CS 1110 SPRING 2016: LAB 02: PRACTICE FOR A1 (Feb 2-3)

http://www.cs.cornell.edu/courses/cs1110/2016sp/labs/lab02/lab02.pdf

First Name: ________________ **Last Name**: ________________ **NetID**: ____________

The lab assignments are very important. *The lab problems feed into the assignments and the assignments define what the exams are all about.*

Getting credit for this lab. Complete all blank boxes and lines on this handout. When you are finished, you should show your written answers to one of the CS 1110 lab staff in your section. 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.

For this lab (02) only: since the material here is very important for your understanding of A1, and since A1 is due Friday the 12th, and since there are no labs next week: finish what you can during the lab section and we will check you off during this period. *If you still have questions on this lab after your lab section, definitely seek out help at office hours or on Piazza!!*

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 various python files included in the zipped `Lab_2.zip`.

Note that students might have trouble finding where they put things; Proactively intervene if you see students seeming to spend a lot of time looking at their directory structure.

Download `Lab_2.zip` and unzip it. This should create a new directory (folder) with some Python files we've created for you.

Open a command shell and use the `cd` commands you learned about last time to move in the command shell to the directory where you put the python files from `Lab_2.zip`. *Ask a staff member immediately if you need help with this!*

1. Print statements for checking that variables have the expected values

Here is a slightly different version of the hyphenation program you saw in class.

```
# Hyphenator_even.py
# the CS 1110 profs (cs-1110profs-L@cornell.edu)
# Feb 2016

""" Inserts hyphens into a non-empty even-length input string as follows:
The hyphen splits the first and second halves.
"""
```

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

```
## We've added a number of so-called "debugging" print statements here

s = input('Enter an even-length string (remember to put quotes around it): ')
n = len(s)
print 'n is',n
if n%2 == 0:                            # Line A
    # s has even length
    m = n/2
    print 'even case. m is',m
    first = s[0:m]                      # Line B
    print 'even case. first is',first
    second = s[m:]                      # Line C
    print 'even case. second is',second
    h = first + '-' + second

# final output
print s,'becomes',h
```

Notice the inclusion certain `print` statements that are designed to show you the contents of key variables:

- `print 'n=',n`
- `print 'even case.  m=',m`
- `print 'even case.  first=',first`
- `print 'even case.  second=',second`

These are *debugging* print statements. (The final line `print s + ' becomes ' + h` is *not* a debugging print statement, but rather, one that creates the desired final result of the program.)

In the command shell, see what the program does: run the program by typing `python Hyphenator_even.py`. When asked for input, try `'abcdef'` ; the last line of the output should say that the program turns this into `'abc-def'`.

Now, let's artificially insert some errors and see how these print statements show that some variables then no longer have the "right" values in them.

Open `Hyphenator_even.py` in Komodo Edit.

Lab heads: YOU MAY WANT TO START CLASS BY WALKING THEM THROUGH Hyphenator_even.PY AND TELLING THEM THAT IT'S TRYING TO BREAK THE STRING INTO "FIRST" AND "SECOND" AND GLUE THOSE PIECES TOGETHER APPROPRIATELY.

(1). In Line B, assign `s[:m-1]` to `first` instead of `s[:m]`. Then, in the command shell, run `Hyphenator_even.py`. (That is, type `python Hyphenator_even.py` in the command shell. Don't forget to save the file in Komodo Edit first.) Give the input `'abcdef'`.

Which variable has an unexpected value, and what should the variable's value have been? Which print line tells you this?

WALK AROUND AND LOOK AT THE STUDENTS' HANDOUTS TO MAKE SURE THE STUDENTS SEE WHAT'S HAPPENING HERE BEFORE THEY GO ON. first should be "abc", not "ab". It's the printout of first that tells them this.

(2). Fix the mistake you made above. Now, introduce a new artificial mistake via Komodo Edit: in Line C, assign `s[m+1:]` to `second` instead of `s[m:]`. Suppose you run `Hyphenator_even.py` and give it input `'abcdef'`.

Which variable has an unexpected value, and what should the variable's value have been? Which print line tells you this?

first should be "abc", but second is now "ef" instead of "def". If the students see two errors, make sure that they remembered to fix the mistake they introduced in the part above.

2. Finding errors using print statements

Now, we give you practice in adding `print` statements to see if there are errors in a program. This is something we ask you to do in A1, as well. *The point: learn to use print statements to figure out if your code is working or not.*

Read the docstring for `Hyphenator_broken.py` (the comment in triple quotes) in the following program to understand what the program is supposed to do.

```
# Hyphenator_broken.py
# the CS 1110 profs (cs-1110profs-L@cornell.edu)
# Feb 2016

""" Inserts hyphens into a non-empty odd-length input string as follows:
A hyphen is inserted on either side of the middle character.

Example: "abcde" becomes "ab-c-de"

"""
### This program intentionally has at least one error in it!

s = input('Enter an odd-length string (remember to put quotes around it): ')
```

```
n = len(s)


m = n/2


first = s[0:m-1]


middle = s[m+1]


second = s[m+1:]


h = first+'-'+middle+'-'+second

# final output
print s,'becomes',h
```

Run program `Hyphenator_broken.py` and give it input `'abcde'` to see that it currently doesn't work correctly.

Open the file `Hyphenator_broken.py` in Komodo Edit. Insert enough `print` statements so that when you run the program in the command shell, you know what lines are incorrect.

