

PHY307, Science and Computers I

Lab #2, September 5, 2002 (due by end of lab period)

and

HWK #3 (due September 10, 11:30AM)

Loops and 3D objects

Summary of what you will do:

The goal is for you to gain more practice at Python by writing loops to handle repetitive tasks and to start using the **visual** module for animation. You will (a) write a simple loop to make a table, (b) modify the plane position calculating program, (c) experiment with the **visual** module, and (d) use the **visual** module to make a flying cylinder.

Your report for this lab will not be on this handout, but in a document that you create.

If you have extensive programming experience, I encourage you to go beyond the examples here, trying more complicated exercises. (See suggestions at the end and try out some of the Visual library.) [607 students: see extended homework.]

PYTHON NOTES:

- ?? There will be a brief lecture on comments, loops, and block statements.
- ?? You should have reviewed “LiveWires J” on loops. You might want to call it up from the course webpage (under Books & Resources.)
- ?? Animation in 3D can be done by executing loops to change the attributes of an object, such as its size, position or color.

LOOPING

1. Log in to your workstation using your CMS/SUnix name & password.
2. Open up some editor or word processor to start your report. Put in a title, your name, and date. As you write the report, make it clear what each section is. At the end of the lab, print your report out, staple it together, and hand it in.
3. Get IDLE for VPython going by going through the Menu: Start ? SU Academic Departments ? Physics.
4. Open a program file editor by “New” under the “File” menu and save your (currently empty) work to your I: or K: drive as “loop1.py” or a similar name.

5. Write a program with a loop that will print out a table of the squares of the integers from 1 to 10 (the square of 5 is 5 times 5 or “**5 * 5**” in Python), using a **for** or **while** loop. You should have comments in your program. Your output should look something like:

```
1 1
2 4
3 9
[etc...]
```

- Put your program and output in your report. (Did I already say that you should have comments in your program?)
6. Next, run and then modify the plane program from today’s opening comments. Part of this is practice in loading a Python file from the course web page.
- Open up Navigator or Explorer and go to the PHY307 web page.
 - Click on codes to “Codes”
 - Right click on the “planetest.py” link and save the file to your home drive (I: or K:).
 - “Open” (under the “File” menu) the program from IDLE and run the code.
 - What results do you get? (Include in your report.)
 - Modify the program by changing the length of time that the plane changes speed and changing the rate at which the speed changes. Add comments to indicate your changes.
 - Run the program. Include the modified results and the program in clearly indicated sections in your report.

ANIMATING WITH LOOPS

7. Objects that have no net force on them move at constant speed in a constant direction. This motion is exhibited by the center of a ball rolling on a table surface or an asteroid in space (over short times where the deflection due to gravity from the Sun is small.) So to simulate a coasting object, we just draw it, move it some amount, draw it again, move it the same amount, draw it again, move it the same amount, draw it again, etc., until we get bored. This repetition is taken care of nicely by a loop in a programming language. Note that when you modify an attribute of a geometric shape in VPython, the scene is automatically updated; you don’t need to request that the scene be redrawn. Here, you will make a visualization of a cylinder coasting through space.
- First, in your report, clearly formulate the goal of the program you will write. Rephrase the above description. Also consider the approximations you will be using: will the object be subject to friction, which slows objects down? Will the object spin? What color will you make the object?

- b. Experiment with the elements of VPython that you will need, using a shell window. Carry out the following and note in your report how you do each of these:
1. Open a shell window from IDLE.
 2. As you did last week, you will first need to bring in the tools of the visual library.
 3. Turn off autoscaling with “**scene.autoscale = 0**”.
 4. You can see the “Reference Manual” under “Documentation” at www.vpython.org, but, among other attributes, cylinders have **x**, **y**, and **z** coordinates that give the position of their center, a **color**, a **radius**, and a **length**. Draw a cylinder and assign it to a variable.
[Example: to do this with a box, you might say
`mybox = box(color=color.red, x=3, y=2, z=0, width=1, height=4, length=9)`
The “**box()**” on the right creates a **box** object with the requested properties; the “**=**” then assigns the object to the “**mybox**” variable.]
 5. Change the cylinder’s color.
 6. Add in a new cylinder with position **x=5**.
 7. Move the first cylinder to a new position.
 8. Copy your shell interaction to your report. Make a couple of appropriate comments.
- c. ANIMATION. Now open a file window and write a program that:
1. Imports the visual module.
 2. Creates a cylinder.
 3. Turns off autoscaling.
 4. Executes a **for** loop that moves the cylinder by a distance of **0.01** in the x-direction each time through the loop. In the loop, you want to set the rate. Do this by making the statement **rate(20)** part of the block statement, to limit the number of steps to 20 per second at most.
- d. Record your working program and describe what you see in your scene.

