CS 1110 Spring 2018, Assignment 1: Class Status Lookup*

http://www.cs.cornell.edu/courses/cs1110/2018sp/assignments/assignment1/a1.pdf

February 14, 2018

Navigating links in this pdf. Text in any shade of blue in this document is a click-able link.

Watch the course website for any announcements or updates.

Contents

1 Rules

1.1 How To Register a Partner (You Only Get One)

You may work alone or with exactly one other person.

If you are partnering, **the two of you must form a group on the CS1110 course management system CMS BEFORE submitting, which will link your submission "portals"**. More details are in Section 2.

If your partnership dissolves, see the course Academic Integrity description about "group divorce" on what to do.

1.2 What Collaborations Are (Dis-)Allowed And How To Document Them

Our policies are laid out in full on the course Academic Integrity page, but we re-state here **the main principles**: where "you" means you and, if there is one, your one CMS-registered group partner,

1. Never look at, access or possess any portion of another group's code in any form.
2. Never show or share any portion of your code in any form to anyone except a member of the course staff.
3. Never request solutions from outside sources, for example, on online services like StackOverflow.

*Authors: Lillian Lee, with some instructions derived from previous assignments by Walker White and David Gries.

4. DO specifically acknowledge by name all help you received, whether or not it was "legal" according to (1)-(3).

Rule (4) above boils down to "citing your sources": the header comments of your code must accurately describe the entire set of people and sources[1] that contributed to the code that is submitted.

Example:

```
# a1.py
# Anne Bracy (AWB93) and Lillian Lee (LJL2)
# Sources/people consulted: discussed strategy for assign_grades with Walker White
# Feb 14, 2018
```

1.3 Python You Are NOT Allowed To Use In This Assignment

Everything that you need to complete this assignment has been covered by the labs and lectures to date.

You may not use, and do not need, any other Python constructs.[2] We want you to demonstrate your skills with the Python we have taught so far.

2 Deadlines

There are several deadlines on Monday, February 26.

1. If you are partnering: well before submission, follow the instructions in the "How to form a group" instructions on the course Assignments page. Both parties need to act on CMS in order for the grouping to take effect.
2. By 2pm on Monday, February 26, submit whatever you have done at that point to CMS, following steps 1-3 in the "Updating, verifying, and documenting assignment submission" section of the course Assignments page. It is OK if you haven't finished working on the files yet.
3. By 11:59pm on Monday, February 26, make your final files submission on CMS, once again following the aforementioned steps 1-3.

The 2pm checkpoint on Monday, February 26 provides you a chance to alert us during business hours if any problems arise. Since you've been warned to submit early, do not expect that we will accept work that doesn't make it onto CMS on time, for whatever reason, including server delays stemming from many other students trying to submit at the same time as you.[3]

Of course, if some special circumstances arise, contact the instructor(s) immediately.

2.1 Is there going to be a chance to revise in response to grader feedback?

Stay focused on hitting the deadlines listed above. Later, we'll talk again.