Write in the blank spaces of the code above what print statements you included.

Suggest the students have print statements with appropriate documents (like, print "middle is: ", middle" instead of just "print middle")

Run your altered program in the command shell. *From the output of your print statements*, explain what are the errors in the program.

assignments to `first` and `middle` are wrong. MAKE SURE THEY COULD ACTUALLY TELL THIS FROM THEIR PRINTS.

3. Boolean Expressions

For each example in this section, write down what you think is the value of the expression. Then use Python in interactive mode to find out if your mental reasoning is OK or not. If not, 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!

3.1. **Numerical Examples.** Assume that x,y and z are initialized as follows

```
>>> x = 1
>>> y = 5
>>> z = 10
```

Now complete this table:

	Expression	**I Think the Value Is**	**Python Says**	**Notes**
1	`x < z`	`True`	`True`	
2	`2*y >= z`			
3	`2*y < z`			
4	`(x>1) or (z!=7)`			
5	`y != (z/2)`			
6	`(x>0) or ((y>0) and (z<0))`			
7	`((x>0) or (y>0)) and (z<0)`			

Recall that if you just enter an expression in Python interactive mode, then Python will display the value of that expression. Thus, something like >>> `x < z` will result in the display of either "`True`" or "`False`".

3.2. **String Examples.** Assume that x, y, and z are initialized as follows

```
>>> x = 'Cornell'
>>> y = 'Harvard'
>>> z = 'Yale'
```

Now complete this table:

	Expression	I Think the Value Is	Python Says	Notes
1	`x != z`	True	True	
2	`x == 'cornell'`			
3	`len(x) > len(y)`			
4	`y[1:] > z[1:]`			
5	`len(x+z) > len(y)`			

4. The If-Else Construction

students need to be able to "edit" the back end of a string and do number-string conversions in A1

(a) Assume that `s` is initialized with a string. Complete the following so that it prints `Plural` if `s` ends with '`es`' and `Not Plural` if it does not.

(s[len(s)-2:] == "es")

```
if ________________________________________________________________________________:

    print 'Plural'
else:
    print 'Not Plural'
```

Try this out in Python interactive mode to see if you are correct. This will involve doing: assign a string ending in "es" to variable `s`; type in the entirety of the `if-else` statement you have completed above; and see what the answer is. Then do the same thing, except first assigning to `s` a string that ends in, say, "ss". (You can actually hit the up-arrow key to recover your previous Python commmands, which is convenient.)

Help students with noticing that they have to indent correctly in the Python "'..." when testing.

(b) Assume that `s` is initialized with a string of digits, e.g., `'12345'`. Complete the following (write code in the blank spaces) so that it prints seven times the value of the middle digit if `s` has odd length. Thus, if `s` is the string `'12945'`, then 63 should be displayed. If `s` has even length then it should print 3 times the value of the last two digits. Thus, if `s` is `'1246'` then 138 should be printed.

```
if                                        :
   else:
```

```
# lab 02 digit question

length=len(s)

if (length%2 == 1):
    i_mid = length/2

    n = 7*int(s[i_mid]) #stub
else:
    last_two = int(s[len(s)-2:])
    n = 3*last_two #stub

print n
```

In working on this problem, you may find it easier to enter your code in a new python program in Komodo Edit, so you can make changes more easily. Remember to run your new program in the command shell.

Use print statements to discover the sources of any errors, like you learned earlier in this lab.

Make sure the students are testing on a STRING, not an int. Encourage the students to use print statements to debug their programs, like they just learned!

5. Pretty printing

Another file in the zip archive we provided us is `FormatPlay.py`. Here it is:

```
# FormatPlay.py
# the CS 1110 profs (cs-1110profs-L@cornell.edu)
# Feb 2016

""" A short script that illustrates formatted print."""

from math import pi
x = 355
y = 113
z = float(x)/float(y)
err = abs(z - pi)
print '\nNumerator   Denominator   Quotient      Error'
print '-----------------------------------------------'
```

```
print'%3d %3d    %22.15f %10.6e' % (x,y,z,err)
```

It produces the following output:

```
Numerator   Denominator   Quotient      Error
-----------------------------------------------
355 113         3.141592920353983 2.667642e-07
```

Modify the last line in `FormatPlay.py` so that the following output is reproduced:

```
Numerator   Denominator   Quotient      Error
-----------------------------------------------
    355          113      3.1415929    2.67e-07
```

In other words, center the numbers under the column headings, display the Quotient through seven decimal places, and display the Error with three significant digits. Do this by playing with the format specifications and blanks in the string `'%3d %3d %22.15f %10.6e'`. What does your new print statement look like?

something like this: `print '%7d %11d %10.7f %12.2e' % (x,y,z,err)`

Change the first `print` statement to

```
print '\n\n\n\nNumerator   Denominator   Quotient      Error'
```

What does `\n` seem to do?

6. If-Elif-Else

Consider the following code:

```
 x = input('Enter x')
 y = input('Enter y')
```

```
if 1<=x<=3 and 1<=y<=3:
    print 'A'
elif x>3:
    print 'B'
elif y<1:
    print 'C'
elif y>3:
    print 'D'
else:
    print 'E'
```

Complete the following table:

x	y	Output
2	2	`A`
1	0	
4	1	
1	5	
0	3	
3	0	