CS 1133, LAB 1: EXPRESSIONS, VARIABLES AND MODULES

http://www.cs.cornell.edu/courses/cs1133/2013fa/labs/lab01.pdf

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

This first lab is usually designed to get you started with Python. Because of the nature of this class, many of you are already up and running, and so this lab will seem a bit old hat to you. Rest assured, we will have more interesting labs after the first week. We simply have not covered enough yet to work on more complicated problems.

The purpose of this lab is essentially a *grammar drill* to get you used to Python as a language. If you finish early, feel free to ask your instructor about anything that you might be interested in.

1. Getting Started with Python

Most of you have installed Python on your laptops by now. Therefore, you know that you to open up the command shell on your computer (Terminal on OS X, Command Prompt) on Windows, and type `python`. Depending on your OS, you should see output that looks something like this:

```
ActivePython 2.7.2.5 (ActiveState Software Inc.) based on
Python 2.7.2 (default, Jun 24 2011, 12:22:14)
[MSC v. 1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

To make Python evaluate an expression, type the expression in after the `>>>`. When you hit "return", Python will give you the evaluation, and then show with another `>>>` prompt.

1.1. Useful Shortcuts.

If you press the *up-arrow key*, you obtain the previous expression that you typed in. Pressing up-arrow repeatedly scrolls through all the expressions that you have typed earlier in the session; if you go too far back, you can press the *down-arrow key* to get to a later expression. The *left and right-arrow keys* move the text cursor on a single line. Using all of these keys together, you can take a previously-used expression, modify it, and try it again.

2. Lab Instructions

The following pages have a list of expressions. For each expression, first *compute the expression in your head, without Python.* Write down what you think the value is in the second column of the table. If you have no idea, write "?".

Next, use Python to compute the same expression. You may find it easier to cut-and-paste from the online version of these instructions; a clickable link is just under the title of this document. Write down Python's result in the third column. **You should always fill in the second and**

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

third column of a row before moving on to the next row. You want to learn from earlier examples before moving on to the next one.

If the two values are different, you should try to figure out why Python gave the answer that it did. Come up with a reasonable explanation and put it in the final column. You are not really being graded on complete correctness in these labs (see below) so make your best guess at what is happening; your answer will help the staff understand how to better aid you.

2.1. **Getting Credit for the Lab.** We have decided that labs are required after all; it gives us a quick way to make sure that everyone is following the material. Therefore, we promise that they will not be so boring after this first week.

With that said, labs graded on effort, not correctness. When you are finished, you should show your written answers to this lab to your lab instructor. The instructor will ask you a few questions to make sure you understand the material and then record your success. This physical piece of paper is yours to keep.

If you do not finish during the lab, you have **until the beginning of lab next week to finish it**. You should show it to your lab instructor at the *beginning* of that next lab.

3. Expressions

Expression	**Expected Value**	**Calculated Value**	**Reason for Calculated Value**
81.85 - 60.0			
-4 - -4 - -4			
2 ** 2 ** 0			
(2 ** 2) ** 0			
6 / 4			
6 / 4.0			
9.0 ** 0.5			
7 % 2			
6.2 % 4			

Expression	Expected Value	Calculated Value	Reason for Calculated Value
$3 < 5$ and $5 < 3$			
`True`			
`true`			
`not False and True`			
`not (False or True)`			
`True and False and True`			
`True or (False and True)`			
$(5 / 0 == 1)$ `and False`			
`False and` $(5 / 0 == 1)$			

Why does the last expression in the table above "work" but the one above it doesn't?

4. Variables and Assignment Statements

The last part of this lab involves assignment statements. You need to know the difference between expressions, which you've been working with so far, and assignment statements. An assignment statement like

$$b = 3 < 5$$

is a command to do something. In particular, this command

(1) evaluates the expression on the right-hand side of the = (in this case, $3 < 5$), and

(2) stores its value in the variable on the left-hand side of the =, in this case, b.

Because it is not an expression, Python will not actually output a result when you type it in; it will just perform the command silently.

In the table below, the first column contains *either* an expression or a command. If it is an expression, write the value. If it is a command, you should just write "None" (we have done the

first one for you). Because some of the entries are commands, it is important that you *enter the expressions or commands in exactly the order they are given.*

Statement or Expression	Expected Value	Calculated Value	Reason for Calculated Value
i = 2	None		
i			
j			
j = 1			
j			
j = j + i			
j			
i			
w = 'Hello'			
i + w			

5. Strings

We talked about strings 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

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:]`			
`len(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()`			

6. Using a the `math` Module

As we saw in class, the `math` module. contains essential mathematical functions like `sin` and `cos`. It also contains mathematical constants such as π. 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 below, using the same approach as before.

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?