VP1: Simple Start for Vpython, Projectile [basic of Python, while, if, and nest structure]

Follow the steps (1) in "安裝" at http://tcjd71.wixsite.com/vpython/install to install python3 + vpython7, or (2) anaconda + vpython (https://vpython.org/presentation2018/install.html)

Video: https://www.youtube.com/playlist?list=PLxowpOHFnGyM_fBj8r3JE9sZL_NITEMJR

I. Free Fall:

Type (typing instead of "cut-and-pasting" allows you to know python faster) and then run the codes. Holding the right mouse button and moving the mouse can change view angle. Holding both buttons and moving the mouse can zoom in or out.

```
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

scene = canvas(width=800, height=800, center =vec(0,height/2,0), background=vec(0.5,0.5,0))   # open a window
floor = box(length=30, height=0.01, width=10, color=color.blue)                               # the floor
ball = sphere(radius = size,  color=color.red, make_trail = True, trail_radius = 0.05)        # the ball
msg =text(text = 'Free Fall', pos = vec(-10, 10, 0))

ball.pos = vec( 0, height, 0)                 # ball center initial position
ball.v = vec(0, 0 , 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.pos = ball.pos + ball.v*dt
   ball.v.y = ball.v.y - g*dt

msg.visible = False
msg =text(text = str(ball.v.y), pos = vec(-10, 10, 0))
print(ball.v.y)
```

Code explanation:

1. Declaring using VPython module. All our simulation programs will have this first line

```
from vpython import *
```

2. Setting constants. For convenience, all physical quantities in the simulation world are always with SI units. Texts after # will not run but are remarks

```
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
```

3. Opening a window.

```
scene = canvas(width=800, height=800, center = vec(0,height/2,0), background=vec(0.5,0.5,0))
```

opens a window named scene with 800 horizontal pixels and 800 vertical pixels. Initially in the simulation world, +x axis is to the right is, +y to the top, +z pointing out of the screen. center is the position vector of the center of the simulation world. vec(x, y, z) represents a vector.

background sets the background color to vec(red, green, blue), each of the color is scaled from 0.0 to 1.0. **Always set this attribute to some color, otherwise the background defaults to black, making results difficult to see.**

4. Objects in simulation world.

```
floor = box(length=30, height=0.01, width=10, color=color.blue)                               # the floor
```

This draws a box of length = 30 (in x), height = 0.01(in y), and width = 10 (in z) called floor. You may use

floor.pos to assign its center, e.g. floor.pos = vec(1,0,0). Without this, the center defaults at vec(0,0,0). Note that, in Python, A.B means the "attribute B of A". vector() or vec() is used to present a vector, such as a=vector(1, 2, 3), in which all three components are float (i.e. here 1 is 1.0,…). More, a.x means the x component of a. We can use print(a.x) to show the x component of vector a or a.x = 5 to set the x component of a to 5. floor.pos is also a vector, therefore floor.pos.x is the x component of floor.pos.

ball = sphere(radius = size, color=color.red, make_trail = True, trail_radius = 0.05)
This draws a sphere named ball, with radius = size and color=color.red. Later, we may assign the center position of the ball, such as ball.pos = vector(1, 0, 0), and we can also attach more attributes to ball, such as ball.v = vector(2, 0, 0). Attribute make_trail = True will draw a trail for the object. The thickness of the trail is set by trail_radius.

msg =text(text = 'Free Fall', pos = vec(-10, 10, 0))
This code shows a message with a text content 'Free Fall' at pos = vec(-10, 10, 0).

