CS 1110, LAB 1: GETTING STARTED

http://www.cs.cornell.edu/courses/cs1110/2016fa/labs/lab01/

First Name: ____________________ **Last Name**: ____________________ **NetID**: ______________

This lab serves two purposes. First, it is designed to get you started with Python immediately, particularly with the command shell. Second it gives you hands on experience with Python expressions, which we talked on the first day of class. Learning a computer language is a lot like learning a new human language, and this lab essentially works as a *grammar drill.*

Lab Materials. This handout is not the only part of this lab. In addition, we have two files for you to play with. Lab files are always located on the Labs section of the course web page:

http://www.cs.cornell.edu/courses/cs1110/2016fa/labs

For today's lab you will notice two files.

- `hello1.py` (a text-based Python script)
- `hello2.py` (a graphical Python script)

Create a *new* directory on your hard drive and call it `lab01`. **Put this directory on your Desktop**. In future labs, you can put the folder wherever you want, but this lab is a lot easier if this folder is in your Desktop folder. Download all of the files to that directory.

Getting Credit for the Lab. There are two ways to get credit for any lab this semester.

The first is to simply use this lab handout. The handout has several empty boxes that prompt you to answer a question. As part of the lab, you are to write the answers to these questions inside the boxes. When you are finished, you should show your written answers to your lab instructor.

As an alternative, we are testing a new online system this semester. You can access the online system from the web page for this lab.

http://www.cs.cornell.edu/courses/cs1110/2016fa/labs/lab01/

Labs are graded on effort, not correctness. However, the online page will only give you credit if you are completely correct (and this is why **the online version is slightly different**). If you want more flexibility, you should use this worksheet and not the online version.

If you are using the worksheet, you should write your answers on the handout and show it your 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, you have **until the beginning of next lab next week** to work on this lab. Over the next week, you may either (1) complete the lab online, (2) show your lab to a consultant during consulting hours, or (3) show your lab to an instructor at the *beginning* of the next lab session.

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

1. Getting Started with Python

If your primary computer is a laptop, bring it to the lab to work on, as lab is an *excellent* opportunity to install Python on your machine. Follow the instructions on the course website:

http://www.cs.cornell.edu/courses/cs1110/2016fa/materials/python.php

Ask a consultant for help if you have problems with your installation; that is why they are there. However, you do not need to spend lab time installing Python on your machine. If you want get started with Python now and install it later, you are welcome to work on a machine in the lab.

Make sure that you have created the folder `lab01` on your Desktop with the two files for this lab: `hello1.py` and `hello2.py`. You should not have any other files in this folder.

1.1. Working with the Command Shell.

Start the command shell for your computer. On Windows, this is called the *Command Prompt* on Windows (Found in `Accessories` in the Start Menu on Windows 7, or `Apps > Windows System` in Windows 10), or the *Terminal* on OS X (Found in `Utilities` in the `Applications` folder). This program works like a normal Window, allowing you to manipulate files and folders. However, it uses typed text commands instead of a mouse.

Like a normal Window, the command shell looks at a specific folder. We call this folder the *active directory*. You can use the command shell to list, copy, and move files in the active directory. You can also change the active directory, just like you can change the current folder of a Window.

When the command shell starts, the active directory is your *home directory*. In both Windows and OS X, this is the folder with your name on it. Open up a Window for your home directory, and put it next to the command shell. Go back to the command shell. If you are using Windows, type

```
dir
```

and hit the **Return** key. If you are using OS X, type

```
ls -l
```

(that is a "dash ell") and hit **Return**. In a few words, describe what you see and how it compares to the Window next to the command shell.

Now let us move to the Desktop. This is actually a folder stored inside your home directory. To get there from the home directory, type the command (on either OS)

```
cd Desktop
```

and hit **Return**.

Once again enter either `dir` or `ls -l` (depending on your operating system) into the command shell. In a few words, explain what is different this time and why.

We want you to change the active directory to the folder `lab01`. If you followed the instructions above, this folder should be on your Desktop. Now type

```
cd lab01
```

and hit **Return**. One last time, type either `dir` or `ls -l` (depending on your operating system) into the command shell and tell us what you see.

To go back to the previous directory, you can use the command "`cd ..`" This should be enough to familiarize you with the command shell for now. If you want to learn more, you should read the more detailed introduction on the course web page:

http://www.cs.cornell.edu/courses/cs1110/2016fa/materials/command.php

1.2. **Working with Python.** There are two ways to use Python. One is to execute "scripts", or files that contain Python commands. The other is to use Python *interactively*, typing in just one command at a time.

The files `hello1.py` and `hello2.py` are both scripts. To run a script, you type `python`, followed by the name of the script, into the command shell. Type

```
python hello1.py
```

and hit **Return**. What do you see happen?

Now open up the file `hello1.py` with Komodo Edit. It is a file with just one line in it:

```
#print 'Hello World!'
```

Delete the `#` character and save the file. Once again, run the script `hello1.py` by typing "`python hello1.py`". What you do see this time?

Finally, run the script `hello2.py`. What happens this time?

