VP2: Bouncing Ball with Air Drag [if, nested structure]

Video: https://goo.gl/dvJyPf

I. Bouncing Ball:

```
from vpython import *

g=9.8      # g = 9.8 m/s^2
size = 0.25 # ball radius = 0.25 m

scene = canvas(center = vec(0,5,0), width=600, background=vec(0.5,0.5,0))
floor = box(length=30, height=0.01, width=4, color=color.blue)
ball = sphere(radius = size,  color=color.red, make_trail=True, trail_radius = size/3)

ball.pos = vec( -15.0, 10.0, 0.0)       # ball initial position
ball.v = vec(2.0, 0.0 , 0.0)          # ball initial velocity
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.pos.y <= size and ball.v.y < 0:          # new: check if ball hits the ground
      ball.v.y = - ball.v.y                         # if so, reverse y component of velocity
```

1. “if” command

```
   if ball.pos.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 is executed once. If the condition (between if and colon :) is “True”, the associated codes (indented codes below colon) are executed once. Then the program moves on.

*** Here shows the nested 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 many levels of nests in a program. Therefore, it is really important that you line up the codes accordingly, then you know which subordinate sections belong to which nests. You can use Tab key on the keyboard to yield the proper indention.

Here this ‘if’ checks whether the ball’s central position is smaller than the ball’s radius and still going down. If so, it means the ball touches the ground and gets reflected by the ground. Here, we assume this is an elastic collision, therefore ball.v.y is reversed in sign with the same magnitude.

2.

```
ball.pos +=  ball.v*dt
```

Inside the while nest, you see the above code. It yields the same result as the one, `ball.pos =  ball.pos + ball.v*dt`, which is used in VP1 but with a subtle difference. This difference sometimes causes some undetectable troubles to newbies. More of this will be discussed when we introduce list.

II. Graph:

```
from vpython import *

scene1 = canvas(width = 200, align = 'left', background = vec(0, 0.5, 0.5))
scene2 = canvas(width = 300, height = 300, align = 'left', background = vec(0.5, 0.5, 0))
box(canvas = scene1)
sphere(canvas = scene2)
oscillation = graph(width = 450, align = 'right')
```

```
funct1 = gcurve(graph = oscillation, color=color.blue, width=4)
funct2 = gvbars(graph = oscillation, delta=0.4, color=color.red)
funct3 = gdots(graph = oscillation, color=color.orange, size=3)
t = 0
while t < 80:
   rate(50)
   t = t+1
   funct1.plot( pos=(t, 5.0+5.0*cos(-0.2*t)*exp(0.015*t)) )
   funct2.plot( pos=(t, 2.0+5.0*cos(-0.1*t)*exp(0.015*t)) )
   funct3.plot( pos=(t, 5.0*cos(-0.03*t)*exp(0.015*t)) )
```

In this program, we see multiple plots and graphs on the same page of the browser. In certain physics simulations, this is very necessary, because we want to see the trend or change of some physical quantities.

1.

```
scene1 = canvas(width = 200, align = 'left', background = vec(0, 0.5, 0.5))
scene2 = canvas(width = 300, height = 300, align = 'left', background = vec(0.5, 0.5, 0))
box(canvas = scene1)
sphere(canvas = scene2)
```

We want to have three figures on the same page, so we align the first two to the left by using the option 'align' in canvas. The objects 'box' and 'sphere' are to be drawn in scene1 and scene2, respectively, so in their creation, the option 'canvas' are designated accordingly.

2.

```
oscillation = graph(width = 450, align = 'right')
```

This creates a graph window called 'oscillation', with 450 pixels on screen width and aligned to the right.

```
funct1 = gcurve(graph = oscillation, color=color.blue, width=4)
```

funct1 creates a curve plot to be drawn in 'oscillation', with color blue and line width equal to 4.

```
funct2 = gvbars(graph = oscillation, delta=0.4, color=color.red)
```

funct2 creates bars to be drawn in 'oscillation', with color red and bar width equal to 0.4.

```
funct3 = gdots(graph = oscillation, color=color.orange, size=3)
```

funct3 creates dots to be drawn in 'oscillation', with color orange and dot size equal to 3.

3.

```
   funct1.plot( pos=(t, 5.0+5.0*cos(-0.2*t)*exp(0.015*t)) )
```

will generate data point at position = pos, with the first value (t) being the horizontal-axis value and the second value (`5.0+5.0*cos(-0.2*t)*exp(0.015*t)`) being the vertical-axis value. Similarly for

```
   funct2.plot( pos=(t, 2.0+5.0*cos(-0.1*t)*exp(0.015*t)) )
   funct3.plot( pos=(t, 5.0*cos(-0.03*t)*exp(0.015*t)) )
```

III. Air_drag

```
from vpython import *
g=9.8              # g = 9.8 m/s^2
size = 0.25        # ball radius = 0.25 m
height = 15.0      # ball center initial height  = 15 m
C_drag = 1.2
scene = canvas(width=600, height=600, center =vec(0,height/2,0), background=vec(0.5,0.5,0))
floor = box(length=30, height=0.01, width=10, color=color.blue)
ball = sphere(radius = size, color=color.red, make_trail = True)
ball.pos = vec(-15, size, 0)
ball.v = vec(16, 16, 0)                        # ball initial velocity

dt = 0.001                              # time step
while ball.pos.y >= size:                                  # until the ball hit the ground
   rate(1000)                                  # run 1000 times per real second
   ball.v += vec(0, -g, 0) * dt - C_drag*ball.v*dt
```

```
    ball.pos += ball.v*dt
msg = text(text = 'final speed = ' + str(ball.v.mag), pos = vec(-10, 15, 0))
```

We know that the acceleration due to the air-drag force acted on an object is of the opposite direction to the object's velocity and the acceleration is positively correlated to the object's speed. Here we assume that the correlation is linear, therefore an additional velocity change - C_drag*ball.v*dt (- C_drag*v is the deceleration due to air drag) is added to the velocity formula, ball.v += vec(0, -g, 0) * dt - C_drag*ball.v*dt. With this, we will be able to simulate a flying object under the influence of air-drag.

msg = text(text = 'final speed = ' + str(ball.v.mag), pos = vec(-10, 15, 0))

This code shows in the end the final speed of the ball. The text expression is 'final speed = ' + str(ball.v.mag). ball.v.mag is the magnitude of the velocity, i.e. the speed. The function str() change the number into a text string and this is added to 'final speed = ' for the entire message.

IV. Homework

(must): A ball, whose radius is size = 0.25, at initial position = vec(-15, size, 0), with initial velocity ball.v = vec(20*cos(theta), 20*sin(theta), 0) and theta = pi/4, is launched with a linear air drag of drag-Coefficient C_drag = 0.9 . When the ball hits the ground, it bounces elastically (i.e. without losing any energy). Plot the trajectory of the ball before the ball has hit the ground for 3 times, and (1) find the distance of the displacement of the ball and (2) show the height (ball.pos.y) when the ball has reached the highest point. In the same page, also plot a graph of speed (magnitude of the velocity) of the ball versus time.

(optional):

If a ball is dropped freely, i.e. the initial velocity = vec(0, 0, 0), in air with drag-Coefficient C_drag = 0.3, plot a graph of its speed vs time and print its terminal velocity in the shell.