Updates to Assignment 1

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

Feb 17, 9am: Page 9 Section 9: missing preconditions for `exchange_amount` added.

Feb 15, 2:45pm: Page 9, Section 10: URL for instructions on how to break up long lines has been fixed.

CS 1110 Spring 2017, Assignment 1: Currency Conversion[*]

http://www.cs.cornell.edu/courses/cs1110/2017sp/assignments/hw1.pdf

February 17, 2017

Figure 1: BTC stands for the cryptocurrency Bitcoin, but we sorta wish it stood for BaTCoin.

Thinking about that trip overseas? If you can swing it, it is best to go when the exchange rate is in your favor, i.e., when your dollars buy more in the foreign currency. So, it would be nice to have a function that, given your current amount of cash in US dollars, tells you how much your money is worth in another currency.

However, there is no set mathematical formula to compute this conversion. The value of one currency with respect to another is constantly changing. In fact, in the time that it takes you to read this paragraph, the exchange rate between the dollar and the Euro has probably changed several times. How can we possibly write a program to handle something like that?

One solution is to make use of a *web service*. A web service is a program that, when you send it web requests, automatically generates a web page with the information that you asked for. In our case, the web service will tell us the exchange rate for most major international currencies. Your job will be to use string-manipulation methods to read the generated web page and extract the exact information we need.

Primary learning objectives. You will exercise the following: use of string operations and methods on a real-world problem; use of iterative development and testing for a larger-scale project than we have tackled before.

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

Contents

[*]Authors: Walker White, Lillian Lee, Steve Marschner, Molly Feldman, Qin Jia, Stephen McDowell, Alex Parkhurst, Dana Warmsley, Dongwook Yoon, Erik Andersen

1 Rules

1.1 How To Partner (You Only Get One)

You may do this assignment with *at most one* other person.

If you choose to work with a partner, *before* you submit any files, the two of you must link your A1 files/fates on CMS by forming a CMS group. This requires that *one person must issue a CMS invite and the other must accept via CMS*. Brief instructions for how to form a CMS group are on the course Assignment page.

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 rules**: 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.
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". See the aforementioned website for more detail, but basically, 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. **An incomplete listing of contributors in your code headers constitutes fraud.**

1.3 Python You Are NOT Allowed To Use In This Assignment

Everything that you need to complete this assignment should have been covered by the lecture on Objects.

In fact, even if you happen to know what they are, *you may not use if-statements, conditional statements, loops, or recursion anywhere in this assignment.*[2]

[1]Other than the course staff or course materials.
[2] In fact, they would constitute inelegant overkill for the assigned tasks.

2 Getting Credit: You Have Multiple Tries to Get All The Points

2.1 Start Early! You'll All Be Working With the Same Machine!

This assignment involves everybody's programs contacting a lone computer, `cs1110.cs.cornell.edu`. All 550 or so CS1110 students doing this at the last minute will slow things down even more than the typical response time of several seconds for connection and data retrieval.

2.2 Initial File Submission Before The Initial Deadline

Your initial solutions must be submitted to CMS by **Thursday, February 23rd at 11:59pm**.

But, we **strongly** recommend that *you first submit whatever preliminary progress you have to CMS by 2pm on Thursday, February 23rd.*[3] You can replace older submissions with improved ones up to the deadline.

2.3 Special For A1 Only: Revision In Response To Grader Feedback

We want everyone to master this assignment, and so allow (repeated) revisions in response to grader feedback — our hope is that everyone eventually gets a 10/10. This iterative process, which can go around multiple times but terminates by Thursday, March 2nd, consists of repetition of the following steps.

Grading step. (Us.) We evaluate your code according to the following criteria, in order:

- Adequate test cases
- Correctness of the code (does it pass our test cases?)
- Good program format (style), according to the course style guidelines page

If there's a problem, we put grading feedback on CMS and increment your grade *by just one point.*[4] We stop checking once we find the first few errors (we don't flag them all).

If instead you've mastered the assignment, we give you 10/10 and the process halts.

Revision and regrade request step. (You.) When you get email from CMS notifying you that your grade has changed, get the grader's feedback: click on the Assignment 1 link in CMS. On the resultant page, if necessary, click on the red word "show" in the line "Grading Comments & Requests (show)." Contact your grader (netid displayed by CMS) if you have questions about their feedback.

As soon as possible — in 24 hours would be great[5] — do **R*5**: Read the feedback, Revise your program accordingly, Resubmit. Finally, Request a Regrade using the CMS; instructions are on the assignments webpage.[6]

When we see your revision, we go back to the grading step.

3 The Currency Exchange Web Service

For this assignment, you will use a simulated currency exchange service that never changes values. This is important for testing in Part C: Currency Query; if the answer is always changing, it is hard to test that you are getting the right answers.

