CS-UY 1114

Final Review

The Story

Professor and Pokémon trainer Katz wants to be the very best that no one ever was. To that end, he has travelled across the land and caught and trained a bunch of **Eevees**, which have now evolved. His team looks like this at the moment:

- Blitzeon -> Andy
- Empatheon -> Ricky
- Acideon -> Apoorva
- Poppeon -> Cindy
- Arcteon -> Kevin
- Sireon -> Kimberly
- Scaldeon -> Tinos
- Clairon -> Neha

He feels confident enough to take on the strongest trainers in New York: The Tandon League, whose leader is **Gary Oak**. Gary's team looks like this:

- Mytheon -> Jun
- Tsuneon -> Meghana
- Infernon -> Anna
- Polareon -> Amalie
- Faeron -> Selina
- Blitzeon -> Semi
- Spectreon -> Tanvi
- Aureon -> Trisha
- Pitayeon -> Victor

Source

Our job is to make that happen through the power of all of the concepts that you have learned throughout the semester. Good luck.

***P.S.**: If you've never played/watched Pokémon, or a similar game/show, I'm so sorry.*

***P.P.S.**: Please don't post this online anywhere else. I don't want to get sued by Nintendo* 😃

NOTE: All of our code will go in one file

Problem 1: The `Pokemon` Class

Let's start with creating our professor's Pokémon. In our file, create a class called `Pokemon`. The `Pokemon` class will have the following methods, which we will tackle one-by-one:

Method	Type of Method	Returns	Notes
`__init__()`	Dunder / Special	`None`	The constructor of the `Pokemon` class.
`__str__()`	Dunder / Special	`str`	Returns informal string representation of `Pokemon` object.
`__gt__()`	Dunder / Special	`bool`	Returns whether this `Pokemon` object has a greater **speed** stat than another `Pokemon` object.
`attack()`	Public	`float` / `int`	Returns the damage that the `Pokemon` object will inflict in a single attack.
`_will_land_critical_hit()`	Private	`bool`	Returns whether a Pokemon will land a critical hit.

It might look like a lot to do, but a lot of these methods are very short and easy. So take it one step at a time and don't move on to the next section until you understand everything that's going on in what you've already implemented!

1.1: The `__init__()` Method

Write your `Pokemon` class's constructor so that it contains the following attributes (note that type 1 and type 2, along with strength and speed are accepted as separate parameters in the constructor, but are not necessarily a separate attribute in the class):

Attribute	Type	Comments
`pokemon_name`	*`str`*	The name of our Pokémon.
`nickname`	*`str`*	The nickname of our Pokémon, given by Prof. Reeves.
`level`	*`int`*	The level of our Pokémon.
`types`	*`(str, str)`*	The two types of the Pokémon. Will be passed as **two individual parameters** to the constructor.
`hp`	*`int`*	The health of our Pokémon.
`stats`	*`{str: int}`*	The battle statistics of our Pokémon. These key-values include: `"attack"`: an `int`, and `"speed"`: an `int`

***Table 1**: `Pokemon`'s attributes. Note that these are **not** necessarily the parameters passed into the constructor.*

If all goes smoothly, your `Pokemon` class will behave as follows:

```
def main():
    your_TA = Pokemon("Acideon", "Apoorva", 50, "Fire", "Water", 180, 135,
70)
    print(your_TA.pokemon_name)
    print(your_TA.nickname)
    print(your_TA.level)
    print(your_TA.types)
    print(your_TA.hp)
    print(your_TA.stats)

main()
```

Output

```
Acideon
Apoorva
50
('Fire', 'Water')
180
{'attack': 135, 'speed': 70}
```

A couple of things to note here:

- You may assume that all will be passed as their intended types (i.e. `pokemon_name` will always receive a string, `attack` will always receive an integer, etc.) but keep in mind that, inside the file, everything is text. You have to convert values explicitly to integers, floats, etc..
- Pokemon *may* not have a second type. Your constructor should check if an empty string is passed in instead of `type2`. If this is the case, the second element inside our `types` tuple should be the keyword `None`:

```
your_TA = Pokemon("Acideon", "Apoorva", 50, "Fire", "Water", 180, 135, 70)
print(your_TA.types)  # will print ('Fire', 'Water')
```

1.2 The `__str__()` Method

Let's, like last time, get this one out of the way quickly. Define `Pokemon`'s `__str__()` method so that it behaves, ***exactly***, as follows:

```
def main():
    pokemon_1 = Pokemon("Gothorita", "Saya", 75, "Psychic", "", 160, 55,
80)
    pokemon_2 = Pokemon("Hatterene", "Elaina", 100, "Psychic", "Fairy",
160, 55, 80)
    print(pokemon_1)
    print(pokemon_2)
```

```
main()
```

