4CS001 - Coding Challenge 3

Functions and File Handling

Due: Sunday 29th November 2020 at 14:00

This assignment is worth 20% of the overall module grade

Introduction

This coding challenge will assess your knowledge of Python functions, file handling operations and exceptions. It also builds on many of the topics we have covered over the course of the module.

The marking scheme for this task is on Canvas, make sure you check back on a regular basis as you work through the assessment. **There is an additional challenge segment that can earn you extra credit, which you will need to complete to achieve the highest possible mark. Students who do not attempt this cannot achieve grades above 80% for this task.**

Task Overview

Your task is to implement a variation of the classic word game Hangman, which involves players guessing the letters in a word chosen at random with a finite number of guesses. While there are alternate versions such as category Hangman and Wheel of Fortune, which involve players guessing idioms, places, names and so on, we will be sticking with the traditional version.

If you are unfamiliar with the rules of the game, please read the following before starting: http://en.wikipedia.org/wiki/Hangman_(game). You can also play an online version here. Don't be intimidated - You'll be given some skeleton code to get you started, it's easier than it looks!

Getting Started:

Start by downloading the files `hangman.py` and `words.txt` from Canvas, **saving them in the same folder**. Start by looking through the main Python file and reading the included documentation. Add your name and student number to the top of the file.

Requirements:

You will implement a function called `main` that allows users to play an interactive hangman game against the computer. The computer should pick a word, and players should then try to guess letters in the word until they win or run out of guesses.

Here is the overarching behaviour we expect:

1. The program should load a list of available words from the text file provided. **Note that the file you have been given contains words in lowercase.**
2. The computer should then select a word at random from the list at random.
3. The user is given a certain number of guesses at the beginning.
4. The game is interactive; the user inputs their guess and the computer either:
    a. reveals the letter if it exists in the secret word
    b. penalizes the user and updates the number of guesses remaining
5. The game ends when the user guesses the word, or runs out of available guesses.

You are going to make use of a common approach to computational problem solving, which involves breaking the problem down into several logical subtasks to make things more manageable. The file `hangman.py` contains several existing helper variables and functions which you should use to help you complete the task.

Step 1 – Loading the Words File:

Implementation Details:

You should start by implementing the `load_words` function, this should return a list of valid words (strings) from the text file you have been given. Implementation guidelines:

1. Print a message to let the user know that the word list is being loaded, this can sometimes take some time depending upon the size of the file and the speed of the CPU.
2. Open `words.txt` and read the contents into a variable. The words in the file are delimited (separated) by a single space.
3. Ensure that any exceptions raised when attempting to open the file are handled such as the file not existing or having the wrong permissions. If an exception is encountered, print a message to the user identifying the problem and end the program.
4. Print the total number of words loaded. There should be 55900 in `words.txt`.
5. Return the wordlist from the function.

You should test that your implementation works before moving on. Make sure that the correct number of words are loaded and that no exceptions are raised.

Hints:

- You can use the `split` function to separate a string into a list of strings.
- Make sure you close the text file once you've loaded the word list.

Example Usage:

```
>>> load_words()
Loading word list from file: words.txt
55900 words loaded.
```

Step 2 – Identifying Unused Letters:

Implementation Details:

Your next task is to implement the `get_remaining_letters` function. This will be used to generate a string comprised of letters that have not yet been guessed. The function takes a single argument `letters_guessed`, a list of letters (strings) that the user has previously guessed. You should implement behaviour to compare these letters against the full alphabet to determine the letters that remain and return them as a string.

Hint: You may find the `string.ascii_lowercase` variable useful, which stores a list of alphabetical letters in lowercase. The string library has been imported for you.

```
>>> from string import ascii_lowercase
>>> print(ascii_lowercase)
abcdefghijklmnopqrstuvwxyz # All lowercase letters
```

Example Usage:

```
>>> letters_guessed = ['a', 'e', 'c']
>>> print(get_remaining_letters(letters_guessed))
bdfghijklmnopqrstuvwxyz # Letters minus letters_guessed
```

Step 3 – Testing the Win Condition:

Implementation Details:

Your next task is to implement the `is_word_guessed` function. This should check whether the user has guessed all the letters in the secret word chosen by the computer and to determine whether the game has been won. The function takes 2 arguments:

a) `word` - the word that the player is attempting to guess.
b) `letters_guessed` - the letters that the player has previously guessed.

You will need to implement the logic to determine whether the letters in `word` are present in `letters_guessed`. Remember that when the player guesses a letter, all instances of that letter in the secret word are revealed to the player. The function should return a Boolean value (True/False) based on whether the letters in `word` are present in `letters_guessed`.

