CSE 231 **Fall 2013**

Programming Project 10

This assignment is worth 60 points (6% of the course grade) and must be **completed and turned in before 11:59 on Monday, November 25, 2013.**

Assignment Overview
This assignment will continue our work with Object Oriented Programming. In this project, you will design your own class that will interact with the Google Finance currency calculator API.

Background
The acronym API is short for "Application Programming Interface." It is usually a collection of functions, methods or classes that supports the interaction of the programmer (the application developer, i.e., you) with some particular program. Google has many API's and it doesn't advertise how you can use them from other applications, but they are there for anyone to use. To interact with the Google Finance currency calculator, a fairly simple web API has been developed. Our goal is to create a user class that, when instantiated, has local values to be used in a personal query. The instance will query the Google Finance API's for user-specified information as needed, and report back some useful information.

The API
We will be using the calculator API for the purpose of converting currencies. The API is based on creating a query using the http protocol of the web. Essentially, you send a web address (with the query embedded in the web address) and a web page is returned with your information, albeit in a rather odd form. Thus, you have to create a particular web query (a particular kind of web page request) to get the information back. The nice part is that you can do the queries on the web (to see the results) and can then do it in Python (discussed below).

The general form of a currency conversion query is shown below:

https://www.google.com/finance/converter?a=X&from=YYY&to=ZZZ

where X is the amount to be converted, YYY is the 3-character currency code of the amount to be converted, and ZZZ is the 3-character currency code that it should be converted to. There are no spaces in the URL. Some examples are listed next. Copy and paste each one into a browser to see what is returned. Then create your own to test.

```
https://www.google.com/finance/converter?a=100&from=USD&to=SEK
https://www.google.com/finance/converter?a=1&from=USD&to=EUR
https://www.google.com/finance/converter?a=10&from=GBP&to=CAD
https://www.google.com/finance/converter?a=20&from=USD&to=CNY
```

Here is one that doesn't work properly because the last currency code is invalid (xxx):

```
https://www.google.com/finance/converter?a=100&from=USD&to=xxx
```

To see the string that is returned before being displayed by your browser view the *source*. For example, in Firefox use Tools ➔ Web Developer ➔ Page Source. Similar tools exist on all browsers.

Parsing the Result
The entire result is returned as a LARGE single string. You will need to parse this string to find the information required. There are over 350 lines to the string and the one line you are interested in is about 8 lines from the bottom. For example, for the first sample above that converts 100 USD to SEK the line of interest is listed below and all you want is the converted value: 665.1200

```
<div id=currency_converter_result>100 USD = <span class=bld>665.1200 SEK</span>
```

Make sure you examine the invalid case (the last example above) to see what is returned.

Querying the web from Python
The library urllib.request has tools to create a connection to a web server and download data from the web server as if it were a file. The particular function in the library is urlopen, which we can use to create the connection. Like any file, you open it, read the contents, and then close it. Here is the example converting 100 USD to SEK :

```
import urllib.request
web_obj =
urllib.request.urlopen("https://www.google.com/finance/converter?a=100&from=USD&to
=SEK")
results_str = str(web_obj.read())
web_obj.close()
```

It is important to note that the result of read() is not a *string*, but an *array of bytes*. It must therefore be converted to a string, or you will not be able to do anything with it. Once it is converted, you will be able to perform normal string methods on the result in order to extract the information desired. You can use this approach to confirm that the currency types passed are both valid, and what the result of the conversion is. Note that if either currency type is invalid, you will receive no results on the line of interest. There is no indication of which currency type was not valid.

Try that sequence of code in the Python Shell to see how it works. Examine the string `results_str` to understand what is returned and find the line of interest (mentioned above). Then replace the three values that vary (100, USD, SEK) with variables such as `amount`, `from_currency`, and `to_currency`. Practice in the shell assigning values to those variables and getting results. Then experiment with splitting, finding, and slicing of `results_str` to extract the value of interest, 665.1200 in the case of converting 100 USD to SEK.

Specification for the Class Currency
Create a class called Currency. This class is to be instantiated with an amount and currency type (three character currency code), and is used to fetch information from the web. The class details:

1. `__init__` The constructor takes the following arguments (with defaults indicated):
   a. An amount. Default: 0 (CHANGE: default changed from 1 to make app easier)
   b. A currency code. If the currency code provided is invalid, set the currency code to '' (an empty string) and the amount to 0. Default: USD
   c. CLARIFICATION: If either argument is invalid, set to default.
2. `convert_to`. This method takes a single argument, a currency code, with no default. It *returns a new Currency object* with the new currency code and the converted amount. This method is where you will use urllib functions to get data from the web to convert this instance's currency to the currency specified in the parameter.

