CSE 231 Lab Exercise #2

(Complete the Pre-Lab on D2L before your lab session)

PART 1: Debugging

Assignment Background

This lab exercise provides practice with debugging, meaning the process of identifying and removing errors from your code.

★ **Demonstrate your results to your TA. On-line (Section 730) students should do it on their own (Do not submit to mimir).**

The art of debugging is an integral part of programming. The following exercise is designed to help you understand the basics of the Spyder `IPython` debugger. The debugger is described in Appendix C of the text. Familiarize yourself with the debug control buttons in Spyder. They are blue and appear in the middle of the top bar of the Spyder window. Hover your mouse over each one to identify what each button does.

Set breakpoints on lines by double clicking the left pane of the Spyder IDE next to the line numbers. Try it and you will see a red circle; double-click again to make it disappear. Run the program in *debug* mode by pressing the Debug file (`CTRL + F5`) button (leftmost). Execution "freezes" when a breakpoint is encountered. You can then progress the program one statement at a time by pressing the Run Current Line (`CTRL + F10`) button (second from left). A pointer (`--->`) in the console (in the lower right corner of Spyder) shows you the next statement to be executed. You can use this pointer along with `CTRL + F5` to understand which lines of your code are executed during conditional statements. Notice the values stored in variables in the *Variable Explorer* window (above the console).

Assignment Overview

Open up the program `debug2.py` in Spyder. In the upper-right window click on Variable Explorer to open that window. Click 'Remove all variables' button () in Variable Explorer to clear all variables. Set a breakpoint at the first executable line of the program (`input`), and start the debugger by pressing the "start debugger" button (leftmost blue button). Answer the following questions for the input provided below and compare to the results:

Input1: `AES`
Input2: `n`

1) As you step through the program, record the sequence of line numbers executed? (e.g. 2, 4, 9)
   ANSWER: 8, 10, 12, 14, 15, 18, 21, 23, 10, 25

Input1: aes
Input2: n

2) A. Sequence of line numbers executed?
   ANSWER: 8, 10, 12, 14, 16, 18, 21, 23, 10, 25
2) B. When the pointer (`--->`) shows that the next instruction to be executed is 14, which variables are held in memory and what are their values?
   ANSWER: Variables → values

1. algorithm → aes
2. go_again → y

Note: Please *do not* use the `print()` statements during the debug exercise.

Additional Resources

https://docs.spyder-ide.org/debugging.html

Part 2: Programming with Control Structures

Assignment Background

This lab exercise provides practice with selection (***if***) and repetition (***for***, ***while***) in Python.

You will work with a partner on this exercise during your lab session. Two people should work at one computer. Occasionally switch the person who is typing. Talk to each other about what you are doing and why so that both of you understand each step.

The ***if*** statement is used to choose between alternatives. The ***for*** and ***while*** statements are used to repeatedly execute blocks of statements.

The ***for*** statement is most useful when you know exactly how many times you want to repeatedly do something (for example, you know how many sides of a polygon you want to draw). Here is an example showing how to print something **n** times:

```
n = 4
for i in range(n):
   print("Hi!")
```

Here is the corresponding output:

```
Hi!
Hi!
Hi!
Hi!
```

The ***while*** statement is most useful when you don't know in advance how many times the program will be required to repeat a task (for example, prompting the user to enter a value until correct input is provided). Here we will use ***while*** to repeat a fixed number of times simply to practice using the statement. To compare and contrast with the previous example, we will print something **n** times:

```
n = 4
count = 0
while count < n:
   print("Hi!")
   count = count + 1
```

Here is the corresponding output:

```
Hi!
```

```
Hi!
Hi!
Hi!
```

In this example, note that you must manage the counter variable ("count") yourself. If you make a mistake with the counter management, the loop may end up repeating forever. In that case, in the upper right of the IPython shell window is an icon with two choices, "Interrupt Kernel" and "Restart Kernel". Try "Interrupt Kernel" first and if that doesn't work, "Restart Kernel" will.

Assignment Overview

Download the file "`lab02a.py`" and develop a Python program which inputs a series of integers and processes them. The program will:

a) Continue to process values until the user enters the value 0
b) Ignore all negative integers
c) Count the number of odd integers entered
d) Count the number of even integers entered
e) Calculate the sum of the odd integers in the series
f) Calculate the sum of the even integers in the series
g) Display the sum of odds
h) Display the sum of evens
i) Display the count of odds
j) Display the count of evens
k) Display the total number of positive integers entered
l) Optional: print a message whenever a negative integer is entered

Sample output:

```
Input an integer (0 terminates): 1
Input an integer (0 terminates): 3
Input an integer (0 terminates): -2
Input an integer (0 terminates): 2
Input an integer (0 terminates): 6
Input an integer (0 terminates): 5
Input an integer (0 terminates): 0

sum of odds: 9
sum of evens: 8
odd count: 3
even count: 2
total positive int count: 5
>>>
```

Commentary

a) A nice characteristic of this problem is that it can be developed in small pieces. Remember: always try to break a problem into smaller pieces that are easier to solve.

b) First write a program that prompts for an integer, displays the integer, and stops asking for more integers when 0 is entered. Use ***while***. Under what condition do you continue to loop? Here is a suggested outline:

```
prompt for an integer  # (and convert the string to an int)
while some_Boolean_condition:
    # do something
    prompt for another integer
```

c) Once that simple program is tested and working, add in another piece such as (c) to count the number of odd integers entered. Create a variable with an appropriate name, such as `odd_count`, and assign it an initial value of 0 (before the while loop). When an odd number is entered, add one to `odd_count` (for example, `odd_count += 1`). Display the count.

d) Next: count the number of even integers and display the count.

e) Next: calculate the sum of the odd integers. The approach is similar to counting: choose a variable name, initialize it to 0; when the integer is odd, add the integer to the variable. Display the sum.

f) Next: calculate the sum of the even integers and display the sum.

g) Finally, ignore all negative numbers which the user inputs. One approach: *if* the integer is positive, do all the counting and summing, e*lse* do nothing (that is, you can use an *if* statement without an *else* clause).

h) If you have time, try this: print a message if the integer is negative.

★ **Demonstrate your completed program to your TA. On-line (Section 730) students should submit the completed program (named "lab02a.py") for grading via the Mimir system.**

Note (from the syllabus): Labs are credit/no-credit. To receive credit for a lab

1. You must complete the Pre-Lab, before your lab or the pre-lab deadline, whichever comes first. Pre-labs are "warm-up" for the labs and are not expected to be perfect -- our expectation is necessarily fuzzy for pre-labs: "you are expected to get most of them correct most of the time."
2. When Mimir tests exist, they test perfection. The Mimir tests allow you to verify that your code is correct. However, you can get credit for correct Lab code that isn't perfect, i.e. it is possible to get credit for code that fails Mimir tests. (Note that Projects are more strict with respect to Mimir tests.)
3. Adhering to the Coding Standard is expected, but expectations are less strict than for Projects.