Part1: analyser et modifier

Your objective is to correct the python files in `part1` folder so that they can pass the tests. Read this document carefully as it explain the rules, and docstring might contain useful information.

```
Point abordés:
  - Class: [Concrètement, un objet est une structure de données qui contient un état (Attr
  définis dans la classe), et des comportements (méthodes ou fonctions définies dans la cl
  org/wiki/Programmation_orient%C3%A9e_objet)
    - un attributs public est defini par `self.variable_name`
    - un attributs privé est defini par `self._variable_name`
    - properties are methods: getter and setter `@properties`
    - public methods are function defined inside class, the first arguments is always `sel
    - private methods are function in class whose name start with "_" `def _private()`
    - special methods are builtins one they start with __funcname__, `__len__ __getitem__`
    syntax like `len(obj)` `obj[x]` they are plenty of them.

  - inheritance/Heritage: prenons l'exemple du loup et du mammifère
    - Imaginons une class Mammifere avec les attributs et méthodes comme suit
  ```
    class Mammifere:
      def __init__(self, gender):
        self.gender = gender
      @abs.abstractmethod()
  ``` def reproduce(cls, *args):
        return cls(*args)

  - Prenons la class fille (qui herite d'une class, ici 'Mammifere') `Loup`. Qui Hérite de M
```

ce cas les attributs et méthodes de `Mammifere` sont present dans `Loup`

```
class Loup(Mammifere):
  def __init__(self, gender):
    super().__init__(gender)
  def reproduce(self, loup):
    if self.gender != loup.gender:
      return super().reproduce(Loup, rand.choices("Male", "Woman"))
  def protect(self):
    pass
  def hunt(self):
    # Do Something
    pass

loup_m = Loup("Male")
loup_f = Loup("Female")
louveteau = loup_f.reproduce(loup_m)
loup_m.protect()
```

```
loup_f.hunt()
```

Warrior

Let's start with a simple Warrior: - Health: 50. - Attack: 5.

Fight

The function `fight()` defined inside the file `fight` initiates a duel between 2 warriors and define the strongest of them. The fight ends with the death of one of them. The `unit_1` attacks first then the `unit_2`, etc...

If the first warrior is still alive (and thus the other one is not anymore), the function should return True, False otherwise.

Defender

Defender is a subclass (inheritance from) of the Warrior Class. - It should have his attack at: `3`. - His life point to `55` - He has 2 point of defense.

The defender receive less damage `damage=(damage-defense)`

Army

The new class should be the Army and have the method `addUnits()` - for adding the chosen amount of units to the army. The first unit added will be the first to go to fight, the second will be the second, ... We should be able to access elements from the army with the `[]` syntax. create a method `pop` which remove the first warrior of the army and return it.

Battlefield

Also you need to create a Battlefield() class with a `simpleFight()` function, which will determine the strongest army. The battles occur according to the following principles: At first, there is a duel between the first warrior of the first army and the first warrior of the second army. As soon as one of them dies - the next warrior from the army that lost the fighter enters the duel, and the surviving warrior continues to fight with his current health. This continues until all the soldiers of one of the armies die. In this case, the `simpleFight()` method should return True , if the first army won, or False , if the second one was stronger.

Specs: Battlefield could be a class with only one methods but it is not required.