CMSC 201 Spring 2016
Lab 11 – Modules and Libraries

Assignment: Lab 11 – Modules and Libraries
Due Date: During discussion, April 25th through April 28th
Value: 10 points

Part 1: Introduction to Modules

For most of the semester, we have been using **`emacs`** (the text editor in GL), to write a number of lines of code. These "scripts" are then run in python and interpreted as we go. As part of these scripts, we have been using a variety of functions in Python that we have said are "built-in." The list of functions and types that are always available includes some of our most commonly used keywords such as: **`print()`**, **`int()`**, **`str()`**, **`list()`**, and **`len()`**. There are over 50 of these built-in functions in Python.

What happens if we want to use something other than these 50 built-in functions? We have two choices: we can write our own function, or we can try and find a function that has already been written for us. If we write our own functions, we may want to organize them by splitting them into several files (e.g., one file for user input, one for file I/O, one for data validation, etc.). You may want to organize our files so that we can more easily reuse our code. You may want to use a handy function that you've written in several programs without copying its definition into each program.

From a vocabulary standpoint, a "***library***" is a collection of code that has been written by someone else, ready for you to use in your program. For example, a "standard library" is a library that is considered part of the language and must be included with any Python system. Python's standard library has hundreds of modules that are available with minimal effort. You can see the complete list for Python 3.3 here: https://docs.python.org/3.3/py-modindex.html. On a lower level, a standard library is broken up into "modules." Related functions and data types are grouped into the same module. Functions defined in a module must explicitly be imported into your program before they are able to be used.

Part 2: Examples of Modules

For example, say we wanted to calculate the exponent of a number. If we wanted to do this, we really have three choices: 1) we write a function to calculate it and store it in our file with **`main`**, 2) we write a function to calculate it and store it in a separate module, or 3) we use a function that is built into the **`math`** module.

Option 1: New Function

In this option, we are writing a new function and placing it inside of the same file as **`main`**. This is the first way that you were introduced to functions, and this is how you should have used functions in Project 1. The function takes in two parameters and returns the value of base1 raised to the power of exp1.

lab11_1.py

```
    def funExponent(base1, exp1):
        return base1**exp1

    def main():
        base1 = 5
        exp1 = 2
        print( funExponent(base1, exp1) )
```

Option 2: New Module

In this option, we write a module with a function called **`exponent()`** that takes in two parameters. The first parameter is the base, and the second is the exponent. The function exists in a completely separate file from **`main()`**. Even though the file is named **`lab11_exp.py`**, we import it by using the import lab11_exp without the file extension. Also, in this case, the file of **`lab11_exp.py`** is in the same folder as the **`lab11_2.py`** file. Finally, because we imported the entire library, when we call the function we have to explicitly tell Python where the function exists. In other words, when we call the function, we use **`lab11_exp.exponent(x,y)`**.

lab11_exp.py

```
    def exponent(base1, exp1):
        return base1**exp1
```

lab11_2.py

```
    import lab11_exp

    def main():
        base1 = 4
        exp1 = 3
        print( lab11_exp.exponent(base1, exp1) )
    main()
```

Option 3: Python's Math Module

In this final option, we are using a function called **`pow(x,y)`** from the math library that is available by default in Python. In order to use this function, we need to import the **`math`** library. When we call the **`pow()`** function, we also have to make sure that we tell Python where that function exists.

lab11_3.py

```
    import math

    def main():
        base1 = 4
        exp1 = 3
        print( math.pow(base1, exp1) )
    main()
```

Part 3: Ways to Import Functions

In class, we discussed the three main ways to import a function from a module. The three ways that we discussed were:

```
    import somefile
    from   somefile import *
    from   somefile import specificThing
```

Importantly, the difference between the three ways is what gets imported from the file and what name refers to it after importing.

Option 1: `import somefile`

If we choose this option, we import the entire module, including all functions that exist inside that module. Remember, some modules can have dozens or even hundreds of functions in them so we have to be careful what we import. For this example, we can go ahead and just use the same code example that we saw earlier. Notice that when we use the **`import`** command, we have to use the module name whenever we use one of the functions from that module. For example, we have to use **`math.pow(x, y)`** in order to use the **`pow()`** function inside of the **`math`** module.

lab11_3.py

```
    import math

    def main():
        base1 = 4
        exp1 = 3
        print( math.pow(base1, exp1) )
    main()
```

Option 2: `from somefile import *`

In this option, we again import the entire module, including all functions that exist inside that module. We do this by specifying the file we want to import using the **`*`**. In this situation the **`*`** character indicates a wildcard. A wildcard character can be used to substitute for any other character or characters in a string – in this case, it imports everything in the module. When we use the **`import *`** command, we do not need to include the module name, and we can just access the function directly.