Output:

```
Gothorita Pokemon object 'Saya' of type(s) Psychic, level 75
Hatterene Pokemon object 'Elaina' of type(s) Psychic/Fairy, level 100
```

1.3 The `attack()` and `_will_land_critical_hit()` Methods

The next method we'll define for the `Pokemon`'s class will be the `attack()` method, which itself will make use of another method we have to define: the `__will_land_critical_hit()`. We'll borrow from one of our previous labs here, so I will just adapt the instructions from there below:

There is always a chance that our Pokemon will land a critical hit when attacking. This basically means that, based on your Pokemon's stats, your attack **might** roughly double in damage. The "might" here is actually based on probability:

> Whether a move scores a critical hit is determined by comparing a 1-byte **random number (0 to 255)** against a 1-byte threshold value (also 0 to 255); if the random number is less than the threshold, the Pokemon scores a critical hit.

— ***Critical hit***, *Bulbapedia*.

Basically, we'll have two values:

- A random value, `R` from 0 to 255.
- A threshold value, `T` from 0 to 255. If this value is higher than `R`, the `Pokemon` object lands a critical hit; we can calculate a critical value as follows:

```
T = (speed / 2)
```

***Figure 1**: Threshold formula, where `speed` is equal to the `Pokemon` object's `"speed"` stat.*

If it is determined that the `Pokemon` has landed a critical hit, the damage multiplier (that is, the number by which the `Pokemon`'s `"attack"` stat will be multiplied by) will be equal to:

```
Multiplier = (2L + 5) / (L + 5)
```

***Figure 2**: Multiplier formula, where `L` is equal to the `Pokemon` object's `level`.*

So, with this knowledge in hand, first write a method in the `Pokemon` class called `_will_land_critical_hit()` that will determine whether our `Pokemon` object will land a critical hit.

Naturally, the `random` module is necessary here. That's all this method needs to do: return `True` if the **`Pokemon`** object lands a critical hit, and `False` if it doesn't.

Next, write a second method called `attack()`, which will return the `attack` stat of the **`Pokemon`** object if **`_will_land_critical_hit()`** returns `False`, or the `attack` stat of the **`Pokemon`** object multiplied by the critical hit multiplier if it **`_will_land_critical_hit()`** returns `True`.

Check out the following sample behavior, where I added a `print` function message every time a critical hit is landed. This is not necessary, but feel free to add it to your code if it helps you:

```
def main():
   starter_pokemon = Pokemon("Aegislash", "Napoleon", 75, "Steel",
"Ghost", 160, 55, 80)

   number_of_tests = 10
   for test_number in range(number_of_tests):
       attack_power = starter_pokemon.attack()
       print("In move #{}, {} '{}' attacked with {}
power.".format(test_number + 1, starter_pokemon.pokemon_name,
starter_pokemon.nickname, attack_power))

main()
```

Possible output (I added a `print` statement in my **`attack()`** function for clarity of what's going on):

```
Napoleon attacks!
Napoleon lands a critical hit!
In move #1, Aegislash 'Napoleon' attacked with 106.5625 power.
Napoleon attacks!
In move #2, Aegislash 'Napoleon' attacked with 55 power.
Napoleon attacks!
In move #3, Aegislash 'Napoleon' attacked with 55 power.
Napoleon attacks!
Napoleon lands a critical hit!
In move #4, Aegislash 'Napoleon' attacked with 106.5625 power.
Napoleon attacks!
In move #5, Aegislash 'Napoleon' attacked with 55 power.
Napoleon attacks!
In move #6, Aegislash 'Napoleon' attacked with 55 power.
Napoleon attacks!
Napoleon lands a critical hit!
In move #7, Aegislash 'Napoleon' attacked with 106.5625 power.
Napoleon attacks!
In move #8, Aegislash 'Napoleon' attacked with 55 power.
Napoleon attacks!
In move #9, Aegislash 'Napoleon' attacked with 55 power.
Napoleon attacks!
In move #10, Aegislash 'Napoleon' attacked with 55 power.
```

Note that `starter_pokemon` lands a critical hit in moves 1, 4, and 7.

