T-Th 9:05
or
T-Th 11:15
in Olin 155

CS 1110: Introduction to Computing Using Python
Fall 2017

Main

About:
Announcements
Staff
Times & Places
Calendar

Materials:
Texts
Python
Command Shell
Terminology
VideoNote

Coursework:
Lectures
Assignments
Labs

Assessment:
Grading
Exams

Resources:
CMS
Piazza
AEWs
FAQ
Python API
Python Tutor

Style Guide

Academic Integrity

Assignment 1:
Currency

Due to CMS by Sunday, September 17th at 11:59 pm.

Thinking about that trip overseas? If you can swing it, it is best to go when the exchange rate is in your favor. When your dollars buy more in the foreign currency, you can do more on your vacation. This is why 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 on Earth do we 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 current exchange rate for most of the major international currencies. Your job will be to use string-manipulation methods to read the web page and extract the exact information we need. Full instructions are included below.

Learning Objectives

This assignment is designed to give you practice with the following skills:

- How to write a self-contained module in Python
- How to use string and object methods in Python
- How to connect Python to a web service
- How to read specifications and understand preconditions
- How to use docstrings appropriately for specifications
- How to follow the coding conventions for this course
- How to thoroughly test a program

The functions we ask you to write in this assignment are relatively short and straightforward. The emphasis is testing and "good practices", not complicated computations. You will find the most recent lab very helpful in understanding this assignment.

Table of Contents

Authors: W. White, D. Yoon, Q. Jia, L. Lee, and S. Marschner.

Image Credit: Petr Kratochvil

Academic Integrity and Collaboration

This assignment is a slightly modified version of an assignment given in previous semesters. Please do this assignment without consulting (or seeking) previous solutions. Since you are allowed to revise and resubmit, with help from us, until you have mastered this assignment, there is no reason for you to look at previous solutions.

While academic integrity policies apply if you do not cite your source, we will take off points if you borrow code form another solution (such as from another semester). If this happens, you will not be able to make a perfect, even with revisions.

We also ask that you do not enable violations of academic policy. Do not post your code to Pastebin, GitHub, or any other publicly accessible site.

Collaboration Policy

You may do this assignment with one other person. If you are going to work together, form your group on CMS as soon as possible. **This must be completed before you submit the assignment. Both people must do something to form the group**. The first person proposes, and then the other accepts. You have to do this early because CMS does not allow you to form groups once grades are released. Once you have grouped on CMS, only one person submits the files.

If you do this assignment with another person, you must work together. It is against the rules for one person to do some programming on this assignment without the other person sitting nearby and helping. Furthermore, this assignment is not designed for you to split the work cleanly between partners. Sit together and take turns "driving"; alternate using the keyboard and mouse.

With the exception of your CMS-registered partner, we ask that you do not look at anyone else's code or show your code to anyone else (except a CS1110 staff member) in any form whatsoever. While it is only an academic integrity violation if you do not cite the collaboration, extra collaboration could prevent you from earning a perfect score.

Before You Get Started

Read Carefully

These instructions may seem long, but that is because we have tried to give you all the information you need in one document. Your chances of completing the assignment quickly will be increased by reading carefully and following all instructions. Many requests for resubmission are caused not by issues with programming but simply by not following instructions.

Pay particular attention to the section on Iterative Development, as it contains important instructions for the remaining sections, and we will not repeat these instructions for each section.

Start Early!

600 students trying to contact a web service at once will slow everybody down. Connecting to, and reading from, a web page is not instantaneous. It will take several seconds for some of the functions you will write to complete their calculations. Furthermore, if you wait until the last minute to test this assignment, you will be connecting to the same web page as everyone else in the class, so things could slow down even more.

Grading Policy (Revise-and-Resubmit Cycle)

To ensure that everyone masters this assignment, we will use an iterative feedback process. If one of the objectives below is not properly met, we will give you feedback and expect you to revise and resubmit. This process will continue until you are done. This process should be finished by Sunday, September 24th; Once you finish you will receive a perfect score of 10. In our experience, almost everyone is able to achieve a perfect score within two submissions.

