Project 1: Moonlander (Part One)

Purpose

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

Description

Create the following files

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

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

Function Definitions

In the first part of this project, you must write **and test** the functions that will eventually be used to complete the finished Moonlander program. All function definitions must be written in **`moonlander.py`**. These functions fall into two categories: those that calculate and those that do I/O (input/output). For the functions that calculate, write at least three example usages per function in the docstring and test them using the doctest module. You need to test your functions that do I/O using the Python interpreter. Note that input and output occur only in the functions which specifically have that purpose; the functions that calculate do not do I/O and vice versa.

Functions that do I/O See the instructor's version for the exact text to be displayed with these functions.

`show_welcome()` Display the welcome message. You do not need to include tests for this function in **`moonlander_tests.py`**.

`get_fuel()` Prompt the user for a positive integer. If this value is non-positive, display an error message and prompt the user again; otherwise, return the integer.

`get_altitude()` Prompt the user for an integer in the interval [1, 9999]. If this value is invalid, display an error message and prompt the user again; otherwise, return this integer.

`display_state(time, altitude, velocity, fuel, fuel_rate)` Display the state of the LM as indicated by the parameters. (Use Python's string format function to ensure that all values are properly lined up.) time is int, altitude is float, velocity is float, fuel is int, and fuel_rate is int.When time=0, altitude=1000.12, velocity=10.50, fuel=10, and fuel_rate=9, the function shall print the following string:
“time=0, altitude=1000.12, velocity=10.5, fuel=10, fuel rate=9”

`get_fuel_rate(fuel)` Prompt the user for an integer in the interval [0, 9]. If this value is invalid, display an error message and prompt the user again; otherwise, return the lesser of the user-entered value or the amount of fuel remaining on the LM (the argument passed to this function). In other words, **the user cannot use more fuel than the amount remaining on the lunar lander.** fuel is int.

`display_landing_status(velocity)` Display the status of the LM upon landing, conforming to one of three possible outputs depending of the velocity (float) of the LM at landing, which are:

- `Status at landing - The eagle has landed!` Print if the final velocity is between 0 and -1 meters per second, inclusive.
- `Status at landing - Enjoy your oxygen while it lasts!` Print if the final velocity is between -1 and -10 meters per second, exclusive.
- `Status at landing - Ouch - that hurt!` Print if the final velocity is <= -10 meters per second.

Functions that Calculate

The "tp" (time period) subscripts in the formulas below describe how the state at each new second of time (tp_1) depends on the state of the previous second (tp_0) and will help you determine the order in these functions must be called in the second part of the project.

`update_acceleration(gravity, fuel_rate)` Return the new acceleration based on the provided inputs and the formula:

$acceleration_{tp1} = gravity * ((fuel\_rate_{tp1} / 5) - 1)$ (Use 1.62 as the gravitational constant for the moon when calling this function.)

Round float values to 2 digits after the decimal point using Python builtin round(val, d) function, where d is the number of digits after the decimal point, just before returning the values from the functions.

`update_altitude(altitude, velocity, acceleration)`
Return the new altitude based on the provided inputs and the formula:

$altitude_{tp1} = altitude_{tp0} + velocity_{tp0} + (acceleration_{tp1} / 2)$ (As the surface of the moon stubbornly limits an altitude to non-negative values, ensure your code does the same).

Round float values to 2 digits after the decimal point using Python builtin round(val, d) function, where d is the number of digits after the decimal point, just before returning the values from the functions.

`update_velocity(velocity, acceleration)` Return the new

velocity based on the provided inputs and the formula:

$$velocity_{tp1} = velocity_{tp0} + acceleration_{tp1}$$

Round float values to 2 digits after the decimal point using Python builtin round(val, d) function, where d is the number of digits after the decimal point, just before returning the values from the functions.

`update_fuel(fuel, fuel_rate)` Return the new amount of fuel based on the provided inputs and the formula:

$$fuel_{tp1} = fuel_{tp0} - fuel\_rate_{tp1}$$

Testing

Functions that Calculate

Test these functions using the doctest module:

```
if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

You should also test with invalid inputs to functions. For example, try entering a negative number for the initial altitude and ensure the proper error message is displayed. You need not test with inputs other than integers.

Submission

Show your work to your instructor or TA. Then, submit your moonlander.py to Canvas. Your file must include function docstrings with examples. Part 1 is worth 40 points.

Rubric

We will dock 5 points per function if the function is incorrectly implemented, the docstring is missing, or enough examples are not written in the correct format the doctest requires.