To run Python interactively, type the word `python` by itself and hit **Return**. Do that now. You should see several lines of text followed by the symbol `>>>`. This is the *Python prompt.* Like the command shell, it responds to commands that you type. The purpose of the `>>>` is to let you know that you are currently running Python, and that you are no longer working with files and folders.

At the Python prompt, type

```
>>> 1+1
```

and hit **Return** (do not type the `>>>`). What happens?

2. Python Expressions

For the remainder of this lab, you will use Python in interactive mode. The following pages have two sets of tables. The first table challenges you to evaluate an expression. The second table asks you to "fill in the blank", completing an expression to give you the provided value.

In the first table, we first ask that you *compute the expression in your head, without Python.* This activity helps you understand where you might be confused, and can help you study in the future. Write your guess in the second column; if you have no idea, write "?". For the third column, use Python to compute the same expression. If the two values are different, try to understand why before moving on to the next expression.

The second table is looks very similar to the first table, but this time we have given you both the expression and the value. However, the expression is missing a literal, as indicated by the box. In the third column you are to fill in the missing literal. Remember that a literal is an expression that looks like a value. So 12 and `True` are literals, while (10+2) is not.

You may find it helpful to work on both tables simultaneously. The tables do "line up". If you understand a row in the first table, then you have enough information to answer the same row on the second table.

This lab is just supposed to be practice. Do not waste too much time trying to figure things out yourself. If you do not understand something, ask a staff member immediately.

2.1. Useful Shortcuts.

If you press the *up-arrow key*, you get the previous expression that you typed in. Pressing up-arrow repeatedly scrolls through all the expressions that you have typed this 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.

2.2. **int and float Expressions.**

The answers to all of these questions are numbers. However, they may be either ints or floats. Remember, a *literal* is just an expression that is also a value. When we want a number that is part of an expression, we always call it a literal, not a value. It only becomes a value when the expression is evaluated.

Expression	**Expected Value**	**Calculated Value**
2 * 3		
2 ** 3		
5 + 2 * 5		
(5 + 2) * 5		
-4 - -4 - -4		
2 ** 2 ** 0		
(2 ** 2) ** 0		
6 / 2		
6 / 4		
6.0 / 4		
6 / 4.0		
9.0 * 0.5		
9.0 ** 0.5		
6 % 2		
7 % 2		
6.2 % 4		

Expression	**Calculated Value**	**Literal in the Box**
□ * 3	12	
2 ** □	8	
5 + 2 * □	13	
(5 + 2) * □	21	
-4 -□- -4	8	
□ ** 2 ** 0	5	
(2 ** 2) ** □	4	
6 / □	2	
(□ +1) / □	1	
□ / 5	0.2	
3 / □	0.6	
□ * 0.25	2.25	
□ ** 0.5	5.0	
□ % 4	0	
□ % 4	2	
□ % 5	1.5	

2.3. **Comparisons and bool Expressions.**

The answers to all of these questions are either numbers, `True`, `False`, or `Error`. You should answer the last one in the case where Python crashes when it tries to evaluate the expression. Remember, a *literal* is just an expression that is also a value. When we want a boolean that is part of an expression, we always call it a literal, not a value. It only becomes a value when the expression is evaluated.

Expression	**Expected Value**	**Calculated Value**
3 < 5		
3 < 5 `and` 5 < 3		
`True`		
`true`		
`True and False`		
`True or False`		
`not True`		
`not not False`		
`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)		

Expression	**Calculated Value**	**Literal in the Box**
3 != ☐	`False`	
4 < ☐ `or` 5 < 3	`True`	
☐	`False`	
☐	`'true'`	
☐ `and True`	`True`	
`False or` ☐	`False`	
`not` ☐	`False`	
`not not not` ☐	`True`	
`not False and` ☐	`False`	
`not (` ☐ `and True)`	`True`	
`False or False or` ☐	`True`	
`True and (3 <` ☐ `)`	`True`	
(3 < ☐) `or` (5/0==1)	`True`	
(3 < ☐) `or` (5/0==1)	`False`	

2.4. **Types and Casting.**

The answers to all of these questions are either numbers or types. Python represents a type as a string inside angle brackets, like this: ⟨type `'int'`⟩. When we ask for a literal, it may either be a number literal or a type literal.

Expression	**Expected Value**	**Calculated Value**
`float`(4)		
`int`(4)		
`int`(5.3)		
`float`(`int`(5.3))		
`int`(-5.7)		
`float`(7) / 4		
`float`(7 / 4)		
`type`(4)		

Expression	**Calculated Value**	**Literal in the Box**
`float`($\boxed{}$)	1.0	
`float`($\boxed{}$)	1.5	
`int`($\boxed{}$ +0.5)	2	
`int`(`float`($\boxed{}$))	6	
`int`($\boxed{}$ -0.5)	-3	
7 / `float`($\boxed{}$)	1.75	
`float`(7 / $\boxed{}$)	1.75	
`type`(7 / $\boxed{}$)	⟨type `'float'`⟩	