PHY307 HWK ASSIGNMENT #3, due Sept. 10, 2002:

Debugging tips: you will run into at least two types of errors while carrying out programming exercises. Fixing these errors is debugging.

The first type of error is that where the program doesn't even run. This is due to some syntax or dynamic error. Fixing these requires carefully checking your typing (did you type **yellow** or **color.yello** instead of what should have been **color.yellow** ?) These fixes also require familiarity with Python and practice looking at examples.

The second type of error is where the code runs OK (no crashes) but doesn't give you what you expect. One approach to finding why the code is not working as expected is to put in lots of **print** statements. If you think that the width of a cube object assigned to the variable **mybox** should be **0.7**, for example, you can check your assumption with a "**print mybox.width**" statement at the write place in your program.

(The worst kind of error is when everything looks fine, but you are getting the wrong answers and don't know it. Prevention of this type of error takes experience and careful verification of your code. We'll worry about this later.)

Problems:

1. Use a **for** loop to draw 10 evenly spaced cubes, arranged along a line. Remember the attributes **position**, **height**, **width**, and **length**. Cubes are special cases of boxes where the **height**, **width**, and **length** are all the same. (See the VPython documentation at www.vpython.org). You can make the cubes overlapping or not; you can make them of the same size or not. Remember the incantation at the beginning of the program necessary to give your code the **visual** library. Submit the Python code and describe what you see.
2. Race a yellow sphere, blue ring, and white cube. Remember the needed **from/import** statement at the beginning. When you create and move the objects (of types **sphere**, **ring** and **box**), remember that each has **x**, **y**, and **z** coordinates. When you create the objects, give them all the same **x** and **y** coordinates, but have them spaced by a distance of 2 in the **z** direction (write the creation/assignment lines separately for each object, you don't want to use a for loop to start up the objects.) You probably want to use **scene.autoscale=0** so that you can cleanly see the motion. Have the objects move in the **x** direction. You will want to control the animation rate using, say, **rate(40)**. Give each object a different speed. Discuss how your programming experience went and comment on what you might like to do to improve the program (you don't need to modify the program, just write how you would like it to look different.)

3. ***For 607 students:*** Use VPython to construct a virtual orrery for the inner Solar System. Speed up time by a factor of a million and uniformly scale the radii of the planets so that they are visible. You may assume circular orbits and don't worry about the initial positions of the planets.

4. ***607:*** (Due Thursday, Sept. 12, if you want extra time.) You are to model the behavior of two charged, rigid, insulating, elastic spheres of masses 10 kg and 30 kg. Take their mass densities to be 5 g/cc. Each sphere has a charge of magnitude $1e-5$ Coulomb, with one sphere being positively charged and the other being negatively charged. These charged spheres behave like cue balls in that as soon as they touch, they bounce. Create a VPython visualization of these bouncing planets, with some non-trivial initial conditions (that is, they collide, but not straight on.) Use the **testplane** example as a guide on how to incorporate acceleration to update the position and velocity of an object. The function **sqrt()** takes the square root of a number (useful for computing distances – note that you need to import **math** or **visual** to get **sqrt()**.) You might look at the VPython demo codes for inspiration. In a report accompanying your code, comment in *detail* about the approximations that you are using, including the implicit and explicit approximations in this problem description. Outline how you might go beyond those approximations, either to get closer to perfectly correct behavior or to place a bound on the error of your simulation. This last requirement requires thinking about what we know about physical models, including 20th Century Physics and is in principal quite difficult to answer. Don't spend more than a page on this. Submit your code in writing, a narrative description of the results, and your discussions about the approximations used. E-mail me your code so that I can try it out.