Homework: Battle Aardvarks

Version 1.0

The concept for this homework comes from Ben Cohn. Many thanks!

Background

Battle Aardvarks are fierce but lovable creatures whose favorite pastime is engaging in non-lethal head-to-head combat.

Battles

Battles occur between two aardvarks. Each aardvark starts with a certain number of health points and has a certain level of power (power influences the effectiveness of an aardvark's attacks).

The two aardvarks mount simultaneous attacks against each other. Attacks succeed about 50% of the time. When an aardvark lands a successful attack on an opponent, the opponent's health points decrease. The battle continues until one aardvark's health points fall to zero or lower. If the other aardvark has positive health points, they are declared the winner; otherwise, the battle ends in a draw.

Clans

Each aardvark belongs to one of three clans: Green, Orange, or Purple. In a matchup between members of different clans, one opponent will have an advantage over the other. The opponent with the advantage will attack with 25% more power than normal; the disadvantaged opponent will attack with 20% less power. The rules of advantage are as follows:

- Orange has an advantage over Green
- Green has an advantage over Purple
- Purple has an advantage over Orange

Provided files

Stats file

A file of aardvark statistics, `stats.csv`, is provided. This is a plain text file in UTF-8 encoding. Each line in the file contains four values, separated by commas:

- The name of an aardvark
- The aardvark's clan
- The aardvark's starting health points
- The aardvark's power

Program template

A program template, `aardvarks.py`, is provided for you. The template imports some functions and modules for you and implements some functions as well as code to run your program.

- The `main()` function creates an instance of your `Catalog` class and two instances of your `Aardvark` class. If the same name was specified for both Aardvark objects, it adds a `1` or a `2` to the end of each aardvark's name (e.g., `Winifred 1`, `Winifred 2`). Then it runs a battle between the two aardvarks.
- The `parse_args()` function processes command-line arguments. Three arguments are expected: the path to a file such as `stats.csv` and the names of two aardvarks from the file that will do battle.

The `if __name__ == "__main__":` statement calls `parse_args()` to process command-line arguments. It then invokes the `main()` function, using the processed arguments.

Problem statement

Write a program that

- creates a catalog of battle aardvarks from the contents of a CSV file
- creates two aardvark objects based on stats in the catalog
- conducts a battle between the two aardvark objects, writing information about the progress of the battle to stdout

Instructions

Use the provided template, `aardvarks.py`, to build your solution. Define your classes and function after the `import` statements and before the `main()` function (you may delete the comment that serves as a placeholder for your code). You should develop your program in VS Code. Your program should be named `aardvarks.py`.

`Aardvark` class

Define a class called `Aardvark`. Your class will have the following attributes:

- `name`: a string containing the aardvark's name.
- `clan`: a string containing the aardvark's clan (`"Orange"`, `"Purple"`, or `"Green"`).
- `hp`: a float representing the aardvark's health points.
- `power`: a float representing the aardvark's attack power.

`__init__()` method

Write an `__init__()` method that has five parameters, in the following order:

- `self`
- the aardvark's name (a string[1])
- the aardvark's clan (a string)
- the aardvark's health points (a float)
- the aardvark's power (a float)

Use these parameters to define the attributes specified above. You can name your parameters as you see fit, but your attributes should use the exact names specified above.

`advantage()` method

Write a method called `advantage()`. This method should have two parameters:

- `self`
- the aardvark's opponent (an Aardvark)

This method will return an advantage coefficient representing the aardvark's advantage over their opponent. (As a reminder, Orange > Green, Green > Purple, Purple > Orange. See the background section on clans for more details.)

If the aardvark has an advantage over their opponent, this method should return 1.25. If the opponent has an advantage over the aardvark, it should return 0.8. Otherwise, it should return 1.0.

`attack()` method

Define a method called `attack()`. This method should have two parameters:

- `self`
- the aardvark's opponent (an Aardvark)

This method should generate a random number (0 or 1) using the command `randint(0, 1)` (`randint()` has been imported for you at the top of the template).

