CSE 231 Lab Exercise #2

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

Assignment Overview

This lab exercise provides practice with selection (if) and repetition (for, while) in Python and it has two parts: Part 1 (Programming with Control Structures, lab2a.py) and Part 2 (Using Turtle Graphics, lab2b.py).

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:

```python
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:

```python
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.

**Part 1: Programming with Control Structures**

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:

```python
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, else 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.
Part 2: Using Turtle Graphics
This part focuses on rendering different colors and some simple graphics using Python libraries. You will use both selection (if) and repetition (while, for) in this part.

You will use Turtle graphics
1. Loops: To draw a regular polygon given the number of sides provided by the user.
2. Selection:
   a. Add the capability to your program to fill polygons with a color indicated by a combination of the colors red, green and blue provided by the user.
   b. You will check to ensure that correct color values are input.

Originally written as a part of the logo programming language, Turtle graphics (http://en.wikipedia.org/wiki/Turtle_graphics) is one of the oldest graphics programs. It is a 2D graphics package that uses a Cartesian coordinate system and a “turtle,” which you can imagine has a pen attached to its body. The turtle can move around the plane, drawing as it goes. Python has a module that implements the behavior of the original turtle graphics program and this module is simply called “turtle” (see Appendix B of the text).

Download the file “lab02b.py” and modify it such that it will function as follows:
1. Prompt the user for the number of sides of a polygon.
2. Prompt the user for the length of sides of a polygon.
3. Draw a regular polygon using “for”
4. Move away from the first polygon.
5. Draw the regular polygon using “while”
6. Add the capability to your program to fill polygons with a color provided by the user (i.e. Python “input” command).

Using turtle graphics:
In order to use turtle graphics in Python you must first import the turtle module. You can then use the help function in idle to find out what methods this module includes and what they do. Just type “import turtle” in the idle command window, hit enter, and then type help(turtle) and scroll up through the list and information. For more details Google “Python 3 turtle.” A sample Python program “turtleSample.py” is provided in the lab directory.

The basic concept behind the turtle is the pen. The pen is either up or down. When down, the turtle draws as it moves around the Cartesian graph. There are a number of methods that are needed in this project:

turtle.up(),turtle.down(): Set the pen state to be up (not drawing) or down (drawing)

turtle.right(degrees), turtle.left(degrees): Turn the direction that the turtle is facing. The amount of turn is indicated in degrees.

turtle.forward(distance), turtle.backward(distance): Move the turtle forward or backward the amount of distance indicated. Depends on the direction the turtle is facing. Draws a line if the pen is down, not if the pen is up.
turtle.goto(x, y): The goto method moves the turtle to a specified point, drawing a line along the way if the pen is down, and not drawing if the pen is up. Note: The turtle always starts at the point (0,0). The goto method moves to the indicated x,y coordinates.

turtle.color(r, g, b): The color method sets the color that the pen will hold for all drawing until the color is changed. It takes three arguments, each a floating-point number between 0.0-1.0. The first is the amount of red, the second, the amount of green and the third the amount of blue.

turtle.begin_fill(), turtle.end_fill(): Use the command turtle.begin_fill() before you start drawing a figure. Draw the figure, then execute the command turtle.end_fill(). The figure drawn between the two fill commands will be filled with the present color setting.

turtle.bye(): Close the turtle drawing window.

First, try it!
The first thing you should do is try out some of the turtle commands by just typing them in the idle window. You can get a much better feel for how the whole thing works by just trying it.

Here is how to draw a line, turn left 45 degrees, and then draw another line:

```python
>>> import turtle
>>> turtle.down()
>>> turtle.forward(20)
>>> turtle.left(45)
>>> turtle.forward(40)
>>> turtle.bye()
```

Regular Polygon
The formula for the total degrees of all angles on the inside of a regular polygon with n sides is $180 * (n - 2)$, so each interior angle is $180 * (n - 2) / n$.
From that you can calculate how much to turn (exterior angle): $180 - 180 * (n - 2) / n$.
Use a loop to draw the regular polygon.

Adding colors
Your program will read in color values from the user and then use the methods in the turtle module to draw the polygons (above) filled with the color indicated. There are many ways to create a color but a common one used in computer graphics is the process of additive color (see [http://en.wikipedia.org/wiki/Additive_color](http://en.wikipedia.org/wiki/Additive_color)). Combining different amounts of red, green and blue can create most, but not all, colors.

For example, here is how to draw a red circle: all red, no green, no blue.

```python
>>> import turtle
>>> turtle.color(1,0,0)
>>> turtle.begin_fill()
>>> turtle.circle(20)
>>> turtle.end_fill()
>>> turtle.bye()
```
Sample output:
This is the output of a program drawing a pentagon (5 sides) of length 100 and red:

Number of sides: 5
length of sides: 100
Color: red

★ Demonstrate your completed program to your TA. On-line (Section 730) students should submit the completed program (named “lab02b.py”) for grading via the Mimir system.