CS 1133, LAB 1: GETTING STARTED

http://www.cs.cornell.edu/courses/cs1133/2017fa/labs/lab1/

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*.

1. Preparation

For today's lab you will need two files, located on the **Labs** section of the course web page.

- `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.

Setting up Python. If your primary computer is a laptop, bring it to the lab, 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/cs1133/2017fa/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.

Getting Credit for the Lab. 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 next Tuesday to finish it**. If you cannot finish this during lab, you should turn it in during consulting hours instead. We do not have time to check everyone off during lab.

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

2. 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/cs1133/2017fa/materials/python.php

Ask a consultant for help if you have problems with your installation; that is why they are there.

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.

2.1. **Working with the Command Shell.** Before you start, make sure that you have created a folder called `lab01` on the Desktop and placed the files `hello1.py` and `hello2.py` in that folder. If you do not do that, your answers will be wrong.

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.

Start the command shell for your computer. On Mac OS X, this is called the *Terminal*. You can find it inside `Utilities` in the `Applications` folder. If you are using Linux, the `X terminal` program does the same thing (which you should know how to use if you are running Linux).

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. Copy the first few lines of what you see into the box below.

What do you think this text means?

Now we are going to move to the Desktop. This is actually a folder stored inside your home directory. To get there from the home directory, type the command

```
cd Desktop
```

and hit **Return**. Once again enter either `dir` or `ls -l` (depending on your operating system) into the command shell and copy the first few lines of what you see below.

Why are the results different this time?

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 what you see below.

To go back to the previous directory, you can use the command `"cd .."`. This should be enough to familiarize you with the Terminal 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/cs1133/2017fa/materials/command.php

3. 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. In this part of the lab, you will get started with both.

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 `hello1.py` by typing `python hello1.py`.

What you do see this time?

Finally, run the script `hello2.py`.

What happens this time?

Interactive Mode. To run Python interactively, type the word `python` by itself and hit **Return**. 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 this last time?

Useful Shortcuts. You will use interactive mode for the rest of this lab. There are some important shortcuts that you might like to use. 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.

4. Int and Float Expressions

Below are two sets of tables. The first table challenges you to evaluate an expression. Use Python to evaluate the expression and type the result into the box. It is possible that the expression may produce an error. In that case, simply write `error` in the box.

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.

Expression	**Calculated**
`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 / 4.0`	
`2.0 // 2.5`	
`9.0 * 0.5`	
`9.0 ** 0.5`	
`6 % 2`	
`8 % 3`	
`6.2 % 4`	

Expression	**Calculated**	**Missing**
`□ * 3`	12	
`2 ** □`	16	
`5 + 2*□`	13	
`(5 + 2) * □`	21	
`-4 -□ - -4`	8	
`□ ** 2 ** 0`	5	
`(2 ** 2)**□`	4	
`6 // □`	2	
`(□+1)//□`	2	
`□ / 5.0`	0.2	
`5.5 // □`	2.0	
`□ * 0.25`	2.25	
`□ ** 0.5`	5.0	
`□ % 4`	0	
`□ % 4`	2	
`□ % 5`	1.5	

5. Comparisons and Bool Expressions

Once again, you have two sets of tables. In the first table, evaluate the expression. It is possible that the expression may produce an error. In that case, simply write `error` in the box. For the second table, you are supposed to provide the missing literal.

Expression	Calculated
`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	Missing
`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 <` ☐ `) and (5/0==1)`	`False`	

You should have noticed something very weird in the last expression in the table on the left. What is happening here?

6. Built-In Functions

Once again, you have two sets of tables. In the first table, evaluate the expression. It is possible that the expression may produce an error. In that case, simply write `error` in the box. For the second table, you are supposed to provide the missing literal.

Expression	Calculated
`min(25, 4)`	
`max(25, 4)`	
`min(5,max(7,4))`	
`abs(25)`	
`abs(-25)`	
`round(25.6)`	
`round(-25.6)`	
`round(25.64, 0)`	
`round(25.64, 1)`	
`round(25.64, 2)`	
`len('Hello')`	
`chr(65)`	
`chr(66)`	
`ord('A')`	
`ord('AB')`	

Expression	Calculated	Missing
`min(` ☐ `,4)`	2	
`max(` ☐ `,4)`	6	
`min(4,max(` ☐ `,2))`	3	
`abs(` ☐ `)+` ☐	4	
`abs(` ☐ `)-` ☐	4	
`round(` ☐ `+0.4)`	2.0	
`round(` ☐ `+0.4)`	-2.0	
`round(5.18,` ☐ `)`	5.0	
`round(5.18,` ☐ `)`	5.2	
`round(5.18,` ☐ `)`	10.0	
`len(` ☐ `)`	2	
`len(` ☐ `)`	0	
`chr(` ☐ `)`	`'C'`	
`ord(` ☐ `)`	66	
`ord(chr(` ☐ `))`	121	