Computer Work for Unit 6 Rotation [Dict, Class]

I. Python Dictionary

Dictionary is a key-to-value mapping data type. For example,

```
d1 = {1:'a', 2:'b'}
d2 = dict(1='a', 2='b')
d3 = dict(); d3[1] = 'a';  d3[2]='b'
```

d1, d2, and d3 are the same dictionary generated in different ways. The usual operation is shown in below:

```
d = {"earth":1, "mars":2, "halley":3}
print(d)
print(d["earth"])
d["sun"] = "s"
print(d)
del d["earth"]
print(d)
print("earth" in d)
print("earth" not in d)
print(len(d))
```

d[key]	gets the *value* stored in *d* at index *k*
d[key] = value	sets *d*'s value at index *key* as *value*
key in d	gives True if *key* is in *d*, else gives False
key not in d	gives True if *key* is not in *d*, else gives False
len(d)	gives the number of stored data in *d*
dict.copy()	return a complete copy of dict.

II. Python Class

In the following program, we define our *class* for the first time. Actually, we have used ***class*** in every previous homework, including ***sphere, box, helix,*** provided by the visual module. The following is the free fall program (from homework 1) with a modification to include ***class***. Usage of class is addressed in the remarks.

```
from visual import *
g, size, height = 9.8, 0.25, 15.0

class any_ball(sphere):         # declare any_ball a class with properties (attributes and methods) inheriting from sphere

   m = 1.0                      # if objects generated by any_ball do not have their own attributes m and v, it will use the class's
   v = vector(0,0,0)            # attributes here, i.e., m and v

   def __init__(self, *args, **kargs):      # Initializing function is called whenever an object of this class is generated. self means the object
                                            # to be generated. *args and **karges are the arguments passed by positions and by keywords.
      self = sphere.__init__(self, *args, **kargs)   # Let self  (the generated object) be initialized by the sphere's initializing function.
                                                     # It also passes all the arguments there.

   def kinetic_energy(self):    # declaring a method, which is a function belonged to a specific class. The way it works is the same as a function
                                # except that its first argument is always self, which is the object calling this method.
      return 0.5 * self.m * abs(self.v)**2

scene = display(width=800, height=800, center = (0,height/2,0), background=(0.5,0.5,0))
floor = box(length=30, height=0.01, width=10, color=color.blue)

ball = any_ball(radius = size,  color=color.red)    # generating ball as an object belonged to any_ball class, and passing arguments into
                                                    # any_ball's initializing function __init__().
print ball.m, ball.v, ball.kinetic_energy()         # print ball.m, ball.v, but since ball does not have attributes m and v yet. It uses any_ball
                                                    # class's  m and v. ball.kinetic_energy() calls the kinetic_energy method.

ball.pos = vector( 0, height, 0)
ball.v = vector(0, 0 , 0)
ball.m = 3.0
dt = 0.001
while ball.pos.y >= size:
   rate(1000)
   ball.pos += ball.v*dt
   ball.v.y +=  - g*dt

print ball.m, ball.v, ball.kinetic_energy()         # print ball.m, ball.v, now ball has its attributes m and v. And with these two attributes
                                                    # printing the ball.kinetic_energy()
```

III. Practice

You may need several Vpython's vector methods (http://vpython.org/contents/docs/vector.html) for the following work to simulate Kepler's laws. The gravitation force acted on the planets and comets by the Sun is $\vec{F} = -G\frac{Mm}{r^2}\hat{r}$. Now, you use the real values for the Sun, Earth, Mars, and the comet Halley to simulate their orbits around the Sun.

1. Construct a dictionary *mass* to contains the mass of sun, earth, mars and comet halley. Construct dictionaries *d_at_ perihelion* and *v_at_ perihelion* to contain the distances and the velocities of earth, mars, and comet halley relative to the sun, respectively, when they are at the perihelion. Google and find the proper values by yourself.

2. Construct a function G_force(m, position_vector) that returns the force vector exerted on an object of mass *m* by the sun when it is at the position_vector. Test your function for the earth when position_vector = vector(d_at_perihelion['earth'], 0, 0) in G_force(mass['earth'], position_vector).

3. Construct a class as_obj (short for astronomical object) inheriting from sphere. This class has two additional methods, potential_energy and kinetic_energy, which return the potential energy and kinetic energy of the astronomical object, respectively. Test your class by:

(1) Open a window by scene = display(.....).

(2) From class sphere generate an object named sun at position vector(0,0,0) and from class as_obj generate an object named earth at position vector(d_at_perihelion['earth'], 0, 0). **Note 1:** It is better to scale up the radius for the sun and earth, for you to see clearly on the screen the earth and the sun when they are at the real distance. **Note 2:** You may go to http://vpython.org/contents/docs/materials.html to choose suitable material characteristic to make your sun glow and your earth looking like real earth.

(3) Let earth.m = mass['earth'] and earth.v = vector(0, 0, - v_at_perihelion['earth']). Check (by printing) if earth.kinetic_energy() and earth.potential_energy() return the correct result.

(4) Now simulate the earth orbits around the sun. You need to choose the proper dt = 86400.0. You can use scene.forward = vector(0, -1, 0) to change the initial view angle.

IV. Homework submission.

In addition to earth, now generate two more objects, mars and halley, from class as_obj with the proper mass, d_at_perihelion and v_at_periheilion, and let them orbit around the sun.

1) When the earth and marts finish their own first cycle of orbit, obtain by your program and print the values of T^2 / r^3 for Earth and Mars. T and r are the period and radius of the orbit, respectively.

2) When comet halley reaches the perihelion and the aphelion and the half-way point, print its kinetic energy, potential energy and total energy.

3) Obtain by your program and print the areas swept by the connecting line between Halley and the Sun for an earth day when comet Halley reaches the perihelion and the aphelion and the half-way point.