CS 1110, LAB 2: STRINGS, FUNCTIONS, AND METHODS

http://www.cs.cornell.edu/courses/cs1110/2013sp/labs/lab02/lab02.pdf

D. GRIES, L. LEE, S. MARSCHNER, AND W. WHITE

First Name: _____ **Last Name**: _____ **Net-ID**: _____

The purpose of this lab is to get you comfortable with using the Python functions and modules. Python has a lot of modules that come with the language, which collectively are called the Python Standard Library. The Python documentation contains the specifications for all the functions in the library. You can get to it from the link on the course website, or by going here:

http://docs.python.org/library/

However, be warned that the Python library documentation is not written for beginners. While it will make more sense later in the course, most of it will be hard to read now. The purpose of this lab is to guide you around some simpler parts of the Python library.

This lab is very similar to the previous one, in that you will be typing commands into the Python interactive shell and recording the results. All of your answers should be written down on a sheet paper (or on the sheet provided to you in lab). When you are finished, you should show your written answers to this lab to your lab instructor, who will record that you did it.

As with the previous lab, if you do not finish during the lab, you have **until the beginning of lab next week to finish it**. You should always do your best to finish during lab hours. Remember that labs are graded on effort, not correctness.

1. Strings

We talked a lot about strings last week in class. Strings have many handy methods, whose specifications can be found at the following URL:

http://docs.python.org/2/library/stdtypes.html#string-methods

(Just look at section 5.6.1 "String Methods," and don't worry that you'll encounter some unfamiliar terminology. You'll understand it all by the end of the semester!) You use these methods by writing the name of the method after the string you want to operate on, preceded by a period, like this:

```
s.index('a')          # assuming the variable s contains a string
'CS 1110'.index('1')  # nothing stops you from calling methods on a literal value
s.strip().index('a')  # or on the result of a method, when it is a value of type str
```

We will learn more about methods soon, but for now just knowing this syntax for using them will be enough. Strings are a built-in type, so although there is a module named `string`, you do not need it for basic string operations.

To reference substrings (or even individual characters) we can use the bracket notation shown in class. Remember Python counts characters starting from zero—for example, `s[1]` is the *second* character in `s`.

One of the things to remember about strings is that they are *immutable*. Methods or sequence operations return new strings; they never modify the original string object.

Evaluating string Expressions. Before starting with the table below, enter the following assignment statement into the Python shell:

```
s = "Hello World!"
```

Once you have done that, you will use the string stored in `s` to fill out the table below.

Expression	Expected Value	Calculated Value	Reason for Calculated Value
`s[2]`			
`s[15]`			
`s[1:5]`			
`s[:5]`			
`s[5:]`			
`"e" in s`			
`"x" in s`			
`s.index("e")`			
`s.index("x")`			
`s.index("l", 5)`			(optional second argument tells Python where to start looking)
`s.find("e")`			
`s.find("x")`			
`s.islower()`			
`s[1:5].islower()`			
`s.upper()`			

As we saw in the last lab, even though single and double quotes are used to delimit string literals, they also are characters that can occur in strings, just like any other character. The simplest way to get a quote character into a string is to use the other kind of quotes to delimit the string:

```
q = "Don't panic!"
```

When both kinds of quotes need to appear, we need to use the *escape sequences* we saw in class, consisting of a backslash followed by a quote:

```
q = 'The phrase, "Don\'t panic!" is frequently uttered by consultants.'
```

You could also write a string like this using double quotes as the delimiters. Write an assignment statement, using a double-quoted string literal, that assigns the same string to the variable `q1` that is already in `q`:

Test the two strings for equality and print each one using the `print` statement.

When we get a string that contains quotes, we might want to extract the substring inside those quotes. For example, starting with the string `q`, we would like to use string slicing to get the substring `Don't panic!`. Suppose that we gave you an arbitrary string `q` (not just the one above); the only thing you know about `q` is that it contains exactly two double-quote characters in it somewhere. Write a sequence of one or more assignment statements, ending by assigning to a variable `inner`, where `inner` contains the substring between the two quotes (but not including the two quote characters).

Try your sequence out on the specific value of `q` above. When you are done, the value of `inner` should be `Don't panic`. Print it with the `print` statement to remind yourself that the quotes Python is putting around its representation of the string value are not part of the string.

2. Calling functions

Python has several built-in functions (functions that do not require you to *import* a module to use them). The expressions below are just a few of them; for a complete list look at the Python documentation:

http://docs.python.org/library/functions.html

You will notice that the casting and typing operations are listed as functions. While this is true in Python, this is not always the case in other programming languages, so we treat those functions separately.

Fill in the table below just like you did in last week's lab. For each expression, we would like you to first **compute the expression in your head, without Python**. You should that down in the second column, or "?" if you have no idea. Next you should use Python to compute the expression. If the answers are different, try to explain why in the last column.

Expression	Expected Value	Calculated Value	Reason for Calculated Value
`min(25, 4)`			
`max(25, 4)`			
`min(25, max(27, 4))`			
`abs(25)`			
`abs(-25)`			
`round(25.6)`			
`round(-25.6)`			
`round(25.64, 0)`			
`round(25.64, 1)`			
`round(25.64, 2)`			
`len("Truth")`			
`len("Truth " + "is " + "best")`			

3. Module `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 π;. 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.

