CSCI 241 Fall 2015
Project 1: Classes, Objects, and Arrays

DUE 0500 Saturday, 26 September 2015

AT NO POINT MAY YOU WORK ON THIS PROJECT WITHOUT THE PHYSICAL PRESENCE OF YOUR TEAMMATE

For this project, you will work in teams to create two classes. One will represent the abstract data type Score and the other will represent the abstract data type Scoreboard. Your score class will be used to create objects that track several aspects of a player's performance during an arcade-style game. Your Scoreboard class will maintain (as long as the program is running) a leaderboard of high scores sorted in decreasing order. This will help reinforce the concepts of classes and objects, and it will help you develop your skills with Python syntax in general. Here are the items that we will track in our Score class:

- **The player's name.** Because objects of the Score class will be used as values in the high score leaderboard, we need a player name to be associated with each object. That way, when we see then entries in the scoreboard, it is clear who owns each score.
- **The current score.** Our Score class won't be very useful to the game if it doesn't keep track of how many points the player has.
- **The number of lives remaining.** Another score-related concept is how many lives a player has remaining.
- **The current level.** As players advance through a game, it gets more difficult. Higher levels indicate more difficulty. Although you can imagine many ways to do this, for us the level will simply be a function of the current score.
- **The current multiplier.** Many games (including ours) have a concept of a multiplier. One common implementation of this is that if you do an action very well a certain number of times in a row, then the multiplier increases. Perhaps in a music game you hit five consecutive notes exactly on time. This might bump your multiplier to 2x, and then any points you earn while the multiplier is at that level are doubled before being added to your score. Usually, any mistake at all resets the multiplier to 1x.

Some important points to consider are the following:

- Levels are achieved at the following scores. Note the exponential relationship between the level and the score range. Specifically, consider the value of 2^x where x is the current level. Multiply that by 10000, and you have an exclusive upper bound for the score range for level x. While it is possible to compute this in closed form, I suggest you use this opportunity to gain more experience with loops in Python. When the score is changed, modify the value of the level attribute inside of a loop until it corresponds to the current score.
  - 0-9999 = level 0
  - 10000-19999 = level 1
  - 20000-39999 = level 2
  - 40000-79999 = level 3
  - and so on ...
- The currentScore and currentLevel attributes are initially 0. The currentMultiplier attribute is initially 1. The livesRemaining attribute is initially 3.

- currentScore, currentLevel, and livesRemaining should never become negative. currentMultiplier should never fall below 1.
- currentScore can be increased or decreased by an amount that results in more than one level change.

In addition creating a Score class, you will also create a Scoreboard class that maintains a sorted array of the top ten instances of Score. This array should be sorted in decreasing order so that the highest score is in cell 0, the next highest score in cell 1, and so on. It is important to note that this assignment does not deal with any file input/output, so your leaderboard will only last as long as the program is running. If we were developing a real game using these classes, we would have to provide support for writing the Scoreboard values to and from disk so that they could be loaded and displayed each time the game is run. That is beyond the scope of this project.

The Scoreboard class is small. It contains a constructor that takes as a parameter the number of ranks you want to store. This number should be used to allocate spots in an array (such as 10 for the top ten scores). Aside from the constructor, the Scoreboard class has only two methods: one to print the Scoreboard to the display and another to update the Scoreboard with a new Score object. When you update the scoreboard with a new object, it should be placed in the array in the correct position to maintain the sorted order. Note that it is possible to bump lower scores off of the board when doing this, and that is okay.

A significant portion of your code will involve testing various cases in your main section. There are over 30 distinct test cases in my grading program for this assignment. Each tests something different about your program. Your goal is to break your program, so be aggressive with your tests. Consider how calling one method could influence the values returned or used by several other methods.

Write Python classes called Score and Scoreboard in a file called YourTeamName.py that contains the following code. **IT IS CRITICAL THAT YOUR CLASS AND FUNCTION NAMES BE EXACTLY THE SAME AS THOSE LISTED, INCLUDING SPELLING.** I have included a copy of the skeleton Score class for you to fill in to avoid copy-paste errors.

Submission Checklist

1. Include a document generated collectively by your team members that outlines how you envision this class being used. Give examples that show its usefulness. If you have suggestions on how to improve it, make and justify them. Most importantly, include a detailed explanation of why you chose to test using the values that you used. What do you consider when selecting values for your test cases and why? Also include in this document a statement affirming that your submission conforms to the W&M Honor Code guidelines, and that all of your work on this project was completed collaboratively as a team.
2. Include exactly one Python script that contains your class implementations as well as all of your testing code.
3. Create a folder called DeverickProject1 (using your team name instead). Put all files (one .py file and one writeup document) in this folder. Compress this folder as you did for Homework 0 and upload the resulting ZIP file to Blackboard as your submission no later than 0500 Saturday, September 26, 2015. (Treat that as Friday night, please. It is not my intention for you to stay up until 5AM.)

```
class Score:

  def __init__(self, player_name):
    # The required attributes
    self._player_name = player_name
    self._current_score = 0
    self._current_level = 0
    self._current_multiplier = 1
    self._lives_remaining = 3

  # The required methods
  def add_points(self, amount):
    # implement this method by adding the number of points
    # specified by amount times the currentMultiplier value
    # to the currentScore. If the new score value should
    # result in the level changing, then change currentLevel.
    # return the new value of currentScore.

  def subtract_points(self, amount):
    # reset currentMultipler to 1. subtract the number of
    # points specified by amount from currentScore, and update
    # currentLevel if necessary.
    # return the new value of currentScore.

  def get_player_name(self):
    # return the name of the player associated with this object.

  def get_multiplier(self):
    # return the current value of the multiplier attribute.

  def increment_multiplier(self):
    # increase the value of currentMultiplier by one.
    # return the new value of currentMultiplier.

  def get_score(self):
    # return the current value of the score attribute.

  def get_level(self):
    # return the current value of the level attribute.

  def get_lives(self):
    # return the number of lives remaining.

  def lose_life(self):
    # decrement the number of lives remaining. If, after you
    # have decremented the lives attributes, that attribute
    # has a positive value, return True, indicating play can
    # continue. If the number is zero, return false,
    # indicating that the game is over.

  def gain_life(self):
    # increase the current value of the lives attribute
    # by one.

  def __str__(self):
    return self._player_name + ' SCORE: ' + str(self._current_score) +\
        ' LVL: '+ str(self._current_level) +\
        ' MULT: ' + str(self._current_multiplier) +\
        ' LIVES: ' + str(self._lives_remaining)
```

```
class Scoreboard:

  def __init__(self, capacity):
    self._high_scores = [None] * capacity
    self._entries = 0

  def update(self, candidate_score):
    # if candidate_score has a score value higher than the
    # lowest score in the Scoreboard, add it at the correct position.

  def print_scoreboard(self):
    # take advantage of the fact that the Score object implements
    # the __str__() method, and can therefore be passed directly to
    # print(). Use this to print the current score board.
```