Updates to Assignment 1, CS 1110 Spring 2022

The assignment itself, with corrections marked in orange, begins on the next page. On this "page 0", we document the time, location, and nature of the updates, in reverse chronological order.

Updates:

- Friday Feb 18, 12:15pm: Change to the submission procedure originally described in Section 2. Instead of making a new CMS assignment for the second half of A1, we have enabled (re-)submission of all files to a single CMS assignment.
- Tue Feb 15, 5:15pm: Figure 3 has been fixed to have "`\r`" instead of "`/r`".
- Friday Feb 11, 5pm:
  - Pg. 9: missing function header restored.
  - Pg. 10: typo in function name and list of function names corrected.

CS 1110 Spring 2022, Assignment 1: Pedal to the Medal (Get live Olympics medal results)*

http://www.cs.cornell.edu/courses/cs1110/2022sp/assignments/assignment1/a1.pdf

Navigating links in this pdf. Text in any shade of blue in this document is a clickable link.

Updates. Monitor announcements on Canvas[1].[2] (Non-Canvas access at http://www.cs.cornell.edu/courses/cs1110/2022sp/announcements/archive.html.)

Don't fear the length of this writeup! There are lots of pages just because it includes big figures and copies of code we give you.

Figure 1: Screenshot: portion of the webpage for the United States as of midnight Tuesday Feb 9, 2022.

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

I would have dearly loved to be able to entitledthis assignment "Petal to the medal". My best attempt at doing so was to make a tortured connection to the recent Olympics in Japan, where the torch was designed to look like a flower with petals.

[1] https://canvas.cornell.edu/courses/25213/announcements

[2] Throughout, we include both footnotes and clickable links because the clickable links may not work for all readers. The URL in the footnotes can be copy-pasted into a browser as a last resort.

Contents

1 Rules

1.1 Working with a Partner (You Can Have At Most One)

The code you submit can be written by you alone, or you can make a joint submission written by you and exactly one other person.

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

Atom has a "Teletype" feature[3] enabling real-time collaboration. Previous (Spring 2021) staff members Jude Javillo, Jonathan Su, and Emily Parker recommend it for its "Google Docs"-like feel, but observe "one thing to note is that the person who shares the link for teletype will have the updated version but their partner won't be able to save it". Prior (Spring 2021) staff member Ben Rosenberg also recommends repl.it[4] .

If your partnership dissolves, see our Collaboration Policies[5]' item on the "group divorce scenario".

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

The full collaboration policy is on the course Academic Integrity page[6]. **Read it.**

As our advice on working with a partner[7] says, "We strongly advise *against* splitting the work; rather, the two of you should make sure both of you could, by the end, be able to do the assignment individually. Otherwise, you won't get all the practice the assignment is meant to provide."

[3] https://teletype.atom.io/
[4] https://repl.it/
[5] https://bit.ly/3sxzJdV
[6] http://www.cs.cornell.edu/courses/cs1110/2022sp/policies/cs1110integrity.html
[7] http://www.cs.cornell.edu/courses/cs1110/2022sp/resources/doing-assignments.html

1.3 Python You Are NOT Allowed To Use In This Assignment

You may not use, and do not need, any Python constructs not yet covered by the labs, lectures, or posted lecture slides by the time this assignment was released.[8] We want you to demonstrate your skills with the Python we have taught so far.

2 Timeline and Deadlines

Change to item 4: we decided to re-use the same CMS assignment as for the 1st half of A1.

1. If you are partnering: well before Thu, Feb 17, follow our "How to form a group on CMS" instructions[9]. Both parties need to act on CMS[10] in order for the grouping to take effect.

   Once partnered on CMS, only one of you need submit on the partnership's behalf, but you can both submit multiple times. Whichever of you submits the latest before the deadline, that last submission will be what we grade for your group.

2. By 2pm Ithaca time on Thu, Feb 17, submit whatever you have done on `a1_first.py` and `policy_acknowledgment.py` to CMS[11]. Then, do steps 2 and 3 of our "Updating, verifying, and documenting assignment submission" instructions. It is OK if you haven't finished yet; CMS lets you update submissions until the final deadline.

