CIS 41A – Lab 5: Inheritance, Regular expression

The goal of this lab is to learn when to use inheritance, get some practice with regular expression, and do some logic problem solving on your own. There are fewer detailed requirements so that you are free to use what we’ve discussed in class to solve the problem, but along with the freedom to use what we’ve covered is the responsibility to use them appropriately.

Given a Game class, which lets the user play a color memory game, write 2 other game classes: MultiGame and CustomGame.

Background: how the game is played

|  |  |
| --- | --- |
| The Game object appears as a GUI window, where the user starts the game by pressing the Escape key.  The vertical rectangle in the middle will change color, displaying a sequence of colors one after another.  When the display of the color sequence is done, the user enters in the horizontal textbox the names of the colors in the sequence. |  |
| This is a screen shot of the game after the user has pressed Escape to start the game, then the rectangle has changed to the colors blue and then red.  The user typed in the color sequence, with color names separated by a space. |  |
| Because the user entered the right colors in the correct sequence, the user earned 1 point, shown by the Score field.  Then a new color sequence is displayed again. This screenshot shows when red is displayed as part of the new sequence.  Each time a new sequence is displayed, it’s longer by one color. Eventually the user will not be able to enter all the correct colors in the correct order, then the game will be over. |  |

In *lab5\_game.py* you can find the Game class definition and the main block of code to create a Game object and run the game. You can run the code in *lab5\_game.py* to see how the game works.

You don’t need to know all the internal workings of the Game class, but 1 attribute and 1 method are important for your game classes:

* self.\_score: contains the user’s score for the game
* mainloop(): runs the game until the user doesn’t enter the correct color sequence

Additionally, you should look at the first part of the \_\_init\_\_() method (the part in between the 2 lines of ###) to see how the Game object can be created.

The MultiGame class

The MultiGame object lets the user play the game as many times as they want.

Create a MultiGame class that will:

* + - * Ask the user for a choice from the menu: 1. Beginner  
         2. Expert

Keep prompting until the choice is 1 or 2 (numeric value).

* + - * Have a play() method that lets the user play at the chosen level:

Print a welcome message and ask the user if they’re ready to play.

Accept one of the following as a ‘yes’ choice: y, yeah, yah, yup, yes with 1 or more ‘s’: yes, yesss …

Make sure words like ‘yesterday’ or ‘yahoo’ are not considered a ‘yes’ choice.

The user can enter the ‘yes’ word with other words on the line, such as: *yes, please* or *for sure, yeah.*

If the choice is not a ‘yes’ word, print a good-bye message and the game ends.

If the choice is a ‘yes’ word:

Print the level of the game.

Let the user play one game.

When the game is done, ask the user if they want to play another game, and if the answer is one of the ‘yes’ words, start another game.

When the user no longer wants to play another game, print:

the number of games played

the highest score

the average score (rounded to 1 digit after the decimal point)

a good-bye message and the game ends

The CustomGame class

The CustomGame object lets the user play one game and keeps track of the user play history.

Create a CustomGame class that will:

* Ask the user for their name.
  + Read in data from a file named *players.csv* (if it exists) and check to see if the user’s name is in this file.
  + If it is, print a “Welcome back” message, the user’s name, the user’s highest score, and the level of the game
  + If it’s not in the file or if the file doesn’t exist, print a “Welcome to the game” message.

The *players.csv* file has a list of user names, their highest scores, and the levels they played at.

* The file is created when the very first user ends the CustomGame app with a non-zero score.
* With each subsequent player that has a non-zero score:
  + If it’s an existing player in the file, the file is updated with the score and level if the score is the highest for that player
  + If it’s a new player, the file is updated with the name, score, and level of the game.
    - * + Ask the user to choose the game level.
        + Print a menu: b. beginner  
           e. expert

Re-prompt if the user doesn’t enter ‘b’ or ‘e’, case insensitive  
Example of acceptable input: “*I want B level*”, “ *e* ”, “*of course it’s E E E all the way*”   
Example of not acceptable input: “*begin*”, “*either b or e*”

* + - * + Have a play() method that lets the user play one game at the chosen level.

