NYU Tandon School of Engineering

CS-UY 1114 Fall 2022

Homework 10

Due: 11:59pm, Thursday, December 8th, 2022

Submission instructions

1. You should submit your homework on **Gradescope**.
2. For this assignment you should turn in 3 `.py` files named according to the following pattern: `hw10_q1.py`, `hw10_q2.py`, and `hw10_q3.py`.
3. Your Python file should contain a header comment block as follows:

```
"""
Author: [Your name here]
Assignment / Part: HW10 - Q1
Date due: 2022-12-08, 11:59pm
I pledge that I have completed this assignment without
collaborating with anyone else, in conformance with the
NYU School of Engineering Policies and Procedures on
Academic Misconduct.
"""
```

No late submissions will be accepted.

REMINDER: *Do not use any Python structures that we have not learned in class.*

The use of `eval()` and `break` are no longer permitted in this class.

For this specific assignment, you may use everything we have learned up to, **and including**, object-oriented programming. Please reach out to us if you're at all unsure about any instruction or whether a Python structure is or is not allowed.

Do **not** use, for example:

1. Modules we haven't covered.
2. The `with` keyword to open files.
3. Any modules that handle files non-natively (i.e. `csv`, `pandas`, etc.) Using them will result in a zero for that problem. Use `open`, `readline`, etc. instead—not only in this lab, but in the course in general.

Note On Importing Classes and Functions From Other `.py` Files

The problems in this assignment will require you to create two classes and to use them in a standalone function. In order to avoid having everything in a single file, we're going to split them up into three files. The way we transfer structures from one file to another is by using the `import` keyword. Let's say we have file A and file B, and file B uses a class and a function from file A. The way we would set this up is as follows:

```python
# file_a.py
class Something:
    pass


def some_function():
    pass
```

```python
# file_b.py
from file_a import Something, some_function

something = Something()      # We can now create Something objects and call
some_function()
some_function()              # as if they had been defined in file_b.py
```

If Python (read: IDLE) gives you trouble when doing this, check in with our CAs—they should be able get it fixed.

Problems

Problem 1: *Tools Of The Trade*

The whole point of this assignment is to simulate two musicians having a battle (think **Scott Pilgrim vs The World's bass battle). To do this, we're going to create two classes, an `Instrument` class (i.e. their instrument of choice) and the `Musician` class (i.e. the musician using the instrument).

Let's start with `Instrument` class, since this one is simpler.

1.1: *Creating `Instrument` Objects*

Start with the initializer method, which will accept three parameters from the user:

Attribute	Type	Comments
`model`.	*str*	The model of our instrument.
`brand`	*str*	The brand of this instrument.

Attribute	Type	Comments
`strength`	*`float`*	It's "strength" value, represented by a `float` from 0.0 to 1.0.

Table 1: Attributes of the **`Instrument`** class. Please make sure the spelling of your attributes matches those given here. You can assume that the user will always enter a valid value for `strength`.

If you implement your initializer method correctly, your **`Instrument`** objects should behave as follows:

```
def main():
    fender_vi = Instrument("VI Bass", "Fender", 0.99)
    print(fender_vi.model)
    print(fender_vi.brand)
    print(fender_vi.strength)

main()
```

Output:

```
VI Bass
Fender
0.99
```

1.2: Printing `Instrument` Objects

Here, your goal is to simply make sure that the following behavior occurs when printing objects of the `Instrument` class:

```
fender_vi = Instrument("VI Bass", "Fender", 0.99)
four_double_o_one = Instrument("4001C64 Bass", "Rickenbacker", 0.856)

print(fender_vi)
print(four_double_o_one)
```

Output:

```
Fender VI Bass (99.0 / 100 strength)
Rickenbacker 4001C64 Bass (85.6 / 100 strength)
```

Note that your output format must match the examples' exactly.

1.3: *The `does_break()` Method*

This method will do the following:

- If a randomly-generated float value from 0.0 to 1.0 is **smaller** than **1/2** of the `strength` attribute of this **`Instrument`** object, **`does_break()`** will return `True`, meaning the instrument has broken.
- Otherwise, return `False`, meaning the instrument has stood the test of time and not broken.

That is, the stronger a **`Instrument`** object is, the more likely it is to break.

Consider the following *possible* sample behavior:

```
def main():
    danelectro = Instrument("Stock '59", "Danelectro", 0.25)

    number_of_tests = 100
    number_of_breaks = 0

    # I'm testing does_break() 100 times and keeping track of how many times it
breaks
    for i in range(number_of_tests):
        if danelectro.does_break():
            number_of_breaks += 1

    percentage = (number_of_breaks / number_of_tests) * 100

    print(f"The {danelectro.model} broke {round(percentage)}% of the time in
{number_of_tests} tests!")

main()
```

Possible output:

```
The Stock '59 broke 16% of the time in 100 tests!
```

Please make sure you understand and have gotten it to work perfectly before moving on to the next part, as we'll be making use of **`Instrument`** objects.

