CS 1110, LAB 3: TESTING

http://www.cs.cornell.edu/courses/cs1110/2014sp/labs/lab03.pdf

Updates. Feb 16: see orange update in second paragraph of section 5. Feb 20: see orange update in section 6.

Important References. This lab exercises concepts from lectures 3-5, and especially lecture 4. See the lectures page for the relevant slides and sample code: http://www.cs.cornell.edu/courses/cs1110/2014sp/lectures

For guidance on command-line navigation and using text-based commands, see http://www.cs.cornell.edu/courses/cs1110/2014sp/materials/command.php

Lab Materials. Create a *new* directory on your hard drive and download all of the following files into that directory. (On your browser, you can bring up the online version of this pdf file so you can just click links to download: http://www.cs.cornell.edu/courses/cs1110/2014sp/labs)

- http://www.cs.cornell.edu/courses/cs1110/2014sp/labs/lab03/transcript.py
- http://www.cs.cornell.edu/courses/cs1110/2014sp/labs/lab03/testtranscript.py

Getting Credit for the Lab. When you are done, show this handout, your completed `testtranscript.py`, and — if you removed any bugs — your fixed `transcript.py` file(s) to your instructor, who may ask you some check-up questions and ask you to run your test code, and then will swipe your ID card to record your success.

If you do not finish during the lab session, you have *until the beginning of lab in* **TWO** *weeks to finish it.* (There is no lab 4 because of the February break.) You should always do your best to finish during lab hours. Remember that labs are graded on effort, not correctness.

1. The `transcript` module

The file `transcript.py` defines a module `transcript`. At this stage, you will *not* understand much of the code in transcript.py; but for this lab, you *do* need to know about the following, all defined in transcript.py:

• `lettergrade_to_val(lg)`: A function that returns the numerical value of letter grade `lg`. The usual numerical scheme is assumed: A+ → 4.3, A → 4.0, A- → 3.7, etc.

Precondition: `lg` is one of the following 1- or 2-character strings: 'A+', 'A', 'A-', 'B+', 'B', 'B-'. At the university using this code, everyone is above average![1]

• `Titem`: A Titem represents an item on a transcript, such as "CS1110 A+".[2]

Course authors: D. Gries, L. Lee, S. Marschner, W. White

[1] Actually, we made this choice to reduce the number of test cases you'll need to write. You're welcome.

[2] Again, to reduce the number of test cases you need to write, we're ignoring credit hours.

Each Titem has the following two attributes:

- `course` [string]: course name. Always at least 1 character long.
- `gradeval` [float]: the numerical equivalent of the letter grade. The only valid letter grades are those listed in the description of lettergrade_to_val(lg) above.

Assuming that you have imported `transcript`, you use a constructor expression to make a Titem, supplying a course name and a letter grade as arguments. An example is `transcript.Titem('CS1110', 'A+')`, which creates a Titem with "CS1110" as the value of attribute `course` and 4.3 as the value of attribute `gradeval`, and evaluates to the ID of the Titem (what is written on the folder tab on the left). The code for creating a Titem makes use of the function lettergrade_to_val(lg) to convert its second input argument to a float.

• `raise_grade(ti)`: A function that raises the `gradeval` of Titem `ti` by a non-noticeable amount. It doesn't return anything.

We imagine that this function is written by some mischievous entity with access to the university registrar's records. The idea is that raising a grade of B- to a B, or B to a B+, could escape notice. But, someone might notice a change of B+ to A-, since that's very "visible".

So, `ti`'s `gradeval` is raised by .3 unless the `gradeval` already corresponds to a "plus" grade like B+, in which case the `gradeval` is not changed.

Preconditions: `ti` is a Titem.

2. Your job

Add test cases to unit test file `testtranscript.py` to test the correctness of three functions: lettergrade_to_val(lg), the initializer of Titem, and raise_grade(ti). Use print statements to identify errors in raise_grade. Specific instructions follow.

3. Running the Unit Test `testtranscript`

Open the file `testtranscript.py` in Komodo Edit, change the header comments appropriately, and then take a look. Right now, it is just a "skeleton" file, containing three "stubbed-in" test procedures (one for each function you must test); something is said to be "stubbed-in" when it contains almost nothing other than what is syntactically required. The file also has some lines of code after the test procedures. What specifically do you think will happen when this python file is run? (Hint: the code after the "if __name__ ..." line is executed.)