3. By 11:59pm Ithaca time on Thu, Feb 17, make your final submission of `a1_first.py` and `policy_acknowledgment.py`, and save the prove-you-submitted screenshots.

4. Sometime on Fri, Feb 18, we will transfer the CMS groupings you made in Deadline 1 to a "new" CMS assignment for `a1_second.py`, and open it for submission. (This implies that you cannot change partners for `a1_second.py`.)

5. By 2pm Ithaca time on Mon, Feb 21, (re-)submit whatever you have done on `a1_second.py`all three submission files to CMS, and save the prove-you-submitted screenshots. It is OK if you haven't finished yet; CMS lets you update submissions until the final deadline.

6. By 11:59pm Ithaca time on Mon, Feb 21, make your final submission of `a1_second.py`, and save the prove-you-submitted screenshots.

The 2pm checkpoints on Thu, Feb 17 and Mon, Feb 21 provide you a chance to alert us during business hours of any submission problems. **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.[12]

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

2.1 Can we revise in response to grader feedback?

Stay focused on hitting the deadlines listed above. But yes, as long as you submit something for each of the three files by the given deadlines, you will have the chance to revise and resubmit, possibly multiple times! We'll talk more about this later.

3 Task Overview: Extracting Live Olympic Medal Information

The 2022 Olympics in Beijing are updating medal results as we "speak" at https://olympics.com/beijing-2022/olympic-games/en/results/all-sports/medal-standings.htm[13]. But if we want to find out which medals a specific team has won, it takes a few clicks around the site. It would be nice to have a more efficient way to look up such information.

In this assignment, you'll write functions that will plug into a program we wrote. When done, you'll be able to run that program, `query_olympics.py`, which draws on the live medal-results website to have interactions with you

[8] So, no `if`s, no loops, etc., even if for some reason you know what those are.
[9] https://www.cs.cornell.edu/courses/cs1110/2022sp/resources/cms.html#partnering
[10] https://cmsx.cs.cornell.edu/web/auth/
[11] https://cmsx.cs.cornell.edu/web/auth/
[12] There are no so-called "slipdays" and there is no "you get to submit late at the price of a late penalty" policy.
[13] https://olympics.com/beijing-2022/olympic-games/en/results/all-sports/medal-standings.htm

like what's shown in Figure 2. (So when you're finished with A1, compare what `query_olympics.py` outputs for you against what you see in this figure and what the live website displays!)

```
[ljl2@ushuaia assignment1] python solution/query_olympics.py

Enter "S" or "L" (without the quotes. No response = "S").
"S": use sample files on your computer.
  (Avoids hundreds of people bothering the real webserver frequently/simultaneously.
  Also useful in case of Internet access problems.)
"L": use the live Olympics webpages.
Your choice? S

Enter a team code (without quotes)
Or, just hit return for "united-states".
Or, type "q" to quit:
COCHRAN-SIEGLE Ryan, Alpine Skiing, Men's Super-G, Silver
DIGGINS Jessie, Cross-Country Skiing, Women's Sprint Free, Bronze
CHEN Nathan, Figure Skating, Men Single Skating, Gold
United States of America, Figure Skating, Team Event, Silver
STEVENSON Colby, Freestyle Skiing, Men's Freeski Big Air, Silver
KAUF Jaelin, Freestyle Skiing, Women's Moguls, Silver
JACOBELLIS Lindsey, Snowboard, Women's Snowboard Cross, Gold
KIM Chloe, Snowboard, Women's Snowboard Halfpipe, Gold
MARINO Julia, Snowboard, Women's Snowboard Slopestyle, Silver
......

Enter another country code,like "united-states", or <return> for "united-states", or "q" to quit: new-zealand
SADOWSKI SYNNOTT Zoi, Snowboard, Women's Snowboard Slopestyle, Gold
......

Enter another country code,like "united-states", or <return> for "united-states", or "q" to quit: norway
KILDE Aleksander Aamodt, Alpine Skiing, Men's Alpine Combined, Silver
KILDE Aleksander Aamodt, Alpine Skiing, Men's Super-G, Bronze
Norway, Biathlon, Mixed Relay 4x6km (W+M), Gold
BOE Johannes Thingnes, Biathlon, Men's 20km Individual, Bronze
ROEISELAND Marte Olsbu, Biathlon, Women's 15km Individual, Bronze
KLAEBO Johannes Hoesflot, Cross-Country Skiing, Men's Sprint Free, Gold
JOHAUG Therese, Cross-Country Skiing, Women's 7.5km + 7.5km Skiathlon, Gold
Norway, Curling, Mixed Doubles, Silver
RUUD Birk, Freestyle Skiing, Men's Freeski Big Air, Gold
GRAABAK Joergen, Nordic Combined, Individual Gundersen Normal Hill/10km, Silver
ENGEBRAATEN Hallgeir, Speed Skating, Men's 5000m, Bronze
......

Enter another country code,like "united-states", or <return> for "united-states", or "q" to quit: latvia
No medals found for latvia

......

[ljl2@ushuaia assignment1]
```

