CS1110 Summer 2021 — Assignment 2[*]

Due Saturday 7/3 at 11:59pm

Read this before you begin

Learning Goals The learning goals and outcomes for this assignment are listed below. Following each goal is a reference to where relevant material can be found in Canvas.

1. Use string operations to solve problems specified in English (Lecture 6)
2. Employ the *iterative development cycle* of software development and testing (Lecture 7)
3. Invoke object methods using the correct syntax (Lecture 8)
4. Basic flow control with if-then-else statements (Lecture 9)

In addition to the primary learning objectives, you may need to reference the Python documentation. Navigating through and reading the Python documentation is an important skill. We understand that the documentation can be hard to read when you are first learning Python. Don't hesitate to ask questions about the documentation, but please try to *first* look through the documentation on a certain function, module, or type if you have questions.

Submission The files you will submit to Canvas are `lab06.py`, `a2.py`, `a2app.py`, and `a2test.py`.

Submission within 24 hours after the deadline will be accepted with a 10% late penalty. Even if only one file (of several) is submitted late, the assignment as a whole will be marked late. Assignment submission on Canvas closes 24 hours after the deadline and no further submission will be allowed.

Do not put your name in the files that you will submit; only include your NetID. The reason is to help promote fairness in grading. Thank you for heeding our request, even if it sounds a little strange.

Restrictions Since CS1110 is an introductory course that assumes no previous programming experience, and for the purpose of learning specific concepts, we sometimes restrict the use of certain tools or concepts that we consider too advanced given the contents and current status of the course.

Even if you already know about *loops* or *recursion*, do not use them in this assignment. Furthermore, you should only have to use conditionals in Part E; any of the other parts can be done (and are simpler) without flow control.

Experiment It is important to develop a spirit of exploration. If you wonder how Python might deal with a particular kind of input, simply *try it out in the interactive mode*! When you come across something that you don't know how to do, *treat it as a learning opportunity.* Play with code ideas in the interactive mode, or refer to the course examples, or look up the documentation (start from the Python Documentation). Be ready to explore—that's learning! However, if you have run out of ideas, come to office hours—that's why we have office hours!

Group work and academic integrity You must work either on your own or with one partner. If you work with a partner, you must first register as a group in Canvas and then submit your work as a group. For a group, "you" below refers to "your group." You may discuss background issues and general strategies with others and seek help from the course staff, but the work that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions with others. It is not OK for you to see or hear another student's code and it is *certainly* not OK to copy code from another person or from published/Internet sources. If you feel that you cannot complete the assignment on you own, please seek help from the course staff.

[*]Based on an earlier assignment by W. White, L. Lee, S. Marschner, M. Feldman, Q. Jia, S. McDowell, A. Parkhurst, D.Warmsley, D. Yoon, E. Andersen, D. Fan, and A. Kellison.

Code	**Name**	**1 USD =**
BTC	Bitcoin	$9.1667216 \cdot 10^{-5}$
EUR	European Euro	0.84261
JPY	Japanese Yen	105.083
NAD	Namibian Dollars	16.5
NOK	Norwegian Kroner	8.982484
PEN	Peruvian Nuevo Soles	3.547788
SEK	Swedish Kronor	8.772173

Table 1: Examples of Supported Currencies

Currency conversion

Dreaming about an overseas trip once the pandemic is over? In addition to waiting until travel restrictions are lifted, you will also want to wait until the exchange rate is in your favor, i.e., when your domestic currency will buy more in the foreign currency. In this assignment, you are going to write a function that, given your current amount of cash, 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 US 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.

The Currency Exchange Web Service For this assignment, you will use a simulated currency exchange service that never changes values (exchange rates). This is important for testing; 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/2019fa/a1?
```

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):

```
src=currency1 & dst=currency2 & amt=value
```

where `currency1` is a three-letter code for the original currency, `currency2` is a three-letter code for the new currency, and `amount` is a float value for the amount of money in the original currency. For example, if you want to know the value of 2.5 dollars (USD) in Cuban Pesos (CUP), the query is

```
src=USD&dst=CUP&amt=2.5
```

The full URL for this query is thus

```
http://cs1110.cs.cornell.edu/2019fa/a1?src=USD&dst=CUP&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:

```
{ "ok":true, "lhs":"2.5 United States Dollars", "rhs":"64.375 Cuban Pesos", "err":"" }
```

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 extract the relevant data out of the JSON string. Try a few currency queries to familiarize yourself with the service.

