Programming Project 05

This assignment is worth 40 points (4.0% of the course grade) and must be **completed and turned in before 11:59 on Monday, October 17, 2022. After the due date, your score will be deducted by 2pt every 2 hours.**

This assignment will give you more experience on the use of strings and functions.

Assignment Overview

Since ancient times, people have wondered about the possibility of life on planets other than our own. As telescopes have improved, the number of possible life-supporting planets has increased. In this week's project, you will work with files and strings to do calculations on extrasolar planetary data. We will do calculations with this data to obtain some statistics about worlds that orbit other stars. In the process, you will work on files, string methods, and string slicing.

Program Specifications

In this project, you will read a file of extrasolar planetary data. You will apply some formulas that attempt to predict the possibility of life-supporting conditions based on observations that can be made by telescopes and other astronomical sensors.

You only read the file once so you must process the data as it is read. We only read a file once because reading a file is quite inefficient. Soon in the course you will learn new data structures such as lists that allow you to read a file once and then allow you to make multiple passes through the data. Thus, you are not allowed to use lists or any other advanced data structures in this project.

One task is to find the maximum number of stars and planets. Here is the generic algorithm for doing that.

High-level min (max) algorithm

Your high-level minimum algorithm will be:

1. Set `min_value` to some value larger than you expect.
2. Loop through the data

a. If a value is less than `min_value`:
   you have found a smaller value so set `min_value = value`
   we will also want to know the information associated with that `min_value`
   so update the information value, too

3. `min_value` now holds the minimum of all values encountered.

To find a maximum, set the initial value to a smaller number than expected, e.g. -1 for our percent, and replace "less than" with "greater than".

File Specification

The input files have fixed-field columns so you can use slicing to extract data. The files happen to be .csv files which means that they are also comma-separated. The input files have one header line and a number of fixed-field columns. The data we are interested in are:

```
[:25]       planet name (str)
[50:57]     number of stars in a system (int)
[58:65]     number of planets (int)
[66:77]     axis: distance at which planet orbits (float)
[78:85]     planet's radius (float)
[86:96]     planet's mass (float)
[97:105]    star's temperature (float)
[106:113]   star's radius (float)
[114:]      distance (float)
```

All the units here are in terms of either Earth or the Sun (e.g. 5.4 in the mass column means 5.4 * the mass of Earth), with the exceptions being stellar temperature (Kelvins) and distance from our solar system (parsecs).

Your program must also meet the following specifications:

1. You **must** have and use at least these four functions—more are fine. A `proj05.py` file with function stubs is provided.
   a. **`def open_file() → file_pointer`**
      i. Repeatedly prompt for a file name until a file is successfully opened for reading.
      Your function will append the extension `'.csv'` to complete the filename.
      If the file does not exist an error message should be displayed.
      Use the `try-except` command with `except FileNotFoundError`
      The first prompt should be:
      `"Input data to open: "`

      The error message should be:
      `'\nError: file not found.  Please try again.'`

      And the new prompt when an error occur will be:
      `'Input a filename: '`
      ii. Parameters: `None`
      iii. Returns: `file_pointer`
      iv. Display: prompt and error message as appropriate

b. **`def make_float(s):→ float`**
   i. Convert a string to a float, if possible. If the float conversion fails, return -1. Use the `try-except` command with `except ValueError`
   ii. Parameters: `str`
   iii. Returns: `float or -1`
   iv. Display: nothing

c. **`def get_density(mass,radius)→ float`**
Takes mass and radius of a spherical object in terms of Earth units and returns the density. The parameters mass and radius must be converted to metric using the provided constants—simply multiply each parameter by its associated constant. (The constants are in the `proj05.py` file we provide.)

```
EARTH_MASS =  5.972E+24    # kg
EARTH_RADIUS = 6.371E+6    # meters
```

Do not change the given constants and use these formulas

Volume = $\frac{4}{3}\pi r^3$
Density = $\frac{mass\ (kg)}{volume\ (m^3)}$

If any parameter is negative, or if the `radius` is zero return `-1`.
   i. Parameters: `float, float`
   ii. Returns: `float`
   iii. Display: nothing

d. **`def temp_in_range(axis, star_temp, star_radius, albedo, lower_bound, upper_bound) → bool`**
This function returns `True` if we estimate that the planet's temperature is within a range that might support life as we know it; `False` otherwise.
We need to calculate the temperature of a given exoplanet given its distance axis in AU (the distance Earth is from the Sun), the star's temperature (in Kelvin), its radius (in terms of the Sun), and its albedo (how much energy is reflected away) to see if it is in the bounds given (returning true or false). The formula to calculate the temperature is

planet_temp = $star\_temp * (\frac{star\_radius}{2*axis})^{0.5} * (1 - albedo)^{0.25}$

make sure to convert to metric units before applying this formula—using these constants. (The constants are in the `proj05.py` file we provide.)

```
   SOLAR_RADIUS = 6.975E+8   # radius of star in meters
   AU = 1.496E+11            # distance earth to sun in
meters
```

If the planet_temp is within the range specified by the lower_bound and upper_bound parameters (inclusive), return `True`.
If any of the inputs are negative, return `False`.
Do not change the given constants that are used for conversions.

i. Parameters: `float, float, float, float, float`
ii. Returns: `bool`
iii. Displays: nothing

e. `def get_dist_range() → float`

Prompt for a distance. If the distance is less than zero or cannot be converted to a `float`, print an error message and re-prompt. We are asking the user for a farthest distance from Earth to filter the data. It will return the distance as a float if it is valid.
Use the `try-except` command with `except ValueError`
Prompts:
`"\nEnter maximum distance from Earth (light years): "`
Error messages:
`"\nError: Distance needs to be greater than 0."`
`"\nError: Distance needs to be a float."`

i. Parameter: nothing
ii. Returns: `float`
iii. Displays: prompt and possibly an error message

f. `def main()` Your main algorithm will be as follows calling the above functions.

