything within this callback function is one test suite
  afterEach(clearMoves); // this is a `hook` that gets called between `it` blocks to reset the state

  it('a function called setPlayerMoves should exist', function() { // this is an `it` block, everything inside this function is a single test case
    should.equal(typeof setPlayerMoves, 'function'); // tests often start by checking that the right things exist and are of the right type
  });

  it('should set player one\'s moves with valid inputs', function() {
    setPlayerMoves('Player One', 'rock', 11); // here we call a function from the code we are testing that sets play one's move to rock with a value of 11

    should.equal(playerOneMoveOneType, 'rock'); // this is an assertion that tests that after the `setPlayerMoves()` function above is called, playerOneMoveOneType should equal `rock`
    should.equal(playerOneMoveOneValue, 11); // this assertion tests that setPlayerMoves can set the value of playerOneMoveOneValue
  });
})
```

Test Suites

Tests for one feature are grouped together in `describe` blocks. This group of tests, called a test suite, describes the "Main Functionality" of the `setPlayerMoves` function. Describe takes a string and a callback function: the string describes the feature or behavior being tested, and the callback function contains all of the code for the different tests being run.

You'll see that inside the describe block, the `afterEach` function is called. This is called a hook, or a function that is called at certain points in the lifecycle of the process that it is running in. The afterEach function gets called right after each

it block is run, and customizing this function allows us to reset things that we want to reset between different tests.

Here we call the function `clearMoves` which is a helper function that sets all of the moves back to undefined. It is written outside of any of the blocks, but used as a hook in many of them.

```
function clearMoves() {
  playerOneMoveOneType = undefined;
  playerOneMoveOneValue = undefined;
  playerTwoMoveOneType = undefined;
  playerTwoMoveOneValue = undefined;
}
```

It's important for tests to generally start from a clean slate and for each test to be independent of the others, because we want the errors that we get from our tests to give us a specific diagnosis of what is wrong with our code.

Test cases

Each `it` block describes more particular behavior to test. In the first `it` block, we test that `setPlayerMoves`actually exists and that it is a function so that it can correctly be used in the next block.

In the second `it` block, we call the function `setPlayerMoves`, which is a function from the code that we are testing from our Rock, Paper, Scissors game. After `setPlayerMoves` is called with the arguments, the variable `playerOneMoveOneType` should be equal to the string 'rock' and `playerOneMoveOneValue` should be equal to the number `11`.

Assertions

Any individual assertion where we are comparing the actual and expected values can be called an assertion. The words `should`, `expect`, and `assert` in the tests indicate that an assertion is being made.

Each `it` block test includes multiple assertions, because there are multiple scenarios and edge cases that we want to test for. The error messages that you get from failed tests will most likely point to one of these assertions.

Onwards

Those are the basics of reading tests! Learn more from the mocha and chai documentation.

Even if you encounter tests written with a different framework or for a different language, the same principles of writing good independent unit tests still apply.

Also, checkout the article on reading test ouput so that you can interpret the output of tests and turn it into an action plan for fixing your code!