CSCA20 Assignment 1

Introduction

The purpose of this assignment is to give you practice in a number of areas:

- Working with type `list`
- Working with for and while loops
- Writing and calling functions
- Writing programs

This assignment is made up of two parts. You should read the entire assignment handout before starting the assignment. Start early; this assignment is considerably more work than any of the exercises.

A description of the game of Hangman is below. I encourage you to try playing the game at the following URL several times until you feel familiar with it and comfortable: https://hangmanwordgame.com/?fca=1&success=0#/

Hangman rules

(Note that there are many versions of hangman. The version described here is the version we'll use for this assignment, and it is slightly different than the version in the aforementioned link)

Hangman is a two-player game. One player comes up with a word (which I'll refer to as the word in question), which is hidden from the other player. Your program will play the part of this player.

At the start of the game, the second player (ie. the user of your program) sees the word in question, but with every letter replaced by an underscore. The overall objective of the second player is to eventually guess every letter of the word in question by guessing a letter at a time. Obviously, the second player is only allowed to guess a letter they haven't already guessed before.

If the guess is of a letter which appears in the word in question, then all occurrences of that letter in the word in question are revealed to the second player by replacing the corresponding underscores with the letter.

If the guess is of a letter which doesn't appear in the word in question, then the second player has made an incorrect guess. Before the start of the game, the two players will have agreed on a maximum number of incorrect guesses. If the second player makes *more than* this number of incorrect guesses, then they lose the game.

If the second player manages to successfully guess every letter in the word in question without losing the game, they win. The second player repeatedly guesses letters until they either lose or win.

In our version of hangman, the word in question and the maximum number of incorrect guesses will be set as constants in the file `settings.py`. In particular, the word in question

will *not* be randomly chosen. Feel free to modify these settings for your own testing and/or enjoyment; your modifications will not affect our testing because we'll be testing with our own version(s) of the `settings.py` file; not yours.

Hangman game functions

In this part you will write the functions that do most of the work in your game of Hangman. You will write these functions in a file called `hangman.py`. Make sure you fully understand the rules of Hangman as described above.

For each function name and description below, fill in the docstring and body for the Python function of the same name in the starter file `hangman.py`. Notice that now you are responsible for writing the docstring. For all assignments, you will be graded on the quality of your docstrings.

In all of the following function descriptions, you can assume the following about the given function arguments:

- `word` is a str representing the word in question
- `past_guesses` is a list of single-character lowercase strs representing the previous guesses the user has already entered
- `guess` stands for a single character str which was most recently guessed by the user

Here are descriptions of what each function should do:

Function header	**What the function should do**

`obfuscated_word(`
`word,`
`past_guesses)`
	Return a str identical to `word` but where the characters which have been guessed are unmodified and the non-guessed characters have been replaced with underscores.
	
	For example, `obfuscated_word('headphones', ['e', 'd'])` should return the str `'_e_d____e_'`

`get_guess(` `past_guesses)`	Get a letter from the user as input, displaying the prompt: `'Guess a letter: '` If the input isn't exactly one alphabetic character, or if the character has already been guessed before (ie. it appears in list past_guesses), this function will warn the user and repeatedly query the user using the previous prompt until they enter a valid input. The warning will consist of the text: `'You need to enter one alphabetic character which you haven\'t already guessed. Try again'` If the user provides an upper-case alphabetic character, this is a valid guess. This function will convert it to lower-case without notifying the user.

	Then, the function will append the guess to `past_guesses` and also return the guess.
`format_guesses(past_guesses)`	The purpose of this function is to format `past_guesses` into a str so that it can be printed in a compact fashion. The function will return a str that consists of each of the characters in `past_guesses` in the order they appear, where each character is separated by a space. For example, `format_guesses(['a', 'b', 'c'])` should return the str `'a b c'`
`is_guess_good(word, guess)`	If the `guess` is in the `word`, returns `True`, else `False`. For example, `is_guess_good('toe', 'e')` should return `True` while `is_guess_good('toe', 'a')` should return `False`.
`is_word_guessed(word, past_guesses)`	This function will return a bool denoting whether the `word` has been fully guessed (ie. whether every character in `word` appears in `past_guesses`) For example, `is_word_guessed('toe', ['e', 't', 'a', 'o',])` should return `True` while `is_word_guessed('toe', ['e', 't', 'a'])` should return `False`.