i. Call `open_file` to open an input file for reading.
ii. Call `get_dist_range` which returns the maximum distance. This input will be in light-years, but our datasets contain distances in parsecs, so be sure to convert the units by multiplying by the `PARSEC_LY` constant provided.
iii. Initialize variables
    1. The lower bound should be 200, the upper bound should be 350.
    2. Albedo should be 0.5
iv. Loop through the file line by line:
    1. Read the distance (check the file Specification section). If the distance is not a float, assign -1. (hint: call `make_float()`). If it is out of range, `continue` to the next line (ignore that line). A valid distance is a float that is between 0 and maximum distance (the value returned from the `get_dist_range()`)
    2. Read in the other variables, calling `make_float` to convert to `float` (Why should you not simply call `float`?)
    3. Calculate max number of planets and stars (see algorithm above)
    To find the max_planets and max_stars, use the algorithm given above.
    4. Accumulate total mass of planets.
    5. Call `get_density` which returns density.
    6. Use temp_in_range in a conditional to find if the planet is a potential candidate for habitation and if it is determine if it is rocky or gaseous as well as whether it is the closest rocky or closest gaseous (find the lowest distance of a rocky or gaseous planet). A planet is rocky if its mass is between 0 and 10, it's radius is between 0 and 1.5, or its density is over 2000. Otherwise, it is gaseous.
v. Close the file
vi. Print the six lines of cumulative information (see sample output and the `strings.txt` file). To match our output, print to an accuracy of two

decimals for floats and strip away extra space in the name of the exoplanets for the final output.

Deliverables
The deliverable for this assignment is the following file:

`proj05.py` -- your source code solution

Be sure to use the specified file name and to submit it for grading before the project deadline.

Notes and Hints:

1. To clarify the project specifications, sample output is appended to the end of this document.
2. Items 1-9 of the Coding Standard will be enforced for this project—note the change to include more items.
3. You can test functions separately—that can be a huge advantage in developing correct code faster! If you cannot figure out how to do that, ask your TA for guidance.
4. Do not hard code your solutions—the result is a zero. Hard coding means that your program for the specific tests rather than having a generic solution.

Sample Interaction:

Test 1:

```
Welcome to program that finds nearby exoplanets in circumstellar habitable
zone.

Input data to open: small


Enter maximum distance from Earth (light years): 100

Number of stars in systems with the most stars: 3.
Number of planets in systems with the most planets: 4.
Average mass of the planets: 455.26 Earth masses.
Number of planets in circumstellar habitable zone: 1.
No rocky planet in circumstellar habitable zone.
Closest gaseous planet in the circumstellar habitable zone kap CrB b is
98.07 light years away.
```

Test 2:

```
Welcome to program that finds nearby exoplanets in circumstellar habitable
zone.

Input data to open: singlestar


Enter maximum distance from Earth (light years): 50

Number of stars in systems with the most stars: 1.
Number of planets in systems with the most planets: 6.
```

```
Average mass of the planets: 68.36 Earth masses.
Number of planets in circumstellar habitable zone: 26.
Closest rocky planet in the circumstellar habitable zone GJ 887 c is 10.73
light years away.
Closest gaseous planet in the circumstellar habitable zone HD 219134 g is
21.31 light years away.
```

Test 3:

```
Welcome to program that finds nearby exoplanets in circumstellar habitable
zone.

Input data to open: fail

Error: file not found.  Please try again.

Enter a file name: KGF.csv

Error: file not found.  Please try again.

Enter a file name: KGF


Enter maximum distance from Earth (light years): 10000

Number of stars in systems with the most stars: 4.
Number of planets in systems with the most planets: 8.
Average mass of the planets: 531.28 Earth masses.
Number of planets in circumstellar habitable zone: 199.
Closest rocky planet in the circumstellar habitable zone HD 219134 d is
21.31 light years away.
Closest gaseous planet in the circumstellar habitable zone GJ 338 B b is
20.66 light years away.
```

Test 4:

```
Welcome to program that finds nearby exoplanets in circumstellar
habitable zone.

Input data to open: all


Enter maximum distance from Earth (light years): test

Error: Distance needs to be a float.


Enter maximum distance from Earth (light years): -34

Error: Distance needs to be greater than 0.


Enter maximum distance from Earth (light years): 1000000
```

```
Number of stars in systems with the most stars: 4.
Number of planets in systems with the most planets: 8.
Average mass of the planets: 510.35 Earth masses.
Number of planets in circumstellar habitable zone: 268.
Closest rocky planet in the circumstellar habitable zone Proxima Cen
b is 4.24 light years away.
Closest gaseous planet in the circumstellar habitable zone GJ 338 B
b is 20.66 light years away.
```

Scoring Rubric

```
Computer Project #05                              Scoring Summary

General Requirements

_______    4 pts    Coding Standard 1-9
          (descriptive comments, function header, etc...)


Implementation:

__0__    (4 pts)  open_file (manual grading)

__0__    (4 pts)  Function Test make_float

__0__    (4 pts)  Function Test get_density

__0__    (4 pts)  Function Test temp_in_range

__0__    (4 pts)  Function Test get_dist_range (manual grading)

__0__    (4 pts)  Test 1

__0__    (4 pts)  Test 2

__0__    (4 pts)  Test 3

__0__    (4 pts)  Test 4
```