Problem 2: *Artist of The Year*

Next up, we'll be creating our musicians. Since this class will exist in a different file than your `Instrument` class, we will need to import our `Instrument` class into our file. Simply add this line at the top of `hw10_2.py` and you should be good to go:

```
from hw10_1 import Instrument
```

Once you do this, you will be able to create `Instrument` objects in file `hw10_2.py`.

Your new class will be called **`Musician`**, and will contain the following methods:

2.1: *Creating `Musician` Objects*

Similar to our `Instrument` class, define the initializer for our `Musician` class, which will create the following attributes:

Attribute	Type	Comments
`stage_name`.	*`str`*	The name of our musician.
`instruments`.	*`list[Instrument]`*	That is, a list of `Instrument` objects.
`number_of_instruments`	*`int`*	That is, the number of `Instrument` objects inside `instruments`

Table 2: Attributes of the `Musician` class.

Of these three attributes, the user will only pass in values for `stage_name` and `instruments`. Your initializer must create `number_of_instruments` using information from `instruments`.

If you implement your initializer correctly, your `Musician` objects should behave as follows:

```python
# Creating our Instrument objects
danelectro = Instrument("Stock '59", "Danelectro", 0.25)
fender_vi = Instrument("VI Bass", "Fender", 0.99)
four_double_o_one = Instrument("4001C64 Bass", "Rickenbacker", 0.856)

gear = [danelectro, fender_vi, four_double_o_one]

# Creating our Musician object
sad_musician = Musician("Robert Smith", gear)

# Checking the Musician object's attributes
print(sad_musician.stage_name)
print(sad_musician.number_of_instruments)

for instrument in sad_musician.instruments:
    print(instrument)
```

Output:

```
Robert Smith
3
Danelectro Stock '59 (25.0 / 100 strength)
Fender VI Bass (99.0 / 100 strength)
Rickenbacker 4001C64 Bass (85.6 / 100 strength)
```

2.2: *Printing `Musician` Objects*

Implement the `Musician` class such that you get the following behavior when printing objects of its class:

```python
# Creating our Instrument objects
danelectro = Instrument("Stock '59", "Danelectro", 0.25)
```

```
fender_vi = Instrument("VI Bass", "Fender", 0.99)
four_double_o_one = Instrument("4001C64 Bass", "Rickenbacker", 0.856)

gear = [danelectro, fender_vi, four_double_o_one]

# Creating our Musician object
sad_musician = Musician("Robert Smith", gear)

print(sad_musician)
```

Output:

```
Musician object 'Robert Smith', owning a Danelectro Stock '59 (25.0 / 100
strength), Fender VI Bass (99.0 / 100 strength), and a Rickenbacker 4001C64
Bass (85.6 / 100 strength)
```

The output format must match *exactly* as the one above. Note that the number of instruments for any `Musician` object may be more, or less, than 3.

2.3: *The `pick_instrument()` Method*

Define a method for the `Musician` class called `pick_instrument()` that:

- Accepts a single parameter, `instrument_index`, representing a location within the `Musician` object's `instruments` list.
- Returns the `Instrument` object at location `instrument_index`.
  - If the value of `instrument_index` is larger than the size of `instruments`, this method will return the last `Instrument` object in `instruments`.
  - `instrument_index` will have a *default value* of `None`. If the user chooses not to pass in a value for `instrument_index`, `pick_instrument()` will return *a random `Instrument` object from `instruments`*.
  - If `instruments` is an empty list, return `None`.

In other words, all of the following invocations of `pick_instrument()` must work and return either an `Instrument` object or `None`:

```
instrument = sad_musician.pick_instrument(2)
instrument = sad_musician.pick_instrument(100000000)
instrument = sad_musician.pick_instrument()
```

Problem 3: *Battle of The Bands*

Note: This function must be written in the file `hw10_3.py`. In order to make use of the `Musician` class, you'll need to import it from your previous file as such:

```
from hw10_q2 import Musician
from hw10_q1 import Instrument
```

Write a *standalone function* called **`get_name_of_battle_winner()`**, which will do the following:

- Accept two parameters, both of which you can assume will always be `Musician` objects.
- The function will then pick a random `Instrument` object from each of the `Musician` objects in this duel. Be sure to check that each `Musician` object has at least one instrument. If either of them don't have any instruments, the other `Musician` automatically wins.
- If both players don't have any instruments, return the string `"NO CONTEST"`.
- Finally, **`get_name_of_battle_winner()`** will first check which `Instrument` object's `strength` attribute is larger. Let's say musician A's instrument is stronger than musician B's. If so, our program will call musician ***A***'s `Instrument` object's `does_break()` method. If it returns `True` (that is, if it breaks), Musician B wins in an upset. Otherwise, musician A wins. If musician B's instrument was stronger than musician A's, we do the same process, but instead calling musician ***B***'s `Instrument` object's `does_break()` method. If both `Instrument` objects happen to have the same `strength` value, the winner will be decided by a 50/50 random coin-toss.
- **Whichever `Musician` wins, return their `stage_name` attribute**.

