

PHY307, Science and Computers I

Lab #6, September 26, 2002
(due by end of lab TODAY)

Fiddling with Falling and Functions

Summary of what you will do:

One of the most important capabilities of a computer language is the ability to define programs or procedures in terms of other procedures. This makes it much easier to write programs, to read them, and to reuse code for multiple purposes.

You have used this ability many times – you import the **visual** module, you use the ****** operation or other math functions *without having to write them yourself, each time*.

You will often want to write programs using a modular approach.

In Python, this capability for modules includes defining functions (as well as classes and other things..) We will need the ability to define functions especially when we start drawing fractals.

In this lab, you will write functions and you will continue work on the ideas needed for the fireworks show.

In your lab report, give a narrative of what you do and include the codes that you try out.

We will start with a very brief lecture which you will immediately apply. The concepts include: attributes of Python objects, vectors, acceleration in physics, and functions.

FIRST, FALLING THINGS

1. Log in and start a report for Lab #6.
2. Make a falling object. Remember that velocity is how quickly position changes and acceleration is how quickly velocity changes. VPython objects do not have a velocity attribute, but we can add one.
 - a. First step: make a program to display a gliding object.

```
from visual import *
scene.autoscale = 0
dt = 0.05          ## time step will be 1/20th of a second
a = cylinder()     ## set up a cylinder and add an attribute which
a.vel = vector(2,0,0) ## is its velocity—a vector in the x-direction of
                    ## length 10.

for step in range(1000):
    rate(20)
    a.pos = a.pos + dt * a.vel ## update the cylinder's position
```

Run it and record what you observe.

3. Next, add acceleration to the cylinder's motion. You need to add only two lines. In one line, outside of the main loop, you will need to define a vector acceleration for the object – set it to be -1 in the y-direction and 0 in the x- and z- directions. In the loop, you need to update the velocity by adding to it **dt** times the acceleration. Run your modified program and record what you observe.

NEXT, TRY WRITING FUNCTIONS

4. Start a new window, saving the empty program to the file **function1.py**, run it and record the results.

```
from visual import *
```

```
## function definition: turns an object red – this definition is NOT  
## EXECUTED UNTIL NEEDED
```

```
def makered(visobject):  
    visobject.color = color.red
```

```
a = box()  
b = cylinder(y=2)  
c = sphere(x=3)
```

```
makered(a)  
makered(b)  
makered(c)
```

5. Try the next example:

```
from visual import *
```

```
## function definition: spins an object by 0.05 radians about (1,1,1) direction  
def spinit(visobject):
```

```
    visobject.rotate(angle=0.05, axis=(1,1,1))
```

```
a = box()  
b = cylinder(y=2)  
c = sphere(x=3)
```

```
while 1:  
    rate(20)  
    spinit(a)  
    spinit(b)  
    spinit(c)
```

What does the function **spinit()** do? What happens in the **while** loop?

6. Now what you might do is have a whole collection of functions that you want a bunch of programs to share. This collection is a module. Examples of modules that you already use are **visual** and **random**. These modules are used over and over in many programs. You will now make your own module.
 - a. Make a module file – it is just a set of python statements. Open a new file, save it as **mymodule.py** and put the following into this file:

```
from visual import *

def spinit(visobject):
    visobject.rotate(angle=0.05,axis=(1,1,1))

def fall(visobject, accel, dt):
    visobject.pos = visobject.pos + dt * visobject.vel
    visobject.vel = visobject.vel + dt * accel

def spinfall(visobject, accel, dt):
    fall(visobject, accel, dt)
    spinit(visobject)
```

*It is important to save **mymodule.py** to the same directory where you will put your Python codes.*

- b. Use your module file. Here is a sample:

```
from mymodule import *
scene.autoscale = 0

b = box(width=5,length=3,height=1)
b.vel = vector(1,3,0)
gravity = vector(0,-1,0)

while 1:
    rate(20)
    spinfall(b, gravity, 0.05) ## fall and spin for 0.05 seconds
```

Explain what is happening (what does **spinfall()** do?) and what you see when you run this program.

HOMEWORK #6, PHY307/607, Fall 2002 – Due by start of class Oct. 1

1. More fireworks. Improve upon last week's fireworks program. Instead of random initial positions, give each spark a position at (0,0,0) and a **radius** of 0.2. After making the sparks, add a loop to give each spark a random *velocity vector*. Then update the locations using **mymodule.py** from Lab #6.

```
from mymodule import *

scene.autoscale = 0

## Build up a list of sparks by appending to the
## end of an initially empty list.
sparks = []
for i in range(100):
    sparks.append(sphere(radius=0.2))

## Initialize the velocity of each spark
for s in sparks:
    s.vel = vector(random()-0.5,
                   random()-0.5,random()-0.5)
    s.vel = s.vel/mag(s.vel)

## Set the strength of gravity and the time step.
grav = vector(0,-0.1,0)
dt = 0.05

## Main loop - repeat forever
while 1:
    rate(20)
    ### PUT IN BELOW WHAT YOU NEED TO STEP
    ### THE LIST OF SPARKS, UPDATING THEIR
    ### POSITIONS - SHOULD TAKE TWO LINES: A LOOP
    ### OVER sparks AND THE FUNCTION TO MAKE A
    ### PARTICULAR SPARK FALL
```

Submit the following:

- ?? Completed code.
- ?? An explanation or guess what the line `s.vel=s.vel/mag(s.vel)` does. Can you find the mag function under the VPython documentation? What do you see if instead you leave that line out?
- ?? An explanation of what the line `s.vel = vector(...)`, does.

HOMEWORK #6, PHY607, Fall 2002 – Due by start of class Oct. 1

It is useful to be able to draw images directly. See if you can install the Python Imaging Library (www.pythonware.com/products/pil) (put the DLL's etc. into your Python22 folder on your own computer or make it accessible by modifying the classpath under IDLE) and if you can get the following (**logisticpic.py**) to work:

```
import Image, ImageDraw

mx = 600
my = 400
## 1-bit, Black/White image
im = Image.new(mode="1",size=(mx,my))

amin = 3.8
amax = 3.9
xmin = 0.1
xmax = 0.5
awidth = amax-amin
xstep = (xmax-xmin+0.)/(my-1.)
for ia in range(mx):
    xA=0.25
    a = amin+ awidth * ia/float(mx)
    for k in range(4000):
        xA = a * xA * (1.-xA)
    for k in range(2048):
        xA = a * xA * (1.-xA)
        if (xA >xmin and xA < xmax):
            ypos = (my-1) - int((xA-xmin)/xstep)
            im.putpixel((ia,ypos),1)
    if (ia % 10 ==0):
        print ia

im.save("logisticpic.jpg")
```

Once you have this working, indicate how this program works by marking it up with comments. For example, why is there a "+0." in the **xstep** initialization? Submit your marked up version.

Colorize the diagram:

- ?? Set the mode to "RGB" rather than "1"
- ?? Make a list **colors** of 7 colors: tuples with 3 integer entries from 0 to 255 in each entry (RGB strengths out of 256.)
- ?? Replace '1' as the color in put.pixel with a selection from your color palette. Say by the (k%7)th color? View a larger portion of the bifurcation diagram (larger a and x range.)