**All_cislunar_orbits.py**

```
import numpy as np
import math as M
import datetime

#prints the starting time
now = datetime.datetime.now()
print(now)

##constance and variables
pi = 3.14159265358979323846264338327  #cd
km = 1
m = km/1000
kg = 1
sec = 1
minute = 60*sec
hour = 60*minute
day = 24*hour
month = 30*day
year = 365*day
newtons = (kg*m)/(sec**2)
G = (6.67E-11)*newtons*(m**2)/(kg**2)
degree = pi/180
dt = 100*sec
t = 1

#mass and radius of objects
r3 = 385_000.6*km     ## distance to moon
m1 = (5.9736E24*kg)    ## earth = obj 1
m2 = (8.4E12*kg)    ## debris = obj 2
m3 = (7.34767309E22*kg)   ##obj 3 = moon
T3 = M.sqrt((4*(pi**2)*(r3**3))/(G*m1))

#mass and radius of sat.
rs = 8_000*km
T4 = M.sqrt((4*(pi**2)*(rs**3))/(G*m1))

#V of objects 1, 2, and 3 and sat
v1_old = np.array([0,0,0])
r1_old = np.array([0,0,0])

r3_old = np.array([r3*M.cos(2*pi*t/T3), r3*M.sin(2*pi*t/T3),0])
v3_old = np.array([0,2*r3*pi/T3,0])

#used for different debris settings
r_1 = 100_000*km
```

```
r_2 = 50_000*km
r_3 = 200_000*km
r_4 = 300_00*km
r_5 = 405_000*km
r_6 = 384_000*km
r_7 = 50_000*km

#list of debris position to run through
all_r2 = {
    1:np.array([-r_1,0.0,0.0]),
    2:np.array([-r_2,0.0,0.0]),
    3:np.array([r_2,0.0,0.0]),
    4:np.array([r_1,0.0,0.0]),
    5:np.array([r_3,0.0,0.0]),
    6:np.array([r_4,0.0,0.0]),
    7:np.array([r_5,0.0,0.0]),
    8:np.array([0.0,r_3,0.0]),
    9:np.array([r_6*M.cos(30*degree),r_6*M.sin(30*degree),5*km]),
    10:np.array([r_3,0.0,-r_7]),
    11:np.array([r_3,0.0,r_7]),
    12:np.array([0.0,r_3,-r_7]),
    13:np.array([0.0,r_3,r_7]),
    }
#loops for each position and velocity
for positionNum in all_r2:
    r2_old = all_r2.get(positionNum)

    for velocityNum in range(1,9):
        r = M.sqrt(np.inner(r2_old, r2_old))
        T  = M.sqrt((4*pi**2*r**3)/(G*m1))

        #list of debris v to run through
        all_v2 = {
            1:np.array([0, 2*r*pi/T,0]),
            2:np.array([0,-2*r*pi/T,0]),

            3:np.array([0, 2*r*pi/T,1.5*km/sec]),
            4:np.array([0,-2*r*pi/T,-1.5*km/sec]),


5:np.array([(M.sin(30*degree)),-(2*r*pi/T)*(M.cos(30*degree)),0]),

6:np.array([-(M.sin(30*degree)),(2*r*pi/T)*(M.cos(30*degree)),0]),


7:np.array([(M.sin(30*degree)),-(2*r*pi/T)*(M.cos(30*degree)),1.5*km/sec])
,
```

```
8:np.array([-(M.sin(30*degree)),(2*r*pi/T)*(M.cos(30*degree)),-1.5*km/sec]
),
            }
        v2_old = all_v2.get(velocityNum)

        #file saving info

        directory = '/Users/mario/Dropbox/XimenasWork/python
practice/super_computing22-23/debrisOrbits/'
        name = str(f'r{positionNum}v{velocityNum}')
        filepath = directory + name
        f = open(filepath, 'w+')#'w'
        f.close()

        crashed = False
        t = 1
        while t<= 6*month and not crashed:

            #elliptical orbit
            r21 = -(r2_old - r1_old)
            r21_length = M.sqrt(np.inner(r21,r21))
            r31 = -(r3_old - r1_old)  #Vector from 3 to 1
            r31_length = M.sqrt(np.inner(r31,r31))
            r32 = -(r3_old - r2_old)
            r32_length = M.sqrt(np.inner(r32,r32))
            if r21_length <= 6_330 * km:
                f = open(filepath, 'w+')
                f.writelines(f'#debris has crashed into the earth\n')#
                f.close()
                quit()
                crahsed  = True

            if r32_length <= 1_728 * km:
                f = open(filepath, 'w+')
                f.writelines(f'#debris has crashed into the earth\n')#
                f.close()
                quit()
                crashed  = True

            r32_hat = r32/r32_length
            r31_hat = r31/r31_length
            r12_hat = r21/r21_length


