Name: ____________________

Introduction to Programming in Python

Lab material prepared for Harrison High School Physics courses in collaboration with Purdue University as part of the project “Science Education in Computational Thinking (SECANT)”, http://secant.cs.purdue.edu/.[1]
August 2009

Lab 3: For Loops

OBJECTIVES
In this lab you will learn:

- Structure of a for-loop
- Writing and using a for-loops
- For-loops operating on lists and VPython objects

A computer **program** is a collection of instructions with the goal of accomplishing a clearly defined task. To model the physical world, we need to be able repeat specified tasks for as long as we want them repeated. Consider a simple problem as a start: printing the integers from 0 to 99. Using the material from Lab 1, we can do this:

We could write a program that consists of 100 print statements. Two possible programs are sketched below (one in each column):

```
BAD IDEA 1          BAD IDEA 2
print “0”           k = 0
print “1”           print k
print “2”           k = k+1
print “3”           print k
print  “4”          k = k+1
.                   print k
.                   .
.                   .
```

Do we really want to type 100 lines (actually 200 lines for the program in the right column)? What if we want to print all the numbers up to 1000? We need a way to accomplish this task by specifying that something should be repeated 100 or 1000 times. This is what loops are used for. We already know that **range(100)** generates a list of 100 elements (from 0 to 99). A for-loop allows us to iterate; in particular, it allows us to iterate over the list generated by range.

Type this program in your editor window:

```
#  Lab 3 – for_loop1.py
#  Print the integers from 0 to 99
# Written by: Your Name

for k in range(100):
   print k
print "loop finished"
```

[1] This work is supported by the National Science Foundation under Grant No. CCF-0722210. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Save the file as **for_loop1.py** and run the program.

Let's take a look at what you typed and how the computer interprets this:
You will notice that the editor automatically indents the statement on the line after the semicolon. Correct indentation is crucial in Python. Hit backspace to return to the indentation that lines up with the word **for** before typing the last statement. Program **for_loop1.py** contains a loop which contains a print statement. Once the execution of the loop is done, a final print statement is executed.

The loop is a **for-loop**. A for-loop consists of the following parts:

- It starts with the keyword **for**, a **variable** which iterates over something and the symbol ":". In the above code, the variable is called k (short names are common). Variable k iterates of the list the range statement generates, taking on every value in the list exactly once. Below this first line, statements are indented (there needs to be at least one statement). In Python, indentation is used to indicate the statements executed by the loop (called the body of the loop). The final statement is not indented which means it is not part of the loop.
- The variable in the for-loop needs to iterate over something. In our program, it iterates over what the range statement generates. This means variable k takes on every value in the list generated by the range command: 0, 1, 2, …, 99, in this order. This is what we want and it avoids having to type 100 print statements.
- The **body** of a loop consists of instructions indented after the first line. In our program, the loop contains only one statement, a print statement.

The for-loop iterates over "something". In program **for_loop1.py**, it is the list generated by range(100). The program contains a for-loop in which variable k takes on the values generated by a range statement. The values of k are used to access the elements in a list L. Open **for_loop2.py**. Note that not all comment lines are echoed in programs shown.

```
# Lab 3 – for_loop2.py
# use a for-loop to print the elements in L and their position in the list
# Written by: Your Instructor

L = [2.5, 1, 5, 6.5, -5, 12, 88, 7.5, 45, -12, 3, 1000]
print "the list:", L, "contains", len(L), "elements"
```

Run program **for_loop2.py**.

```
for i in range(len(L)):
    print i
```

A.) What did this print and why did it stop where it did?

```
for i in range(len(L)):
   # print i
    print i, L[i]
```

B.) What did **L[i]** do in the program?

Exercises
Write programs using a for-loop to do the following (review the **range** statement from Lab 1):

1. Print all odd integers between 1 and 100 (include 1).
2. Print all integers that are multiples of 5 from150 to 0 (print in decreasing order and include 150, but not 0).
3. Open program **for_loop2.py** and save it as file **for_loop2sq.py**. Change its code so that a for-loop iterates over a list L (containing only numbers) and squares every element in the list. Print the list before and after the

for-loop is executed. If L=[1, 2, 4] before the loop, it should be L=[1, 4, 16] after the loop.

For-Loops and VPython

For objects created in VPython we can use a loop structure to move the objects. Here is a first example of such a for-loop.

```
# Lab 3 – for_loop3.py
# two spheres, one moving
# Written by: Your instructor

from __future__ import division
from visual import *

ball_red = sphere(radius=3, color=color.red )
ball_blue = sphere(pos=vector(-6,0,0), radius=3, color=color.blue)

# the red sphere moves by increasing its x-position 30 times
for m in range(30):
   rate(10)
   ball_red.pos.x = ball_red.pos.x + m
   print "m=", m,  "   ball_red.pos.x=", ball_red.pos.x
```

Type the code shown above and save it in file **for loop3.py. R**un the program. You have seen in Lab 2 how to define and name sphere objects and how one can change attributes once objects are named. The for-loop iterates over the 30 values generated by range(30) and for every value, the x-position of the sphere is updated (how much depends on m). Change the value of 30 to a smaller integer (say, 15) and run the program.