When the game ends:

Print an appropriate message:

If the user’s score is 0, print a “good try” message.

If the user’s score is the highest score of all the players, print a “you’re the top player” message.

If the user’s score is the same as the highest score of all the players, print a “you’re in the top N of all players” message, where N is the total number of players with the highest score.

If the user’s score is lower than the highest score, print a “good game” message.

Save to file the user’s name, score, and level if the score is non-zero and is the user’s highest score.

To consider while writing the MultiGame and CustomGame classes:

* Determine if any of the 2 classes should use inheritance.
* If a class uses inheritance, create a new Python module for the class definition.
* If a class doesn’t use inheritance, create a lab5.py file and put the class definition in the file.

At the end of lab5.py, add these 2 lines to play the game:

MultiGame().play()

CustomGame().play()

* Determine when to use local variables and when to use instance attributes. If a variable needs to save data for the lifetime of the object, it’s an instance attribute. Otherwise, a local variable is more memory efficient.
* Create methods as needed to organize the code into logical blocks.
* Regular expressions can be your friend when you validate (check for valid) user input, but string methods can be useful, too.
* Handle exceptions as needed.
* Documentation: as usual, put your name, lab number, module name at the top of the file, and docstrings for public methods.

Test sample for MultiGame, user input in blue:

1. beginner

2. expert

Enter a level: 1

Welcome to the color memory game

Ready to play? yeah why not

<game played>

Play again? yes!

<game played>

Play again? y

<game played>

Play again? no thanks

You've played 3 games

Highest score 3

Average score 2.0

Test sample for CustomGame, user input in blue.

User ab plays as the first player of the game:

Enter your name: ab

Welcome to the game, ab

b. beginner

e. expert

Enter your choice: b level

Playing at beginner level

<game played>

Your score: 2

Congratulations! You're the current top player # this means only one player has top score

User ab plays again, ending with a lower score:

Enter your name: ab

Welcome back, ab

Your highest score so far is 2

The game was at the beginner level

b. beginner

e. expert

Enter your choice: to b or not to b

Playing at beginner level

<game played>

Your score: 1

Good game

User ab plays again, ending with the same highest score:

Enter your name: ab

Welcome back, ab

Your highest score so far is 2

The game was at the beginner level

b. beginner

e. expert

Enter your choice: bbbbbbbbb

Choose either 'b' or 'e'

b. beginner

e. expert

Enter your choice: b

Playing at beginner level

<game played>

Your score: 2

Congratulations! You are in the top 1 of players # this means sharing the same top score with 1 player

User xy plays as new player, ending with the same highest score:

Enter your name: xy

Welcome to the game, xy

b. beginner

e. expert

Enter your choice: b

<game played>

Your score: 2

Congratulations! You are in the top 2 of players # this means sharing the same top score with 2 players

User xy plays again with a new highest score:

Enter your name: xy

Welcome back, xy

Your highest score so far is 2

Your last game was at the beginner level

b. beginner

e. expert

Enter your choice: b please

<game played>

Your score: 3

Congratulations! You're the current top player # this means only one player has top score

User ab plays again with a 0 score

Enter your name: ab

Welcome back, ab

Your highest score so far is 2

Your last game was at the beginner level

b. beginner

e. expert

Enter your choice: never!

Choose either 'b' or 'e'

b. beginner

e. expert

Enter your choice: b, e

Choose either 'b' or 'e'

b. beginner

e. expert

Enter your choice: ok b

Playing at beginner level

<game played>

Your score: 0

Good try

After the above 6 games were played, the file has ‘ab’ with score of 2 at beginner level, and ‘xy’ with score of 3 at beginner level.