ISTA 130: Fall 2015
Programming Assignment 10
(100 points)
CLASSES

Due : Thursday, December 3rd at 11:59 pm

(submit via D2L dropbox)

Please read the instructions below carefully and follow them closely. All problems (and sub-parts of the problems) are required except as noted in the instructions below. If you have any questions, please email the instructor or one of the section leaders.

Important: Your filenames must be identical to the filenames given below. For any functions you are asked to write, the function signature (header) must be exactly as described in the instructions. That is, you must use the exact function names given in the instructions, you must have the parameters the instructions ask for, and the parameters must be in the order the instructions give.

Important: Make sure you always save a backup of your work somewhere safe (such as your D2L locker, Dropbox, etc).

Important: You are not just graded on whether your code produces the same result as the examples show. You should be using what you've learned to do your best to write good code. For example, things like poorly named variables/functions, duplicating code where you could instead use a loop, and missing documentation will cost you points.

1.0) RPG COMBAT (100 POINTS)

1.1) DETAILS

In this assignment you will write a program to simulate combat for a simple Role Playing Game (RPG). The game will have a single type of character, the Fighter. Each individual Fighter will have an integer value called "hit points" that represents his/her current health (the amount of damage he/she can sustain before death).

The program will simulate combat between two Fighters. Combat will be divided into rounds. During each round each combatant will make one attempt to strike the other. Determining which Fighter attacks first in a round is important because the first attacker might kill his opponent before the opponent has a chance to attack him.

In table top RPGs the order of attacks is often determined by rolling dice. For example, each combatant rolls a six-sided die. The higher roller attacks first. The lower roller attacks second if he is still alive. We'll use random numbers to simulate the rolling of dice. In the event of a tie, we'll consider the attacks to occur simultaneously (i.e. the Fighters attempt to strike each other at the exact same moment). During a simultaneous attack both Fighters will get to attack even if one (or both) is (are) killed during that round.

We'll use a random number to determine whether an attack attempt is successful or not. Each successful attack will inflict damage on the opponent. To simulate damage, we'll reduce the opponent's hit points by another random number. When a Fighter's hit points are reduced to 0 (or less than 0) he is considered to be dead.

Combat rounds will continue until one (or both) combatants are dead. Create a new file called "rpg.py". In the file:

1.) Write a class called **Fighter**. Inside the class:

A.) Write a initializer method called **__init__** that takes 2 parameters: self (all of your methods will have this as the first parameter), and name (a string, the name of a fighter). The initializer method will:

- set a name attribute to the value of the name parameter.
- set a hit_points attribute to 10 (i.e. all fighters begin life with 10 hit points.)

B.) Write a method called **__repr__** that takes one parameter: self. The method returns a string showing the name and hit points of the instance in the following format:

Bonzo (HP: 9)

- In this example “Bonzo” is the Fighter’s name and he currently has 9 hit points.

C.) Write a method called **take_damage** that takes two parameters: self (the Fighter instance that calls the method), and damage_amount (an integer representing the number of hit points of damage that have just been inflicted on this Fighter):

i.) The method should first decrease the hit_points attribute by the damage_amount.

ii.) It should next check to see if the Fighter has died from the damage:

- A hit_points value that is zero or less indicates death. In that case print a message as shown in the following example:

\tAlas, Bonzo has fallen!

  - In this example the Fighter’s name is “Bonzo”

- Otherwise print a message as shown in the following example:

\tBonzo has 5 hit points remaining.

  - In this example the Fighter’s name is “Bonzo” and he has 5 hit points left over after the damage.

iii.) This method returns nothing.

D.) Write a method called **attack** that takes two parameters: self (the Fighter instance that calls the method), and other (another Fighter instance being attacked by self)

i.) The method will print the name of the attacker and attacked in the following format:

Bonzo attacks Chubs!

ii.) Next determine whether the attack hits by generating a random integer between 1 and 20. Use the randrange function from the random module to get this number. A number that is 12 or higher indicates a hit*:

- For an attack that hits:

  a.) Generate a random integer between 1 and 6 (use randrange) to represent the amount of damage inflicted by the attack.

  b.) Print the amount of damage inflicted as in the following example:

  \tHits for 4 hit points!

  c.) Invoke the **take_damage** method on the victim (i.e. on other), passing it the amount of damage inflicted.

- For an attack that misses, just print the following message:

  \tMisses!

iii.) This method returns nothing

E.) Write a method called **is_alive** that takes one parameter: self (the Fighter instance that calls the method).

i.) The method returns `True` if self has a positive number of points, `False` otherwise.