5. Start the simulation

```
ball.pos = vec( 0, height, 0)        # ball center initial position
ball.v = vec(0, 0 , 0)               # ball initial velocity
```

These two set the initial conditions.

dt = 0.001
dt sets how much of the real time elapses in one step in the following while loop. The size of dt depends on the time scale of the simulation events. Too small, the simulation takes too long. Too large, the simulation will be too rough and cause incorrect results. For the free fall, an event of several seconds, dt = 0.001 is just fine. For atom collision events in 10^{-11} seconds, dt should be 10^{-14}. For Earth to circle around the sun it takes about 10^{7} seconds, then dt = 10^{3} is fine.

while ball.pos.y >= size:
We use the "while" loop command all the time. The condition between while and colon(:) is tested. If it is satisfied, all **the indented codes (associated codes)** below colon are executed once. Then the condition will be tested again and the process will repeat until the condition is no longer satisfied (here, it means that the y component of the ball's center position is no longer larger than the ball radius, meaning the ball touches the floor). At this moment, Vpython stops executing the while loop and its associated codes, but then to continue to the next section of the codes (here, it is msg.visible = False)
In Python, **indentation of a section of codes** (you can do this by press tab key) means this section of codes is associated with the previous line of code with colon (:).

rate(1000)
This sets the while loop to run 1000 times per real-world second. With dt=0.001, this simulation runs at a speed of 1000*0.001 = 1 of real-world time, meaning the result is presented as the real-world speed. If rate(500), 500*0.001 = 0.5, then the result is presented at a slow motion of 0.5 real-world speed.

```
ball.pos = ball.pos + ball.v*dt      Let ball.pos to increase ball.v*dt in one dt
ball.v.y = ball.v.y - g*dt           Let ball.v.y to increase -g*dt in one dt.
                                     These are the fundamental equations of the kinetics of moving bodies
```

```
msg.visible = False
msg =text(text = str(ball.v.y), pos = vec(-10, 10, 0))
```

After the "while" loop stop running due to the unsatisfactory condition, these two lines of code make the previous message text "Free Fall" invisible and then show at the same position the y component of the ball's velocity. Here, str() transforms a number to a text. For example, str(5.5) gives you a text string ='5.5'

You can also print this value on the shell screen by

print(ball.v.y)

II. Arrow:

```
from vpython import *

scene = canvas(width=800, height=800, background=vec(0.5,0.5,0))                    # open a window
a1 = arrow(color = color.green, shaftwidth = 0.05)
b1 = arrow(color = color.blue, shaftwidth = 0.05)

a1.pos = vec(1, 1, 0)
a1.axis = vec(1, -1, 0)
b1.pos = a1.pos + a1.axis
b1.axis = vec(2, 1, 0)

c1 = arrow(color = color.yellow, shaftwidth=0.05)
c1.pos = a1.pos
c1.axis = a1.axis + b1.axis
```

In this program, it draws two arrows, a1 and b1, with color being green and blue, respectively and with shaftwidth = 0.05. arrow has attributes like pos, axis, and color, e.g. a1.pos = vector(1, 1, 0) makes the starting point of a1 at (1, 1, 0), a1.axis = vec(1, -1, 0) draws a1 as a vector of (1, -1, 0). Similarly, b1 starts at a1's arrow tip and has an axis of vector vec(2, 1, 0). If you follow the codes for arrow c1, you can find that this is actually a representation of a vector addition, vector a1 + vector b1 = vector c1.

III. 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 when the ball is going down. If so, it means the ball touches the ground and needs to get 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 ball.pos = ball.pos + ball.v*dt, which is used in II but with a subtle difference. This difference sometimes causes some undetectable troubles to newbies. More about this issue will be discussed when we introduce list.

IV. 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)) )
```

V. 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.

VI. Homework:

You need to hand in your homework with a filename extension '.py'. If you are writing your homework in Jupyter, you need to extract the complete runnable codes and save them in just one '.py' file.

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.

(1) Plot the trajectory of the ball and stop the ball when it hits the ground for 3 times. Also add at the center of the ball an arrow, which moves along with the ball and whose length is proportional (proportional constant by your choice) to and parallel to the velocity vector of the ball.
(2) Plot a graph of speed (magnitude of the velocity) of the ball versus time.
(3) Show both the displacement of the ball and the total distance travelled by the ball. Note: in python, x**p means x to the power p = x^p.
(4) Show the largest height of the ball during the entire process.