To use the service, you use your web browser with special URLs that start with the following prefix:

```
http://cs1110.cs.cornell.edu/2017sp/a1server.php?
```

This prefix should be followed by a *currency query*. A currency query has three pieces of information in the following format (*without* spaces; we have included spaces here solely for readability):

[3] This will give you practice with CMS and provide 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. No so-called "slipdays" and no "you get to submit late at the price of a late penalty" for this assignment, due to the tight grading timelines described in the next section.
Of course, if there are extenuating circumstances, contact the instructor(s) immediately to let us know the situation.

[4] Hence, do not be alarmed if you see a "1" for the assignment at first! We're just tracking the number of revisions.

[5] You want to leave time for multiple cycles.

[6] If you do not request a regrade, we have no simple way of knowing that you have resubmitted, and your work will languish.

```
source=currencyYouHave & target=currencyYouWantToConvertTo & amt=amount
```

where *currencyYouHave* is a three-letter code for the original currency, *currencyYouWantToConvertTo* is a three-letter code for the new currency, and *amount* is a float value for the amount of money in the original. For example, if you want to know the value of 2.5 dollars (USD) in Euros (EUR), the query is

```
source=USD&target=EUR&amt=2.5
```

The full URL for this query is thus

`http://cs1110.cs.cornell.edu/2017sp/a1server.php?source=USD&target=EUR&amt=2.5`

Click on the link to see it in action. You will note that the "web page" brought up in your browser is just a single line in the following format:

```
{"success":true, "error":"", "source":"2.5 United States Dollars", "target":"2.3476275 Euros"}
```

This is what is known as a JSON representation of the answer. JSON is a way of encoding complex data so that it can be sent over the Internet. You will use string operations and methods to pull the relevant data out of the JSON string.

Try a few currency queries to familiarize yourself with the service.

3.1 Invalid Queries

If you enter a query that is syntactically well-formatted but invalid (for example, one with a non-existent currency code like "AAA"), you get this error response:

```
{"success":false, "error":"Source currency code is invalid.", "source":"", "target":""}
```

If you enter a query with two valid currency codes but an invalid quantity value, you get this error response:

```
{"success":false, "error":"Currency amount is invalid.", "source":"", "target":""}
```

For all errorful queries, the "source" and "target" values are blank, while "success" is false. The "error" value is a specific error message describing the problem.

4 Files Needed

Make a new folder. Have the command shell and Komodo Edit both open in this new folder before you start.

4.1 Your Lab 3 Files

You will be using two files that you worked with as part of Lab 3 for this assignment: `lab03.py` and `cornelltest.py`. As a reminder, `cornelltest` contains function-level testing tools, which you will use to perform your own unit testing throughout this assignment. A function in `lab03.py` (once you complete and test it) will be used in this assignment.

Place a copy of these files into your new folder.

4.2 Files In a1.zip

Download and unzip `http://www.cs.cornell.edu/courses/cs1110/2017sp/assignments/assignment1/a1.zip`. Move the files in it into your new folder. The two files are:

`a1.py`: a skeleton of the main module for this assignment. Note that it imports `lab03`.

`a1test.py`: a skeleton unit test for module `a1`. We've included the following test procedure *stubs*: `testA`, `testB`, `testC`, `testD`, plus calls to these procedures at the end of the file.

5 Your task

Your primary goal in this assignment is to write a function that will use our webserver to return to its caller what amount of a new currency they will receive in exchange for a given amount of a given currency. This function will involve several steps. You will get the JSON string from the web service, break up the string to pull out the substring containing the numeric value, and then convert that substring to a float.

This process might feel like you are working in reverse: you will write the functions to break up the string first, and the functions to interact with the web service last. This is because we want you to develop the following programming habit: *always complete and test helper functions before finishing the functions that use them.*

5.1 Iterative Development (How to Work Through the Assignment)

This assignment will follow an *iterative development cycle*. That means you will write test cases for a few functions, then write their bodies, then fully test them before you write any more. This process makes it easier to find bugs; you know that any bugs must have been part of the work you did since the last test.

The rest of the assignment is broken into four parts (listed as Parts A, B, C, and D). *For each part*, do the following:

1. **Write a representative set of test cases in `a1test.py`, by examing the function specification(s) for that part in `a1`.** Yes, this means you are writing tests before writing the function bodies. We talked about this in lecture.
   Unless otherwise instructed, each test case should be a call to an `assert` function in `cornelltest`. Furthermore, your tests should be *representative*.

2. **Write the function bodies for that part.** Hint: If the specification says to return something, you need a `return` statement. Make sure that the value returned is of the correct type.

3. **Run the unit test `a1test`.** If errors are found, fix them and re-test. Repeat until no more errors are found.

