VP5: Kepler's Planet motion, and Precession of Moon's orbit [Dict, Class]

I. Python Dictionary

1. Dictionary is a key-to-value mapping data type. You can generate the dictionary by d = {key1:value1, key2:value2, key3:value3,…}. For example, astro_obj is a dictionary with "earth", "mars", and "halley" as its keys, and the corresponding values are 1, 2, and 3, respectively.
2. d[key] gets the value stored in d indicted by key.
3. d[key] = value assigns the key in d to value.
4. del d[key] deletes the key and value all together in d.
5. key in d yields True if key is in d, else gives False (Note: This can also be used for list).
6. key not in d yields True if key is not in d, else gives False (also can be used for list).
7. len(d) gives the number of stored data in d (also can be used for list).
8. dict.copy() returns a complete copy of dict.

```
In [1]: astro_obj = {"earth":1, "mars":2, 'halley':3}
        astro_obj
Out[1]: {'earth': 1, 'halley': 3, 'mars': 2}

In [2]: print(astro_obj['earth'])
        1

In [3]: astro_obj['sun'] = 's'
        astro_obj
Out[3]: {'earth': 1, 'halley': 3, 'mars': 2, 'sun': 's'}

In [4]: del astro_obj['earth']
        astro_obj
Out[4]: {'halley': 3, 'mars': 2, 'sun': 's'}

In [5]: 'halley' in astro_obj
Out[5]: True

In [6]: 'earth' in astro_obj
Out[6]: False

In [7]: 'earth' not in astro_obj
Out[7]: True

In [8]: len(astro_obj)
Out[8]: 3
```

II. Python Class

In the following program, we define a *class* for the first time in this series of course. Actually, we have used ***class*** in every previous homework, which are ***sphere, box,*** and ***helix,*** provided by the vpython module (they are called to work by the code "from vpython import *"). The following is the free fall program (from VP1) with a modification to include ***class***. Usage of class is addressed in the remarks. Only those in **bold red** are new to you.

```
from vpython import *
g, size, height = 9.8, 0.25, 15.0
class any_ball(sphere):          # declare any_ball a class with properties (attributes and methods) inherited from sphere
   m = 1.0                       # if objects generated by any_ball do not have their own attributes m and v, they will inherit class's
   v = vector(0,0,0)             # attributes here, i.e., m and v
   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, the object calling this method.
      return 0.5 * self.m * mag(self.v)**2

scene = canvas(width=800, height=800, 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 = any_ball(radius = size,  color=color.red)    # generate 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, ball.v, ball.m = vector( 0, height, 0), vector(0, 0, 0), 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
                                                    # this prints ball.kinetic_energy() as well
```

III. Kepler's Law

This program uses function, class, and dictionary to construct a solar model including planets Earth and Mars and comet Halley.

```
from vpython import*
G=6.673E-11
mass = {'sun': 1.989E30,'earth': 5.972E24, 'mars':6.4169E23, 'halley':2.2E14}
d_at_perihelion = {'earth': 1.495E11, 'mars':2.279E11, 'halley': 8.7665E10}
v_at_perihelion = {'earth': 2.9783E4, 'mars':2.4077E4, 'halley': 54563.3}
```

```
def G_force(m, pos_vec):
    return -G * mass['sun'] * m / mag2(pos_vec) * norm(pos_vec)

class as_obj(sphere):
    def kinetic_energy(self):
        return 0.5 * self.m * mag2(self.v)
    def potential_energy(self):
        return - G * mass['sun'] * self.m / mag(self.pos)

scene = canvas(width=800, height=800, background=vector(0.5,0.5,0))
scene.lights = []
sun = sphere(pos=vector(0,0,0), radius = 3.0E10, color = color.orange, emissive=True)
local_light(pos=vector(0,0,0))
earth = as_obj(pos = vector(d_at_perihelion['earth'],0,0), radius = 1.5E10, m = mass['earth'], texture={'file':textures.earth}, make_trail = True)
earth.v = vector(0, 0, - v_at_perihelion['earth'])
mars = as_obj(pos = vector(d_at_perihelion['mars'],0,0), radius = 1.0E10, m = mass['mars'], color = color.red, make_trail = True)
mars.v = vector(0, 0, - v_at_perihelion['mars'])
halley = as_obj(pos = vector(d_at_perihelion['halley'],0,0), radius = 0.5E10, m = mass['halley'], make_trail = True)
halley.v = vector(0, 0, - v_at_perihelion['halley'])

stars = [earth, mars, halley]
dt=60*60*6
print(earth.potential_energy(), earth.kinetic_energy())

while True:
    rate(1000)
    for star in stars:
        star.a = G_force(star.m, star.pos) / star.m
        star.v = star.v + star.a * dt
        star.pos = star.pos + star.v * dt
```

In this program, we create a function `G_force(m, pos_vec)` to find the gravitational force from sun acted on an astronomical object with mass `m` and at position `pos_vec`. We also create a class `as_obj`, inheriting all its attributes and methods from `sphere` provided by vpython, with two additional methods that calculate the `kinetic_energy` and `potential_energy`.
We also kill the original global lighting in the virtual world by `scene.lights = []`, and provide the light source at vector(0,0,0), which is the position of the sun, by `local_light(pos=vector(0,0,0))`. This will provide the proper shade for earth and mars. We also plot the earth with its full geographical feature by putting `texture={'file':textures.earth}` in its creating class when the earth object is created.