Many currencies are supported by our currency exchange. The list of currencies can be found here. Table 1 provides a small sample of some of the currencies available and the corresponding exchange rate.

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:

```
{ "ok":false, "lhs":"", "rhs":"", "err":"Source currency code is invalid." }
```

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

```
{ "ok":false, "lhs":"", "rhs":"", "err":"Currency amount is invalid." }
```

For all error responses, the `lhs` and `rhs` values are blank, while `ok` is `false`. The value `err` is a specific error message describing the problem. This will be important for error handling and debugging in this assignment.

Getting Started First, let's have a look at the files that come with this assignment.

- `a2.py`: a skeleton of the main module for this assignment. Note that it imports `lab06`. You will use your file `lab06.py` from labs 6 and 7 in this assignment; make sure to move a copy of it into your new directory.
- `a2test.py`: a skeleton unit test for module `a2`. We've included the following test procedure *stubs*: `testA`, `testB`, `testC`, `testD`, plus calls to these procedures at the end of the file.
- `testcase.py`: contains function-level testing tools, which you will use to perform your own unit testing throughout this assignment. You worked with `testcase.py` in labs 7 and 8

Your task Your primary goal in this assignment is to write an interactive application that queries the user for input and responds as follows:

```
ak2485$ python a2app.py
Enter source currency: USD
Enter target currency: EUR
Enter original amount: 2.5
You can exchange 2.5 USD for 2.2160175 EUR.
```

You will write the script `a2app.py` last, in Part E. We do not want you to put any code in this file until `a2.py` is complete and fully tested, which is what you will do in Parts A to D. So, 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 habit: *always complete and test the helper functions before finishing the functions that use them.*

The central function of `a2.py` is `exchange`. This function will involve several steps. In particular, you will need to (1) get the JSON string from the web service, (2) break up the string to pull out the numeric value (as a substring), and then (3) convert that substring to a float. We are going to take you through this process step-by-step. Note that not every function that we ask you to implement will be used directly by the function `exchange`.

Iterative Development 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 writing the next function. This 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. *For each part*, do the following:

1. **Write a representative set of test cases in `a2test.py` by examining the function specification(s) for that part in `a2.py`.** Yes, this means you are *writing tests before writing the function bodies.* Unless otherwise instructed, each test case should be a call to an `assert` function in `testcase.py`. Furthermore, your tests should be *representative.* See Lecture 7 and Lab 7 for examples of using `testcase.py`.
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 `a2test`.** If errors are found, fix them and re-test. Repeat until all errors are eliminated. Remember that `print` statements are your friend when debugging.

Overall: adhere to the style guide [10 points]

Before you start coding, make sure you have reviewed the Style Guide for assignments in Canvas.

Part A: Breaking Up Strings [18 points]

One subtask is to break up JSON strings. Conceptually, you want to separate the amount from the currency name. For example, if we are given the string `"4.502 Euros"`, then we want to break it up into `"4.502"` and `"Euros"`.

This is the motivation for the two functions `before_space` and `after_space` in `a2.py`. The implementation of these functions should be relatively simple; one or two lines suffices.

Implement these functions according to their specification using the iterative development cycle. In other words,

1. Based on careful reading of the function specification in `a2.py`, place a set of representative test cases in the procedure `testA` of `a2test.py`. To test the functions, make use of function `assert_equals` from the module `testcase` to compare the result of each function with the string that you expect to get back. When you think about what test cases you want to include[1], consider the following:
   - Does the specification allow for strings with more than one space?
   - Does the specification allow for strings that start with a space?
   - Does the specification allow for strings that don't have any spaces?
2. Implement the functions `before_space` and `after_space` in `a2.py`.
3. Test for and correct errors until the code is error free.