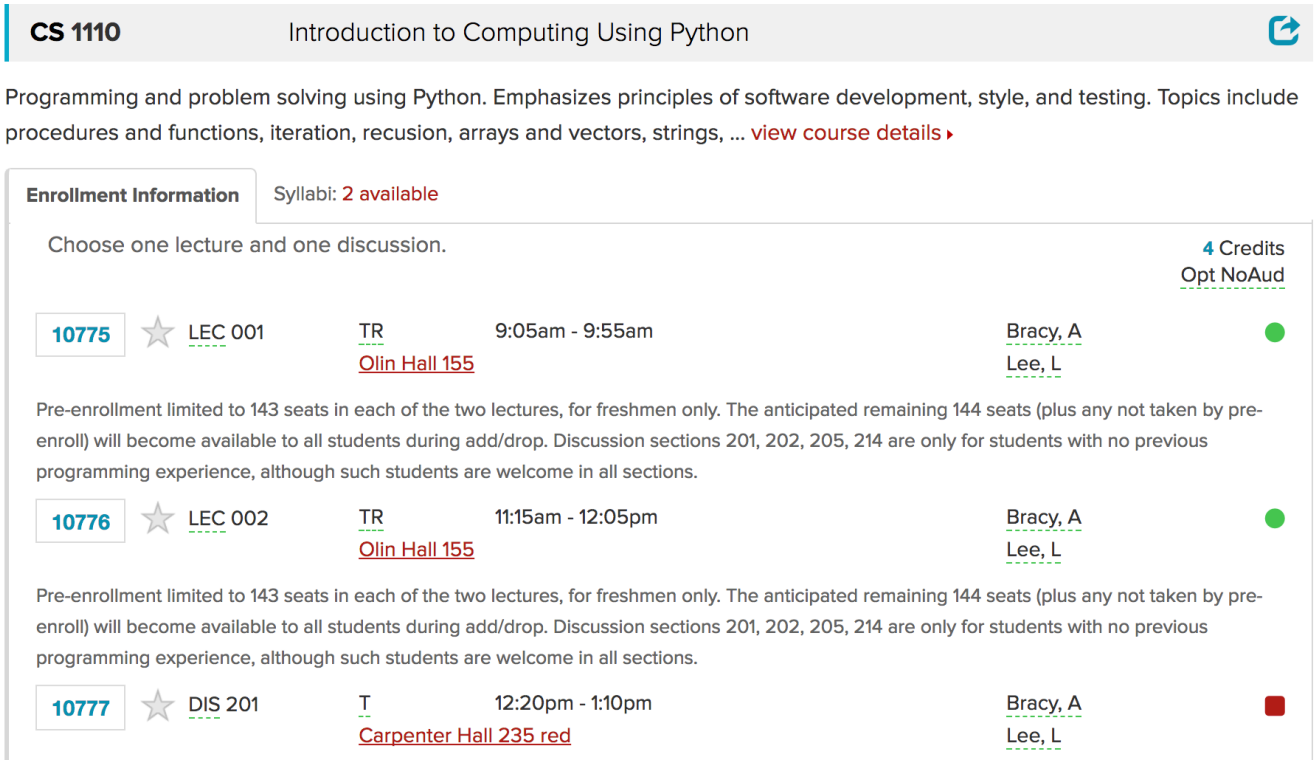
3 Task Overview: Extracting Roster Information

When you look at a Cornell course roster page such as `https://classes.cornell.edu/browse/roster/SP18/subject/CS`:

[1] Other than the course staff or course materials.

[2] So, no `if`s, no loops, etc., even if for some reason you know what those are.

[3] There are no so-called "slipdays" and there is no "you get to submit late at the price of a late penalty" policy.



you can tell that class number 10775 is CS 1110 LEC 001 and, as indicated by green circle, that class component is open; whereas 10777 is CS 1110 DIS 201 and, as indicated by the red square, is closed.

In this assignment, you'll be writing helper code for a program that looks up such information from the online course roster for you. When done, you'll be able to run the program `get_status_from_webpage.py` on live Cornell course roster pages to have interactions like this:

```
[llee@Hy-Brasil] python get_status_from_webpage.py
Enter a roster website URL, including the http part: https://classes.cornell.edu/browse/roster/SP18/subject/CS
The webpage has been loaded.

Note that this program does NOT refresh its data;
if the webpage gets updated, you need to restart this program to get the new information.

Enter the class number, probably a 5-digit number (or "q" to quit): 10775
CS 1110 LEC 001 open
Enter the class number (or "q" to quit): 10776
CS 1110 LEC 002 open
Enter the class number (or "q" to quit): 10777
CS 1110 DIS 201 closed
Enter the class number (or "q" to quit): I am ready to quit now
I couldn't process the input; try a different number (or fix any bugs in a1.py).
Enter the class number (or "q" to quit): q
```

The key to this information access is that many webpages are really just big collections of special strings that your browser displays using formatting information in those strings. You can view the underlying string for a given webpage by using the "view source" functionality of your browser.[4] Here is an excerpt of the source (underlying string) for the webpage above:

[4]Chrome: View » Developer » View Source. Firefox: Tools » Web Developer » Page Source. Safari: Develop » Show Page Source. Or, right-click or ctrl-click in the browser window often brings up a menu with an option to view page source.

We've highlighted with red boxes the places where the information needed by program `get_status_from_webpage.py` is stored. The "common" name for the main course is in the `course-repeater` boxThe first red box starts the info for class 10775. The component LEC is in the `data-ssr-component` box, the component number 001 is in the `data-section` box, and the status (open or closed) is indicated in the `open-status-open` box. (Closed classes say `open-status-closed`.)

