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

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

First Name: _____ **Last Name**: _____ **NetID**: _____

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.

Getting Credit for the Lab. 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.

With that said, it is very important that you complete the last section, **Writing Your First Function**. If you find yourself running out of time in the lab, you should jump to this section. And if you are not able to finish it today, please come to consulting hours, which are Sunday–Thursday, 4:30–9:30pm.

1. Calling functions

Built-in functions are those that do not require you to *import* a module to use them. You can find a list of them in the Python documentation:

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

Note 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. That is why we treated those functions specially in the last lab.

The table on the next page uses several built in functions. Fill out the table just like you did in last week's lab. For each expression, first **compute the expression in your head, without Python**. Write the result in the second column, or "?" if you have no idea. Next use Python to compute the expression. If the answers are different, try to explain why in the last column.

Course authors: D. Gries, L. Lee, S. Marschner, W. White

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')`			

2. Using a Python Module

One of the more important Python library modules is the `math` module. It contains essential mathematical functions like `sin` and `cos`. It also contains 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. To learn more about this module, look at its online documentation:

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

To use a module, you must *import* it. Type the following 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. For example, to access the variable `pi`, you must type `math.pi`.

Fill out the table on the next page, using the same approach as the previous table.

Expression	Expected Value	Calculated Value	Reason for Calculated Value
`math.sqrt(9)`			
`math.sqrt(-9)`			
`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?

3. Strings

We talked 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

For right now, look at section 5.6.1 "String Methods," and do not worry that about all the unfamiliar terminology. You will understand it all by the end of the semester.

Using a method is a lot like using a function. The difference is that you first start with the string to operate on, follow it with a period, and *then* use the name of the method as in a function call.

For example, the following all work in Python:

```
s.index('a')          # assuming the variable s contains a string
'CS 1110'.index('1')  # you can call methods on a literal value
s.strip().index('a')  # s.strip() returns a string, which takes a method
```

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

Before starting with the table below, enter the following statement into the Python shell:

```
s = 'Hello World!'
```

Once you have done that, use the string stored in `s` to fill out the table, just as before.

Expression	**Expected Value**	**Calculated Value**	**Reason for Calculated Value**
`s[1]`			
`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)`			
`s.find('e')`			
`s.find('x')`			
`s.lower()`			
`s.islower()`			
`s[1:5].islower()`			

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. An escape sequence consists of a backslash followed by a quote:

```
q1 = 'The phrase, "Don\'t panic!" is frequently uttered by consultants.'
```

You could also write a string like this using double quotes as the delimiters. **Rewrite the assignment statement for `q1` above using a double-quoted string literal**:

4. Writing Your First Function

In addition to the modules provided by Python, you can make your own. To try this out, we return to our string `q1` above. As you can see, this string has double quotes inside of it. We would like to extract the substring inside those quotes (which is `"Don't panic!"`). But we want to do it in a way that is *independent* of `q1`. That means, whatever Python statements we use, they should still work if we used a different value of `q1`, provided that it still has a pair of double-quote characters somewhere.

In the box below, write a sequence of one or more assignment statements, ending by assigning to a variable `inner`. The contents of inner should be the two substring inside the double-quote characters (but not including the double quotes themselves).

To test that your statements are correct, do the following. First, type in

```
q1 = 'The phrase, "Don\'t panic!" is frequently uttered by consultants.'
```

Then type in your statements from the box above. Finally, `print` the value of `inner`. You should see `Don't panic`, without the quotes (printing always removes the quotes from a string value).

Now, try the process again with

```
q1 = 'The question "Can you help me?" is often asked in consulting hours.'
```

Type in the assignment statement above, then type in your statements from the box, and finally `print` the value of `inner`. You should see `Can you help me?`, without the quotes. If you had to modify your answer in the box for the second `q1`, you have done it incorrectly.

The statements in the box are the start of your first interesting computer program. Given a string `q1` with a value in quotes, we can always extract the substring inside the quotes. Of course, we have to retype it everytime we want to change `q1`, which can be really annoying.

To keep ourself from retyping these all the time, you are going to write a new function and put it in a *module*. To create a module, open up Komodo Edit. Select "New File" from the File menu, and save it (even though it is still empty) as `lab02.py`. Inside of this module you will write the body of a function called `first_inside_quotes()`; this function takes a string and returns the substring inside the first pair of double-quote characters.

Your function should look something like this:

```
def first_inside_quotes(q1):
    # Your assignment statements here

    return inner
```

Replace the comment with your assignment statements in the box on the previous page.

It is now time to try out your function. Navigate the command line to the folder containing this file. Start up the Python interactive shell and `import` the module. **Remember to omit the `.py` suffix when you use the import command**. Call the function

```
lab02.first_inside_quotes('The instructions say "Dry-clean only".')
```

(Remember the module prefix) What happens?

To check off this portion of the lab you should demo your function to the course staff with a few different arguments. Make sure that each argument always has a pair of double-quote characters in it. When you have completed all this, you are done with the lab.