1.4 The `__gt__()` Method

Finally, we'll define the `Pokemon` class's `__gt__()` method. For our `Pokemon` class, this will mean the following: very simply, return `True` when the `Pokemon` object's ***speed*** stat is greater than the other `Pokemon` object's speed stat. Otherwise, return `False`:

```
def main():
   trainer_pokemon = Pokemon("Glaceon", "Miu", 80, "Ice", "", 200, 55, 90)
# speed stat: 90
   rival_pokemon = Pokemon("Sylveon", "Ayaka", 75, "Fairy", "", 160, 55,
80)  # speed stat: 80
   winner = "{}".format(trainer_pokemon.pokemon_name if trainer_pokemon >
rival_pokemon else rival_pokemon.pokemon_name)
   print(winner)

main()
```

Output:

```
Glaceon
```

Problem 2: The `ProfessorTrainer` Class

Oh yeah, it's all coming together. The `ProfessorTrainer` class will contain the following methods:

Method	Type of Method	Returns	Notes
`__init__()`	Dunder / Special	`None`	The constructor of the `ProfessorTrainer` class.
`__str__()`	Dunder / Special	`str`	Returns informal string representation of `ProfessorTrainer` object.
`_load_pokemon()`	Private	`None`	Loads a file passed into the `ProfessorTrainer` constructor and uses its contents to populate the `ProfessorTrainer`'s list of `Pokemon` objects.

2.1: The `__init__()` Method

Each `ProfessorTrainer` object will be equipped with the following attributes:

Attribute	Type	Comments

Attribute	Type	Comments
`professor_name`	*`str`*	The name of our professor.
`party`	*`[Pokemon]`*	That is, a list of `Pokemon` objects.

***Table 2**: `ProfessorTrainer`'s attributes. Note that these are **not** necessarily the parameters passed into the constructor.*

However, each `ProfessorTrainer` object will be instantiated, and will behave, as follows:

```
def main():
   file_path = "save_file_1.csv"
   trainer_name = "Daniel Katz"
   your_professor = ProfessorTrainer(trainer_name, file_path)

   print("Professor {}, and their
Pokemon:".format(your_professor.professor_name))
   for pokemon in your_professor.party:
       print("{}, a {}.".format(pokemon.nickname, pokemon.pokemon_name))

main()
```

Output:

```
Professor Daniel Katz, and their Pokemon:
Jun, a Mytheon.
Meghana, a Tsuneon.
Ollie, a Infernon.
Rihui, a Polareon.
Selina, a Faeron.
Semi, a Blitzeon.
Tanvi, a Spectreon.
Trisha, a Aureon.
Vipul, a Pitayeon.
```

That is, each `ProfessorTrainer` object will accept one string as its `professor_name`, and a second string denoting the name of a file where their Pokemon's information can be found. In the above case, file `save_file_1.csv` looks as follows:

```
Blitzeon,Andy,50,Electra,Dark,384,175,65,69
Empatheon,Ansh,50,Psychic,Fairy,514,195,117,28
Acideon,Apoorva,50,Fire,Water,600,180,135,70
Poppeon,Cindy,50,Psychic,Grass,290,140,45,90
Arcteon,Kevin,50,Dark ,Ice,600,180,135,70
Sireon,Kimberly,50,Dark ,Grass,400,165,73,85
Scaldeon,Luca,50,Water,Fire,238,272,67,63
Clairon, Sebastian,50,Psychic,Ice,400,165,73,85
```

In other words, you have to read this file inside your constructor and create a list of `Pokemon` objects from it. We could do this inside the `__init__()` method itself, but it would clutter it a bit too much.

Instead, initialize the `ProfessorTrainer` object's `party` attribute as an empty list. Then, make a call to a "private" method called `_load_pokemon()`, which will populate this empty list with the file's contents, and which we'll define next...

2.2 The `_load_pokemon()` Method

The `_load_pokemon()` will accept the file path string that was passed into the `ProfessorTrainer`'s `__init__()` method, and:

1. Attempt to open this file in read-mode.
2. If the file does not exist at the provided file path, return immediately.
3. If it does find the file, creates a `Pokemon` object per line in the file (ignore the header). Keep in mind that not every value per line is useful when creating a `Pokemon` object.
4. Append that new `Pokemon` object to the `ProfessorTrainer`'s `party` attribute.
5. Close the file, and return immediately.

Again, this method should be called inside the `__init__()` method ***after*** initializing the `party` attribute to an empty list.