In grading your code, we will focus on the following issues *in order*:

- Correct function specifications and/or formatting
- Adequate test cases
- Correctness of the code (does it pass our test cases?)

Formatting is graded according to the course style guidelines, available on the course web page.

If your code fails one of the three tests above, we will notify you and ask you to resubmit. We stop checking once we find the first few errors, so you should not assume that the errors we point out are the only errors present.

Until we have decided that you have mastered (e.g. 10/10) the assignment, your "grade" on CMS will be the number of revisions so far. This allows us to keep track of your progress. Do not be alarmed if you see a "1" for the assignment at first! The assignment will be considered completed when it passes all three steps outlined above.

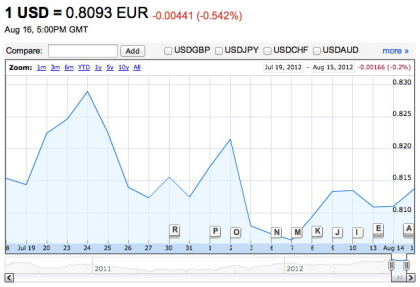
Assignment Scope

Everything that you need to complete this assignment should have been covered by Lecture 6 (Specifications and Testing) in class. In particular, *you may not use if-statements anywhere in this assignment*, as they are not necessary. Submissions containing if-statements will be returned for you to revise. Similarly, students with prior programming experience should not try to use loops or recursion.

Getting Help

If you do not know where to start, if you do not understand testing, or if you are completely lost, please see someone immediately. This can be the course instructor, a TA, or a consultant. Do not wait until the last minute, particularly since this is due just after a weekend. A little in-person help can do wonders. See the staff page for more information.

The Currency Exchange Web Service



Before you do anything at all, you might want to play around with the currency exchange web service. You do not need any Python to do this; just a web browser.

For this assignment, you will use a simulated currency exchange service that never changes values. This is important for testing; if the answer is always changing, it is hard to test that you are getting the right answers. The appendix explains how you can make a few minor changes to get real-time currency-exchange results. However, we do not want you to submit such code for your assignment; stick with the fixed, unchanging server.

To use the service, you employ special URLs that start with the following prefix:

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

This prefix is 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=source & dst=target & q=amount
```

where *source* is a three-letter code for the original currency, *target* 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

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

The query is not enough by itself. To use it, you have to make it part of a web page URL. The full URL for this query is

http://cs1110.cs.cornell.edu/2017fa/a1server.php?src=USD&dst=EUR&amt=2.5

Click on the link to see it in action.

You will note that the "web page" in your browser is just a single line in the following format:

```
{ "success" : true, "lhs" : "2.5 United States Dollars", "rhs" : "2.0952375 Euros", "error" : "" }
```

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 what you know about string operations and methods to pull out the relevant data out of the JSON string.

You should try a few more currency queries to familiarize yourself with the service. Note that if you enter an invalid query (for example, using a non-existent currency code like "AAA"), you will get the following response in error:

```
{"success":false, "lhs":"", "rhs":"", "error":"Source currency code is invalid." }
```

Similarly, if you enter a query with two valid currency codes, but with an invalid quantity value, you will get the following error:

```
{"success":false, "lhs":"", "rhs":"", "error":"Currency amount is invalid." }
```

For all error queries, the "lhs" and "rhs" values are blank, while "success" is false. The value "error" is a specific error message describing the problem. This will be important for error handling in this assignment.

Focus of the Assignment

Your primary goal in this assignment is to use the currency exchange service to write the following function:

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

    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.

    The value returned has type float.

    Parameter currency_from: the currency on hand (the LHS)
    Precondition: currency_from is a string for a valid currency code

    Parameter currency_to: the currency to convert to (the RHS)
    Precondition: currency_to is a string for a valid currency code

    Parameter amount_from: amount of currency to convert
    Precondition: amount_from is a float
    """
```

This function will involve several steps. You will get the JSON string from the web service, break up the string to pull out the numeric value (as a substring), and then convert that substring to a float. As this is the very first assignment, we are going to take you through this process step-by-step. **However, not every function that we ask you to implement will be used by `exchange`.**

This assignment 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 the helper functions before finishing the functions that use them*.