            #a of obj.1,2,3
```

```
            a3_old =  (G*m1/(r31_length**2))*r31_hat +
(G*m2/(r32_length**2))*r32_hat#moon
            a2_old = -(G*m3/(r32_length**2))*r32_hat +
(G*m1/(r21_length**2))*r12_hat#debris

            #v of obj.1,2,3
            v3_new = v3_old + a3_old*dt
            v2_new = v2_old + a2_old*dt

            r2_new = r2_old + v2_old*dt
            r3_new = r3_old + r3_old*dt

            #update to new frame
            r2_old = r2_new
            r3_old = r3_new
            v3_old = v3_new
            v2_old = v2_new
            t = t + dt

            #print info into file
            if  (int(t) % int(sec) == 0):
                f = open(filepath, 'a+')
                print(t, r2_old, r3_old, "t, r2_old, r3_old")
                f.writelines(f'{t},{r2_old},{r3_old}\n')
                f.close()
#prints the finishing time
now = datetime.datetime.now()
print(now)
```

**All_sat_orbit.py**

```
import numpy as np
import math as M
import datetime

#prints start time of program
now = datetime.datetime.now()
print(now)

##constance and variables
pi = 3.14159265358979323846264338327
km = 1
m = km/1000
kg = 1
sec = 1
minute = 60*sec
hour = 60*minute
day = 24*hour
month = 30*day
year = 365*day
newtons = (kg*m)/(sec**2)
G = (6.67E-11)*newtons*(m**2)/(kg**2)
degree = pi/180
dt = 100*sec
t = 1


#mass and radius of objects
r3 = 385_000.6*km     ## distance to moon
m1 = (5.9736E24*kg)    ## earth = obj 1
ms = (8.4E12*kg)    ## sat = object s.
m3 = (7.34767309E22*kg)   ##obj 3 = moon
T3 = M.sqrt((4*(pi**2)*(r3**3))/(G*m1))

#mass and radius of sat.
rs = 8_000*km

#V and R of objects 1, 3, and sat.
v1_old = np.array([0,0,0])
r1_old = np.array([0,0,0])

r3_old = np.array([r3*M.cos(2*pi*t/T3), r3*M.sin(2*pi*t/T3),0])
v3_old = np.array([0,2*r3*pi/T3,0])


#used for setting 1 and 2 of sat. setup
```

```
r1 = 6_330 + 2_000 * km
Ts1 = M.sqrt((4*(pi**2)*(r1**3))/(G*m1))

r2 = 6_330 + 5_000*km
Ts2 = M.sqrt((4*(pi**2)*(r2**3))/(G*m1))

# list of sat. position to run through
all_rs = {
    1:np.array([ r1, 0,0]),
    2:np.array([  0, r1,0]),
    3:np.array([-r1,  0,0]),
    4:np.array([  0,-r1,0]),
    5:np.array([ r1*M.cos(45*degree), r1*M.sin(45*degree),0]),
    6:np.array([-r1*M.cos(45*degree), r1*M.sin(45*degree),0]),
    7:np.array([-r1*M.cos(45*degree),-r1*M.sin(45*degree),0]),
    8:np.array([ r1*M.cos(45*degree),-r1*M.sin(45*degree),0]),
    9:np.array([ r2, 0,0]),
    10:np.array([  0, r2,0]),
    11:np.array([-r2,  0,0]),
    12:np.array([  0,-r2,0]),
    }

#list of sat. velocities for each position
all_vs = {
    1:np.array([0,(2*pi*r1)/Ts1,0]),
    2:np.array([-(2*pi*r1)/Ts1,0,0]),
    3:np.array([0,-(2*pi*r1)/Ts1,0]),
    4:np.array([(2*pi*r1)/Ts1,0,0]),
    5:np.array([-(2*r2*pi/Ts1)*(M.sin(45*degree)),
(2*r2*pi/Ts1)*(M.cos(45*degree)),0]),

6:np.array([-(2*r2*pi/Ts1)*(M.sin(45*degree)),-(2*r2*pi/Ts1)*(M.cos(45*deg
ree)),0]),

7:np.array([(2*r2*pi/Ts1)*(M.sin(45*degree)),-(2*r2*pi/Ts1)*(M.cos(45*degr
ee)),0]),

8:np.array([(2*r2*pi/Ts1)*(M.sin(45*degree)),(2*r2*pi/Ts1)*(M.cos(45*degre
e)),0]),
    9:np.array([0,(2*pi*r2)/Ts2,0]),
    10:np.array([-(2*pi*r2)/Ts2,0,0]),
    11:np.array([0,-(2*pi*r2)/Ts2,0]),
    12:np.array([(2*pi*r2)/Ts2,0,0]),

    }
# repeats for the length of sat. setting
for satnum in all_rs:
```

```
    rs_old = all_rs.get(satnum)
    vs_old = all_vs.get(satnum)

    #file saving info
    directory = '/Users/mario/Dropbox/XimenasWork/python
practice/super_computing22-23/debrisOrbits/'
    name = str(f'RandV{satnum}')
    filepath = directory + name

    f = open(filepath, 'w+')
    f.close()


    crashed = False
    t = 1
    while t<= 6*month and not crashed:
        #elliptical orbit
        # defines vectors for positions of object 1,3, and sat.
        rs1 = -(rs_old - r1_old)
        rs1_length = M.sqrt(np.inner(rs1,rs1))
        r31 = -(r3_old - r1_old)  #Vector from 3 to 1
        r31_length = M.sqrt(np.inner(r31,r31))
        r3s = -(r3_old - rs_old)
        r3s_length = M.sqrt(np.inner(r3s,r3s))

        r3s_hat = r3s/r3s_length
        r31_hat = r31/r31_length
        r1s_hat = rs1/rs1_length

        #writes to file: crashed if crached into moon or earth
        if rs1_length <= 6_330 * km:
            f = open(filepath, 'a+')
            f.writelines('#crashed into earth')
            f.close()
            crahsed  = True

        if r3s_length <= 1_728 * km:
            f = open(filepath, 'a+')
            f.writelines('#crashed into moon')
            f.close()
            crashed  = True

        #a of obj.1,2,3
        a3_old =  (G*m1/(r31_length**2))*r31_hat +
(G*ms/(r3s_length**2))*r3s_hat#moon
        as_old = -(G*m3/(r3s_length**2))*r3s_hat +
(G*m1/(rs1_length**2))*r1s_hat#debris
```

```
        #v of obj.1,2,3
        v3_new = v3_old + a3_old*dt
        vs_new = vs_old + as_old*dt

        rs_new  = rs_old + vs_old*dt
        r3_new = r3_old + v3_old*dt

        #update to new frame
        rs_old = rs_new
        r3_old = r3_new

        v3_old = v3_new
        vs_old = vs_new
        t = t + dt

        #print info into file
        if  (int(t) % int(sec) == 0):
            # change directory for personal computer
            directory = '/Users/mario/Dropbox/XimenasWork/python
practice/super_computing22-23/debrisOrbits/'
            filepath = directory + name

            f = open(filepath, 'a+')
            f.writelines(f'{t},{rs_old},{r3_old} \n')
            f.close()
#displays finished time
now = datetime.datetime.now()
print(now)
```

**Luminosity.py**

```
import numpy as np
import itertools
import csv
import re
#sets up filepath for the debris positions
directory1 =
'C:/Users/Hadwyn/Documents/cislunar-orbit-simulation/orbit-data/'
rIter = 1
vIter = 1
name1 = 'R' + str(rIter) + 'V' + str(vIter) + ".csv"
filepath1 = directory1 + name1
#sets up filepath for the satellite positions
directory2 =
'C:/Users/Hadwyn/Documents/cislunar-orbit-simulation/sat-data/'
satIter = 1
name2 = 'RandV' + str(satIter) + ".csv"
filepath2 = directory2 + name2
#sets up filepath to grab satellite specs
directory3 =
'C:/Users/Hadwyn/Documents/cislunar-orbit-simulation/spec-data/'
name3 = 'C.0.O_text.txt'
filepath3 = directory3 + name3
#sets up directory for saving
newdirectory =
'C:/Users/Hadwyn/Documents/cislunar-orbit-simulation/luminosity-data/'
dataName = 'S' + str(satIter) + 'D' + str(rIter) + "-" + str(vIter)
savedir = newdirectory + dataName

#conversion rates
km = 1000
m = 1
#ballpark positions of the sun, in km, for 8 different points in time
sunPositionX =
[0,-106045189,-150000000,-106100721,0,106100721,150000000,106072958]
sunPositionY =
[150000000,106086841,0,-106086841,-150000000,-106086841,0,106059074]
#wattage of the sun
sunWattage = 3.846*10**26 #watts
#albedo of the debris in question
debrisAlbedo = .14 #percent
#radius/surface area of the debris
debrisRad = 1737.5 #km
debrisSA = 4*np.pi*debrisRad**2 #km^2
#Grabs relevant specs for the satellite
with open(filepath3, 'r+') as spec_file:
  spec_reader = csv.reader(spec_file, delimiter = ',')
```

```
  for row in spec_reader:
    lensRad = float(row[1]) #m
    lensArea = np.pi*lensRad**2 #m^2
    FOV = float(row[3])
    FOVoffset = 0
    shutterSpeed = float(row[2]) #seconds
    satQE = float(row[0]) #percent
#Goes through file and removes unecessary
def dataParse(dataLine):
  workingData = re.split(r',|[[]|[]]| ', str(dataLine))
  while("'" in workingData):
    workingData.remove("'")
  while("" in workingData):
    workingData.remove("")
  for index, item in enumerate(workingData):
    workingData[index] = workingData[index].strip(" \' ")
  return workingData
#Makes sure that the angles used are always within 0 and 360
def toPos(angle):
  while (angle < 0):
    angle += 360
  while (angle > 360):
    angle -= 360
  return angle
#Body of the program starts here
while (satIter <= 12 and rIter <= 13 and vIter <= 8):
  with open(filepath1, 'r+') as deb_file,   open(filepath2, 'r+') as
sat_file:
    deb_reader = csv.reader(deb_file, delimiter = ',')
    sat_reader = csv.reader(sat_file, delimiter = ',')
    #Loop through each row of the data
    for row1, row2 in itertools.zip_longest(deb_reader, sat_reader):
        deb_data = dataParse(row1)
        sat_data = dataParse(row2)
        #print(deb_data)
        #print(sat_data)
        #1. Extract the data into variables
        time = int(deb_data[0])
        debrisPos = [float(deb_data[1]), float(deb_data[2]),
float(deb_data[3])]
        satPos = [float(sat_data[(1)]), float(sat_data[(2)]),
float(sat_data[(3)])]
        sunPos = [sunPositionX[int(time/4269024)],
sunPositionY[int(time/4269024)], 0]
        #2. Find sides and angles needed for future calculations
```

```
        sunDeb =
np.sqrt((debrisPos[0]-sunPos[0])**2+(debrisPos[1]-sunPos[1])**2+(debrisPos
[2]-sunPos[2])**2)
        satDeb =
np.sqrt((debrisPos[0]-satPos[0])**2+(debrisPos[1]-satPos[1])**2+(debrisPos
[2]-satPos[2])**2)
        sunSat =
np.sqrt((satPos[0]-sunPos[0])**2+(satPos[1]-sunPos[1])**2+(satPos[2]-sunPo
s[2])**2)
        cosSat = (sunDeb**2+satDeb**2-sunSat**2)/(2*sunDeb*satDeb)
        debAng = (np.arccos(cosSat))*180/np.pi
        debLeftover = 90-debAng
        debrisPhase = (180-debAng)/180
        satAng = toPos(np.degrees(np.arctan2(satPos[0], satPos[1])))
        #3. Find if FOV overlaps with the angle of the debris
        FOVstartAngle = satAng+(FOV/2)
        FOVendAngle = satAng-(FOV/2)
        if(FOVstartAngle > debAng and FOVendAngle < debAng or
FOVstartAngle > debAng-360 and FOVendAngle < debAng-360 or
FOVstartAngle-360 > debAng and FOVendAngle-360 < debAng):
          #print("Debris is in sight!")
          #4. Find how much light hits debris from sun
          debrisLuminosity = (sunWattage/(4*np.pi*(sunDeb*km)**2))
          debrisWattage =
(debrisLuminosity*(debrisSA/2))*debrisAlbedo*debrisPhase
          #5. Find how much light off debris reaches satellite
          satLuminosity = (debrisWattage/(4*np.pi*(satDeb*km)**2))
          #6. Calculate how much light is intercepted by satellite
            #a. Convert to joules from lens area and shutter speed
          satJoules = satLuminosity * lensArea * shutterSpeed
            #b. multiply by satellite quantum efficiency
          perceivedJoules = satJoules * satQE
          #7. Save data to file
          f = open(savedir,'a+')
          f.writelines(f'{time},{satJoules}\n')
          f.close()
        else:
          #print("Debris is not in sight!")