2.3 The `__str__` Method

Write your `ProfessorTrainer` class's `__str__` method so that it behaves ***exactly*** as follows:

```
def main():
   their_professor = ProfessorTrainer("Gary Oak", "save_file_2.csv")
   print(their_professor)

main()
```

Output:

```
ProfessorTrainer object 'Gary Oak' and contains the following Pokemon
objects:
Blitzeon, called Andy ('Electra', 'Dark')
Empatheon, called Ansh ('Psychic', 'Fairy')
Acideon, called Apoorva ('Fire', 'Water')
Poppeon, called Cindy ('Psychic', 'Grass')
Arcteon, called Kevin ('Dark ', 'Ice')
Sireon, called Kimberly ('Dark ', 'Grass')
Scaldeon, called Luca ('Water', 'Fire')
Clairon, called Sebastian ('Psychic', 'Ice')
```

Pretty simple, right? Give yourself a challenge, and try to use a **list comprehension** in at least part of your implementation! Ask your TAs for guidance if you need it.

And that's it for the `ProfessorTrainer` class. Now for the fun.

Problem 3: Becoming the Tandon League Champion

Let's put our classes to good use. Write a standalone function (that is, not a method of either class) that will accept two `ProfessorTrainer` objects. This function, called `ultimate_showdown()`, will then:

1. Pick a random `Pokemon` object from each `ProfessorTrainer` object's `party` list. You may assume that each `ProfessorTrainer` object will have at least 1 `Pokemon` object inside their list. ***Be sure to not mutate either `ProfessorTrainer` object's `party` attribute when doing this.***
2. Have both `Pokemon` attack each other until either reaches an `hp` of 0. The way we determine who attacks first every turn is as follows: Determine which `Pokemon` object has the highest speed stat. Do this without accessing their `stats` attribute. (Hint: We defined the `__gt__()` attribute for the `Pokemon` class, which is based on the speed stat) If Pokemon A's speed stat is greater than Pokemon B's, then Pokemon A attacks first. Similarly, if Pokemon B's speed stat is greater than Pokemon A's, then Pokemon B attacks first. If they both have the same speed stat, flip a coin (50/50) to determine who attacks first.
3. The way we attack is by subtracting the attack stat of the attacking Pokémon from the `hp` attribute of the defending Pokeémon. ***Remember to check the defending Pokemon's `hp` attribute right after it gets attacked. If it fainted (that is, if it's `hp` attribute reached 0 or below) it can no longer attack. If this happens, go immediately to step 4.***
4. Return the `professor_name` attribute of whichever `ProfessorTrainer` object won.

Be aware that the `battle` function should not actually alter the `Pokemon` objects inside the `party` attribute (hint: the `copy` module is useful here).

Here is some sample behavior to consider. As before, I added some `print` function messages to better illustrate what is going on:

```
def main():
    your_professor = ProfessorTrainer("Daniel Katz", "save_file_1.csv")
    their_professor = ProfessorTrainer("Gary Oak", "save_file_2.csv")

    winner = ultimate_showdown(your_professor, their_professor)
    print("Professor {} wins the ultimate showdown!".format(winner))

main()
```

Possible output:

```
Professor Daniel Katz sends out a Polareon called 'Rihui'!
Professor Gary Oak sends out a Clairon called 'Sebastian'!

Sebastian attacks!
Rihui attacks!
```

```
Sebastian attacks!
Rihui attacks!
Sebastian attacks!
Sebastian lands a critical hit!
Rihui attacks!
Professor Daniel Katz's Pokemon Rihui wins!
Professor Daniel Katz wins the ultimate showdown!
```

Another possible output:

```
Professor Daniel Katz sends out a Spectreon called 'Tanvi'!
Professor Gary Oak sends out a Blitzeon called 'Andy'!

Tanvi attacks!
Andy attacks!
Tanvi attacks!
Andy attacks!
Tanvi attacks!
Andy attacks!
Tanvi attacks!
Andy attacks!
Tanvi attacks!
Tanvi lands a critical hit!
Professor Daniel Katz's Pokemon Tanvi wins!
Professor Daniel Katz wins the ultimate showdown!
```

Where `save_file_2.csv` looks as follows:

```
Pokemon Name,Name,Level,Type 1,Type 2,Total,HP,Attack,Speed
Blitzeon,Andy,50,Electra,Dark,384,175,65,69
Empatheon,Ansh,50,Psychic,Fairy,514,195,117,28
Acideon,Apoorva,50,Fire,Water,600,180,135,70
Poppeon,Cindy,50,Psychic,Grass,290,140,45,90
Arcteon,Kevin,50,Dark ,Ice,600,180,135,70
Sireon,Kimberly,50,Dark ,Grass,400,165,73,85
Scaldeon,Luca,50,Water,Fire,238,272,67,63
Clairon,Sebastian,50,Psychic,Ice,400,165,73,85
```

I have included both of these sample files **here** and **here**, in case you want to use them, but feel free to create your own and have fun with it!