Currency Exchange Table

In order to make it easier to test your program, we have fixed the exchange rates in our web service. That way you can test the answer in a web browser (using a currency query URL) and then compare the results to your Python program, without worrying about rates fluctuating.

The following currencies are supported by our web service:

Code	Name	1 USD =		Code	Name	1 USD =
AED	United Arab Emirates Dirham	3.672878		LBP	Lebanese Pound	1508
AFN	Afghan Afghani	68.3935		LKR	Sri Lankan Rupee	152.610797
ALL	Albanian Lek	111.63		LRD	Liberian Dollar	116.871332
AMD	Armenian Dram	478.16		LSL	Lesotho Loti	12.926476
ANG	Netherlands Antillean Guilder	1.780104		LYD	Libyan Dinar	1.360553
AOA	Angolan Kwanza	165.9215		MAD	Moroccan Dirham	9.34595
ARS	Argentine Peso	17.265		MDL	Moldovan Leu	17.699209
AUD	Australian Dollar	1.25541		MGA	Malagasy Ariary	2948.1
AWG	Aruban Florin	1.794996		MKD	Macedonian Denar	51.61
AZN	Azerbaijani Manat	1.7		MMK	Myanma Kyat	1358.05
BAM	Bosnia-Herzegovina Convertible Mark	1.63925		MNT	Mongolian Tugrik	2433.43053
BBD	Barbadian Dollar	2		MOP	Macanese Pataca	8.059501
BDT	Bangladeshi Taka	81.498388		MRO	Mauritanian Ouguiya	365.04065
BGN	Bulgarian Lev	1.639493		MUR	Mauritian Rupee	32.887
BHD	Bahraini Dinar	0.377064		MVR	Maldivian Rufiyaa	15.404937
BIF	Burundian Franc	1743.05		MWK	Malawian Kwacha	725.605
BMD	Bermudan Dollar	1		MXN	Mexican Peso	17.8686
BND	Brunei Dollar	1.352488		MYR	Malaysian Ringgit	4.228868
BOB	Bolivian Boliviano	6.975127		MZN	Mozambican Metical	61.43
BRL	Brazilian Real	3.113905		NAD	Namibian Dollar	12.91
BSD	Bahamian Dollar	1		NGN	Nigerian Naira	359.066229
BTC	Bitcoin	0.00021745803		NIO	Nicaraguan Córdoba	30.008984
BTN	Bhutanese Ngultrum	64.106522		NOK	Norwegian Krone	7.778974
BWP	Botswanan Pula	10.070807		NPR	Nepalese Rupee	102.553085
BZD	Belize Dollar	2.015882		NZD	New Zealand Dollar	1.38733
CAD	Canadian Dollar	1.240735		OMR	Omani Rial	0.384992
CDF	Congolese Franc	1562.881563		PAB	Panamanian Balboa	1
CHF	Swiss Franc	0.955688		PEN	Peruvian Nuevo Sol	3.238015
CLF	Chilean Unidad de Fomento	0.02311		PGK	Papua New Guinean Kina	3.194708
CLP	Chilean Peso	621.715		PHP	Philippine Peso	51.095
CNY	Chinese Yuan	6.52615		PKR	Pakistani Rupee	105.19
COP	Colombian Peso	2926.1		PLN	Polish Zloty	3.55689
CRC	Costa Rican Colón	576.836761		PYG	Paraguayan Guarani	5656.15
CUC	Cuban Convertible Peso	1		QAR	Qatari Rial	3.689998
CUP	Cuban Peso	25.5		RON	Romanian Leu	3.853287
CVE	Cape Verdean Escudo	92.9		RSD	Serbian Dinar	100.200167
CZK	Czech Republic Koruna	21.884838		RUB	Russian Ruble	57.5206
DJF	Djiboutian Franc	178.77		RWF	Rwandan Franc	830.084566
DKK	Danish Krone	6.234868		SAR	Saudi Riyal	3.7506
DOP	Dominican Peso	47.180379		SBD	Solomon Islands Dollar	7.725799