3. The following operations must be implemented. For each method it should be able to handle operands of type Currency, float, or int (for example, your `__add__` method should be able to handle curr1 + curr2, or handle curr1 + 5, or handle curr1 + 2.71). CLARIFICATION: for addition and subtraction return use self's currency code.
   a. `__str__`: Return the amount and type as a string
   b. `__add__` and `__radd__`
   c. `__sub__` and `__rsub__` (remember that subtraction isn't commutative)
   d. `__gt__` (CHANGE: only handle operands of type Currency for __gt__)
   e. `__repr__`
4. Arithmetic methods (`__add__`, `__radd__`, `__sub__`, `__rsub___`) must all return type Currency.
5. Your class only needs to handle currency codes USD, EUR, SEK, CAD, CNY, and GBP

Specifications for the Test Program

1. You will develop a program to serve as a test bed for "class Currency". That is, the only purpose of the program is to demonstrate that each method of "class Currency" is implemented correctly.

The source code for your test bed will be contained in the file named "proj10-test.py". That file will import "currency.py".

2. Your test bed will not perform any input operations. Instead, all test cases will be embedded in the program itself.

3. The output produced by your test bed will be appropriately labeled so that the reader can understand the purpose and result of each test case without examining the source code.

Specifications for the Application Program

1. You will develop an application program which uses "class Currency" to solve the problem described below. The source code for your application program will be contained in the file named "proj10-app.py". That file will import "currency.py".

2. The scenario for the application program is an account that starts with 1000 USD ($1000) from which you deduct expenses in a variety of currencies (you only need to handle the currencies specified above for class Currency). Deductions will occur in a variety of currencies, but the account will keep track of dollars (USD).

3. The program will

   a. Display a descriptive message about what the program does.
   b. Establish the account with 1000 USD (no input is necessary).
   c. Repeatedly prompt the user for expense deductions until the user enters 'q'
      i. The expense MUST be in the form:
         `XXX YYY`
         where
         1. XXX is an int or a float
         2. YYY is a currency code
      ii. Deduct the expense from the account (using Currency subtraction). Display the account balance in USD.

iii. If an invalid XXX value or an unrecognized currency code or an incorrectly formatted expense is entered, an appropriate error message will be displayed and another prompt will be made.
iv. If the account gets overdrawn (i.e. it holds a negative amount), break out of the loop and print a message that the account is overdrawn. You can do this by creating a Currency instance with value 0 USD and then use ">" which calls the __gt__ method that you wrote.

Assignment Deliverables

The deliverables for this assignment are the following files:

currency.py – the source code for your class Currency
proj10-test.py – the source code for your test program
proj10-app.py – the source code for your application program

Be sure to use the specified file names and submit them for grading via the **handin** system before the project deadline.

Here are some lines of code such as will appear in your proj10-test.py to demonstrate that your class works (this example is incomplete because it doesn't test all the methods):

```
curr = Currency(7.50, 'USD')

print(curr)

curr2 = Currency(2, 'EUR')

print(curr2)

new_curr = curr2.convert_to('USD')

print(new_curr)

sum_curr = curr + curr2

print(sum_curr)

sum_curr2 = curr + 5.5

print(sum_curr2)
```

Notes and Hints:

1. Enter the example URLs in a browser to see the web pages that result from the queries. These give you a good idea of what you are parsing.
2. Experiment in the Python shell with the `urllib.request` functions (as shown above) to fetch a URL and see what you get.
3. Experiment with one such fetch and see how to parse the fetched information to gather the required information.
4. Use various string functions to break the response down, piece by piece, until it is in a form you can easily work with.

5. Start with the easiest methods: `__init__` and `__str__`. Test them before moving on, e.g. test the first four lines of the example main program.
6. Next work on the `convert_to` method using what you learned from steps 1-4. Test is using the first 6 lines from the example main program.
7. Next work on the `__add__` method. Think about this one! *How should you handle the situation when the two currencies being added are different? (Figuring that out is part of your challenge, but note that the right answer is not a programming issue. Instead it is a specification issue: what makes sense?)* You can now test the whole main program. Of course, sample programs from class and the text that implement an `__add__` method will be helpful here.
8. Finally, implement and test the remaining operations: `__radd__`, `__sub__`, etc. Remember that some operations are not commutative!