Computer Work Unit 8 Electromagnetically Induced Transparency [empty class, list representation]

I. Python Usage [List representation]

(1) range(). Notice the number in range should be integer (int)

```
L = range(5)                               # range(5) = [0, 1, 2, 3, 4].
L = range(4, 9)                            # = [4, 5, 6, 7, 8]
L = range(1, 6, 2)                         # = [1, 3, 5]    1 to 6 every two numbers
```

(2) list representation. Sometimes we want to generate a list with some conditions, e.g.

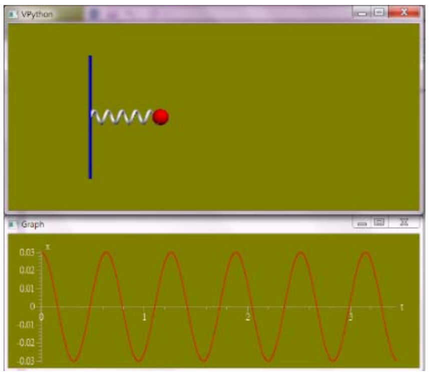
```
L = [i**2  for  i  in  range(5)]           # =  [0, 1, 4, 9, 16]
L = [0.1*i*pi   for  i   in   range(-3, 3)]   # =  [-0.3*pi, -0.2*pi, -0.1*pi, 0, 0.1*pi, 0.2*pi]
L = [i**2  for  i  in  range (5)  if  i != 3]  # =  [0, 1, 4, 16]
```

(3) List representation can be used in a nested structure, or even for dictionary and tuple. e.g.

```
L = [i*10 + j for i in range(3) for j in  range(5) ]   # =  [0, 1, 2, 3, 4, 10, 11, 12, 13, 14, 20, 21, 22, 23, 24]
D = {i:i**2  for i  in [0, 1, 2]}                      # = {0:0,   1:1, 2:4}
```

II. Practice

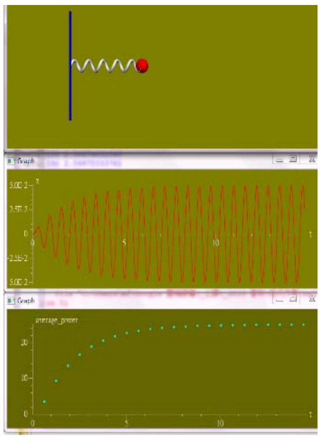
```
from visual import *
from visual.graph import*
size, m = 0.02, 0.2              # ball size =  0.02 m, ball mass = 0.2kg
L, k = 0.2, 20.0                 # spring original length = 0.2m, force constant = 20 N/m
amplitude = 0.03

scene1=gdisplay(y=400,width=800,height=300,xtitle='t',ytitle='x',background=(0.5,0.5,0))
x=gcurve(color=color.red,gdisplay=scene1)
scene = display(width=800, height=400, fov = 0.03, range = 0.5, center=(0.3, 0, 0), background=(0.5,0.5,0))
wall_left = box(length=0.005, height=0.3, width=0.3, color=color.blue)         # left wall
ball = sphere(radius = size,  color=color.red)                                 # ball
spring = helix(radius=0.015, thickness =0.01)                                  #spring
wall_left.pos = vector(0, 0, 0)
ball.pos, ball.v, ball.m = vector(L + amplitude, 0 , 0), vector(0, 0, 0), m
spring.pos = wall_left.pos

t, dt = 0, 0.001
while True:
   rate(1000)
   spring.axis = ball.pos - spring.pos               # spring extended from spring endpoint A to ball
   spring_force = - k * (mag(spring.axis) - L) * norm(spring.axis)      # spring force vector
   ball.a = spring_force / ball.m                    # ball acceleration = spring force /m
   ball.v +=   ball.a*dt
   ball.pos += ball.v*dt
   t += dt
   x.plot(pos=(t,ball.pos.x - L))
```



Modified from hw3, this simulates the oscillation of a horizontal spring, with a given amplitude. The "ball position versus time" is done by the bold part. Reference: http://vpython.org/contents/docs/graph.html

(1) Now, inaddition to the restoring force from the spring, add the air resistance force $\vec{f} = -b\vec{v}$ to the ball with **damping factor** $b = 0.05m\sqrt{k/m}$, m the ball mass, and k the spring constant. What happens then?

(2)Rather than letting the ball to oscillate from the initial amplitude, let the ball to be initially at $x = L$ and a sinusoidal force $\vec{F} = f_a \sin(\omega_d t)$ is exerted on the ball with f_a = 0.1 and ω_d = $\sqrt{k/m}$. When this force $\vec{F}$ is exerted on an object of velocity $\vec{v}$, the power consumed on the object by the force is $P = \vec{F} \cdot \vec{v}$. Calculate the consumed power averaged over a period $T = 2\pi / \omega_d$ at the end of each period. Use the following code to generate a dot graph. gdots() and gcurve() have similar usage, except the former one plots dots and the later one plots curve.