Figure 2: An interaction with our program when all the functions in `a1_second.py` are correctly implemented. Here, we use the un-changing sample files as data — you can see this from the line "Your choice? S" — so you can directly compare your results against this image without worrying about medal counts changing.

How can we, having just had six lectures of introductory Python, manage this task? Well, many webpages are really just big collections of special *strings* 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.[14]

Figure 3 shows an excerpt of the (very slightly edited) source (that is, the underlying string) for the olympics.com medal-standings webpage for the United States.

[14] 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.

3.1 Key intuitions

3.1.1 How to get at a team's data: use string operations to create a URL string

There exists a python function `requests.get()` that can be used to fetch the contents of a webpage, if we just feed it the right URL string as input. And, exploratory clicking around on the Olympics website reveals relevant webpages with URLs like:

https://olympics.com/beijing-2022/olympic-games/en/results/all-sports/noc-medalist-by-sport-**trinidad-and-tobago**.htm
https://olympics.com/beijing-2022/olympic-games/en/results/all-sports/noc-medalist-by-sport-**chile**.htm
https://olympics.com/beijing-2022/olympic-games/en/results/all-sports/noc-medalist-by-sport-**islamic-rep-of-iran**.htm

Referring to the bold-faced substrings as *team codes*, we observe that *given* a team code, we can reproduce the above pattern with an expression like

`<the appropriate "http://...." prefix> + <team code> + ".htm"`

So let's write a function `data_url(prefix, c)` that helps do this!

3.1.2 How to extract data for a given medal: the logic of leveraging patterns in the webpage source

How can we write a function `one_medal_info(s)`, which, for one medal win, will tell us the following four data items?

1. The winner name (which can be an individual or a country)
2. The sport
3. The particular event for that sport
4. The type of medal (1 is gold, 2 is silver, 3 is bronze)

Figure 3: Annotated portion of the source (html) string for the United States webpage. Green highlights: the substrings we want to extract, corresponding to a winner name, a sport, an event for that sport, and a number indicating the kind of medal. The original figure had a typo: it had "`/r`" instead of "`\r`" for the event part. Incidentally, see Ed Discussion post https://edstem.org/us/courses/19140/discussion/1130087 for more on how to deal with the evil `\r`.

Figure 3 highlights patterns in the underlying strings of the relevant webpages that we can take advantage of.

- Notice that for *each of the four data items*, we can do pretty much exactly the same thing: extract the data item by "*scooping out*" the text between a particular pair of "*starter*" and "*ender*" markers.

  So we'll write a function `scoop(text, starter, ender)` which we'll call four times in `one_medal_info(s)`, once for each item.

  - "Scooping" means getting the text *after the first* occurrence of `starter` that is *before the first* following occurrence of `ender`. That suggests writing helper functions `after_first` and `before_first`: with these two helper functions in hand, the body of `scoop()` can be short and sweet, just consisting of calls to those helpers.

