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 2: Introduction to VPython

OBJECTIVES

In this lab you will learn:

- Creating and running a Python program
- Creating sphere, arrow, and box objects in VPython
- Giving VPython objects a name and using their attributes

In Lab 1, you executed simple Python instruction in the interpreter environment. A program is a **set of instructions** which are saved in a file and run/executed as needed. In order for the computer to understand what you want, you must follow a specific writing protocol. The guidelines governing the writing of these instructions are referred to as syntax. A Python program contains Python statements. When running the program, these statements are executed in sequential order, from the first statement to the last. A Python file has the ending .py (SPE and most environments will add the extension for you).

Before we write our first program, we introduce a new module, **visual** (remember, Lab 1 used the **math** module). Modules need to be imported before they are used and we use **import *** to import everything. To use the **visual** module (which imports what is known as VPython), we write **from visual import *.** Every program importing the **visual** module should start with **from __future__ import division** (see Lab 1, the __ are two underscore characters). Importing this module ensures that division on integers does not truncate the result (it produces what is called a floating point number, not an integer).

Start by opening SPE and locate the editing window (not the interpreter window used in Lab 1). The editing window in SPE has the Source tab and it is used to type and edit programs. Type the following:

```
from __future__ import division
from visual import *
```

Before doing anything else, save this statement as a program. In SPE: File>Save As and select the folder to save the file in. On the flash drive, use a folder "MyPrograms" to store your programs. Name the file name **objects1** (SPE as well as PyScripter will add the file extension .py for you). Once the file is saved, you are ready to create your first VPython object.

---

[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.

1. Creating VPython Objects

Go to the edit screen of file **objects1.py**. Leave a blank line (adding blank lines to Python programs does no harm and improved readability) and then write the following line of code:
sphere( )

To run the program; in SPE: Tools>Run without arguments. You will be asked to confirm saving the program if it was changed since the last save: say yes.

Tah-Dah! Two windows will open up: python.exe (ignore it) and the VPython window. There is your sphere!

Position the mouse over the VPython window. If you press and hold the right and left mouse buttons and move up and down (as opposed to right and left), you will notice the sphere seemingly change size. You are actually zooming in or out from the sphere. Next try holding down the right button while moving the mouse around. Notice that you are seemingly able to look at the sphere from any vantage point you choose.

Close the VPython window: the two recently opened windows disappear, but SPE stays open and ready for you to do more.

Your code is saved every time you run a program (SPE will ask you to confirm). If you want to modify a program and still keep the former one in its original form, create a new copy before opening SPE. Or, use the "Save as" feature and save it as a file with a different name. That way, you can keep the old one and modify the new one, too.

Open the file **objects1.py** (if it is not already open), leave another line (for readability) and add:
sphere(pos=vector(0,4,0), radius=0.40, color=color.red)

Save this modified program and run it. You should see a red sphere located above a larger white sphere. VPython created two spheres (they are called objects). For the second sphere, we specified a number of features which were omitted for the original (white) sphere. In particular,

- We selected the **position** of the center of the second sphere by modifying the "pos" parameter of the sphere. The position of the sphere is described in 3-D Cartesian coordinates (in this case, the center of the sphere lies on the y-axis four units above the x-z-plane). When no position is specified, the center of the sphere is (0,0,0), the origin (this is its default position).
- The **radius** of the sphere is set to 0.40 units in length. The default length is 1.
- The **color** is set to red. The default color is white. Eight colors can be written as color.xxx: red, green, blue, yellow, orange, magenta, cyan, black, and white.

Here is what you see for file **objects1.py** in the editor window (note that the line numbers are not part of the file, but are added by SPE):

```
1  from __future__ import division
2  from visual import *
3
4  sphere()
5
6  sphere(pos=vector(0,4,0), radius=0.40, color=color.red)
```

Next, let's see what happens when we change the syntax of the statement in line 6. Try running the program after making each of the following changes (change it back to the original before altering another line). All but the last one will produce a syntax error (make notes so you have an idea of what to look for when you see these error messages again):

- Write **radius 0.40** (eliminate the =); run the program
- Replace one comma by a period; run the program
- Add an unnecessary); for example, **sphere(pos=vector(0,4,0)), radius=0.40, color=color.red)**; run the program
- Write **radius = 0.40** (add spaces); run the program

Correct syntax is extremely important. Think of these exercises as helping you communicate better. For instance, if you claimed that "mass is the same as weight", your physics teacher would stop you from going any farther until this statement is corrected. Mass is different from weight. Mass is the same as linear inertia. The force of gravity is the same as weight. You might have meant any of these. You might have meant something different. The problem is that your teacher can only respond to what you said, not what you meant. The same is true when writing a program. A program will react to exactly what you communicate and not what you meant to communicate. It does exactly what you tell it to. The good news is that Python is equipped with the ability to let you know if you have incorrect syntax (most of the time).