          #8. Save data to file
          f = open(savedir,'a+')
          f.writelines(f'{time}, no debris detected\n')
          f.close()
  deb_file.close()
  sat_file.close()
  vIter+=1
  if (vIter == 13):
    vIter = 1
```

```
  rIter += 1
  if (rIter == 14):
    rIter = 1
    satIter += 1
name1 = 'R' + str(rIter) + 'V' + str(vIter) + ".csv"
name2 = 'RandV' + str(satIter) + ".csv"
filepath1 = directory1 + name1
filepath2 = directory2 + name2
dataName = 'S' + str(satIter) + 'D' + str(rIter) + "-" + str(vIter)
savedir = newdirectory + dataName
```

**Specs.py**

```
# Goal of program/satelite: Should be able to be done with a "large" group
of college kids with funding from sponsor(s)/Sat should be of 1U-1.5U
frame size

import colorama
from colorama import Fore, Back, Style
from colorama import init
import pygame
import tkinter as tk
from tkinter import simpledialog
from tkinter import *
import csv
import math
import decimal
import time

# variables
check = False
check2 = False
count = 0

#Space Launch Vehicles
SLV = "Falcon 9"  #From the americansecurityproject.org

#Orbit Type
LEO = 22800  #Payload capacity to low-earth orbit (kg)
orbit_type = "LEO"  #should be inputted?

#Sat specs/costs
Transponder_Type = "Regenerative Transponder"
Transponder_Mhz = 2290
Transponder_Mhz_per_Month_Cost = Transponder_Mhz * 3472.22  # cost per
month
Transponder_Mhz_per_Year_Cost = Transponder_Mhz_per_Month_Cost * 12  #
cost per year

#Lens specs for sat (piCAM)
quantum_efficiency = 0.95  # QE of the best scientific camera
lens_radius = 0.5  # Theoretical lens radius (m)
shutter_speed = 5  # satellite is taking photograps not actively tracking
debris through space; lens takes in light for 5 seconds
fov = 76

# general specs for sat
sat_weight = 2.3  # weight in kilograms
run_time = 1.1  # avg. lifespan of a cube sat in years
```

```
num_sats = 4
prompts = {
  "FOV": fov,
  "Shutter Speed": shutter_speed,
  "Lens Radius": lens_radius,
  "CubeSat Weight": sat_weight,
  "Run time": run_time,
  "Number of Satellites": num_sats
}


#takes user input for theoretical specs asside from base camera (piCAM)
def input():
  ROOT = tk.Tk()
  ROOT.withdraw()

  for x in prompts:
    USER_INP = simpledialog.askstring(title="Sat Specs", prompt=x)
    if USER_INP == "N/A" or USER_INP == prompts[x] or len(USER_INP) == 0:
      prompts[x] = prompts[x]

    else:
      prompts[x] = USER_INP

  for i in prompts:
    print(Fore.CYAN + i + ": " + Fore.WHITE + str(prompts[i]))


input()

Transponder_Mhz_Life_Cost = 16 * 12 * float(prompts["Run time"])
Tot_part_cost = 55000  # hard estimate due to finding part prices being
extremely difficult and actual part compatibilaty being undetermined

sat_weight_cost = 5000 * float(prompts["CubeSat Weight"])
cost_factors = [sat_weight_cost, Transponder_Mhz_Life_Cost, Tot_part_cost]

# saves data to a text file
directory =
r"C:\\Users/Hadwyn\\Documents\\cislunar-orbit-simulation\\spec-data\\C.0.O
_text.txt"
filepath = directory
f = open(filepath, 'w+')
f.writelines(
  f'{quantum_efficiency}, {prompts["Lens Radius"]}, {prompts["Shutter
Speed"]}, {prompts["FOV"]}'
```

```
)
f.close()
print(Fore.RED + "File Saved Succesfully")


# outputs estimated cost of the satellite
def sat_cost():
  global cost
  cost = sum(cost_factors)


sat_cost()

# calculates total cost
Total_cost = cost * float(prompts["Number of Satellites"])
Total_cost = '{:.2f}'.format(Total_cost)
Total_cost = float(Total_cost)

win = Tk()
win.geometry("300x50")
text = Text(win)
text.tag_configure("tag_name", justify='center')
text.insert("5.0", "Cost of Operation: $" + '{:,}'.format(Total_cost))
text.tag_add("tag_name", "1.0", "end")
text.pack()
win.after(5000, lambda: win.destroy())

win.mainloop()
```