The statement you have not seen is **rate(10).** Change it to rate(1) and run the program. Comment out the rate statement and run the program. What does **rate** do? It controls the speed of the visualization, with rate(1) being very slow and rate(10) being a little faster. Not setting rate will visualize depending on the speed of the computer the program runs on. As you use VPython, you will get experience on what values to use to get the visualization you want.

Take another look at program **for_loop3.py**:

i. The x-position of the red sphere is changed by the statement **ball_red.pos.x = ball_red.pos.x + m** Alternatively, we can change the position by using vector addition (we are adding the vector (m,0,0)): Change the position of **ball** by using vector addition. Run the changed program.
ii. Change program **for_loop3.py** so that all three coordinates of **ball** change (add m to all three). Run the changed program.

Have the instructor check you work. You may continue if your instructor is busy.

Ball moving on a track

We now use a for-loop to move a ball on a track, a setup we will use later on in a number of scenarios. Load program **for_loop4.py** (found on website http://secant.cs.purdue.edu/hhs) which contains:

```
# Lab 3 – for_loop4.py
# a sphere moving on a track

from __future__ import division
from visual import *

my_le = 100
my_he = 2
my_wi = 10

track = box(size=(my_le, my_he, my_wi), color=color.red)      # the center of the track is at (0,0,0)

rad_ball = 2                  #defines the radius of the ball

ball = sphere(pos=vector(-my_le/2, (my_he/2 + rad_ball), 0), radius=2, color=color.blue)
```

C.) Where does **(-my_le/2)** put ball.pos.x?
D.) Where does **(my_he/2 + rad_ball)** put ball.pos.y?

```
for m in range(my_le):
   rate(10)
   ball.pos.x = ball.pos.x + 1
   # ball moves by one unit per iteration, reaching the end of the track after my_le iterations
   print ball.pos
```

Run program **for_loop4.py**. Feel free to change the colors of the objects. Reduce and increase the rate so you can change the view in the display window as the sphere is moving.

Remember that the size of the box is given by specifying its length, width and height. The default value of attribute pos is (0,0,0). The box will be centered at pos and the box will aligned with the coordinate axes.

Program **for_loop4.py** defines three variables – **my_le, my_he, my_wi -** representing the length, height, and width of the track. The two VPython objects are defined in terms of these variables.
Why are we doing it this way?
Change the length of the track from 100 to 200.
Only the statement **my_le = 100** needs to be changed to **my_le = 200**. The initial position of the sphere will change automatically as it gets it x-value from **my_le**.
Run the program with the changed length of the track.

If we had "hard-wired" these values, we would have to make more changes. Always think about whether to use variables to represent quantities (the assignment statements are generally listed at the beginning of the program). Your code will then refer to these variables. It is a good idea to use variables for parameters when one plans to change values and there are dependencies.

Ball moving on a track: Version 2

In the physics problems, we will see examples where we move objects from one specified positions to the next. The positions will be stored in a list. Our next program prepares you for this. We are again rolling a sphere on a track using the definitions given program **for_loop4.py** (so copy and paste **for_loop4.py** into a new page and save it as **for_loop5.py**). Insert the list **track_pos**, which contains the x-positions on the track after defining the track and ball. Let's use

```
track_pos = [-40, -30, -5, 0,  5,  10,  20,  35,  40, 43.5]
```

Remember that the sphere is initially at x-position **-my_le/2** which is -50 in the program. To make it easy, we start with the positions expressed in the list as numbers.

Replace the original for-loop with this

```
for i in range(len(track_pos)):
   rate(2)
   ball.pos.x = track_pos[i]
   print ball.pos
```

Run the program.

E.) How many times is the body of the for-loop executed? ____

Rearrange the order of the elements in list **track_pos**, add some new ones. Run it.

Challenge

Edit **for_loop4.py** program (save it as **for_loop_two.py**) so it does the following:

- Initially, two spheres are positioned on opposite ends of the track. You can use the code in **for_loop4.py** as a template**.**
- Write a for-loop moving in which the spheres across the track with the same speed.
- Once the spheres have reached their opposite side, move one of the spheres back to the other end of the track. Both spheres are now both positioned on the same side.

Have your instructor check it.

Copy the code and print it out along with the answers to questions A through E