Problem Set 5

Python - Decorators and Context Managers

For this homework, you will write a code to describe the behavior of gases.

Ideal gases are commonly used to introduce thermodynamic concepts. Their behavior can be described by the equation

$$PV = nRT$$

where `P` represents pressure, `V` is the volume, `n` is the number of moles, `R` is the ideal gas constant, and `T` is the temperature.

This equation can be modified to describe the behavior of real gases by adding terms a and b to account for particle volume and interactions. The **van der Waals gas equation** is

$$\left(P + \frac{an^2}{V^2}\right)(V - nb) = nRT$$

You will first write a class to describe the behavior of ideal gases, and use Test Driven Development to develop your code. Test Driven Development is a development technique where tests are created before the code is developed.

This homework provides you with a set of tests in `test_ideal.py` which can be run using the PyTest testing framework. To run these tests, navigate to the folder with your files and execute

```
pytest -v
```

Your tasks will be to

1. Write a class called `IdealGas` which passes the provided tests.
2. Modify the provided tests to use context managers and decorators which are part of the `pytest` package.
3. Write code (implementation method of your choice) to model a real gas.

Part 1 - Ideal Gas Class

Write a class called `IdealGas` which fits specifications from the provided tests. You should also write additional tests to ensure that your code is robust.

**Add an alternate constructor called `from_pressure`. Be sure to also add appropriate tests for your method (there are no tests for this requirement.)

Part 2 : Decorators and Context Managers - PyTest

The provided tests are written as individual tests. You should refactor the tests to use the pytest parametrize. You will also identify a context manager used in the provided tests that is part of pytest which will allow you to test for expected exceptions.

Part 3 - Modeling a Real Gas

Write code to account for the behavior of real gases. The code for this should be very similar, except that you will need `a` and `b` parameters for the model.

You will also need to account for different mixing behavior of van der Waals gases. For your addition function, assume constant volume (i.e. `v = v1 + v2`). However, you will need to calculate `a` and `b` for the mixture.

$$a = \left(\sum_{i=1} x_i, a_i^{1/2} \right)^2 \quad b = \sum_{i=1} x_i, b_i;,$$

There are a few ways to approach this. You could simply add `a` and `b` parameters to your ideal gas class and set them equal to 0 for an ideal gas. Another possibility might be to use inheritance to create two different types of classes. This implementation detail is left up to you (but you will have to discuss your choice!)

Documentation and Discussion

Include a `README.md` file which describes the project and software usage. **You should include a Makefile in the repository which has at least two targets - one to make an environment, and another to run your tests.** Note that for your environment, you should use approaches we have learned in this course (`conda`).

Include answers to the following questions in your README.

1. What is a decorator? Look up the documentation to write an explanation **in your own words** of what the `pytest parametrize` decorator does.
2. What are the context managers? What context manager is used in these tests, and what is its purpose? How could the same behavor be used without a context manager? Write this in your own words.
3. Where did you use `@property` in your class and why? Did you use any other decorators?
4. What (if any) additional tests did you add to ensure that your implementations were robust?
5. How did you choose to implement code for your van der Waals gas? Would your method change if you had to write code for real gases using other real gas equations? Why or why not? Can any of your tests be used for both classes (particularly through using decorators)?

Debugging a C++ application

The file `crashes.cpp` contains a program that sometimes crashes. Use your debugging knowledge to find the issue.

Write up the cause of the issue in the README file. Include any output from debuggers or sanitizers that helped you.

Profiling C++ Code

In this repo is a file `mcsim_cpp.cpp` which contains code for the `mcscim` package from the bootcamp. Using `gprof`, profile this code and include the output in a file in this repo. Iinclude anything you find surprising or interesting. Put this information into the README.

Do the above for two optimzation levels - `-O0` and `-O3`. Is there anything interesting about one compared to the other?

Profiling Python Code

Similar to the above, but profiling the `mcsim_psl.py` program with python's cProfile. There are no optimization levels, so only one profiler run needs to be done. You can visualize the results using a program called snakeviz. Is there anything taking a surprising amount of time?

Documentation and Discussion

Add answers to the questions from the sections above in your `README`. Your code should also contain a `Makefile` with targets for profiling the Python and C++ code.