The module `math` has a lot of functions. Instead of listing them all here, it is best that you discover them for yourself in the documentation at the following URL:

http://docs.python.org/library/math.html

Read a bit to familiarize yourself with the page. Look at the functions in the table below and see if you can find them on that page. You might find it easiest to take advantage of the search capabilities in your browser when looking for specifications.

Evaluating `math` Expressions. To use the `math` module, you need to import it. For this part of the lab, type the following command into the Python interactive shell:

```
import math
```

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.

Fill out the table below, using the same approach that you took in the previous part of the lab.

Expression	Expected Value	Calculated Value	Reason for Calculated Value
`math.sqrt(5)`			
`math.sqrt(-5)`			
`math.floor(-3.7)`			
`math.ceil(3.7)`			
`math.ceil(-3.7)`			
`math.copysign(2,-3.7)`			
`math.trunc(3.7)`			
`math.trunc(-3.7)`			
`math.pi`			
`math.cos(math.pi)`			

In addition to the above expressions, type the following code into the Python interactive shell:

```
math.pi = 3
math.pi
```

What happens and why?

4. Scraping from the Web

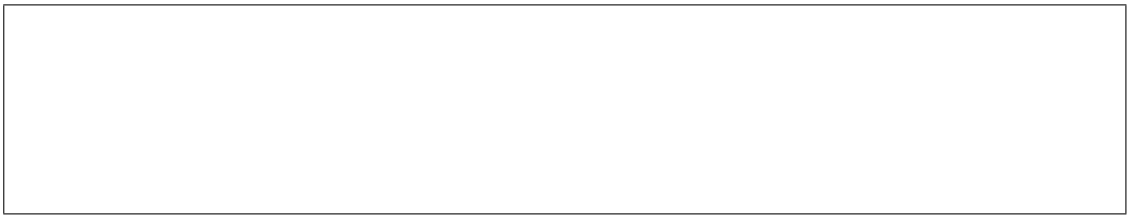
In the last part of the lab you will experiment with some bigger strings that are supplied by a module that can read pages from the web. The module is called `urllib2`, and it gives you access to a universe of data.

We'll use just one function from this module, called `urllib2.urlopen`. You give this function a URL (Uniform Resource Locator—the name of a web page, which usually starts with `http://`) and it returns an *object* representing the web page at that URL. 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. Try this, using whatever web page you like (we used the front page of CNN's site):

```
import urllib2
u = urllib2.urlopen('http://www.cnn.com')
s = u.read()
```

Look at the string `s` that you got, using the methods we looked at above. Check its length; print it out; count the number of occurrences of some words that you know, by visiting the page in your Web browser, are in the page (we used "Obama" on the CNN page and found six occurrences on Monday). Note that the basic string searching methods require exact matches, including uppercase vs. lower case (we say they are *case sensitive*), so "market" will match a headline "Asian markets up in early trading" but not "Markets up in heavy trading."

Using a word that occurs at least twice in your page, write a short sequence of statements (we call this a *code fragment*) thats extracts a substring starting 20 characters before the beginning of the second occurrence of your word and ending 20 characters after it, and assigns it to a variable `word_context`. (It is OK to assume that the word occurs more than 20 characters from the beginning of the page.)



For a more practical example, we turn to a page that may already be familiar:

http://registrar.sas.cornell.edu/Sched/PRELS.html

We as human users can look up prelim dates and locations on this page, and so can a Python program. Your task for this last part is to write some Python code to do this, but since it will be a few lines long, it is tiresome to keep typing it at the Python interactive prompt. Instead, you'll put your code in a *script* and run it from there.

A script is a Python module that you run by itself, rather than importing it using an `import` statement. To get started with your script, open Komodo Edit, select "New File" from the File menu, put a statement like `print 'Hi!'` into the file, and save it using a name ending in `.py`. You should be able to run it using the "Run Python Module" command you created when you installed

Komodo Edit following the instructions on the course web page. You should see your output appear in the "Command Output" pane at the bottom of the Komodo Edit window.[1]

Now erase your test print statement and start your script with the statements

```
import urllib2
course_name = 'CS 1110'
```

Fill in the rest of the script with a series of statements that fetches the Registrar's prelim schedule page, finds the index of the string stored in the variable `course_name`, extracts the date and location that appear on the same line on the web page, and prints out the information. At this stage of the class, don't worry about what happens if the class does not exist, does not have a prelim scheduled, there is no web connection, etc.

With the script written this way, you can just change the class name at the top and re-run it to look up the prelims for your other courses. To check off this part of the lab, demo your script to the course staff by looking up a couple of different courses.

Hints:

- You can use the `.strip()` method to remove any extra whitespace from the strings you collect.
- The longest course name is 10 characters long.
- The prelim location always starts a fixed number of characters from the start of the course name.
- The line ends at the first newline (`'\n'`) character that occurs after the index where you find the course name.

[1] You can also run a script from the command prompt by ensuring the script is in the working directory and typing `python` *script-name*`.py`. For more on the command prompt see http://www.cs.cornell.edu/Courses/cs1110/2013sp/materials/command.php