WARNING: When picking `Instrument` objects from each `Musician` object in the duel, make sure not to remove it from that `Musician` object's `instruments` list. In other words, each `Musician` object's `instruments` list must never change once it is initialized.

If you successfully implement this method, you should see similar behavior to the following example. I added a few `print()` function calls in my **`get_name_of_battle_winner()`** method to better illustrate what is happening behind the scenes. Feel free to do this as well if it helps you, but it is **not** necessary. As long as the function returns the correct name, that is enough:

```
def main():
    danelectro = Instrument("Stock '59", "Danelectro", 0.25)
    fender_vi = Instrument("VI Bass", "Fender", 0.99)
    four_double_o_one = Instrument("4001C64 Bass", "Rickenbacker", 0.856)

    gear = [danelectro, fender_vi, four_double_o_one]

    # Let's say both musicians have access to the same gear
    sad_musician = Musician("Robert Smith", gear)
    less_sad_musician = Musician("Miki Berenyi", gear)

    # Testing the get_name_of_battle_winner method a few times
    number_of_duels = 10

    for duel_number in range(number_of_duels):
        winner_name = get_name_of_battle_winner(sad_musician,
less_sad_musician)
        print(f"THE WINNER OF DUEL #{duel_number + 1} IS {winner_name}!",
end="\n\n")

main()
```

Possible output:

```
Robert Smith picked a Fender VI Bass (99.0 / 100 strength)!
Miki Berenyi picked a Fender VI Bass (99.0 / 100 strength)!
Both musician's instruments are the same strength. The winner will be decided
by the whim of chance.
THE WINNER OF DUEL #1 IS Robert Smith!

Robert Smith picked a Danelectro Stock '59 (25.0 / 100 strength)!
Miki Berenyi picked a Rickenbacker 4001C64 Bass (85.6 / 100 strength)!
THE WINNER OF DUEL #2 IS Miki Berenyi!

Robert Smith picked a Danelectro Stock '59 (25.0 / 100 strength)!
Miki Berenyi picked a Danelectro Stock '59 (25.0 / 100 strength)!
Both musician's instruments are the same strength. The winner will be decided
by the whim of chance.
THE WINNER OF DUEL #3 IS Miki Berenyi!

Robert Smith picked a Fender VI Bass (99.0 / 100 strength)!
Miki Berenyi picked a Rickenbacker 4001C64 Bass (85.6 / 100 strength)!
THE WINNER OF DUEL #4 IS Robert Smith!

Robert Smith picked a Rickenbacker 4001C64 Bass (85.6 / 100 strength)!
Miki Berenyi picked a Fender VI Bass (99.0 / 100 strength)!
THE WINNER OF DUEL #5 IS Miki Berenyi!

Robert Smith picked a Fender VI Bass (99.0 / 100 strength)!
Miki Berenyi picked a Fender VI Bass (99.0 / 100 strength)!
Both musician's instruments are the same strength. The winner will be decided
by the whim of chance.
THE WINNER OF DUEL #6 IS Miki Berenyi!

Robert Smith picked a Fender VI Bass (99.0 / 100 strength)!
Miki Berenyi picked a Danelectro Stock '59 (25.0 / 100 strength)!
Robert Smith's VI Bass broke!
THE WINNER OF DUEL #7 IS Miki Berenyi!

Robert Smith picked a Rickenbacker 4001C64 Bass (85.6 / 100 strength)!
Miki Berenyi picked a Danelectro Stock '59 (25.0 / 100 strength)!
Robert Smith's 4001C64 Bass broke!
THE WINNER OF DUEL #8 IS Miki Berenyi!

Robert Smith picked a Rickenbacker 4001C64 Bass (85.6 / 100 strength)!
Miki Berenyi picked a Fender VI Bass (99.0 / 100 strength)!
Miki Berenyi's VI Bass broke!
THE WINNER OF DUEL #9 IS Robert Smith!

Robert Smith picked a Danelectro Stock '59 (25.0 / 100 strength)!
Miki Berenyi picked a Rickenbacker 4001C64 Bass (85.6 / 100 strength)!
Miki Berenyi's 4001C64 Bass broke!
THE WINNER OF DUEL #10 IS Robert Smith!
```