Open in new tab

Friends Exercise/Lab

Lesson Duration: 60-120 minutes

Learning Objectives

- Be able to write our own tests
- Understand what the setup() method is used for in MiniTest

Using Setup()

Up until now every test we wrote, we instantiated the objects we were testing within our tests.

```
def test_join_string()
  string_1 = "Mary had a little lamb, "
  string_2 = "its fleece was white as snow"
  joined_string = join_string( string_1, string_2 )
  assert_equal( "Mary had a little lamb, its fleece was white as snow", joined_string )
end
```

This was working nicely, but we got introduced to more complex data structures, like arrays and hashes, not to mention next week we will create our own custom objects.

What problems can this cause?

In case we want to calculate something using a more complex data structure (like a hash representing a person) or we want to manipulate said hash, we might end up repeating ourselves every time we write a new test. Even if our person hash is only one key-value pair, I'd rather not repeat myself by creating it for every test - since we know, we should write one test for one function!

Even if we find a solution to the repeating problem, and have a way to use one variable in all my tests, I might run into another problem: what will happen if I manipulate my variable? If I change, let's say, a hash's "name" value in one of my tests, but my next test checks the value of "name", which one should it check for? The one before the change, or the one after?

Luckily, MiniTest gives us a solution for this!

Enter setup()

```
#Copy this into a test file, no need for students to type along:
require("minitest/autorun")
require("minitest/reporters")
Minitest::Reporters.use! Minitest::Reporters::SpecReporter.new

class TestExample < MiniTest::Test

  def setup()
    @person1 = {"name" => "Alex", "age" => 30}
  end

  def test_can_change_name
    @person1["name"] = "Alexander"
    assert_equal("Alexander", @person1["name"])
  end

  def test_person_has_name
    assert_equal("Alex", @person1["name"])
  end

end
```

What the setup() method does is it runs before each test, setting us up one or more variables that can be used in all our tests. To make it visible for all tests, we have to put the @ symbol before the variable (more on this next week)

Since we cannot guarantee that tests will run in order, this will make sure that even if my name changing test runs first, the name checking test will have a brand new, default hash to work with - ensuring I know the expected outcome.

Important: You have to name your setup() properly. It is a function that's given to us by MiniTest, misspelling it will cause MiniTest to miss it.

Lab Time!

Hand out starting_point

We have a bunch of information about a set of friends. We're going to find out some interesting information about them using TDD.

We need to to complete the tasks in the comments of the `friends_spec.rb` file. Write your tests in that file first, then write the function to pass the test in `friends.rb`. There is one example test and function already. If you get stuck, there are hints on how to approach the problem for each question.

A few extra notes…

Firstly, remember we can pass more than one argument into a function and this will need to be done for some of the questions (check for the hints)

Secondly, this is probably the first time where we might not test the return value of a function, but instead check that it's completed the action we want it to (read the hint for question 4)

Lastly, in the spec file remember to name all your tests with test_ at the start of the function declaration (e.g. def test_getting_name)

Published with GitHub Pages