To check your intuitions, navigate on the command line to the folder containing this file. Then, type "`python testtranscript.py`" in the command shell (*not* the "`>>>`" Python prompt) and hit return to see whether you were right. Ask a staff member if you need help with any of this.

4. Testing lettergrade_to_val

Since the test functions (procedures) were just stubs, the fact that they all succeeded doesn't prove that `transcript`'s functions are correct. Let's use test cases to determine whether or not lettergrade_to_val is correct.

Test cases consist of legal inputs and desired outputs. For your first test case, note that for input "A+", lettergrade_to_val should return the float 4.3. You implement this testcase by replacing the line `pass` in function `test_lettergrade_to_val` by the following line, indented appropriately.

```
cornelltest.assert_floats_equal(4.3,transcript.lettergrade_to_val("A+"))
```

Run the unit test now (type `python testtranscript.py` on the command line), and you should see the same reassuring messages get printed out, indicating that at least the function lettergrade_to_val is correct on one input.[3]

Let's examine the line you added more carefully. Why do you need to have "`cornelltest.`" and "`transcript.`" in it? Write the answer below. If you aren't sure, try deleting them, saving the file, and then running the unit test file again — but make sure to restore the line afterwards.

Now try changing "4.3" to "4.33", saving, and running to see what happens when a test case fails. Then restore the line.

What other test cases do you need to be completely sure that the function works as specified? Implement them all in test_lettergrade_to_val (copy-and-paste is your friend, here), sve, and run the unit test file. Hopefully you'll still see the same reassuring messages printed out, because ... we guarantee that lettergrade_to_val is correct!

5. A test case for Titem's initializer

We now need to see whether Titems are created correctly by the Titem initializer function, which is called when a construction expression for a Titem is evaluated. We do this by creating Titems and checking that their attribute values are created correctly.

We already explained above what `transcript.Titem('CS1110', 'A+')` does. So, replace the line `pass` in test procedure test_Titem_init with `testobj = transcript.Titem('CS1110','A+')`. Then, add lines that use the convenience functions cornelltest.assert_equals and cornelltest.assert_floats_equal to check that `course` and `gradeval` of testobj are correct.

Now run testtranscript.py and see whether your test case passes. (It should.)

To keep the lab short, we do not ask you to implement more than this one test case. Just assume the initializer function is correct.

6. Fully Test and Identify Bugs in raise_grade

Add all necessary test cases to test_raise, save, and run testtranscript.py. Implementing test cases should involve creating Titems, running the raise_grade function on them, and seeing if their new `gradeval` attribute value is correct. If all your testcases are correctly implemented and pass,

[3] If instead you get a message that *cornelltest* isn't found, download the following file into the same directory: http://www.cs.cornell.edu/courses/cs1110/2014sp/labs/lab03/cornelltest.py . That file comes from the installation of CornellExtensions, but we haven't finalized the installation files for that yet.

you are done. But they shouldn't the first time around, because there *is* at least one error in raise_grade.

While there are errors in raise_grade, identify as many errors as you can, using print statements as follows.

Open `transcript.py` with Komodo Edit and note the comments in raise_grade, which — luckily — explain what the code's author intended. We need to check (at least) whether `bval`, `newdec`, and `newval` are storing the right values. (You *don't* have to understand what the code is doing, just what it's *supposed* to do.)

For `newval`, add the following line, properly indented, right after the statement where `newval` is assigned to:

```
print 'newval is: "' + str(newval) + '"'
```

Add similar lines for `bval` and `newdec`, and then run testtranscript.py. You should glean enough information from the output of these three print statements to see which lines of raise_grade are incorrect. Which are they?

It's also helpful to add a line right before any of the other lines of code in raise_grade that tells you what the Titem's gradeval was before raise_grade was called. Try adding this (properly indented), saving, then running:

```
print 'original gradeval is: "' + str(ti.gradeval) + '"'
```

Why is this line's output helpful?

6.1. **Clean up `raise_grade`.** Fix the errors in raise_grade if you can (ask a staff member for help if you need) and run the test unit again; hopefully it will pass!

Unlike unit tests, using print statements to isolate an error is quite invasive to the program being tested. Also, you do not want those print statements showing information on the screen every time you run the program. So once you are sure the program is running correctly, you should remove all of the print statements added for debugging. You can either comment them out (fine in small doses, as long as it does not make your code unreadable), or you can delete them entirely.

However, once you remove these, it is important that you test the procedure one last time. You want to be sure that you did not delete the wrong line of code by accident. Run the unit test one last time, and you are done.