Code	Name	1 USD =	Code	Name	1 USD =
DZD	Algerian Dinar	110.94	SCR	Seychellois Rupee	13.646594
EGP	Egyptian Pound	17.645	SDG	Sudanese Pound	6.677259
ERN	Eritrean Nakfa	15.333155	SEK	Swedish Krona	7.963411
ETB	Ethiopian Birr	23.415537	SGD	Singapore Dollar	1.3524
EUR	Euro	0.838095	SHP	Saint Helena Pound	0.766307
FJD	Fijian Dollar	2.008905	SLL	Sierra Leonean Leone	7537.33542
FKP	Falkland Islands Pound	0.766307	SOS	Somali Shilling	579.49334
GBP	British Pound Sterling	0.766307	SRD	Surinamese Dollar	7.438
GEL	Georgian Lari	2.482075	STD	São Tomé and Príncipe Dobra	20565.521652
GGP	Guernsey Pound	0.766307	SVC	Salvadoran Colón	8.750313
GHS	Ghanaian Cedi	4.41291	SYP	Syrian Pound	515
GIP	Gibraltar Pound	0.766307	SZL	Swazi Lilangeni	12.905459
GMD	Gambian Dalasi	46.075	THB	Thai Baht	33.148
GNF	Guinean Franc	8968.85	TJS	Tajikistani Somoni	8.810921
GTQ	Guatemalan Quetzal	7.298907	TMT	Turkmenistani Manat	3.50998
GYD	Guyanaese Dollar	207.79838	TND	Tunisian Dinar	2.425088
HKD	Hong Kong Dollar	7.82541	TOP	Tongan Pa‘anga	2.201704
HNL	Honduran Lempira	23.391297	TRY	Turkish Lira	3.439969
HRK	Croatian Kuna	6.224506	TTD	Trinidad and Tobago Dollar	6.730511
HTG	Haitian Gourde	63.021504	TWD	New Taiwan Dollar	30.07725
HUF	Hungarian Forint	256.667	TZS	Tanzanian Shilling	2239.4
IDR	Indonesian Rupiah	13332.915894	UAH	Ukrainian Hryvnia	25.992936
ILS	Israeli New Sheqel	3.56023	UGX	Ugandan Shilling	3618.908069
IMP	Manx pound	0.766307	USD	United States Dollar	1
INR	Indian Rupee	64.075	UYU	Uruguayan Peso	28.764997
IQD	Iraqi Dinar	1169.000369	UZS	Uzbekistan Som	8100.55
IRR	Iranian Rial	33225.5	VEF	Venezuelan Bolívar Fuerte	10.09029
ISK	Icelandic Króna	106.210839	VND	Vietnamese Dong	22732.614564
JEP	Jersey Pound	0.766307	VUV	Vanuatu Vatu	103.774063
JMD	Jamaican Dollar	128.871658	WST	Samoan Tala	2.492203
JOD	Jordanian Dinar	0.709001	XAF	CFA Franc (BEAC)	549.754252
JPY	Japanese Yen	108.85857545	XAG	Troy Ounce of Silver	0.05568549
KES	Kenyan Shilling	103.236159	XAU	Troy Ounce of Gold	0.0007466
KGS	Kyrgystani Som	68.494341	XCD	East Caribbean Dollar	2.70255
KHR	Cambodian Riel	4047.1	XDR	Special Drawing Rights	0.705231
KMF	Comorian Franc	413.45	XOF	CFA Franc (BCEAO)	549.754252
KPW	North Korean Won	900	XPD	Troy Ounce of Palladium	0.00103738
KRW	South Korean Won	1132.9975	XPF	CFP Franc	100.011331
KWD	Kuwaiti Dinar	0.301493	XPT	Troy Ounce of Platinum	0.00098665
KYD	Cayman Islands Dollar	0.833299	YER	Yemeni Rial	250.48907
KZT	Kazakhstani Tenge	341.47	ZAR	South African Rand	12.9032
LAK	Laotian Kip	8289.75	ZMW	Zambian Kwacha	9.125
LBP	Lebanese Pound	1508	ZWL	Zimbabwean Dollar	322.355011