6 Part A: Breaking Up Strings

One subtask is to separate currency amounts from currency names. For example, given the string

```
"0.8963 Euros"
```

we want to be able to break it up into “0.8963” and “Euros”.

This is the motivation for the two functions below. The implementation of these functions should be relatively simple; one or two lines suffices.

`pre_space(s)`	
Returns	Substring of `s`; up to, but not including, the first space
Parameter `s`	the string to break
Precondition	`s` has at least one space in it
`post_space(string)`	
Returns	Substring of `string` after the first space
Parameter `string`	the string to break
Precondition	`string` has at least one space in it

Implement these functions according to their specification, as described in “Iterative Development (How to Work Through the Assignment)”. In other words,

1. Based on careful reading of the function specification, place a set of representative test cases in the procedure `testA()` of `a1test.py`. To test the functions, make use of `assert_equals` in the module `cornelltest` to compare the result of each functions with the string that you expect to get back. When you think about what test cases you want to include[7], consider the following:

   - Does the specification allow for strings with more than one space?

[7] We have four per function.

   - Does it allow for strings that start with a space?
   - Does it allow for strings that don't have any spaces?

2. Implement the two functions.

3. Test for and correct errors until no errors remain.

7 Part B: Processing A JSON String

All of the responses to a currency query, whether valid or invalid, contain the keywords "source" and "target". If it is a valid currency query, then the answer is in quotes after the keyword "target". If it is invalid, then the quotes after "target" are empty. Hence the next step is to extract the information in quotes after these keywords.

While working on each of the functions below, remember to write the test cases in `at1test.py` before implementing the body. All test cases in this section go in the procedure `testB()`, which you should remember to specify. You should thoroughly test each function before implementing the next one.

7.1 Warning: Test For Stray Spaces

We never said that JSON strings might not have extra spaces around the colons that separate values and their value names.

So, test that your functions work on JSONs with spaces before and after the colons and without spaces before and after the colons.

7.2 `first_in_double_quotes(s)`

You have the specification of this function **in file** `lab03.py`, and need to have (or write) a completed and tested version of this function.

Do *not* place this function in `a1.py`; it already has an `import` of `lab03.py`. Also, you do not need to submit test cases for this function.

7.3 `get_source(json)`

Returns: The source value in the response to a currency query.

Given a JSON response to a currency query, this returns the string inside double quotes (`"`) immediately following the keyword `"source"`. For example, if the JSON is

```
'{"success":  true,"error":"","source":"2 United States Dollars","target":"1.878102 Euros"}'
```

then this function returns `'2 United States Dollars'` (not `'"2 United States Dollars"'`). It returns the empty string if the JSON is the result of on invalid query.

Parameter json: a JSON string to parse
Precondition: json is the response to a currency query

7.4 `get_target(json)`

Returns: The target value in the response to a currency query.

Given a JSON response to a currency query, this returns the string inside double quotes (`"`) immediately following the keyword `"target"`. For example, if the JSON is

```
'{"success":true,"error":"","source" :"2 United States Dollars","target":  "1.878102 Euros"}'
```

then this function returns `'1.878102 Euros'` (not `'"1.878102 Euros"'`). It returns the empty string if the JSON is the result of an invalid query.

Parameter json: a JSON string to parse
Precondition: json is the response to a currency query

7.5 `has_error(json)`

Returns: True if the query has an error; False otherwise.

Given a JSON response to a currency query, this returns the opposite of the value following the keyword `"success"`. For example, if the JSON is

`'{"success":false,"error":"Source currency code is invalid.", "source":"","target":""}'`

then the query is not valid, so this function returns `True` (it does NOT return the message `'Source currency code is invalid'`, nor the string `"True"`).

Parameter json: a json string to parse

Precondition: json is the response to a currency query

As always, write your unit tests before implementing the two functions. *Look carefully at the specifications.* You only need to test valid JSON queries. To get some JSON responses for testing, send queries to the webserver via a query URL (as shown in the web service instructions) and copy the result into a test case.

Your approach should be simply to find the position of the appropriate keyword and extract the value in quotes immediately after it.[8]

Your implementation **must** make use of the the helper function `lab03.first_in_double_quotes()`, and the string method `find()` or `index()`.

8 Part C: Currency Query

Now it is time to interact with the web service. In this part, you will implement a single function. The test cases for this function should go in procedure `testC()` in `a1test.py`.

```
def currency_response(source_currency, target_currency, source_amount):
    """Returns: A JSON string that is a response to a currency query.

    A currency query converts source_amount money in currency
    source_currency to the currency target_currency. The response
    should be a string of the form

        '{"success":true,"error":"","source" :"<old-amount>","target":"<new amount>"}'

    where the values old-amount and new-amount contain the value
    and name for the original and new currencies. If the query is
    invalid, both old-amount and new-amount will be empty.

    Preconditions:
        source_currency is a string
        target_currency is a string
        source_amount is a positive float"""
```

While this function sounds complicated, it is not as bad as you think it is. You need to use the `urlopen` function from the module `urllib2`. This function takes a string that represents a URL and returns an object that represents the web page for that url. This object has the following methods:

Method	Specification
geturl()	**Returns**: The URL address of this web page as a string.
read()	**Returns**: The contents of this web page as a string.