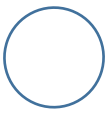
IV. Homework

(must) Moon Orbit Precession

```
mass = {'earth': 5.97E24, 'moon': 7.36E22, 'sun':1.99E30}
radius = {'earth': 6.371E6*10, 'moon': 1.317E6*10, 'sun':6.95E8*10}    #10 times larger for better view
earth_orbit = {'r': 1.495E11, 'v': 2.9783E4}
moon_orbit = {'r': 3.84E8, 'v': 1.022E3}
theta = 5.145*pi/180.0
```

The above code provides the parameters that you will need for the moon orbit precession simulation. In these, the radii of earth, moon, and sun are deliberately magnified by 10 times for better viewing. The earth_orbit provides the orbit radius ('r') and orbit speed('v') for the orbit of the earth around the sun. The moon_orbit provides the orbit radius ('r') and orbit speed('v') for the orbit of the moon around the earth. theta is the angle between the normal vector of earth orbit plane and the normal vector of moon orbit plane. Here, it is demonstrated how to construct a simulation project, building from a simple system to a more complex one step by step, to eliminate unintentional errors and to reduce debug time.

1. Construct two sphere objects, earth and moon. Let `earth.m = mass['earth']` and `moon.m = mass['moon']`. Assign `earth.pos = vector(0, 0, 0)`, `moon.pos = vector(moon_orbit['r'], 0, 0)`, `moon.v = vector(0, 0, -moon_orbit['v'])`. Applying the gravitational force acting on moon only by fixing the position



of earth, you will observe the moon orbiting around earth. You can change the initial view angle by setting scene.forward = vector(0, -1, 0) for a top-view observation. Note: Do NOT leave a trail for the moon orbit, because when later earth is going around the sun, the trail will be very messy.

2. However, we know that 1. is not entirely correct since (1) earth's position is not fixed and (2) gravitational force is also acted on the earth. Therefore, you need to modify your program for these conditions. However, if you simply add the gravitational force on earth, you will find two interesting results. One is that the earth also does a small orbiting motion, which is correct, because the center of mass of the earth-moon system is not on the center of earth. The other one is that the earth-moon system continues to move in –z direction. This is not surprising at all, because in the initial total linear momentum = earth.m * earth.v + moon.m*moon.v is non-zero and toward –z direction. You need to modify your initial conditions (i.e. earth.pos, earth.v, moon.pos, moon.v) to make the center of mass of the earth-moon system to be fixed at the Origin, i.e. vector(0, 0, 0).

3. Now, change the positions of both earth and moon to make the normal vector of the moon's orbit tilted toward +x direction with the tilting angle = theta. Then, you will see the moon orbiting earth with a tilted angle. To see this, remove(or remark) the line scene.forward = vector(0, -1, 0) to observe the tilted orbit.

   To make the observation from the point of view of earth, we can set the center of the world at the earth by scene.center = earth.pos. This way, we will see on screen a standing-still earth though the position of earth is changing all the time.

4. Now, you can add the sun, with its designated mass and its position at Origin, i.e. vec(0, 0, 0). Also set the center of mass of the earth-moon system at vector(earth_orbit['r'], 0, 0) by changing accordingly the positions of earth and moon to their suitable positions. Since now the earth-moon system is rotating around the sun, therefore remember to add the earth-orbiting velocity vector(0, 0, -earth_orbit['v']) to both earth.v and moon.v. Also remember to add the gravitational force by sun acted on both earth and moon. Keep the scene.center = earth.pos.

   Run your simulation, you will at first not see the earth and moon because as compared to the sun-earth distance, the earth and moon system is too small. Use your mouse to move closer, then you will observe the moon orbiting the earth. Wait for a bit longer, you will see that the tilting angle of the moon orbit is changing, which indicates the precession of moon's orbit.

5. Find and show the period of the precession of moon's orbit around the Earth.

Note: Now you have the orbits of the moon around the earth and the related precession, and the orbit of the earth around the sun. Combining these two will allow you to calculate the timing for moon eclipse and sun eclipse.