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 testing your functions; not for writing them. 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.

Iterative Development (How to Work Through the Assignment)

One of the most important outcomes of this assignment is that you understand the importance of testing. This assignment will follow an *iterative development cycle*. That means you will write a few functions, 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.

In this section we help you get started with this process. We also provide an overview of the rest of the assignment.

Setting up Python

To do this assignment, Python must be set up properly. If you have not already done this, follow the installation instructions to set it up on your computer. Alternatively, you can just work in the ACCEL lab.

You should also create a folder on your hard drive that is dedicated to this assignment and this assignment only. Every time that you work on a new assignment, we want you to make a new folder, to keep things organized and avoid problems with naming collisions. Make sure that the command shell and Komodo Edit are both open in the current folder before you start.

The Module `a1`

In your newly created directory, you should create the module `a1` (with file name `a1.py`). This will be the main module for this assignment. Following the style guidelines your file should start with a descriptive docstring, and the last two lines should be (1) the name and netid of the authors and (2) the date the file was last editted. This is the docstring that we would like you to use:

```
"""
Module for currency exchange

This module provides several string parsing functions to implement a
simple currency exchange routine using an online currency service.
The primary function in this module is exchange.

YOUR NAME AND NETID HERE
THE DATE COMPLETED HERE
"""
```

Cut-and-paste this docstring into `a1`, making sure to insert your name and date as appropriate.

The Module `a1test`

Iterative development hinges on proper unit testing, which was covered in lecture and lab. In the same folder as `a1.py`, create the module `a1test` (with file name `a1test.py`). This will be the unit test for the `a1` module.

As with `a1.py`, this file should start with a docstring specification that includes (1) your name and netid and (2) the date the file was last editted. This is the docstring that we would like you to use:

```
"""
Unit test for module a1

When run as a script, this module invokes several procedures that
test the various functions in the module a1.

YOUR NAME AND NETID HERE
THE DATE COMPLETED HERE
"""
```

You will get experience writing your own docstrings in a later assignment. After this docstring, add the following two lines.

```
import cornell
import a1
```

Finally four *procedure stubs* to the file `a1test.py`: `testA`, `testB`, `testC`, and `testD`. Remember that a procedure stub should have the keyword `pass` (indented) after the header, but nothing else. For example, here is the code for the first one.

```
def testA():
    """Test procedure for Part A"""
    pass
```

We will add our test cases to these procedures later.

These procedures will eventually contain your unit tests, which we talked about in Lecture 6. If you are curious about how they are supposed to work, look at the example test2.py in that lecture.

Finally at the end of `a1test`, you will need to add code to active the test procedures. Add these lines:

```
testA()
testB()
testC()
testD()
print("Module a1 passed all tests")
```

Again, see the lecture slides for an explanation of why you are doing this. The script code will call your four test procedures, which are (currently) empty. If everything is working, then the module will print out the message

```
"Module a1 passed all tests"
```

Try this out.

Instructions for the Remainder of the Assignment

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

Write a function header into `a1`

We will give you the header to write. We will also give you a detailed docstring specification for the function. You should copy-and-paste the specification into the function body, indented.

