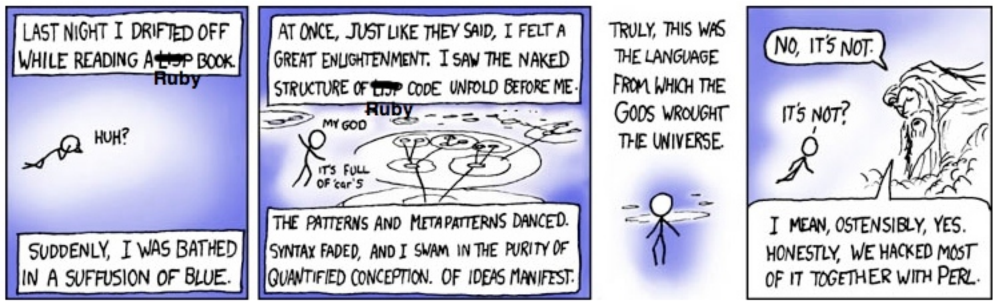
CS 312 - Assignment 3
Unit Testing: Where failure is always an option

Due 11:59pm on Monday, March 5


LAST NIGHT I DRIFTED OFF WHILE READING A ~~LISP~~ Ruby BOOK.
HUH?
SUDDENLY, I WAS BATHED IN A SUFFUSION OF BLUE.
AT ONCE, JUST LIKE THEY SAID, I FELT A GREAT ENLIGHTENMENT. I SAW THE NAKED STRUCTURE OF ~~LISP~~ Ruby CODE UNFOLD BEFORE ME.
MY GOD
IT'S FULL OF 'car'S
THE PATTERNS AND METAPATTERNS DANCED. SYNTAX FADED, AND I SWAM IN THE PURITY OF QUANTIFIED CONCEPTION. OF IDEAS MANIFEST.
TRULY, THIS WAS THE LANGUAGE FROM WHICH THE GODS WROUGHT THE UNIVERSE.
NO, IT'S NOT.
IT'S NOT?
I MEAN, OSTENSIBLY, YES. HONESTLY, WE HACKED MOST OF IT TOGETHER WITH PERL.

http://xkcd.com/224/

For this assignment you will be writing unit tests for the functions that you wrote for the last assignment. It is generally good practice to write the unit test *before* you write the code that it tests, however, to give you some practice writing tests, we're going to do it in the opposite order this time.

For this assignment you should also use `Git` to track your code as you go. Setup an initial repository when you start and commit occassionally as you work. I don't expect to see tons of commits, but I do want to see some. See the **When you're done...** section to see how to submit this work.

After you have written your tests, you will be working with a partner to create a combined set of tests. *Even though this will happen after your write your own tests, find a partner now.* If you can't find a partner, e-mail me before Friday and I will find you a group.

Before starting, read through the entire handout to make sure you understand what is required of you.

1 Unit tests

This first part should be done individually.

Write at least one unit test per function for the functions listed below from Assignment 2. Separate your tests into three separate files corresponding for testing the functions from the three different sections in files named: `test_blocks101.rb`, `test_blocks201.rb` and `test_blocks301.rb` respectively. Inside these files you should use `require` to import the file to be tested and the required files should be named `assign2_1.rb`, `assign2_2.rb` and `linked_list.rb` respectively.

Make sure you follow this specification exactly!

As you're writing unit tests keep the following in mind:

- Make sure to test corner cases, such as being called with an empty array/hash.
- When appropriate, make your tests more robust by using iteration to generate larger data sets.
- You should be writing your tests based on the specification (that is the handout for Assignment 2).
- If the specification assumes some format about the data (i.e an array of numbers) you do not need to (and should not) test input that does not meet this requirement.
- For some of the functions it may make sense to write multiples tests.
- For testing similar function (e.g. `invert_sign` and `invert_sign!` setup your tests to reuse code).
- You can run your tests on your Assignment 2 submission as you go.

Bocks 101 functions to test: all *except* `square_hash_values` and `longest_words`

Blocks 201 functions to test: all *except* `print_sorted`

Blocks 301 functions to test: test both functions

You will be partially graded based on how good your tests are at catching code with issues. In addition, I will likely run your unit tests over all of the Assignment 2 submissions from last time and you may also get graded on how many issues you catch on other people's assignments.

2 Sweet Test Suite

Once you have your three different files, create a test suite called `ts_assignment2.rb` that incorporates all of your three tests above.

3 Adjoining Suites

Once both you and your partner have finished writing your unit tests do the steps in this section

1. Run your test on your partners program and pipe the results to a file with your first name and last name on it followed by "_test_partner_name" (or copy and paste the results into a .txt file). For example:

   ```
   $ ruby -I lib test/ts_assignment2.rb > DavidKauchak_test_Bobby_Z.txt
   ```

2. Sit down together and look at the results of the tests. Then look at the unit test code for each partner. In a file called `partner_evaluations.txt` jointly come up with a score between 1 and 10 for each of your unit tests based on the test results and the thoroughness of the tests. Include one or two sentences justifying your scores. For example, my partner and I might submit the following:

   ```
   David Kauchak: 9
   Bobby Z: 2

   Dave's tests caught mistakes in both his code and Bobby's
   ```

```
code.  In reviewing code, they were robust and handled
almost all of the cases we thought of.

Some of Bobby's tests failed erroneously and in reviewing
the code, many of them didn't test key functionality.
```

3. Finally, combine your two sets of tests into a final set of test files and test suite that you will submit jointly as a partnered group. Note that you should not simply copy and paste the tests into a single file. Instead, you should go through the tests one by one and if two tests cover basically the same issues, pick the best of the two (or merge the two into a single test).

Style/Comments

Comments

For unit tests, you can use comments sparingly. You should include comments at the top of the file with BOTH partners names and a short description of the tests. If there are any complicated parts in the code, include a short comment.

Style

- You code should be succinct and follow the general ruby conventions, including variable, method and class naming schemes.
- Unit test names should be very descriptive of what they are testing. In the case of unit tests, it's better to err on the side of too long/descriptive.

When you're done...

Make sure you've followed the specifications above. *You should only submit one assignment per partnered pair.* To submit, follow the instructions on the course web page.

Your submission should include the following:

- A single copy of your tests (`test_blocks101.rb`, `test_blocks201.rb`, `test_blocks301.rb`) and your test suite file (`ts_assignment2.rb`). These files should represent your groups *combined* tests and should be in a folder called `test`.
- The output from the individual unit test runs. There should be one for each partner (e.g. `DavidKauchak_test_Bobby_Z.txt` and `Bobby_Z_test_DavidKauchak.txt`).
- Your partnered evaluation file (`partner_evaluations.txt`).
- Each partner should also submit a Git log file showing their progress as they constructed their individual tests. For example:

  ```
  $ git log > DavidKauchak.git.txt
  ```

Grading

You will be graded based on:

- How well you followed the specification above.
- The effectiveness of your tests (both on artificially created examples as well as examples from the class).
- Your partnered evaluation.
- Your use of Git.
- The style and quality of your code/tests.