lab11_4.py
```
from math import *

def main():
    base1 = 4
    exp1 = 3
    print( pow(base1, exp1) )
main()
```

Option 3: `from somefile import specificThing`

In this final example, we are going to import a specific function from the module. This is very efficient because rather than importing the entire module, we are only going to import the function that we need. Again, in this situation, we do not need to refer to the module name after we import it. The biggest downside to importing functions this way is that we would have to list each and every function that we want to use from that specific module.

lab11_5.py
```
from math import pow

def main():
    base1 = 4
    exp1 = 3
    print(pow(base1, exp1))
main()
```

Part 4: Using the Python Standard Library (`math`)

As we have seen throughout the semester, there are a number of functions that we need to use on a pretty regular basis. Many common functions have already been programmed (and exhaustively tested) for the Python standard library. Python's standard library is very extensive, and includes more than 200 individual modules for everything from math to file management. The library contains built-in modules (written in C) that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardized solutions for many problems that occur in everyday programming.

You can see the complete Python 3.3 library here:
https://docs.python.org/3.3/library/

Part 5: Using the Python Standard Library (`math`)

One of the more important modules in Python is the **`math`** module. This contains a lot of essential mathematical functions like **`sin`** and **`cos`**. It also contains valuable mathematical constants such as pi. These values are stored in global variables; global variables are variables in a module (created via an assignment statement) that are not part of any function.

As we have mentioned previously, the math module has many functions (far too many than to list here). Take a look at the URL below for additional information about the math module.
https://docs.python.org/3.3/library/math.html

Take a few minutes to look over the functions that are associated with the math module. See if you can find the functions **`sqrt`**, **`floor`**, and **`ceil`** on that page. You might find it easiest to take advantage of the search capabilities in your browser when looking for specifications.

For this part of the lab, create a file called **`lab11_math.py`**. Make sure to import the **`math`** module. You can now access all of the functions and global variables in math. However, they are still in the math namespace. That means that in order to use any of them, you have to put "math." before the function or variable name. To complete this part of your lab, print out the results of each of the following commands to practice using the math module.

```
math.sqrt(5)
math.sqrt(-5)
math.floor(-3.7)
math.ceil(3.7)
math.ceil(-3.7)
```

Using information in the URL above that shows all of the math functions, **write two additional lines that print pi (using math) and print the cosine of pi (using math).**

Part 6: Using the Python Standard Library (`urllib.request`)

In the last part of the lab you will experiment with some bigger strings that are supplied by a module that can read pages from the web. The module is called **`urllib.request`** and the details of the module can be found here: https://docs.python.org/3/library/urllib.request.html#module-urllib.request

We'll use just one function from this module that you need to figure out. What we want is a function that allows you to provide it with a URL (*i.e.*, a Uniform Resource Locator—the name of a web page, which usually starts with http://) and returns an object representing the web page at that URL. (The same way that using **`open()`** returns an object that represents a file.) You can then call methods of that object to get the data that you want; the simplest way is to call the read method with no arguments, which just returns the whole web page as a string.

`lab11_url.py`

```
    import urllib.request

    def main():
        # step 1: add function from urllib.request to
        #         open http://www.cnn.com

        # step 2: read in the webpage as a string

        # step 3: print the string and print the
        #         length of the webpage


    main()
```

To complete this part of the lab, you need to figure out which functions will allow you to open the URL (http://www.cnn.com) and to read in the entire contents of the webpage as a string. Then you need to print the string and the length of the string.

Part 7: Completing Your Lab

Your lab has two parts: Using the Standard Library (math) and Using the Standard Library (urllib.request).
To test your program, first enable Python 3, then run **`lab11_math.py`**. Test your code to make sure that it outputs the math examples as expected.
To test the second part of the assignment, first enable Python 3, then run **`lab11_url.py`**. Test your code to make sure that it outputs the contents of the website and the number of characters in the website.

Since this is an in-person lab, raise your hand to let your TA know that you are finished.

They will come over and check your work – they may ask you to run your program for them, and they may also want to see your code. Once they've checked your work, they'll give you a score for the lab, and you are free to leave.

IMPORTANT: If you leave the lab without the TA checking your work, you will receive a **zero** for this week's lab. Make sure you have been given a grade before you leave!

References:

Gries, D, Lee, L., Marschner, S. and White, W. (2013). CS 1110, Lab 2: Strings, Functions, And Methods. Retrieved from: http://www.cs.cornell.edu/courses/cs1110/2013sp/labs/lab02/lab02.pdf

Python Software Foundation. (2016). The Python Standard Library. Retrieved from: https://docs.python.org/3.3/library/