CS 1110 SPRING 2016: GETTING STARTED (Jan 27-28)

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

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

In retrospect:

- many students read "ls" in "tt" font as "one-ess" and figured we wanted them to see a "command not found error ...". Solutions below have been corrected.
- It's harder now with newer OSs to find one's home directory. Help students do this, and perhaps (for Mac) help the students add their home directory to the Finder.
- Students were confused by the word "directory", until I realized they knew them as "folders".

Goals. Learning a computer language is a lot like learning a new human language, and this lab essentially works as a *grammar drill*. Lab goals: (1) get you started with Python immediately, particularly with the command shell; (2) give you hands-on experience with Python types and errors.

Getting credit for this lab. This lab 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 one of the CS 1110 lab staff. The staff member will ask you a few questions to make sure you understand the material, and then swipe your Cornell ID card to record your success. This physical piece of paper is yours to keep.

If you don't finish, you have until *the beginning of your lab next week*: you may either show your lab to a consultant during consulting hours, or to a lab staff member at the beginning of your lab section on Feb 3-4.

What computer to use? If your primary computer is a laptop, bring it to the lab to work on, as lab is an *excellent* opportunity to get started with Python on your machine. You should follow the instructions on the course website. Ask a staff member for help if you have problems with your installation.

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 should use one of the machines in the lab.

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

Lab Materials. Lab materials can always be found at

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

For today's lab, you need this handout (which is also online), plus the file `hello1.py`, which is a text-based 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 a lab's materials in whatever folder you want, but this lab is a lot easier if this folder is in your Desktop folder. Download `hello1.py` into that directory.

1. Files, Directories, Running Python, and Editing

1.1. **Working with the Command Shell.** Start the command shell for your computer. This is the Command Prompt on Windows (use the Start Menu to get to Search and type in "cmd"), or the Terminal on OS X (located in `Utilities` in the `Applications` folder). This program works like a Window in your operating system, allowing you to manipulate files and folders. However, it uses typed text commands instead of a mouse.

Like a Window in your operating system, 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 of the command shell, just like you can change the current folder of a Window. When the command shell starts out, the active directory is your *home directory*. In both Windows and OS X, this is the folder with your login name on it.

Open up a Window for your home directory using either File Explorer (Windows) or Finder (Mac), 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 "ell, ess, space, hyphen, ell") and hit the **Return** key. In a few words, describe what you see and how it compares to the Window next to the command shell.

They should NOT see an error message here. If you see an error message here, tell the student that that they have (probably) mistaken ell's for the number one.

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. Your Desktop is actually a folder stored inside your home directory. So the first thing to do is to move to the Desktop. 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.

Since `lab01` is a folder inside the Desktop, you can now type

```
cd lab01
```

and hit **Return**. Your active directory should now be `lab01`. One last time, 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, visit

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

1.2. **Working with Python.** There are two ways to run 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 file `hello1.py` is a script. 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**. Does anything happen?

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

```
print 'Hello World!'
print 'Welcome to CS1110!'
```

Add the `#` character at the beginning of the second line, so the file now contains

```
print 'Hello World!'
#print 'Welcome to CS1110!'
```

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

If students call the character a "hashtag", you might want to tell them that it's a "hashmark" (or pound sign, or octothorpe). It's definitely not a hashtag.

(Incidentally, what you just did is "comment out" a line of the script: the `#` character tells Python to ignore everything on the rest of the line.)

To run Python *interactively*, type the word `python` by itself and hit **Return**. Do that now. You should see several lines of text forming a "banner" describing Python, followed 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. Using the Interactive Mode to Learn Some Basics

In is section, you use Python in interactive mode. Proceed **row-wise** through the tables below. For each example (row), write down what you think the result should be. It's OK to say "I don't know". Then use Python in interactive mode to find out if your mental reasoning is OK or not. You may want to cut-and-paste from the online version of this handout.

If what you see doesn't match what you expected, then you are having a learning moment! Try to figure out why Python gave the answer that it did. Ask a staff member for clarification if necessary. Leave a paper trail of your mistakes and recoveries in the "notes column" as a reminder not to make the same mistakes in the future!

The labs will often be like this. You are acquainted with a Python construction in lecture. You then nail down the rules associated with that construction in the lab by working examples two ways: mentally and then with the computer.

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. **Integer and Float Arithmetic.**

	Expression	**I Think the Value Is**	**Python Says**
1	`10/4 2:  int vs float`		
2	`10./4`		
3	`float(10/4) 2.0:  order of operations`		
4	`1.0 + 10/4 ASK: why a decimal (float) result?`		
5	`1 + 10/4.  ASK: why different?`		
6	`int(10/4)`		
7	`int(3.9) ASK: does int round up?  (nope)`		
8	`float(int(10.)/4)`		
9	`int(float(10/4))`		
10	`1.0/3.0`		

	Expression	I Think th Value Is
11	`10 % 4` ASK: what operation is '%'?. If they can't remember from class, have them experiment: 8 % 2, 9 % 2, 10 % 2	
12	`4 % 10`	
13	`9**2`	
14	`9**.5`	
15	`9**(1/2)` ASK: why different? (b/c 1/2 is 0)	
16	`4**2**0` ASK: why not 1? (right associative)	

2.3. **Strings.** Reproduce this example of string slicing by entering the first two commands

```
>>> s = 'abcdef'
>>> s[0:3]
'abc'
```

The very astute student might notice that here, there are single-quotes around the string, whereas for the script which used print statements, there are no quotes around the output. I guess we just don't want to get into the difference between evaluating an expression and printing a string right now. Now look at these other examples.

	Expression	I Think the Value Is	Python Says	Notes
1	`s[1]`			
2	`s[0]`			
3	`s[1:2]`			
4	`s[0:2]`			
5	`s[:3]`			
6	`s[3:]`			

2.4. **Strings and Numbers.** Reproduce this "dialog": ASK: why is s2+s2 NOT 24? (+: string concat)

```
>>> s1 = '12.34'
>>> 2*float(s1)
24.68
>>> 2*int(s1[0:2])
24
>>> s2 = str(12)
>>> s2+s2
'1212'
>>> s3 = str(12.34)
```

Now look at these other examples

	Expression	**I Think Value**
1	`s2+s2+s2`	
2	`float(s1[2:4])` 0.3	
3	`s3[2]`	
4	`s3+s3` ASK: why not '..'? (because previous expression didn't change s3)	

2.5. **Errors.** If you do something illegal, Python complains. Roughly, it tries to tell you the type of error and where it occurs. In time, you will learn how to decipher error messages. For now, let's ignore the "where it occurs" part and interpret the rest of the message.

1. If I enter

```
>>> 10*(5+6) = x
```

then here is what Python tells me and why it is complaining:

```
File "<stdin>", line 1
SyntaxError:  can't assign to operator
```

In an assignment statement, the expression has to be to the right of the = .

2. If I enter

```
>>> s = 'abcdef'
>>> c = s[7]
```

then here is what Python tells me and why it is complaining:

3. If I enter

```
>>> s = 'abc'
>>> t = s - s
```

then here is what Python tells me and why it is complaining:

4. If I enter

```
>>> s = 'abc'
>>> t = s(2)
```

then here is what Python tells me and why it is complaining: (write small!)

TypeError: 'str' object is not callable.
Students do NOT have to know what a callable object is. But they should be told it's a common error to type parentheses instead of brackets (and vice versa)