- Note that, as stated in Figure 3 box 1, there is an extra step needed for the winner name: pull out from the scooped-out text the part after the last occurrence of '>'. Hence, it would be nice to have a function `after_last` that `one_medal_info(s)` can use to clean up the winner name.

3.2 The files you need

Create a new directory on your computer. Download and unzip into that directory this zip file. The contents are:

`policy_acknowledgment.py`
`a1_first.py`
`a1_second.py`
`query_olympics.py`
`cornellasserts.py`
`sample_data`, a folder of some downloaded medals-per-country webpages, as html strings.

We've written the entire program `query_olympics.py` for you! But it won't work "out of the box", because it calls functions in file `a1_second.py` that are currently mostly just "skeletons": only the function-definition headers and docstring specifications you'll need — **don't change those** — plus function bodies containing only the do-nothing command `pass`.

We did `after_last()`, for you; you should find it useful, as per Section 3.1.2.

We've also given you a partially-completed testing file `a1_first.py` **It has some purposely erroneous test cases in it**; more on that later.

3.3 Desired/required function-call structure for the entire assignment

- The program in file `query_olympics.py` repeatedly calls `data_url()` and `one_medal_info()` in module `a1_second`.
- Function `one_medal_info()` in `a1_second.py` should/must call "helper" function `scoop()`, explicitly or implicitly, potentially multiple times.
- Function `scoop()` in `a1_second.py` should/must call "helper" function `after_first()` and `before_first`.[15]
- The testing functions in `a1_first.py` each call the corresponding functions from `a1_second.py` multiple times.

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 `a1_first.py` .

3.4 Your task: the one-line description (full descriptions in Section 4 and Section 5)

Fix and **complete** files `a1_first.py` and `a1_second.py`, following all directions given as comments starting "`STUDENTS`" in the `.py` files and all directions in this document.

[15] Such calls can be implicit if you create and use another helper function that calls `scoop()` .

4 Collaboration/Academic Integrity Policy Acknowledgment

Read our Collaboration and Academic Integrity Policies[16]. We want you to understand points (1)-(4).

Open `policy_acknowledgment.py`. Insert your NetID(s) and the date into the header comments.

Paste into the file, between the first set of three double-quotes and the series of dots, the lines from the collaboration policy starting with "Until all students' . . ." and ending with "acknowledge the course staff".

Change all the pronouns appropriately so that the subjects of all the relevant sentences are in the first person, not the second person (implicit or explicit). In other words, "you" statements and imperatives should be changed to "I" or "we" statements.

Below the series of dots but above the second set of three double-quotes, write down any questions you have about policies on the aforementioned webpage; we would be more than happy to clarify anything you are wondering about! (OK to not have any questions).

Save the file and submit it.

5 Iterative Development (How to Work Through the Assignment)

We said in Section 3.3 that there are dependencies between the functions you will write. The wisest course of action: write *and test* the basic functions first before moving on to the functions that build on that basis.

Hence, develop and test the functions in `a1_second.py` one at a time, in the given order (since that's the order we put the testing functions in.) For each function, do the following.

1. **Carefully read the specification for the function.** In the specification docstrings, backquotes are used to visually distinguish variable names, like this: `` `c` `` or `` `pref` `` (because we can't use font changes or other visual aids in comment strings). We don't always use angle brackets the way we do in lecture because html strings often themselves contain angle brackets.

   In the below, a "team code" is how each team is referenced in the Olympics site's URLs. The US has team code "united-states", Nigeria has team code "nigeria", and so on; Section 9 has the full list.

```
def data_url(prefix, c):
    """Returns: new string of the form prefix-c.htm
    Precondition: `c` and `prefix` are non-empty strings

    Example: If we had
        prefix: "https://olympics.com/noc-medalist-by-sport"
        c: "united-states"
    Then this function would return the string
        "https://olympics.com/noc-medalist-by-sport-united-states.htm"
    """
```

```
def after_first(text, marker):
    """Returns: portion of `text` starting just after the 1st occurrence of
    `marker`.

    Preconditions:
     `text` [str]: contains at least one instance of `marker`
     `marker` [str]: length > 0

    Examples:
        after_first("ab+c", "+") ---> 'c'
            To be clear, that's a length-one string.

        after_first('faith <cough> hope <cough>Charity', '<cough>') --->
            ' hope <cough>Charity'
            To be clear, the returned string starts with a space.
    """
```