If the number generated was 0, your method should print a statement in the following format (replace `Winifred` with the name of the attacking aardvark and `Chester` with the name of the opponent):

```
Winifred fails to do damage to Chester.
```

If the number generated was 1, your method should calculate the damage to the aardvark's opponent by multiplying the aardvark's power (using the `power` attribute) by the aardvark's advantage coefficient (which you can obtain by calling the `advantage()` method). Deduct the damage from the opponent's `hp` attribute. Print a statement indicating the amount of damage done, in the following format (replace `Winifred` with the name of the attacking aardvark, `161.0` with the amount of damage done, and `Chester` with the name of the opponent):

```
Winifred does 161.0 damage to Chester.
```

`Catalog` class

Define a class called `Catalog`. This class will read a CSV file of aardvark stats such as `stats.csv` and will store those stats in a dictionary. The class will allow the user to instantiate an Aardvark object using the name of an aardvark from the CSV file.

Your class should have an attribute called `catalog` (a dictionary). Each key in `catalog` will be the name of an aardvark. The corresponding value will be a tuple of three values, in the following order:

- the aardvark's clan (a string)
- the aardvark's initial health points (a float)
- the aardvark's power (a float)

`__init__()` method

Write an `__init__()` method that has two parameters:

- `self`
- the path to a file containing aardvark stats (a string)

This method should open the specified file for reading (please make sure your method opens the file specified in the parameter; don't hard-code the string `"stats.py"` into your program anywhere). Each line in the file contains four values, separated by commas:

- the aardvark's name
- the aardvark's clan
- the aardvark's initial health points
- the aardvark's power

The method should read each line from the file and split the line into its four values. It should convert the health points and power to floats. It should create build a dictionary so that each key is the name of an aardvark from the file and the corresponding value is a tuple containing the aardvark's clan, health points, and power. This dictionary should be stored as `self.catalog`.

`get_aardvark()` method

Write a method called `get_aardvark()`. This method should have two parameters:

- `self`
- the name of an aardvark in the catalog (a string)

If the specified name is not in the catalog, your method should raise a `KeyError`. Otherwise, the method should instantiate an Aardvark object with the name, clan, health points, and power of the aardvark specified in the second parameter. The method should return this Aardvark object.

`battle()` function

Note: this is a function, not a method. It should be defined outside of your classes.

Write a function called `battle()`. The function should have three parameters:

- the first participant (an Aardvark)
- the second participant (an Aardvark)
- a pause time in seconds (a float); this should have a default value of 2.0

A battle proceeds as follows:

1. the first aardvark uses their `attack()` method to attack the second aardvark
2. the second aardvark uses their `attack()` method to attack the first aardvark
3. for each aardvark, print a message with the aardvark's name and health points, in the following format (replace `Winifred` or `Chester` with the aardvark's name and `643.0` or `1021.0` with the aardvark's health points):

   ```
   Winifred has 643.0 health points.
   Chester has 1021.0 health points.
   ```

4. print a blank line by calling `print()` with no arguments
5. wait for the specified pause time to give the user a time to read the messages; you can do this by passing the pause time to the `sleep()` function (the template imports this function for you)

Repeat these steps until at least one aardvark's health points are non-positive. If both aardvarks have non-positive health points, print

```
The battle ends in a draw!
```

Otherwise, the aardvark with positive health points is the winner; print a message in the following format (replace `Winifred` with the winning aardvark's name):

```
Winifred wins!
```

Other instructions

Length of individual lines of code

Please keep your lines of code to 80 characters or less. If you need help breaking up long lines of code, please see https://umd.instructure.com/courses/1299872/pages/how-to-break-up-long-lines-of-code.

Docstrings

Please write docstrings for each class, method, and function. Docstrings were covered in the first week's lecture videos (https://youtu.be/jHTv83PlQYw?t=1415) and revisited in the OOP lecture videos (https://youtu.be/Oq9ssywHMPg). There's an ELMS page about them here: https://umd.instructure.com/courses/1299872/pages/docstrings.

