CSE101

DUE JUNE 01

23:59 KST

ASSIGNMENT V

PROFESSOR

FRANCOIS

RAMEAU

OOP

General instructions

- Please add your name and email in the dedicated location at the top of your code
- Do not use any external library, except when explicitly requested
- Try to follow the naming convention proposed by PEP-8 seen in class
- Use ONLY what we have already seen in class. If not, you will get ZERO points
- Use meaningful names for your variables and functions
- If you face problem submitting your code via GitHub please contact the professor and the TA by email
- Note that the received code will be tested on a classifier to detect potential usage of Large Language Model. We will also pay particular attention to plagiarism
- Leave comments in your code to explain your code and describe the difficulties you faced

INVITATION LINK

https://classroom.github.com/a/p1U1Uo4V

In this assignment, we will create a turned-based game simulating a gladiator fight. Our game will consist of two gladiators, each equipped with a weapon and armor. Each gladiator will have different attributes, including strength and luck, which will affect the outcome of the fight.

The fight will take place in an arena where the gladiators will take turns attacking each other. The damage each gladiator deals will be determined by their weapon's damage, their strength, and a bit of luck. A gladiator's armor and luck will reduce the incoming damage when defending.

Your task is to implement the Weapon, Armor, Gladiator, and Arena classes, and to create a fight simulation. This assignment will give you a chance to apply object-oriented programming concepts and see how they can be used to create an interactive program. Note that we will not use any inheritance in this homework (it is for simplicity, but inheritance could be very interesting for such a task).

Part I: The Equipment (2 points)

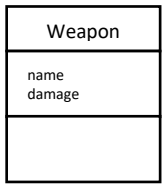
The very first thing your fighters will need is equipment to attack opponents and protect themselves. Therefore, we will start creating a class weapon and armor.

1. The Weapons (1 points)

You will "forge" your weapons thanks to your `Weapon` class! The `Weapon` class is responsible for storing information about a gladiator's weapon. This includes the name of the weapon and the damage it can deal. The damage is a base value that will later be adjusted by a Gladiator's strength and luck during an attack.

You will create this class weapon with an `__init__` method that will be used to initialize the weapon name and its damage (Remember to use self to refer to the class variables) in two class attributes called `name` and `damage`. The game developer (you!) will manually select the name and damage of each weapon in your armory. For instance, after your class is finalized, you will create various weapons to equip your gladiators. We want our game to be historically correct so we can get inspiration from real gladiator's weapons: list. For instance, we will create the following:

```
pugio = Weapon("pugio", 2)
spatha = Weapon("spatha", 10)
fascina = Weapon("fascina", 12)
```



Be creative and "forge" as many weapons as you judge required for your fighters. It is okay if you are not completely historically accurate 😉.

What should you implement for the Weapon class?

1.	An __init__ method that initializes the name and damage	1 point
2.	Declare multiple weapons in the main	0 point

2. The armors (1 point)

Your gladiators will also have to be protected. The `Armor` class is similar to the `Weapon` class, but instead of dealing damage, it reduces it. This class holds information about a gladiator's armor, specifically the name of the armor and its defense value that you will save in two class attributes `name` and `defense`. The defense is a base value that will later be adjusted by a Gladiator's luck when defending against an attack. Follow the same instructions as for the Weapon class.

As for the weapon, you will fill your armory with various types of armor to equip your gladiators:

Armor
name defense

```
cuirass = Armor("cuirass", 5)
helmet = Armor("helmet", 2)
```

What should you implement for the Armor class?

1. An __init__ method that initializes the name and defense — 1 point
2. Declare multiple armors in the main — 0 point

Part II: The gladiators (4 points)

Gladiator
name health weapon armor strength luck
attack() defend()

Now that we have the Gladiator's equipment ready, it's time to create the Gladiators themselves. A Gladiator in our game is created using our `Gladiator` class, which has certain **attributes** and **methods**.

1. Attributes of the gladiators (2 points)

Each Gladiator has a name, health points, a weapon, armor, strength, and luck.

- The `name` attribute is a string representing the Gladiator's name. The name is provided by the programmer when initializing the gladiator
- The `health` attribute is an integer initialized at 100 for each gladiator, representing the Gladiator's current health points.
- The `weapon` attribute is an instance of the Weapon class we have previously defined. The programmer will assign a weapon to his fighter during instantiation.
- The `armor` attribute is an instance of the Armor class we have previously defined. The programmer will assign an armor to his fighter during instantiation.
- The `strength` attribute is a float representing the Gladiator's base strength which affects the damage they deal in battle. This attribute will also be provided during instantiation. Typically, this strength characteristic will range between 1.0 to 2.0 (it is a damage multiplier), but here it is the developer's choice when building his characters.
- Finally, the `luck` attribute is a float representing the Gladiator's luck factor that randomly affects their attacking and defending power.