Part B: Processing a JSON String [18 points]

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

There are three functions in Part B of `a2.py`: `get_rhs`, `get_lhs`, and `has_error`. While working on these functions, remember to write the test cases in `a2test.py` *before* implementing the body. All test cases in this section go in the procedure `testB`. You should thoroughly test each function before implementing the next one.

Note that three of the functions have **json is the response to a currency query** in the precondition. This just means that your test cases should only consider strings that can be returned by the server. So the string

```
{ "ok":true, "lhs":"2 Namibian Dollars", "rhs":"2 Lesotho Maloti", "err":"" }
```

is okay to test, but the string

```
' "lhs":"2 Namibian Dollars", "rhs":"2 Lesotho Maloti" '
```

is not.

> **Important note:** Your implementation for Part B must make use of
>
> 1. the `find` or `index` string methods, and
> 2. the helper function `first_inside_quotes` from `lab06.py`; see Part B.1.

B.1 `first_inside_quotes(s)`

You have the specification of this function in file `lab06.py`, and already should have a completed and tested version of this function (if not, complete that part of Lab 6 now).

Do *not* place this function in `a2.py` since `a2.py` already imports `lab06.py`. Also, you do not need to submit test cases for this function.

[1]Our solution has four representative cases for each of the functions `before_space` and `after_space`.

Part C: Currency Query [18 points]

Now it is time to interact with the web service. In this part, you will implement the function `currency_response`. The test cases for this function should go in procedure `testC` in `a2test.py`.

While this function looks complicated, it is not as bad as you think. To implement this function, you should first use string techniques to construct the query URL based on the parameters. Then you need to use the `urlopen` function from the module `urllib.request`. The function `urlopen` 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 *bytes* object.

One or both of these methods is enough to complete the implementation of the function above.[2]

The best way to test this function 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/2019fa/a1?src=USD&dst=EUR&amt=2.5
```

Copy the value from this web page into a test case in `testC` **(do not forget to add quotes)**. Then check that the function returns the same JSON string. Remember to be thorough with your 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.

Part D: Currency Exchange [18 points]

Next, you will implement the functions `is_currency` and `exchange` according to the specifications in `a2.py` and using our test-case-before-function-body approach. The test cases should go in procedure `testD` in `a2test.py`. When making your test cases, reread the `testcase` documentation.

In implementing `is_currency`, you **must** use `currency_response` and `has_error` as helper functions. You may find Table 1 useful in in determining correct answers for your test cases for the function `is_currency`.

You may also use Table 1 to craft some test cases for the function `exchange`. 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` is that real numbers cannot be represented exactly (some of you had questions about this in Assignment 1). This creates problems when you try to test equality between floats. To solve this problem, `testcase` provides a function called `assert_floats_equal`. You should use this function to test `exchange` instead of `assert_equals`. There is an example of this issue in the exercises of Lab 7.

Finally, bear in mind that, like `currency_response`, these functions connect to the web service, and so are not instantaneous. In our solution, with complete test procedures for everything, it can take up to 2 seconds to run the unit test on campus. This will be a bit slower if you are working closer to the deadline.

Part E: Putting It All Together [18 points]

Now it is time to reap the fruits of your labor, and create the currency conversion application. As sketched earlier, this application should (1) ask for the source currency, (2) ask for the target currency, and (3) ask for the original amount, before (4) printing out the exchange amount, or whether an error resulted. To ask the user for input, you will need to use the `input` function, which we saw in Lab 3; refer to your work there to find an example.

You should write this script in the file `a2app.py`, which you need to create and store in the same directory as all of the other files for this assignment. When you create `a2app.py` file, use the following docstring:

[2] 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 access to something in a module, you first have to make the module accessible. What Python command do you use to do that? (3) You can convert from `bytes` to `str` using the function `str(byte-object,'utf-8')`

```
"""
User interface for module currency

When run as a script, this module prompts the user for two currencies and
an amount. It prints out the result of converting the first currency to
the second.

Author: YOUR NETID(S) HERE
Date:   THE DATE COMPLETED HERE
"""
```

The script `a2app.py` should be fairly easy for you to write, based on what you have done before.