Docstrings are not comments; they are statements. Python recognizes a string as a docstring if it is the first statement in the body of the method, function, class, or script/module it documents. Because docstrings are statements, the quotation mark at the start of the docstring must align exactly with the start of other statements in the method, function, class, or module.

▶ General instructions for class docstrings

▶ General instructions method and function docstrings

Testing your code

Running your program at the command line

The template is designed to use command-line arguments. To run your program within the VS Code built-in terminal, first make sure you have opened (in VS Code) the folder where your program is saved. If necessary, you can go to the VS Code File menu and select "Open…" on macOS or "Open Folder…" on Windows, and navigate to the directory where your program is.

Then, open the VS Code built-in terminal. Type `python3` (on macOS) or `python` (on Windows) followed by a space, the name of the program, another space, and the name of the CSV file containing a customer's electricity usage over time (a sample file, `stats.csv`, is provided with the assignment). Below is an example. The example assumes that `stats.csv` is in the same directory as your program.

```
python3 aardvarks.py stats.csv Winifred Chester
```

You can specify the number of seconds to pause between attacks in a battle, using the `-p` option. Here is an example of how to do that:

```
python3 aardvarks.py stats.csv Winifred Chester -p 1.25
```

Testing your program with pytest

Three test scripts are provided to allow you to test different parts of your program:

- `test_aardvark.py`: tests the `Aardvark` class
- `test_catalog.py`: tests the `Catalog` class
- `test_battle.py`: tests the `battle()` function

You should save these scripts in the same directory as your program. To run these scripts within the VS Code terminal, first make sure you have opened VS Code to the folder where your program is saved. Then open the terminal. To run a single test script at the command line, type `pytest` followed by a space and the name of the script you wish to run; for example:

```
pytest test_aardvark.py
```

If that does not work, try

```
python3 -m pytest test_aardvark.py
```

(Windows users, type `python` instead of `python3`.)

It's possible to run multiple test scripts at the same time:

```
pytest test_aardvark.py test_catalog.py test_battle.py
```

Submitting your work

Submit your work using Gradescope. Please upload only `aardvarks.py` (do not upload any test scripts or CSV files).

`aardvarks.py` will be partially auto-graded by Gradescope. If you are not happy with the results, you may revise your code and resubmit as many times as you like until the deadline.

Extensions

If you need an extension on the deadline, please email the instructor prior to the deadline, indicating when you propose to turn in the assignment. There is no penalty for requesting or receiving an extension, and no explanation is required. Once granted, an extension will not be extended. For full details on the policy for homework extensions are provided in the syllabus.

Grading

This assigment is worth 30 points in the homework category, allocated as follows:

14 points are allocated to automatic tests of your code functionality. 4 points are allocated to automatic tests of your docstrings. The remaining 12 points are awarded based on the degree of completeness of your program and docstrings.

Category	Points	Notes
Automatic tests of code functionality	14	Tests will evaluate instance attributes; return values; side effects; and whether errors are raised when expected
Automatic tests of docstrings	4	Tests will look for existence of docstrings and presence of expected sections in each docstring
Manual evaluation of code completeness	8	-1 pt per missing class, method, or function; -0.5 pt per incomplete method or function
Manual evaluation of docstring completeness	4	-0.5 pt per missing docstring; -0.25 pt per incomplete docstring

Academic integrity

This assignment is to be done by you individually, without outside help of any kind (including, but not limited to, help from classmates, tutors, or the internet). Disseminating these instructions in whole or in part without written permission of the instructor is considered an infraction of academic integrity.

Revision history

- 1.0 (2021-02-23): Added instructions for using test scripts
- 0.9 (2021-02-22): Original version

5. For this parameter and all the others in the assignment, I've specified the data type so you know what assumptions you can make and so you don't have to convert during marking. You don't need to convert the value to this data type; just assume the value already conforms to the specified data type.