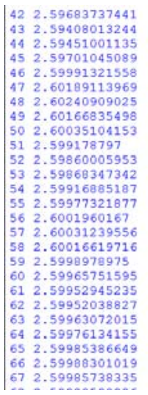
```
scene2=gdisplay(y=700,width=800,height=300, xtitle='t',ytitle='average_power',background=(0.4,0.4,0))
p=gdots(color=color.cyan,gdisplay=scene2)                #add these two lines before scene1=....
……
p.plot(pos=(t, average_power))          # put this where it plots a data point when the power averaged over a period is calculated.
```

Observe your simulation with different settings, e.g. ω_d =0.8, 0.9, 1.0, 1.1, or 1.2 times of $\sqrt{k/m}$ or with different *b*. Think about the results. Before doing (3), change ω_d and *b* to their original values.

(3) Often, we want to change a graphical simulation, which is slow due to graphical plotting, to a calculation. We can delete the codes that open graphical windows, delete rate(1000), and replace codes that generate visual objects by codes that generate objects from an empty class.

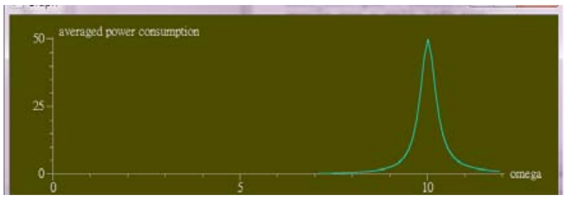
```
class obj:   pass

wall_left, ball, spring = obj(), obj(), obj()
```

With these, we still do the same simulation but do not plot them. This speeds up the simulation. Here, instead of plotting the average consumed power, we now only print the result at the end of each period. As you can see after certain number of periods, the system reaches a **steady state** and the consumed power is almost a constant.



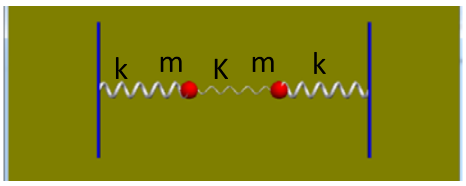
```
42 2.59683737441
43 2.59408013244
44 2.59451001135
45 2.59701045089
46 2.59991321558
47 2.60189113969
48 2.60240909025
49 2.60166835498
50 2.60035104153
51 2.599178797
52 2.59860005953
53 2.59868347342
54 2.59916885187
55 2.59977321877
56 2.6001960167
57 2.60031239556
58 2.60016619716
59 2.5998978975
60 2.59965751595
61 2.59952945235
62 2.59952038827
63 2.59963072015
64 2.59976134155
65 2.59985386649
66 2.59988301019
67 2.59985738335
```

(4) Let `omega = [0.1*i + 0.7*sqrt(k/m) for i in range(1, int(0.5*sqrt(k/m)/0.1))]` and by using "`for omega_d in omega:`", perform the calculation of the consumed power as practice (3) for different ω_d. Do not print the result. Instead, for each ω_d, when the system reaches steady state, plot the result in the "**steady-state** power consumed averaged over a period versus omega_d" and then go to calculate for the next ω_d. You will get something similar to the figure, which shows clearly the system's response to different driving frequency ω_d. When the driving frequency is near the resonance frequency, the consumed power is the highest. See https://goo.gl/Y5VZTi



Homework submission:

(1) From Practice (2), add an addition ball, two more springs, and a right wall. All the parameters are the same as in the original program. The middle spring is of constant K = 5.0. The damping factor for ball 1 (the left one) $b_1 = 0.05$ and for ball 2 (the right one) $b_2 = 0.025$. The external force $\vec{F} = f_a \sin(\omega_d t)$ is exerted on ball 1 with f_a = 0.1 and ω_d = $\sqrt{(k+K)/m}$ as in Practice (2) except that the driving frequency is corresponding to the resonance frequency of the combining effect due to the left and the middle springs. Observe the oscillation and the power consumption, what do you find?



(2) Let `omega_d` be each element in `omega = [0.1*i + 0.7*sqrt((k+K)/m) for i in range(1, int(0.5*sqrt((k+K)/m)/0.1))]` (a bit different from Practice(4)) plot the "**steady-state** power consumed averaged over a period versus omega_d" as you did in Practice (4) for the system in Homework (1). What do you find? You can compare this to the result of fixing the position of ball 2 you obtained from practice(4).

Save(1) and (2) in different files but zip them together and submit the zipped file.

Working out the homework (1) and (2), we will observe the principle behind "EIT" (Electromagnetically Induced Transparency), a very interesting and advanced research topic in optical physics that won many awards. You may read https://en.wikipedia.org/wiki/Electromagnetically_induced_transparency for the mechanism of EIT and see the similarity between the EIT and our simulation here.