Now assume you want only the red ball to appear in the VPython window. Instead of deleting the line with the statement creating the white sphere, put the pound sign (#) in front of the statement:
#sphere( )

When Python sees the symbol #, it ignores everything after it on that line. You can use this property to write comments in your program. Commenting programs is very important. For that reason, SPE provides this as a feature: select the lines you want to comment out and click the *Comment* icon on tools bar (click the *Uncomment* icon to reverse). It will actually insert two #'s to make it stand out even more.

Comments identify your program if handed in for grading, they allow you to remember why you did things a certain way, and you can use them to make notes so you can make sure that a program is doing everything you want it to do. As a requirement, you always have to start a program with comments that contain your name and its purpose. Something like:

```
## Lab 2 – objects1.py
## Prof. Bunsen Honeydew
##
from __future__ import division
from visual import *

##sphere()
sphere(pos=vector(0,4,0), radius=0.40, color=color.red)
```

Go ahead and add comments to your code in file **objects1.py**. Also comment out the initial sphere object and run the program again. You end up with just a red sphere.

Next, open a new file and write code which will produce the following spheres: One technique that will help you go faster is copying and pasting lines of code from programs you have previously written to use in your new program. Underneath the toolbar at the top of the page are tabs that have files that you have open, so you can quickly go between existing programs and your new one.

- A blue sphere on the x-axis with radius 0.50 units and center 2 units to the left of the origin.
- A magenta sphere centered at (0,3,-3) and whose shell extends half-way to the y-axis.
- A yellow sphere centered at (1,4,1) and whose shell includes the point (0.5,3,-1). You will have to use your knowledge of vectors to calculate the radius.

Save these three statements in a new program called **objects2.py**.

Have your instructor check your program and the VPython window it generates. (keep going if your instructor is busy, but make sure you can reproduce it).

Create a new file **objects3.py** (File>New). In this file we will add statements making VPython draw arrows. Go back to the editor window. Add the initial comment lines and the import statements. Skip a line and write the following line of code:

arrow( )

Run your program. In the VPython window, move the arrow around and zoom in and out to familiarize yourself with the arrow. Just like with the sphere object, the arrow object has default settings. Unless you tell it otherwise, the arrow will start from the origin, point in the positive x-direction, be white, and have a magnitude of one unit.

Add the following statement to file **objects3.py:**
arrow(pos=vector(1,1,1), axis=vector(1,1,1), color=color.green)

Run the program. What have you instructed VPython to draw? The "pos" vector works as for the sphere; it describes a point with respect to the origin. For a sphere, it defines the location of the center. For an arrow, it represents the location of the tail. The "axis" vector is another vector which defines a point with *respect to the "pos" vector*. The "axis" determines where the head of the arrow will be displayed.

Both the magnitude and direction of the arrow can be determined from the "axis" of the arrow. Using your knowledge of vectors and how VPython interprets them, answer the following questions:

A. What is the magnitude of the green arrow?

B. What is the unit vector of the green arrow pointing?

C. What Cartesian point (with respect to the origin) would the tip of the arrow occupy?

2. Naming VPython Objects and Accessing their Attributes

Our goal is to not only create objects, but to also manipulate them; in particular, see them move in the VPython window. To do this, we need to give VPython objects names. Naming an object is done in an assignment statement. You can choose any legal Python name. We will choose your instructor's favorite organic foods.

Start a new program and call is **objects4.py** Create a sphere called bambi.

```
# Lab 2 – objects4.py
# spheres bambi and thumper and the arrow tnt in between
# Written by: Your Name
from __future__ import division
from visual import *

bambi = sphere(pos=vector(0,4,0), radius=1, color=color.red)

thumper = sphere(pos=vector(-6,0,0), radius=3, color=color.blue)
```

Using the print command allows us to get immediate feedback to what the computer is interpreting from each line of code…this can be very helpful for trouble shooting a program. One can also use the interpreter window, the bottom window.

Once a VPython object has a name, we can access its attributes.
print bambi.radius Now run the program. Click on the black SPE window behind Vpython to see the result.
print bambi.pos Now run the program
print "Bambi's x component is", bambi.pos.x Now run the program
print "y=", bambi.pos.y, **"z=", bambi.pos.z** Now run the program

We can also change the attributes of an object:
bambi.color = color.green Now run the program
bambi.pos = (-1, 2, 0) Now run the program
bambi.radius = 2 Now run the program
bambi.pos.x = bambi.pos.x + 20 Now run the program

Change the **bambi** back to the original values.