2.) Outside of the class write a function called **combat_round** that takes two parameters. The first is an instance of Fighter. The second is another instance of Fighter.

i.) The function determines which of the two fighters attacks first for the round by generating a random integer (use randrange) between 1 and 6 for each fighter:

- If the numbers are equal:

a.) Print the following message:

Simultaneous!

b.) Have each fighter instance call his attack method on the other fighter (the order of the two method calls doesn't matter since we're considering these to be simultaneous attacks)

- If one number is larger:

a.) The fighter with the larger roll attacks first (by calling his attack method and passing it the fighter being attacked)

b.) If the second fighter survives the attack (call is_alive), he then attacks the first.

ii.) This function returns nothing.

3.) In **main**:

i.) Create two instances of the Fighter class (you can choose the names of your Fighters)

ii.) Next repeat the following process until one (or both) fighters have been slain (remember, a Fighter is dead when it has less than 1 hit point):

a.) Print the combat round number (start with 1) as shown in the following example:

========== ROUND 1 ==========

NOTE: There are 19 equal signs on each side

b.) Print each Fighter's information (remember how the **__ repr__** method works). E.g.:

Bonzo (HP: 4)

Chubs (HP: 7)

c.) Use the input function to pause the program (like we did in our turtle graphics programs) until the user presses enter. Prompt the user with the following message:

Enter to Fight!

d.) Use your **combat_round** function to simulate a single round of combat.

* For the curious: The numbers are from Advanced Dungeons & Dragons in which you roll a 20 sided die to determine a hit. They assume the attacker is a 1st level Fighter swinging a club and the defender is wearing leather armor.

iii.) Finally print a message indicating the battle has ended, followed by the information for each of the Fighters, as shown in the following example:

The battle is over!

Bonzo (HP: 3)

Chubs (HP: -2)

4.) Adjust all of your output so that it looks pretty. See the example output below.

5.) Verify that your documentation makes sense and that you've added documentation to each of your functions.

6.) **Verify that your program works**

7.) Upload your file to the Program 10 dropbox folder on D2L

1.1) EXAMPLE OUTPUT

The following is an example of the output you might see when running this program:

NOTE: All indents are done by a single tab character

```
================== ROUND 1 ==================
Bonzo (HP: 10)
Chubs (HP: 10)
Enter to Fight!
Simultaneous!
Bonzo attacks Chubs!
	Hits for 1 hit points!
	Chubs has 9 hit points remaining.
Chubs attacks Bonzo!
	Misses!

================== ROUND 2 ==================
Bonzo (HP: 10)
Chubs (HP: 9)
Enter to Fight!
Chubs attacks Bonzo!
	Hits for 4 hit points!
	Bonzo has 6 hit points remaining.
Bonzo attacks Chubs!
```

```
        Hits for 3 hit points!
        Chubs has 6 hit points remaining.

================== ROUND 3 ==================
Bonzo (HP: 6)
Chubs (HP: 6)
Enter to Fight!
Bonzo attacks Chubs!
        Hits for 3 hit points!
        Chubs has 3 hit points remaining.
Chubs attacks Bonzo!
        Hits for 1 hit points!
        Bonzo has 5 hit points remaining.

================== ROUND 4 ==================
Bonzo (HP: 5)
Chubs (HP: 3)
Enter to Fight!
Simultaneous!
Bonzo attacks Chubs!
        Hits for 5 hit points!
        Alas, Chubs has fallen!
Chubs attacks Bonzo!
        Hits for 2 hit points!
        Bonzo has 3 hit points remaining.

The battle is over!
Bonzo (HP: 3)
Chubs (HP: -2)
```

Here is a second example of running the program:

========== ROUND 1 ==========

Bonzo (HP: 10)

Chubs (HP: 10)

Enter to Fight!

Bonzo attacks Chubs!

    Hits for 3 hit points!

    Chubs has 7 hit points remaining.

Chubs attacks Bonzo!

    Hits for 3 hit points!

    Bonzo has 7 hit points remaining.

========== ROUND 2 ==========

Bonzo (HP: 7)

Chubs (HP: 7)

Enter to Fight!

Bonzo attacks Chubs!

    Misses!

Chubs attacks Bonzo!

    Hits for 4 hit points!

    Bonzo has 3 hit points remaining.

========== ROUND 3 ==========

Bonzo (HP: 3)

Chubs (HP: 7)

Enter to Fight!

Simultaneous!

Bonzo attacks Chubs!

    Misses!

Chubs attacks Bonzo!

    Misses!

========== ROUND 4 ==========

Bonzo (HP: 3)

Chubs (HP: 7)

Enter to Fight!

Chubs attacks Bonzo!

Hits for 4 hit points!

Alas, Bonzo has fallen!

The battle is over!

Bonzo (HP: -1)

Chubs (HP: 7)