CS 1110, LAB 2: ASSIGNMENTS AND STRINGS

http://www.cs.cornell.edu/courses/cs1110/2014fa/labs/lab02.pdf

First Name: ____________ **Last Name**: ____________ **NetID**: __________

The purpose of this lab is to get you comfortable with using assignment statements, strings, and their associate methods. These are all a major part of the first assignment. The purpose of the early labs is to get you ready for that assignment.

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. At each step you should first **compute the result in your head, without Python**. Write the result in the second column, or "?" if you have no idea. Next type the expression or command into Python and record the result in the third column. If the answers are different, try to explain why in the last column.

Unlike the last lab, there are no files to download this time.

Getting Credit for the Lab. All of your answers should be written down on a sheet paper (or on the sheet provided to you in lab). There are no files to download or turn in. When you are finished, you should show your written answers to your lab instructor, who will record that you did it.

This (and the lab that follows it) are two of the longest labs. There is a lot of work getting you ready for the first assignment. It is very possible that you will not finish during class time. If you do not finish, you have **until the beginning of lab next week** to finish it. You can turn in the lab during consulting hours, or at the beginning of lab. Remember that labs are graded on effort, not correctness.

1. Variables and Assignment Statements

As we saw in class, assignment statements are different from expressions. A 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. It is important that you understand the difference.

In the table on the next page, 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.*

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

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			

2. String Expressions

Throughout this section, pay close attention to spaces and to the different types of quotation marks being used. We use both ' (single quote) and " (double quote).

Expression	**Expected Value**	**Calculated Value**	**Reason for Calculated Value**
'Truth ' + 'is ' + 'best'			
'Truth' + 'is' + 'best'			
"Truth " + "is " + "best"			
"Truth " + ('is ' + "best")			
'A double quote: "'			
"A single quote: '"			
'A single quote: ''			

Expression	Expected Value	Calculated Value	Reason for Calculated Value
'' + 'ok'			
'' + '4 / 2'			
'' + 4 / 2			
'' + `str`(4 / 2)			

3. 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.

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')`			

4. 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 variables for 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 variables in math. However, 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`. Keep this in mind as you fill out that table below.

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?

5. String Methods

Now that you understand strings and (calling) functions, you can put the two together. 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
```

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')`			

5.1. **Extracting Internal Strings.** 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**:

For your last exercise, we want you to write Python code to *extract* the substring inside the double quotes (which is `"Don't panic!"`). But we want to do it in a way that is *independent* of `q1`. That means, even if you change the contents of `q1`, your answer should still work, provided that `q1` still has a pair of double-quote characters somewhere.

In the box below, write a sequence of one or more assignment statements, ending with an assignment to a variable called `inner` (the other variables can be named whatever you want). The assignment statements should use string slicing to remove the unwanted parts of `q1`. When you are done, the contents of inner should be the 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.