The Game

In this section you will write a main program that calls your functions to create the game. ALL code for this section MUST be in the body of the `main()` function of the `game.py` starter file. In this section, your code will consist primarily of a while loop, interactions with the user and calls to functions from `hangman.py`.

Take extra care to follow these instructions precisely. If you don't, the auto-marker may trip on your code and you'll get a far lower grade than you expect.

Where possible, call the functions you wrote in `hangman.py`. You will be graded on how appropriately you call your functions. Also, make sure that you use the settings from `settings.py` as appropriate.

The program order is as follows:

1. Display the following text to the user:
   `'Word: W'` where W is the word in question with all un-guessed letters replaced by underscores

2. Display the following text to the user:
   `'Number of bad guesses: N out of M'` where N is the number of incorrect guesses the user has made so far, and M is the maximum number of incorrect guesses the user is allowed to make
3. Display the following text to the user:
   `'Guesses: G'` where G is the letters guessed by the user so far and each guess is separated by a space. For example, if the user has guessed a, b, and c, then display the text `'Guesses: a b c'`. Display these letters in the order they were guessed by the user.
4. Ask the user for a guess of a letter with the prompt:
   `'Guess a letter: '`
5. If the user enters an invalid guess (ie. a valid guess is a single alphabetic character which they haven't already guessed), print a warning to the user and repeat the previous step. Invalid guesses *don't* count towards the total number of incorrect guesses. The warning should be the following text (where the backslash is meant to escape the quote; it's *not* meant to be displayed to the user):
   `'You need to enter one alphabetic character which you haven\'t already guessed. Try again'`
6. If the user enters an uppercase character, convert the character to lowercase without chastising the user. The word in question and all guesses (after conversion) should all be in lowercase.
7. If the user successfully guesses a letter in the word, display the following text:
   `'L is in the word'`, where L is the letter being guessed
8. If the user makes an incorrect guess, display the following text to the user:
   `'L is not in the word'`, where L is the letter being guessed
9. Repeat all of the previous steps until the user has either won or lost the game
10. If the user has won, display the following text to the user:
    `'Congratulations! It took you N incorrect guesses to guess the word'`, where N was the number of incorrect guesses they made
11. If the user has lost, display the following text to the user:
    `'Sorry, you failed to guess the word after M incorrect guesses'`, where M is the maximum number of incorrect guesses they were allowed
12. Finally, regardless of whether the user has won or lost, display the text:
    `'The word was W'`, where W is the word in question.

Testing

As usual, you're strongly encouraged (but not required) to test your own code. You may use any method we've shown you to test your functions, eg. print statements in the `if __name__ == '__main__':` part at the bottom or doctests. Feel free to consult the exercise handouts and the week 7 lab handout for guidelines on how to test.

Grading

Here are some general outlines on how we'll be grading you:

- Correctness: your functions should perform as specified. Correctness, as measured by our tests, will count for the largest single portion of your marks.
- Style: the style requirements for assignments are stricter than those for exercises. Follow the guidelines in this document: https://docs.google.com/document/d/1QNoQ1Yt7QLk4vaKrQ8W5G0aDVw4ql6SlQhoQ394fW44/edit?usp=sharing
  Note that style will be worth slightly less on A1 than A2 or A3. This is to allow you to make mistakes now without being penalized too heavily, while allowing you to learn from your mistakes in time for later assignments.
- Commenting: for each function, you should write a docstring comment that describes its arguments, what the function does, and what is returned by the function as well as appropriate examples. Follow the docstring format from the Google Style guide (see the link in the style point)
- Programming style: Your variable names should be meaningful and your code as simple and clear as possible. You should avoid duplicate code by calling other functions as helpers. Your functions should be written so that they would work properly even if different values are used for the constants.

Deadline(s)

There will be two pre-grading runs by the automarker:

1. One on all code submitted by Wednesday October 24, 11:59pm.
2. One on all code submitted by Friday October 26, 11:59pm.

The automarker will only check for correctness and adherence to the PEP8 style guide. You will only receive grades for the adherence to the rest of the style requirements after the final deadline because we will grade those manually.

The final deadline for this assignment will be **Sunday October 28 at 11:59pm**.

What to submit

Submit your fully implemented `hangman.py` and `game.py` to MarkUs under Assignment 1. As usual, make sure to submit *exactly* these files; don't modify their names! We will only grade files with the *exact* correct filenames.

Sanity check

As usual, you are strongly suggested to make sure your program runs *without any errors* (eg. syntax errors) before you submit..