# CSCI 141 Computational Problem Solving
# Project 2: Keeping Score

For this project, you will create one class that represent the data type Score. Your score class will be used to create objects that track several aspects of a player's performance during an arcade-style game that you will implement in your final project. Here are the items that we will track in our Score class:

- **The player's name.** So that we can include the player's name when we display (print, for now) the score, we will store it as an attribute of the class. Its value should be provided as a parameter to the class constructor, and is expected to remain the same throughout the lifetime of the object.
- **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 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 `current_score` and `current_level` attributes are initially 0. The `current_multiplier` attribute is initially 1. The `lives_remaining` attribute is initially 3.
- `current_score`, `current_level`, and `lives_remaining` should never become negative. `current_multiplier` should never fall below 1.
- `current_score` can be increased or decreased by an amount that results in more than one level change.
- In the `__str__()` method, construct a string of the following format, **noting spacing, case, order of values, etc.** Strings that do not match this format will be considered incorrect by our test cases. This method will be invoked by Python when you try to print a Score object or when you explicitly ask for a conversion with `str(my_score_object)`.

  ```
  "Player: <player name>, Score: <current_score>, Level: <current_level>, Multiplier: <current_multiplier>, Lives: <lives_remaining>"
  ```

  For example,

  ```
  "Player: Jim Deverick, Score: 0, Level: 0, Multiplier: 1, Lives: 3"
  ```

  ```
  "Player: Michael Lewis, Score: 51829, Level: 3, Multiplier: 4, Lives: 6"
  ```

- None of the methods in your class should print anything to the screen, including the `__str__()` method.

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.

Because the name of the class and its methods **must be exactly as specified** in order for our testing to work, we have provided a skeleton file for you to fill in. Be certain not to change any names, and do not modify the numbers of parameters for each specified method.

## SUBMISSION EXPECTATIONS

**`Score.py`**: The skeleton file attached to this project, but with your implementations and test code added. The file must contain only the methods provided, and you must not change their names or parameters. Your test code must be conditioned on being in the main namespace as we discussed in lecture. The conditional that supports this is provided in the skeleton file.