3.1 What we've already given you

3.1.1 Files download

Create a new directory on your computer. Download the necessary files from the Assignments page into that new directory.

3.1.2 Files description

We've written the entire program `get_status_from_webpage.py` for you! But it won't work as expected yet, because it makes calls to some unfinished functions in file `a1.py`.

`a1.py` does contain the function definitions and docstring specifications you'll need. **Don't change those.** But the function bodies just consist of the do-nothing command `pass`, so the functions don't work correctly yet.

We've also given you a partially-completed file `a1test.py` for testing the functions in `a1.py`. All that's missing from it are some test cases. **Don't change the ones that are already there**.

3.1.3 Overall intended calling structure

- The program in file `get_status_from_webpage.py` calls functions `open_status` and `label` in file `a1.py`.
- Functions `open_status` and `label` in `a1.py` both should call helpers `after` and `before_first_double_quote` (or helper functions you write that use those helpers). Function `label` should probably do so multiple times.
- The testing functions in `a1test.py` should each call the function from `a1.py` that they are testing, multiple times.

3.2 Your task: the short version

In a nutshell, you must complete files `a1.py` and `a1test.py`, following all directions given in comments with all-caps commands.

You are allowed to write your own helper functions, but if you do, you must (a) provide clear specification docstrings for them, and (b) provide adequate testing for them in `a1test.py`

4 Iterative Development (How to Work Through the Assignment)

As outlined in Section 3.1.3, there are dependencies between the functions you will write. The wisest course of action is to work on the basic functions first, and make sure they are correct, before moving on to the functions that build on that basis.

Hence, develop and test the functions in `a1.py` one at a time, in order. For each function, do the following.

1. **Carefully read the specification for the function and look at the test cases we've given you.**[5] Although some of the specifications are long, don't panic! Working through the examples given in the docstrings and in the test cases will clarify your understanding.
2. **Add representative test cases for that function in the appropriate place in `a1test.py`.** You want cases that represent valid inputs, but exhibit different aspects of the problem the function is trying to solve, so that using your suite of test cases can catch different types of bugs. Look at the reasoning behind the test cases in Lab 03 for some guidance. You can also use the examples given in the function specifications for some of your test cases.
3. **Write the function body.** Hint: If the specification says to return something, you need a `return` statement returning something of the correct type.
   Another hint: double-check that if the instructions said to call a certain helper, that you did indeed use that helper function.
4. **Run python on the unit test file `a1test.py`.** If you see there are errors with the function you're currently working on, fix them and re-test.

Note that once a testing function completes without errors, `a1test.py` will ask you whether you want to move on to testing the next function.

5 Grand Finale

If you're convinced that your code is correct, you should be able run Python on the file `get_status_from_webpage.py` and re-enact the interaction in Section 3.

6 Code Cleanup

Before submitting, ensure your code obeys the following.[6]

- Lines are short enough (˜80 characters) that horizontal scrolling is not necessary.
- You have indented with spaces, not tabs (this is not an issue if using Komodo Edit).
- Functions are separated from each other by two blank lines.
- You have removed any debugging `print` statements.
- You have removed all `pass` statements.
- You have removed "instruction" comments, such as "`# IMPLEMENT THIS FUNCTION`".
- If you added any helper functions, these have good docstring specifications and you have put sufficient testing code for your functions in `a1.py` .

[5] This step makes sure you know what a function is supposed to do. You don't get points for writing correct code for the wrong task!
[6] One reason for these requirements is that they speed up the process of reading hundreds of files.

7 Pre-Submission Checklist and What to Submit

The files you will submit to CMS are `a1.py` and `a1test.py`.[7]

Make sure the following are all true before you submit.

1. You've changed the header comments in all files to list the entire set of people and sources that contributed to the code.
2. You (and your partner) have included your names and NetIDs in the header of all files.
3. The date in the header comments has been changed to when the files were last edited.
4. You have set your CMS notifications settings to receive email regarding grade changes, and regarding group invitations.
5. (reminder) If working with a partner, you have grouped on CMS. (One has invited on CMS, and one has accepted on CMS.)

[7] Do not submit any files with the extension/suffix `.pyc`. It will help to set the preferences in your operating system so that extensions always appear.