FIRST LAB OF THE SEMESTER
LAB EXERCISES Lab Wk2

Algorithm to follow for these lab exercises:

- **read the "Labs general instructions"**
- **review class notes as needed,**
- **read all the explanations in this lab exercises**
- **work on these exercises**
- **if you have doubts: ask peers, TAs and/or instructor.**
- **keep working after lab time.**

Keep in mind that at the time when the labs are posted some of the topics in the exercises have not yet been seen in class (marked +)

Start with #1, then #2, ... address the exercises in order!

1) **INTRODUCTORY VIDEO**

Watch the video posted in the course website "Starting to use IDLE". NOTICE that this video is introducing the IDLE environment for Python version 2.7.

Important difference for this lab between Python 2. 7 and 3.5:
For Python version 3.5, print is a function (not a statement) and therefore you need to include parenthesis, as in >>> print ("Hello!!")

IMPORTANT! There are other details where Python v 2.7 and Python v3.5 differ. The differences that are most relevant to our course will be addressed. If you want to see more information, you may check:
http://sebastianraschka.com/Articles/2014_python_2_3_key_diff.html

2) **TEST CODE IN THE SHELL VS. CREATE A PROGRAM?**

As was seen in class and is shown in the video, the IDLE environment uses two windows: The Python shell and the IDLE editor.

Several exploratory exercises are possible to do in the shell, in particular those that involve one line statements.

However, as the code gets longer (i.e. including several statements) it is **best that you create a program in the editor window** and then run it. Thus, you will use the shell window to run the program and see the results from executing the program.
To avoid re-running code that you tested you are recommended to "comment out" code already executed. See below.

3) **WHAT TO SAVE FROM THE LAB EXERCISES**

a) PROGRAMS CREATED IN THE IDLE EDITOR as .py files

b) You may also save for future reference WHAT YOU TESTED IN THE Python shell as .txt files (or as .py).

c) Keep in mind that the shell contents (that you may save) are just showing the dialog of what already was executed (including the prompt >>> and the responses).

The saved SHELL cannot be executed again. IT IS NOT A PROGRAM. It is historical information.

The reason to save the shell is for future reference for you to review, and to show to TAs what work you did.

4) COMMENTING OUT CODE THAT YOU ALREADY TESTED

Parts of code that you have already tested can be commented out (so that they are not executed again) but you still should save them to show to the TAs and for your own reference.

a. Comment out in your file code (including multi-lines) that you have already tested by surrounding by three single apostrophes. Example:

```
'''
(some statement) ...
(some statement) ...
'''
```

b. You can also mark a whole region in your file with the cursor and then comment out by using the Format→ comment out option in the IDLE editor

Test running the example file example_commenting_out.py

5) INCLUDE PRINT OR NOT?
FROM THE PYTHON SHELL, IF YOU WRITE AN EXPRESSION, PYTHON EVALUATES AND ALSO ASSUMES A PRINT. I.e. you can omit the word "print" if you are working in the IDLE python shell, i.e. when preceded by the prompt >>>. Test the following:

```
>>> 2*10 + 25
>>> print (2*10 + 25)
```

FROM A PROGRAM, HOWEVER, you need to include a print to be able to see any results printed in the shell when the program is run (or executed)

IDLE ENVIRONEMNT
TESTING SOME INITIAL CODE. KEEP IN MIND:

For 1 LINE of code: Use the PYTHON SHELL
For more than 1 LINE: Create a Python program with the editor, save it, and run it.

6) **+ To practice and investigate:** What is printed in each of the following statements (a) to (l) below? FOR THESE EXAMPLES TEST ONE LINE AT A TIME

NOTICE: Some expressions in the examples below may cause an error (on purpose). When there is an error there is a comment indicating so. Try out these examples. Type exactly what is presented here. You can copy/paste from here. **If there is an error, figure out what the problem may be.**

a. BASIC CALCULATIONS
Note: each print in these examples is aiming to print one single value, resulting from evaluating the calculation in the expression

```
>>> print (123 + 111)
>>> print (123*2/2+4-8/2)
>>> print ("123" + "111")    (notice the difference with the 1st
                              calculation!)
```

b. TYPES OF VALUES IN A PROGRAM
The data (the values) we use in a program can be of different **types**. The type of the value can be investigated using the type() function. Values can be converted from one type to another using conversion functions. Let's explore integers and strings:

```
>>> print ( type(123) )
>>> print ( type("ABC") )
>>> print ( type("456") )
```

AGAIN: NOTICE THAT 123 and "123" are different types of values!! The former is an integer (int) and the latter is a string (str).

For the next examples print is omitted since it is executed from the shell

```
>>> type(str(123))          (what is the str function doing??)
>>> type(int("789"))        (what is the int function doing??)
>>> type(int("Hello"))      (why does this cause an error?)
```

c. Another type of value: Boolean (test one example at a time)

```
>>> (5>1)
>>> (1>5)
>>>  type(5>1)
```

d. TEST ONE print AT A TIME
EXPLORING THE "+" OPERATOR WITH DIFFERENT TYPES OF VALUES

```
print ( "Hello" + "    " + "everyone" )
print ( "Hello! " + 3  )            # (!error!  what happened?)
print ( "Hello! " + "3")

print ( 1 + 1)
print ( "1 + 1")
print ( "1" + "1")

print ( "Hello! " + str(3) )
print ( "Hello! " + str(1+2+3)  )
print ( "Hello! " + str(1) + str(2) + str(3)  )
```

e. + EXPLORING THE "*" OPERATOR USING DIFFERENT TYPES OF VALUES (What does the * operator do?)

```
print ( 2 * 4)
print ( 2 + 3 * 4)
print ( 2+3 *4)
print ((2+3)*4)
```

f. AN INTERESTING CASE OF THE OPERATOR *

```
print ( "Hello! " * 3)
print ( 3 * "Hello! " )
```

g. print ("2" * "3") (!error! what may the problem be?)

