CS 1110, LAB 2: FUNCTIONS AND OBJECTS

Name: _____ **Net-ID**: _____

There is an online version of these instructions at

http://www.cs.cornell.edu/courses/cs1110/2012fa/labs/lab2

You may wish to use that version of the instructions.

The purpose of this lab is to get you comfortable with using the Python functions and modules. Python has a lot of built-in modules, which collectively are called the Python library. This library comes with an API (Application Programming Interface), which is a list of *specifications* telling us how to use the library; we will learn more about specifications later.

If you want, you can access the Python library API from the course website, or by going here:

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

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

Requirements For This Lab. The very first thing that you should do in this lab is to download the file `window.py` from the course web page:

http://www.cs.cornell.edu/courses/cs1110/2012fa/labs/lab2/window.py

This is a module that provides some GUI functionality; it will be used at the end of the lab. You should put it in a new directory. When you start the Python interactive shell, make sure that you are in the same directory as this module file.

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. Function Calls

Python has several built-in functions (e.g. 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")			

---

2. Module `math`

One of the more important modules in Python is the `math` module. This contains a lot of advanced 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 Python API specification 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.

2.1. **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?

3. Strings

We talked a lot about strings in the last class. Strings are objects. They do not have any attributes, but they have methods. The specifications for string methods can be found at the following URL:

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

In addition, strings are sequences. That means that we can reference substrings (or even individual characters) using the bracket notation shown in class. For example, `s[1]` is the *second* character in `s`.

In order to be consistent with other Python sequence types, the length of a string is determined not by a method but by a function called `len`, which we saw at the beginning of this lab. This is different from most object oriented languages (such as Java), where you would call a method to determine the length.

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.

3.1. **Evaluating string Expressions.**

Strings are a built-in type; there is no module to import for them.

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

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

Once you have done that, you will use the object whose name is 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:]			
s.find("e")			
s.find("x")			
s.islower()			
s[1:5].islower()			
s.upper()			
s.isupper()			

¡p¿ As we saw in the last lab, we use backquotes (`) to turn anything that is not a string into a string. Suppose `x` is a variable that contains an int. Write an expression that produces the string `"The value of x is "` followed by the value of `x`. So if `x` is 2, the expression produces the string `"The value of x is 2"`.

` Another thing we saw in the last lab is that strings can contain quotes as characters. To do that, we put a slash before the quote (
"). For example consider the following string `q`:

```
q = "The phrase \"Hello World!\" is inside"
```

Alternatively, we could put the string in single quotes to distinguish them from double quotes.

```
q = 'The phrase "Hello World!" is inside'
```

But we find this confusing. It is best to use " when you want to start or end a string, and
" when you want to use a quote as a character.

When we get a string like that, we often want to extract the substring inside the inner quotes. For example, starting with the string `q`, we would like to use string slicing to get the substring `"Hello World!"`. Suppose that we gave you an arbitrary string `q` (not just the one above); the only thing you know about `q` is that it has a pair of quotes inside of it. Write a sequence of one or more assignment statements, ending in 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 `"Hello World!"`.

4. Windows

In the last part of the lab you will experiment with a simple graphical user interface (GUI) object. There are many competing GUI APIs for Python, and none of them are accessible to someone just starting Python. Therefore, we have provided you with the module `window`, which provides you with a very simple GUI type, called Window.

All of the types that we have seen so far have natural ways to represent their values. We represent ints by whole numbers; we represent strings as characters inside double quotes. But some types, such as Window, do not have natural ways to represent them. For these types, we have to use a special functon – called a constructor – to create a new object of that type.

The name of a constructor function is generally the same name as the type, which in this case is Window. Import the `window` module and type the assignment statement

```
w = window.Window()
```

See what happens? The function `Window()` (which is in namespace `window`) creates a new Window object and displays it on your screen. The variable `w` now holds the "folder name" of this Window object.

4.1. **Window Attributes.** As we discussed in class, attributes are named variables that are stored inside an object (thinking of an object as a manilla folder). You can use attributes in expressions, or even assignment statements.

One of the interesting thing about GUI objects is that assigning new values to an attribute can have visible effects. In the table below we have a list of expressions and assignment statements. **Enter these into the Python shell in exactly the order presented**. If it is an expression, give (or guess) the value that Python returns. If it is an assignment statement explain (or guess) the result of the assignment.

Statement or Expression	**Expected Result**	**Actual Result**	**Reason for Actual Result**
w.x			
w.x = 100			
w.y			
w.y = 100			
w.width = 10			
w.height			
w.title			
w.title = "window"			

4.2. **Window Methods.** Window objects also have methods. Unlike string methods, which are functions, these methods are procedures that do something to the object. Execute calls for the three methods shown in the table below. Explain what happens when you call them.

Method	**Result When Called**
w.beep()	
w.iconify()	
w.deiconify()	

4.3. **Positioning a Window.** You have already seen the Window size and position is controlled by attributes. For the Window whose name is in `w`, look at the attributes for the x-coordinate and y-coordinate. Write their values here:

Next, create a second Window object, storing its name in another variable. This should pop up a new window. What are the x-coordinate and y-coordinate for this window?

Is there something unusual about how screen coordinates work? What do you notice about the difference in coordinates between the two windows?

4.4. **Resizing a Window.** Create a new Window, storing its name in variable `w`. Try resizing the Window with your mouse to make it bigger. Look at the attribute function `resizable` of `w` to see whether the Window is resizable. What is the value of this attribute?

Now execute the assignment

```
w.resizable = False
```

Try resizing the Window whose name is in `w` with your mouse. Is it resizeable now?

Assign the attribute `resizable` to True. Once you have done that, call the procedure

```
w.setMaxSize(50,100)
```

What happened to your Window?

What happens when you try to resize this Window?