[16] http://www.cs.cornell.edu/courses/cs1110/2022sp/policies/cs1110integrity.html

```
def before_first(text, marker):
    """Returns: portion of `text` ending just before the first occurrence of
    `marker`.

    Preconditions:
     `text` [str]: contains at least one instance of `marker`
     `marker` [str]: length > 0

    Examples:
        before_first("ab+c", "+") ---> 'ab'

        before_first('faith <cough> hope <cough>Charity', '<cough>') --->
            'faith '
    """
```

```
def scoop(text, starter, ender):
    """Returns substring of `text` that:
     * starts just after the end of the 1st occurrence of `starter` in `text`
     * ends just before the beginning of the 1st following occurrence of `ender`.

    Preconditions:
     `text` [str]: length > 0.
     `starter` and `ender` [str]: both non-empty and occur in `text`.
     At least one `ender` appears after a `starter` in `text`.

    Examples:
     scoop('+a+b+c!+4def+5','+', '!') ---> 'a+b+c'
     scoop('good job :) good example foo(0) ', '(', ')' ) --> '0'
     t = '<li style="color:purple">python</li><span> the < is intentional</span>'
     scoop(t, '<span>','</span>') ---> " the < is intentional"
    """
```

```
def one_medal_info(s):
    """Returns string of the form
        <winner name>!<sport>!<event>!<medal type>
    where the relevant data is pulled from `s`.

    These four data items should have exactly the capitalization, punctuation,
    and spacing as in `s`.

    See a1_first.test_one_medal_info() for examples.

    Preconditions:

    `s` is a string of the following form, where WN, SP, EV, and MT
    indicates non empty word(s) with no double-quotes, angle brackets ("<" or ">"),
    or '!', and "..." stands for anything:

    1. `s` starts with
            <div class="name">...>WN</a></div>
        with no "<div>" or "</div>" in the middle.
        There are no other occurrences of <div class="name"> in `s`.

        [Technicality: in real data, for individual (e.g., non-team)
        winners), there a </span> right before the </a>.  Our
        "wrapper" code will delete this for students.]

    2. After that, `s` has a portion
            daily-schedule - SP">
        There are no other occurrences of 'daily-schedule' in `s`.
    3. After that, `s` has a portion
            <td class="StyleCenter">
            EV</td>
        There are no other occurrences of 'StyleCenter' in `s`.
    4. After that, `s` has a portion
            medals/big/MT.png
        There are no other occurrences of 'medals/big/' in `s`.
    """
```

2. **Fix the bad test cases in `a1_first.py`.** We've given you a number of test cases, so you have examples to look at. But we also planted bad ones to ensure you carefully read the given function specifications.

   We guarantee that:

   - you do not need to do any fixing or adding to `test_data_url()`, `test_before_first()`, or `test_one_medal_info()`. (We want to save you some time.)
   - Function `test_after_first()` contains *at least* two bad test cases.[17]

   Here's how to fix:

   - For a test case where the "expected answer" is wrong, add the comment `# ... STUDENT-FIXED ERROR ...` under the comment that numbers the test case; comment out the incorrect `assert_equals` call; and add the fixed call below, like this:

[17] To be clear: maybe there are just two, maybe there are three, maybe there are more...

```
# 0. first input has multiple spaces in it
# ... STUDENT-FIXED ERROR ...           <--- added
result = a1_second.some_wonderful_function('s t a', 'st')
# cornellasserts.assert_equals('', result)    <--- commented out
cornellasserts.assert_equals('a', result)     <--- fix added
```

- For a test case where the situation should not have been tested, add the comment `# ... STUDENT-DELETED CASE ...` under the comment that numbers the test case; add a comment giving your reasoning, and comment out the entire test case, like this:

```
# 1110. first input is an int
# ... STUDENT-DELETED CASE ...           <--- added
# ... REASON: violates precondition      <--- added
# result = a1_second.brilliant_function(2, 3)     <--- commented out
# cornellasserts.assert_equals('3', result)    <--- commented out
```

3. **Add missing representative test cases for that function in the appropriate place in** ~~`test_tag_endi()`~~ `test_after_first()` **and** `test_scoop()`.[18] 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.