Use the values stored in the pos attribute of the spheres to create an orange arrow **tnt** pointing from the center of **thumper** (blue) to the center of **bambi** (red). Example: use **bambi.pos + vector(thumper.radius, 0, 0)** instead of **vector(0,4,0)+vector(3,0,0)** or **vector(3,4,0)**.

The concept of naming objects and using the attributes associated with the name, as done for arrow **tnt**, is a crucial one. We will soon begin writing programs for dynamic systems of multiple objects. For the sake of Physics, you will be required to describe a change in various physical attributes of one object as they relate to attributes of other objects.

```
tnt = arrow( ……)      #an orange arrow pointing from thumper to bambi
```

Change the values of **bambi** and **thumper** and see if **tnt** updates correctly.

Have your instructor check your code. While you wait, read about drawing a box and other objects. You may then start working on the challenge project for this lab.

What other objects can one create in VPython?

The course will introduce objects as needed. Here is how we draw a box. We will be using the interactive interpreter window to try it out (*that is the bottom window*). This allows us to get immediate feedback to what the computer interpreting from each line of code…this can be very helpful for trouble shooting individual lines of code in a program.

>>> **from visual import ***

>>> **dice = box ( )**

Use the zoom and rotate function on the mouse to play around with it. *DON'T close the VPython window or you will shut down SPE all together!*

>>> **dice.pos** (the "pos" for a box refers to the position of the center of the box)

vector(0, 0, 0)

>>> **dice.length**

1.0

>>> **mybox = box(pos = vector(4,4,4), length=8, height=4, width=2, color=color.green)**

Select the VPython file tab at the bottom of the screen to view the box.

Question 4. What dimension corresponds with the x-axis? the y-axis? the z-axis?

>>> **my_volume = mybox.length*mybox.height*mybox.width**

>>> **my_volume**

64.0

Note that writing **length=8, height=4, width=2** is equivalent to writing **size=(8,4,2) when defining a box.** A box can also get an axis attribute which we will make use of later.

You can see a complete list along with additional explanations at http://www.vpython.org/webdoc/visual/index.html. Below is summary of all objects available in VPython (with their default values) starting with the three you have already seen:

arrow( ) is equivalent to	**arrow(pos=vector(0,0,0), axis= vector(1,0,0), radius=1)**
box( ) is equivalent to	**box(pos=vector(0,0,0), size=(1,1,1))**
sphere( ) is equivalent to	**sphere(pos=vector(0,0,0), radius=1)**
cylinder( ) is equivalent to	**cylinder(pos=vector(0,0,0), axis= vector (1,0,0), radius=1)**
cone( ) is equivalent to	**cone(pos=vector(0,0,0), axis=vector (1,0,0), radius=1)**
pyramid( ) is equivalent to	**pyramid(pos=vector(0,0,0), size=(1,1,1), axis=vector(1,0,0))**
ring( ) is equivalent to	**ring(pos=vector(0,0,0), axis=vector (1,0,0), radius=1)**

Close the VPython window. You will have to start SPE again…kind of frustrating, I know.

Challenge

You may use the handout from Lab 1 and this handout as references. If you get hung-up, you may raise your hand for assistance. You are prohibited from asking a classmate. This project focuses on the foundation blocks for modeling Physics and you must learn them independently.

Open **objects5.py** and copy it into a new program **objects6.py.** Change the **thumper** and **bambi** back to the original coordinates, and then write a program to do the following:

a. Magnitude
   - Write a program that calculates the magnitude of the arrow (using the definition of magnitude). Use the name of the arrow to determine the values, rather than numbers! Print the magnitude.
   - VPython provides a function **mag(**vector**)** computing the magnitude of a vector. Use this function to compute the magnitude and print the magnitude. It should be the same as above. Example: **mag(mybox.pos)** = **mag(**vector(4,4,4)**)** = 6.9282

b. Direction
   - Write a program that calculates the direction (unit vector) the arrow is pointing.
   - Print the unit vector of the arrow.

c. The arrow starts at the center of **thumper**, but ends at a point on the *surface of sphere* **bambi**.

d. Change the radius of bambi and the location of both thumper and bambi. Run program **objects6.py** again. Is the correct picture generated by VPython?

When you are finished, have your instructor check your program. They will modify the location of your objects and make sure the program still functions properly. Once you have had your work approved, copy the code for your challenge and type the answers to questions A through C on a Word.doc and print a copy with your name on it.