Add test cases to `a1test`

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 the `cornell` module. Furthermore, your tests should be *representative*. While we talked about this in class, you might be a little unsure of what we are asking for here. If so, you might want to work on the optional exercises in lab 4 with a consultant before going any further.

Write the function bodies

Make sure that the function satisifies the specifications exactly. If the specification says to return something, you need a `return` statement. Make sure that the value returned is of the correct type.

Run the unit test `a1test`

If errors are found, fix them and re-test. Keep doing this until no more errors are found.

Writing Function Specifications

The descriptions that we provide in each part below represent the level of completeness and precision we are looking for in your docstring comments. In fact, it is best to copy-and-paste these descriptions to create the first draft of your docstring comments. If you do not cut and paste, please adhere to the conventions we use, such as using a single line, followed by a blank line and a more descriptive paragraph, or by using "Returns: ..." for fruitful-functions. Using a consistent set of good conventions in this class will help us all.

If you want to see if your specifications are written correctly, start an interactive Python shell and type

```
>>> import a1
>>> help(a1)
```

This should list all the functions with their specifications.

Part A: Breaking Up Strings

A large part of this assignment is breaking up a JSON string. Conceptuually, you want to separate the currency amount from the currency name. For example, if we are given the string

```
"0.838095 Euros"
```

Then we want to break it up into "0.838095" and "Euros".

This is the motivation for the two functions below. The implementation of these functions should be relatively simple. We were able to implement them both in one or two lines.

---

before_space(s)

Returns: Substring of s; up to, but not including, the first space

Parameter s: the string to slice
Precondition: s has at least one space in it

---

after_space(s)

Returns: Substring of s after the first space

Parameter s: the string to slice
Precondition: s has at least one space in it

Implement these functions according to their specification, as described in the Instructions for the Remainder of the Assignment. In other words,

- Write the header and specification in `a1.py`
- Place test cases in the procedure `testA()` of `a1test.py`
- Implement the functions in `a1.py`.
- Test for and correct errors until no errors remain.

To test the functions, you should make use of `assert_equals` in the module `cornell` to compare the result of each functions with the string that you expect to get back. Our solution has four test cases for each of the two functions above. When you think about what test cases you want to include, consider the following:

- Does the specification allow for strings with more than one space?
- Does it allow for strings that start with a space?
- Does it allow for strings that don't have any spaces?

Finally, do not forget to add a specification to `testA()`. Just because it is used for testing does not mean that it should not be properly specified.

Part B: Processing a JSON String

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 keyword "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.

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.

---

first_inside_quotes(s)

Returns: The first substring of s between two (double) quote characters

A quote character is one that is inside a string, not one that delimits it. We typically use single quotes (') to delimit a string if want to use a double quote character (") inside of it.

Example: If s is `'A "B C" D'`, this function returns `'B C'`
Example: If s is `'A "B C" D "E F" G'`, this function still returns `'B C'` because it only picks the first such substring.

Parameter s: a string to search
Precondition: s is a string with at least two (double) quote characters inside.

You should have completed the function above in lab 3. Because this function is technically part of the lab, and not the assignment, you may talk to students other than your partner about it (collaboration is always allowed on labs). **This is the only function for which this is allowed. The rest of the functions are part of the assignment, so you may only collaborate with your partner.**

Once you have this function completed, you should move on to the following functions.

