Project 1: Moonlander (Part Two)

Purpose

To gain experience writing both functions that return values and those that only perform I/O operations.

Description

Continue working on the following files

- **`moonlander.py`** a file in which you put the program's functions
- **`moonlander_tests.py`** `a file in which you write tests.`

Download a zip file containing the following files from here:

- **`test[1-3].txt`** test cases for moonlander.py
- **`test[1-3].out`** `the expected output from your program.`

Background

You are tasked with writing a program that simulates landing the Lunar Module (LM) from the Apollo space program on the moon. The simulation starts when the retrorockets are cut off and the astronaut takes control of the LM. The LM starts at a user-specified altitude with an initial velocity of zero meters per second and has a user-specified amount of fuel onboard. With the manual thrusters off and the LM in free-fall, lunar gravity is causing the LM to accelerate toward the surface according to the gravitational constant for the moon. The pilot can control the rate of descent using the thrusters of the LM. The thrusters are controlled by entering integer values from 0 to 9 which represent the rate of fuel flow. A value of 5 maintains the current velocity, 0 means free-fall, 9 means maximum thrust; all other values are a variation of those possibilities and can be calculated with an equation that is provided below.

To make things interesting (and to reflect reality) the LM has a limited amount of fuel - if you run out of fuel before touching down you have lost control of the LM and it free-falls to the surface. The goal is to land the LM on the surface with a velocity between 0 and -1 meters per second, inclusive, using the least amount of fuel possible. A landing velocity between -1 meters per second and -10 meters per second (exclusive) means you survive but the LM is damaged and will not be able to leave the surface of the moon - you will be able to enjoy the surface of the moon as long as your oxygen lasts but you will not leave the moon alive. Velocities faster than (less than) or equal to -10 meters per second mean the LM sustains severe damage and you perish immediately.

The initial conditions of 1300 meters altitude and 500 liters of fuel were used in the original simulation and were chosen so that an optimal landing should use approximately 300 liters of fuel. It's a considerable challenge at first, and you may enjoy trying it.

Implementation

Reference Solution

A reference solution of the moonlander simulator is provided here:

https://cpe202.toshikuboi.net/moonlander.html

Start at 1300 meters with 500 liters of fuel and see how you do.

class Moonlander

In moonlander.py add the class definition of Moonlander near the top in moonlander.py:

```
class Moonlander:
    """Moonlander
    Attributes:
        altitude (float): the distance from the surface of the moon.
        fuel (int): the fuel
        velocity (float): It is positive when the Moonlander is
                          moving away from the moon.
        acceleration (float):  It is positive when the Moonlander is
                               moving away from the moon.
    """
    def __init__(self, alt, fuel):
        pass #replace this line with your code to initialize a Moonlander object
```

#YOU ARE NOT REQUIRED TO IMPLEMENT __eq__ and __repr__ for Moonlander in this project.

main() function

Begin this part with your fully tested `moonlander.py` file. Now you must write a `main` function that simulates the advancement of the LM from time period zero to landing, calling the functions of Part One along with some additional code to connect them together. Don't hesitate to ask your instructor questions early in the process to help make sure you understand what you are supposed to be doing. You will probably go down some dead ends and need to remove and rewrite code, so be sure to allocate enough time.

In `main`, you do not need to write nested loops, nor do you need to write a loop followed by another loop. The problem may be solved in both of those ways, but it can be solved more simply by writing a single loop.

Start by asking the user to input the initial altitude and fuel. Then, create an object of Moonlander with the initial altitude and the fuel. The initial velocity and the acceleration will be 0. You need to update the attributes of a Moonlander object in the loop.

Ask the user to enter a fuel rate only when the moonlander has some fuel: **DO NOT PROMPT USER TO INPUT FUEL RATE IF fuel = 0.**

For example, to update the fuel of a **Moonlander** object named **ml**, write a line of code like this:

ml.fuel = update_fuel(ml.fuel, fuel_rate)

To update the acceleration of a **Moonlander** object named **ml**, write a line of code like this:

ml.acceleration = update_acceleration(gravity, fuel_rate)

Use 1.62 as the gravity.

To update the altitude of a **Moonlander** object named **ml**, write a line of code like this:

ml.altitude = update_altitude(ml.altitude, ml.velocity, ml.acceleration)

Testing

Use the `diff` command to compare your program's output against supplied sample output files: test[1-3].out. You have three sample test cases (`test[1-3].txt`).

1. Run the reference solution until you are familiar with how the code should behave. Then run your code by typing the command below, making sure your `moonlander.py` file is in the same directory.

```
python3 moonlander.py
```

If your version does not behave the same as the reference solution, modify the appropriate functions in `moonlander.py` and test some more. Once you are confident that your code behaves just like the reference solution (the result of rounding may be slightly different, but do not worry about it), go to step 2.

2. On a terminal or gitbash, run your `moonlander.py` module with the `test1.txt` file:

```
python moonlander.py < test1.txt
```

The < directs the content of the test1.txt file to moonlander.py as inputs.

The landing status displayed on the screen should be the same as in test1.out. If not, go to step 3.

3. Run your `moonlander.py` module with the `test1.txt` file and save the result to a file:

```
python3 moonlander.py < test1.txt > my_test1.txt
```

The > directs the output from moonlander.py to the my_test1.out file.

4. Compare the final landing status outputted from your program and that in the supplied sample output. The landing statuses must match. If not, compare the saved result to the supplied output file to see where they differ on a terminal or gitbash:

```
diff test1.out my_test1.txt -wB
```

The `-wB` flags above ensure that whitespace is ignored. If both files contain the same text, nothing will display in the terminal. If, however, the files differ on some lines, the text from those lines as well as their line numbers will be displayed. Please note that it is normal that the numbers in your output are slightly different from the numbers in the sample output.

5. Continue modifying `moonlander.py` and comparing its output to the supplied output until the `diff` command ceases to display file differences (slight differences are ok as long as the landing status are the same).

6. Be sure to perform the above steps with the other provided test cases as well as other cases that you create.

Submission

Show your work to your instructor or TA.
After your work gets checked off, submit all your files including moonlander.py, and the three output files (my_test1.txt, my_test2.txt, and my_test3.txt) to Canvas before the due.

Rubric

- We will dock 20 points per wrong landing status.
- Additionally, we will dock 10 points if you do not use the Moonlander class object in your main() function.
- **Additionally, we will run Pylint against your submitted file, moonlander.py (it checks your code against the Python coding style guide PEP 8), and dock up to 10 points from your grade based on Pylint score.**

To install Pylint on Mac, type the following command in a terminal:

```
sudo pip install pylint
```

After Pylint has been installed, type the following command in terminal to run Pylint against your file:

```
python3 -m pylint moonlander.py
```

To install Pylint on Windows, open a gitbash and type the following command in gitbash:

```
py -m pip install pylint
```

After Pylint has been installed, type the following command in gitbash to run Pylint against your file:

```
py -m pylint moonlander.py
```