Thus, the definition of your `__init__` method should look like this:

```
def __init__(self, name, weapon, armor, strength, luck):
```

After your gladiators class is declare with the proper attributes in the **`__init__`** method, you will instantiate multiple gladiators in the main part of the code (try to look for a database of common gladiator's names if you want to add realism to your simulation):

```
spartacus = Gladiator("spartacus", spatha, helmet, 1.5, 0.2)
flamma = Gladiator("flamma", fascina, cuirass, 1.7, 0.1)
attilius = Gladiator("attilius", pugio, helmet, 1.1, 0.5)
```

What should you implement for the Gladiator class attribute?

1. An __init__ method that initializes all the attributes mentioned — 2 point
2. Declare multiple gladiators in the main — 0 point

2. Methods of the gladiators (2 points)

Your gladiators are now equipped and in good health, they are ready for action! The behavior of your gladiators is simple; they can either attack or defend. These two behaviors will be defined in two different methods called respectively **`attack()`** and **`defend()`**.

II.2.1 - The attack method

The attack method calculates the damage dealt by the Gladiator. It multiplies the weapon damage by the Gladiator's strength and a luck factor (the luck factor is a random multiplier in the range [0 to 1+luck] (uniform distribution)). This behavior will be programmed in the method **`attack().`** For every call of this function, the luck factor is estimated using the Python random module, and the quantity of damage is calculated as follows:

$damage = w * s * l$, where w is the weapon damage, s the strength of the gladiator and l the luck factor sampled before. The method **`attack()`** returns the quantity of damage dealt for this call!

II.2.1 - The defense method

The damage computed for the gladiator's attacker can be used to inflict damage on the other gladiator. This process is implemented in the **`defend()`** method. The **`defend()`** method takes the incoming damage as a parameter and calculates the net damage after the armor defense is subtracted.

This net damage is also affected by a luck factor (the luck factor is recomputed for each call of the method), similar to the attack method. The formula for the net damage is the following: $net\_damage = damage * armor\_defense * luck\_factor$. Then, we subtract this net damage from the Gladiator's health. If the net damage is lower than zero, then enforce it to be zero.

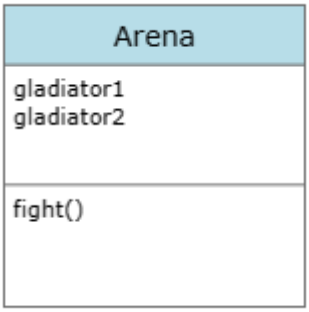
Note that this method is void since it only affects an object's attribute: `self.health`

What should you implement for the Gladiator class methods?

1.	Define an attack() method with the behavior described	1 point
2.	Define an defend() method with the behavior described	1 point

In the next section, we will place our Gladiators into an Arena and let the fight begin!

Part III: The Arena (4 points)



The final piece of our Gladiator Fight Simulation is the Arena. The Arena is where our Gladiators will engage in their fight. It's represented by the Arena class, which has its own attributes and methods.

1. Attributes of the arena (1 point)

An Arena instance has two main attributes: `gladiator1` and `gladiator2`, representing the two Gladiators who will fight in this Arena.

What should you implement for the Arena class attribute?

1.	An __init__ method that initializes the two attributes mentioned	1 point
2.	Declare an arena in the main (already done)	0 point

2. Method of the arena (3 points)

The Arena class has one main method: `fight`().

The `fight` method initiates a turn-based fight between the two Gladiators. The Gladiators take turns attacking each other until one of them has no more health points (i.e., their health reaches 0 or below); you might need a while loop here. The method should print out a message for each attack, showing who

attacked whom and how much damage was dealt. Once a Gladiator's health points reach 0 or less, the fight ends, and the method prints out the name of the winner.

What should you implement for the Arena class method?

1.	Implement the method fight() in the arena class	3 points
2.	Run a fight in the main (already done)	0 point

With this, we have all the pieces in place for our Gladiator Fight Simulation. Your final task is to create a few Gladiators, equip them with Weapons and Armors, place them in an Arena, and start a fight.

Part IV? (for fun)

Feel free to continue this project further if you like it, for instance:

- The arena can have spectators who can give grace (50% chance) if one of the fighters reach less than 10 points of life. His life can be spared, and the fight can end before his death.
- Multiple specific types of gladiators existed in reality, each with different characteristics (high speed, heavy armor, use of specific weapon). You can add more attributes to the gladiator and also create new classes inheriting from the gladiator class with specific methods and characteristic
- Make new specialized weapons inheriting from the class weapon which will inflict the fighter in different manners
- Add critical hit randomly
- Make more than two fighters join the arena