CW2: Bouncing ball with Air Drag [module, if, nested structure]

(1) Modify your computer homework 1, change the code that draws the ball to

```
ball = sphere(radius = size,  color=color.red, make_trail=True)
```

Now the ball leaves a trail. More about trail is at http://vpython.org/contents/docs/trail.html.

(2) Ball hitting and bouncing off the ground. Underlined codes are new to you. Again typing the entire program rather than cut-and-pasting can accelerate your learning.

Before you run the following codes, save the program and choose "模組" in http://VPhysics.ntu.edu.tw and download "ruler.py" into the directory where your program is.

```
from visual import *
import ruler

g=9.8               # g = 9.8 m/s^2
size = 0.25         # ball radius = 0.25 m
scene = display(title='bouncing projectile', center = (0,5,0),width=1200, height=800, background=(0.5,0.5,0))
floor = box(length=30, height=0.01, width=4, color=color.blue)                    # floor
ball = sphere(radius = size,  color=color.red, make_trail=True)                   # ball
ruler1 = ruler.ruler(vector(-15, 0, 1), vector(1,0,0), unit = 2.0, length = 30.0, thickness = 0.2)       # ruler1
ruler2 = ruler.ruler(vector(-15, 0, 1), vector(0,1,0), unit = 1.0, length = 10.0, thickness = 0.2)       # ruler2

ball.pos = vector( -15.0, 10.0, 0.0)   # ball initial position
ball.v = vector(2.0, 0.0 , 0.0)        # ball initial velocity, vx = 2.0 m/s, vy=0, vz=0
dt = 0.001
while ball.pos.x < 15.0:     # simulate until x=15.0m
   rate(1000)
   ball.pos +=  ball.v*dt
   ball.v.y +=  - g*dt
   if ball.y <= size and ball.v.y < 0:            # check if ball hits the ground
      ball.v.y = - ball.v.y                       # if so, reverse y component of velocity

print 'end'
```

1. module

```
import ruler
```

*** Python and VPython provides many useful modules, such as the "visual" at the first line. However, sometimes when we need some frequently-used feature not provided, we can write the codes down and save it for future use. This piece of codes saved in the name of xxx.py is called a module, such as "ruler.py". When we need it, we just import it by writing "import xxx". We will talk about how to write module in the future.

Ruler is a feature that VPython does not provide, therefore I wrote one for you. The following code is an example

```
ruler1  = ruler.ruler(pos=vector(-1,0,0), axis=vector(0,1,0), unit=1.0, length=10.0, thickness=0.1)
```

This draws a ruler with an end point at (-1, 0, 0), pointing in direction of (0, 1, 0). The unit of the ruler is 1.0, the total length is 10.0 and the thickness of the ruler is 0.1. You can modify this to suit for your purpose.

*** In the code, the first ruler in "ruler.ruler" is the module name, the second ruler is the real thing (called class, will be taught later) that draws the ruler. The ruler object's name is ruler1. The way of presentation is very similar to that of ball.pos.

2. "if" command

```
   if ball.y <= size and ball.v.y < 0:            # check if ball hits the ground
      ball.v.y = - ball.v.y                       # if so, reverse y component of velocity
```

*** "if" is similar to "while" but only executed once. If the condition is satisfied ("True"), the associated codes (indented codes below colon) are executed once. Then the program moves on.

*** Here shows the nest structure of Python. Below the colon of “while” is the first-level nest of codes, which include several lines and “if”. Below the colon of “if” is the second-level nest of codes. There can be as many levels of nests in a programs. Therefore, it is really important that you line up the codes according to the codes’ nest level, then you know which subordinate section belongs to which part. In Python editor, we use Tab key to set the indention, one Tab for one indention and two for two, …

Practice:
(1) Modify the program,
Now, you are going to do a program that simulates the ball (initially on the ground) flying over a flat horizontal ground and the ball stops when it hits the ground for the first time, while the air drag $F_{air\_drag} = Cv^{power}$ needs to be considered, where *C* is the coefficient, and *v* is the velocity.

You need to set the following initial condition (Notice that all the physical quantity must be in SI unit)

```
v_initial          # for the initial velocity magnitude
theta              # for the initial angle that the projectile is launched over the ground
                   # you can use  sinusoidal functions to set up the initial velocity vector, such as
                   # ball.v = v_initial * vector(cos(theta_initial), sin(theta_initial), 0)
drag_coef
drag_power         # the drag force exerted on the projectile sets an acceleration  with
                   # magnitude = drag_ coef * v ** drag_power and the acceleration
                   # is anti-parallel to the velocity
```

You will make a simulation of the projectile flying over the distance. And in the end of the program, you need to print how far the projectile flies.

(2) Continue from (1), Find the optimal angle of theta such the ball reaches the longest distance when drag_coef = 0.3 and drag_power = 1.0

Homework submission:
Continuing from Practice (2), simulate and print the distance that the ball travels horizontally when the ball touches the ground for the third time after launching.
Hint: You need to add a counter to record how many times the ball touches the ground.