---

get_lhs(json)

Returns: The the LHS 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 `"lhs"`. For example, if the JSON is

```
'{"success":true, "lhs":"2 United States Dollars", "rhs":"1.825936 Euros", "error":""}'
```

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

---

get_rhs(json)

Returns: The RHS 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 `"rhs"`. For example, if the JSON is

```
'{"success":true, "lhs":"2 United States Dollars", "rhs":"1.825936 Euros", "error":""}'
```

then this function returns `'1.825936 Euros'` (not `'"1.825936 Euros"'`). 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

---

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, "lhs":"", "rhs":"", "error":"Source currency code is invalid."}'
```

then the query is not valid, so this function returns `True` (It does NOT return the message `'Source currency code is invalid'`).

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, enter a query URL into the web service and copy the result into a test case.

You should not need a conditional statement to implement these functions; simply find the position of the appropriate keyword and extract the value in quotes immediately after it. Your implementation must make use of the find() or `index()` string methods, plus the helper function `first_inside_quotes()`.

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`. Do not forget to specify `testC()` properly.

```
def currency_response(currency_from, currency_to, amount_from):
    """
    Returns: a JSON string that is a response to a currency query.

    A currency query converts amount_from money in currency currency_from
    to the currency currency_to. The response should be a string of the form

        '{"success":true, "lhs":"<old-amt>", "rhs":"<new-amt>", "error":""}'

    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, while "success" will be followed by the value false.

    Parameter currency_from: the currency on hand (the LHS)
    Precondition: currency_from is a string

    Parameter currency_to: the currency to convert to (the RHS)
    Precondition: currency_to is a string

    Parameter amount_from: amount of currency to convert
```

```
    Precondition: amount_from is a float
    """
```

While this function sounds complicated, it is not as bad as you think it is. There is a function inside of the `cornell` module called `urlread`. This function takes a single web address as an argument, and returns the contents of the web page. Try this now in the interactive shell by typing

```
>>> import cornell
>>> cornell.urlread('http://www.cornell.edu')
```

You will notice that this function does exactly what you want. So what is the challenge? The challenge is coming up with the correct web address. Revisit our explanation of how the currency service works to see why this is a potential challenge.

Testing

You need to ensure that the function returns exactly the right JSON string for the value given. The best way to test this is to use a web browser to manually get the right JSON answer. For example, one test case can be constructed by seeing the result of going to the URL

http://cs1110.cs.cornell.edu/2017fa/a1server.php?src=USD&dst=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.

Part D: Currency Exchange

We are now ready for the final part of the assignment. Implement the following specifications, again using our test-case-before-function-body approach. The test cases should go in procedure `testD()` in `a1test`, which you should properly specify. You may wish to use `assert_true()` instead of `assert_equals()` in some of your test cases. There is also a case in which you will want to use `assert_floats_equal()`. If you are confused by this, you can get help by working through the optional exercises in lab 4 with a consultant.

```
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 **should not** use the table of currencies. That would make a very large function with a lot of if-statements. You are not allowed if-statements in this lab. Instead, you **must use the functions `currency_response` and `has_error` as helper functions**.

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

    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.

    The value returned has type float.

    Parameter currency_from: the currency on hand (the LHS)
    Precondition: currency_from is a string for a valid currency code

    Parameter currency_to: the currency to convert to (the RHS)
    Precondition: currency_to is a string for a valid currency code

    Parameter amount_from: amount of currency to convert
    Precondition: amount_from is a float
    """
```

Testing

In the case of `iscurrency()`, you will find the exchange table 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`. 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 problem that we saw in class: real numbers cannot be represented exactly. This creates problems when you try to test equality between floats. To solve this problem, `cornell` 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 in the optional exercises of lab 4.

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.

Finishing the Assignment

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

- You have indented with spaces, not tabs (this is not an issue if using Komodo).
- Functions are each separated by two blank lines.
- Lines are short enough (~80 characters) that horizontal scrolling is not necessary.
- The specifications for all of the functions are complete.
- Function specifications are immediately after the function header and indented.
- Your name(s) and netid(s) are in the comments at the top of the modules.

One of the things that you may have the biggest difficulty with is breaking up long lines. First, you may not be aware when your lines are too long. There is an easy way to test. In Komodo Edit choose **Edit** > **Preferences** > **Editor** > **Smart Edition**. Select the checkbox that says **Draw the edge line column**.

As for breaking up long lines, there are two solutions. First, Python allows you to "hit Return" within any expression inside of parentheses. So if you are adding together several expressions together, like