Using one or both of these methods (you might not need them both), plus string techniques to construct a query URL string to give to `urlopen`, is enough to implement the function above.[9]

[8] Recall from Python You Are NOT Allowed To Use In This Assignment that conditional and if-statments are banned.

[9] Hints: (1) If `x` stores an object that has method `sing()`, then you can call the method by the expression `x.sing()`. (2) If you need

8.1 Testing: Some Example Values

There are a vast number of currencies supported by our currency exchange.[10] Here is a sample:

Code	Name	1 USD =
BTC	Bitcoin	0.001028329131
EUR	European Euro	0.939051
JPY	Japanese Yen	113.2341
NAD	Namibian Dollars	13.41125
NOK	Norwegian Kroner	8.366577
PEN	Peruvian Nuevo Soles	3.252093
SEK	Swedish Kronor	8.902149

Note, however, that you should **not use this table in any of the functions that you write in `a1.py`**. The table above is for constructing test cases, not for use in your actual functions.[11]

8.2 Better: Testing Against the Webserver

The best way to test this is to use a web browser to manually get the right (i.e., expected) JSON answer. For example, one test case can be constructed by seeing the result of going to the URL

`http://cs1110.cs.cornell.edu/2017sp/a1server.php?source=USD&target=EUR&amt=2.5`

Copy the value from this web page into a test case in `testC()`. Then check that the function returns the same JSON string. Remember to be thorough with your choice of test cases; one is not enough.

Important: Fetching a web page takes time, especially if too many people are trying to do so simultaneously. You should give each call to this function at least 5-10 seconds to complete before restarting any tests.

9 Part D: Currency Exchange

We are now ready for the final part of the assignment. Implement the following, again using our test-case-before-function-body approach. The test cases should go in procedure `testD()` in `a1test`.

```python
def iscurrency(currency):
    """Returns: True if currency is a valid (3 letter code for a) currency.
    It returns False otherwise.

    Parameter currency: the currency code to verify
    Precondition: currency is a string."""
```

In implementing `iscurrency()`, you **must use the functions `currency_response` and `has_error` as helper functions**.

access to something in a module, you first have to make the module accessible. What Python command do you use for that?

[10] The list of currencies can be found here. But remember, to get the exchange rates themselves for this assignment, you should be querying our CS1110 server.

[11] There is no reason for you to waste your time hard-coding in all of the currencies listed in this table into your program, since the web service you will contact already knows them all anyway.

```
def exchange_amount(currency_from, currency_to, amount_from):
    """Returns: amount of currency received in the given exchange, as a float.

    In this exchange, the user is changing amount_from money in
    currency currency_from to the currency currency_to. The value
    returned represents the amount in currency currency_to.

    Preconditions:
        currency_from is a string FOR A VALID CURRENCY CODE
        currency_to is a string FOR A VALID CURRENCY CODE
        amount_from is a positive float"""
```

UPPERCASE text added above to the preconditions.

9.1 Testing

In the case of `iscurrency()`, you will find "Testing: Some Example Values" useful in determining correct answers for your test cases. While it is not okay to use the table in the body of `iscurrency()` itself, it is okay to use the table to to decide on some test cases.

You may also use the table to craft some test cases for the function `exchange_amount`. However, you might find it easier to use a currency query URL to look up the correct answer, and then paste the answer into your test case.

A bigger issue with testing `exchange_amount` is that real numbers cannot be represented exactly on your computer. This creates problems when you try to test equality between floats. To solve this problem, `cornelltest` provides a function called `assert_floats_equal()`. You should use this function to test `exchange_amount()` instead of `assert_equals()`.

10 Code Format Requirements

Once you have everything working you should go back and make sure that your code meets the class coding/style conventions, including the following:

- Lines are short enough (~80 characters) that horizontal scrolling is not necessary. See `http://www.cs.cornell.edu/courses/cs1110/2017sp/materials/style.php#line-length` [URL fixed to include "materials"]. on how to break long lines up.
- You have indented with spaces, not tabs (this is not an issue if using Komodo).
- Functions are separated from each other by two blank lines.

11 Pre-Submission Checklist

The files you will submit to CMS are `lab03.py`, `a1.py`, and `a1test.py`.[12]

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.)

[12]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.