h.
```
print ( int("2") * "3")
print ( int("2") * int("3"))
```

i. + NEXT WE ARE INVESTIGATING THE INTEGER DIVISION OPERATOR (//) and the REGULAR DIVISION (/) AND COMBINING INTEGERS AND FLOAT NUMBERS. OBSERVE THE RESULTS

```
print ( 21/3)
print ( 23/3)
print ( 21//3)   (This is the integer division operator)
print ( 23//3)
print ( round(23.0/3))
```

j. + HERE WE ARE INVESTIGATING THE % (reminder or modulus) OPERATOR. OBSERVE THE RESULTS. – What does the % operator calculate?

```
print ( 23 % 3)
print ( 21 % 3)
print ( 3 % 23)
```

k. + Printing ON THE SAME LINE with two print's.
For this code type all these statements in your python (.py) file and run, (as opposed to testing directly in the shell one at a time) to better see the effect when they are all run together (one after the other).

```
print ( "a")
print ( "b", end="" )
print ( 123)
print ( "b",123, end="")
print ( "c",1,"d",2 )   # different types of values may be printed
                        # when separated by commas
```

NOTICE that the possibility of including end="" is a Python 3.5 characteristic. Python v2.7 does not have the possibility of 'end=', and for this purpose it uses an extra comma). We will work with Python 3.5 in the course.

l. Printing extra lines

```
print ("a")
print ()
print ("b")
print ("c", "\n\n\n", "d")
```

7) **To reflect about the organization of an algorithm, and applicable to code:** Check the "sandwich example" algorithm versions #3 and #4 seen in class (posted notes in the Modules section). What characteristics distinguish the two versions? Version 4 would be preferable; why?

8) + ASSIGNING VALUES TO VARIABLES. USING VARIABLES IN EXPRESSSIONS…

We will start exploring variables and expressions in class after these lab exercises were originally posted. To continue with these exercises you are recommended to check the "Readings on variables and expressions".

For the next exercises you may keep testing one statement at a time after the prompt >>> in the shell, or you may also create a program. If in a program, write what you test (or copy paste) but test one group (i.e. the statements placed together) at a time. After you tested, comment out to not execute everything over and over again.

HERE WE ARE ASSIGNING THE VALUE 48 TO THE VARIABLE NAMED var1. var1 is one possible name for a variable (not an informative name!).

```
var1 = 48
print (var1 )      (what did we print ( here??)

print ((var1 * 10) + 1000)
print (var1 )      (did the contents of variable change after printing?)

print ("var1" )    (what did we print ( here??)

var1 = var1 *2
print (var1)          (the contents of the variable var1 changed. Why?)

print (new_var)   (what happened here?)
print (VAR1 )     (and here?)
```

9) **+ IF STATEMENT –** AN EXAMPLE – WE WILL SEE THIS IN DETAIL!! (the colon ":" and the spaces are mandatory). Do this example in a program

```
age = 15
if (age >= 16):
     print ("You can start practicing how to drive")
else:
     print ("No driving for you yet!")
```

10) **+ Testing a brief program**

Save the example provided (example_basic_calc.py) in your folder and then open it in the IDLE editor. Complete it as indicated, run it, test it.

11) **++ Creating a brief program**

a. Write a Python program (a .py file in the editor) that initializes (assigns initial values to) two variables (provide adequate naming of variables!) one consisting of a (positive, integer) number of years, and the other a (positive, integer) number of months (not necessarily less than 12). Assign some values directly to these variables, no input is required from the user.

Example:

```
years = 10
...
```

b. Then the program should calculate a single value in months (equivalent to the total years and months) and then print all the values (original and calculated), with adequate messages. After printing the values, if the resulting age is greater or equal than 20 years show the message "you are not a teen anymore!". Otherwise, print "you are so young!". Finish by printing 15 stars ("*").

   Note. To print 15 stars do not type 15 stars! You may want to review some of the examples above with string examples

c. ++ How would you do the inverse process? (From a total number of months calculate the number of years and months (with 11 as the maximum number of months in the final result)? *Hint*: Consider the integer operation and the modulus operator presented above.

SAMPLE RUN

So that you can better imagine what this program should do, see an example of the execution of this program (a "**Sample run**") next.

```
>>>
18 years and 27 months are equivalent to: 243 months
243 total months are equivalent to: 20 years and 3 months
* * * * * * * * * * * * * * *
>>>
```

Notes and recommendations for this brief program and in general!

- **IMPORTANT! To test and debug (eliminate the "bugs" or problems)** include intermediate printing messages so that as you test the program it shows as output not only the final result but also the contents of the variables in the middle of the calculations.
- For now INITIALIZE VARIABLES (give values to variables) directly in the program (for this exercise the user is not asked any data), such as var = 32
- **IMPORTANT! To best test your program consider different cases and special limit cases**: A limit case could be 0 years and some months, 0 years and 0 months, for example. Change the values you assign to the variables (that is, revise the program various times and re-run with the new values) to test how the program works with different values.
- A comment line is preceded by a symbol #. Get used to including comments in your program, identifying who the author is and to remind yourself what your program is doing.
- In a program, do not write sentences that are so long that you cannot see them in the IDLE editor original screen, this may cause a problem and it is not easily readable. If needed, to continue a sentence in the next line you can use the symbol \
- To print values in different lines in your output you can have different print sentences or you can also print the string "\n".

End of lab exercises (first lab this semester, week 2)