```
a = 'Hello ' + name + ', it is good to meet you'
```

you can break it up over several lines, using parentheses, as follows:

```
a = ('Hello ' + name +
     ', it is good to meet you')
```

Another solution is to use the backslash symbol \. Remember that this is the escape character for making special characters in strings. It also has a special effect outside of a string. If you type this symbol, immediately followed by a return, then Python will know to continue to the next line. So you can rewrite the addition above as

```
a = 'Hello ' + name + \
    ', it is good to meet you'
```

Turning it In

Upload the files `a1.py` and `a1test.py` to CMS by the due date: **Sunday, September 17th at 11:59 pm**. 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.

Check the CMS daily until you get feedback from a grader. Make sure your CMS notifications for CS 1110 are set so that you are sent an email when one of your grades is changed. To find the feedback, click on the Assignment 1 link in CMS. On the page you are brought to, click on the red word "show" in the line "Grading Comments & Requests (show)." You can contact your grader if you have questions about their feedback; you can see their netid in the place you can see their feedback.

Within 24 hours, do **RRRRR**: **Read** the feedback, **Revise** your program accordingly, **Resubmit**, and **Request** a **Regrade** using the CMS. If you do not request a regrade, we have no simple way of knowing that you have resubmitted.

This whole process, starting from first submission on **Sunday, September 17th**, continues until you submit a solution that demonstrates complete mastery; in some cases this may require multiple additional resubmits by you. You need to complete this process within one week. You need to have submitted a final, correct version by **Sunday, September 24th**, which means you will probably want to have re-submitted at least once before then.

Survey

In addition to turning in the assignment, we ask that you complete the new survey posted in CMS. These assignments are still rather new(ish), and we would like some understanding of how long you spent on the assignment, your impression of the difficulty,

and what could be done to improve it.

Please try to complete the survey within a day of turning in this assignment. Remember that participation in surveys compromise 1% of your final grade. We also ask that you be honest in your answers.

Appendix: Getting Real-Time Exchange Rates

This section is not part of the assignment. It is optional. Furthermore, **do not** make the changes in this section to the file that you submit for grading. It will be sent back to you to fix.

This assignment was first designed back in 2012, to take advantage of a service called iGoogle. iGoogle was a JSON service provided by Google (hence the name) which supported simple Python programs. It was intended for any data that might be changing often, such as currency exchange rates, weather data, or similar types of things. Unfortunately, Google discontinued the service in November 2013, two months after we ran the assignment for a second time.

This meant that we could still simulate a fake currency exchange service, but we no longer had a real-world example to show off the power of this assignment. Most replacements to iGoogle typically charge for their service (because they are used by currency traders), and we could not justify the subscription cost for a single assignment.

Fortunately, there is a great service called Open Exchange Rates. This service still charges, but it is free if you only need a new currency value once an hour. This is a pretty good compromise, because that is frequent enough for anyone who is not a currency trader.

The data from Open Exchange Rates is not in a format usable by this assignment. However, it does allow your instructor to turn our fake currency service into a real currency service. We are actually running two currency servers in this class. In the web service instructions we told you to use the URL prefix

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

If you change that prefix to

```
http://cs1110.cs.cornell.edu/2017fa/exchange.php?
```

you will get our real server instead. Try that out on converting dollars to Euros (pick small values for now).

The server updates once an hour at 30 minutes after the hour. To see this in action, run a query just before this time, at say 8:28. Wait 5 minutes and run the same query again. See how it changes? This is one of the reasons we did not use the real service in development; it is too hard to test against. In fact, even professional software engineers would do what we did: write a program against an unchanging exchange service before deploying it against the real thing.

We promise to keep the real server running for at least the next year, should you wish to show this off to other people.

Course Material Authors: D. Gries, L. Lee, S. Marschner, & W. White (over the years)