   For each test case you add, include in a comment a short justification of what the test case represents.

4. **Write the function body in** `a1_second.py`.[19]

5. **Run python on the script** `a1_first.py`.[20] If errors are revealed with the function you're currently working on, fix them and re-test.

6 Grand Finale

If you're convinced that your code is correct, you should be able to run Python on the file `query_olympics.py`[21] and reproduce the interaction in Figure 2!

And then, quit the program, restart it, this time enable "Live" mode, and for a given team code, like "united-states" (see Section 9 for full list), compare your program's results with what you see if you visit the URL that you create with your own textttdata_url() function!

Finally, use your program to watch those medals pile up in real time!

7 Code Cleanup

Before submitting, ensure your code obeys the following.[22]

1. Lines are short enough (~80 characters) that horizontal scrolling is not necessary.
2. You have indented with spaces, not tabs.

[18] We have constructed enough test cases for you for ~~`test_data_url()` and `test_one_medal_info()`~~ all the other testing functions, so you don't need to add any more. You're welcome.

[19] Sanity checks:

- If the specification says to return something, you need a `return` statement returning something of the correct type.
- Double-check that if the instructions said to call a certain helper, that you did indeed use that helper function.
- The functions in `query_olympics.py` do some string processing, so you may find it useful to look in that file for inspiration/examples. But it is definitely not necessary to do so, and if you do choose to check that file out, don't be intimidated by the Python in there that you don't know (yet).

[20] At the command prompt, *not* the >>> Python interactive prompt, enter `python a1_first.py` .

[21] At the command-shell prompt, enter `python query_olympics.py` .

[22] These requirements speed up the process of reading/grading hundreds of files.

3. Functions are separated from each other by at least two blank lines.
4. You have commented out any debugging `print` statements.
5. You have removed all `pass` statements.
6. If you added any helper functions, these have good docstring specifications and you have put sufficient testing code for your functions in `a1_second.py` .

8 Pre-Submission Checklist and What to Submit

The files to submit to CMS[23] are `policy_acknowledgment.py`, `a1_first.py`, and `a1_second.py`.[24]

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 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 the other has accepted on CMS.)

9 Team codes

Here are the "codes" that the Olympics website is using in its URLS, and which we borrow for our program:

albania, american-samoa, andorra, argentina, armenia, australia, austria, azerbaijan, belarus, belgium, bolivia, bosnia-herzegovina, brazil, bulgaria, canada, chile, chinese-taipei, colombia, croatia, cyprus, czech-republic, timor-leste, denmark, ecuador, eritrea, estonia, finland, france, georgia, germany, ghana, great-britain, greece, haiti, hong-kong-china, hungary, iceland, india, ireland, islamic-rep-of-iran, israel, italy, jamaica, japan, kazakhstan, kosovo, kyrgyzstan, latvia, lebanon, liechtenstein, lithuania, luxembourg, madagascar, malaysia, malta, mexico, monaco, mongolia, montenegro, morocco, netherlands, new-zealand, nigeria, north-macedonia, norway, pakistan, china, peru, philippines, poland, portugal, puerto-rico, republic-of-korea, rep-of-moldova, roc, romania, san-marino, saudi-arabia, serbia, slovakia, slovenia, spain, sweden, switzerland, thailand, trinidad-and-tobago, turkey, ukraine, united-states, uzbekistan, virgin-islands-us

[23] https://cmsx.cs.cornell.edu/web/auth/

[24] 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.