Hint: Iteration and use of sets are both viable options for this step.

Example Usage:

```
>>> letters_guessed = ['q', 'v', 'd', 'e', 'n', 'u']
>>> print(is_word_guessed('queen', letters_guessed))
True # All of the letters are in letters_guessed

>>> letters_guessed = ['q', 'v', 'd']
>>> print(is_word_guessed('queen', letters_guessed))
False # Not all of the letters are in letters_guessed
```

Step 4 – Tracking Players Progress:

Implementation Details:

Your next task is to implement the `get_guessed_word` function. This should return a string, comprised of letters, underscores `(_)`, and spaces that represents players progress.

The function takes two arguments:

a) `word` - the word that the player is attempting to guess.
b) `letters_guessed` - the letters that the player has previously guessed.

You will need to write additional code to generate and return a string containing the letters that have previously been guessed and placeholders for those that haven't. In this case you should use an underscore followed by a space `(_ )`. Other symbols could also be used, but this is easily recognisable. You may be wondering why the space is required. The reason is that if several consecutive letters are hidden, it would not be possible to discern between them, imagine seeing this: `M______`, it would be difficult to determine the number of remaining letters. However, with a space added, it is much easier to tell what is going on: `M_ tt_ e_`.

Example Usage:

```
>>> letters_guessed = ['q', 'v', 'd']
>>> print(get_guessed_word('queen', letters_guessed))
q_ _ _ _ # Only q is revealed as the other letters are incorrect

>>> letters_guessed = ['q', 'e', 'd']
>>> print(get_guessed_word('queen', letters_guessed))
q_ ee_ # When e is guessed, both occurrences are revealed
```

Step 5 - The Main Game Logic:

Implementation Details:

Now it's time to tackle the `main` function, the part of the program responsible for starting and running your interactive hangman game. The function takes a single argument: `word`, which is used to store the secret word that the player is trying to guess. You will need to use the functions you implemented in the previous steps to tie the application together.

Begin by printing a welcome message to the player followed by the number of letters in the secret word, you should consider defining a `welcome_message` function to do this.

You should use a loop to drive the main game logic. The game should consist of several 'rounds' in which the player guesses a new letter in the secret word. The game should end when the player guesses the word or uses all their available guesses.

In each round, you should:

1. Check if the player has guessed all the letters in the secret word. If so, print a message congratulating the user, reveal their score (see below) and end the game.
2. Check the players number of remaining guesses. You should either:
   a. Print a message informing the player they have run out of guesses. Print the secret word and end the game.

   - **b.** Proceed to the next step if the player has 1 or more guesses remaining.
3. Print the number of guesses the player has remaining, traditionally players start with 6 guesses. However, you may wish to change this.
4. Print the remaining letters that the player has not yet guessed.
5. Prompt the user to enter a new guess. Uppercase and lowercase letters should be accepted.
6. Check if the guess is a valid alphabetic letter. You should either:
   - **a.** Print a message informing the player their guess was invalid. Decrement the players remaining guesses by 1 and skip to the next round.
   - **b.** Proceed to the next step if the guess is valid.
7. Check if the guess is the same as a previous one. You should either:
   - **a.** Print a message informing the user they have repeated a previous guess and skip to the next round. Do not penalise the player for repeated guesses.
   - **b.** Proceed to the next step if the guess is not the same as a previous one.
8. Give the user feedback on their guess. You should either:
   - **a.** Print a message congratulating the player on a successful guess and display the partially guessed word with the appropriate letters revealed.
   - **b.** Print a message informing the player of an incorrect guess and display their current progress. If the guess was a vowel decrement the players remaining guesses by 2 otherwise decrement the players remaining guesses by 1.
9. Print a separator such as '------------' or an alternative to indicate the end of a round.

Scoring:

Users should receive a score if they win the game. The score should be calculated by multiplying the players remaining guesses by the number of unique letters in the secret word.

Example Implementation (user input in red):

```
Loading word list from file...
55900 words loaded.
Welcome to Hangman Ultimate Edition
I am thinking of a word that is 3 letters long
-------------
You have 6 guesses left.
Available letters: abcdefghijklmnopqrstuvwxyz
Please guess a letter: a
Good guess: a_ _
------------
```

```
You have 6 guesses left.
Available letters: bcdefghijklmnopqrstuvwxyz
Please guess a letter: e
Good guess: a_ e
------------
You have 6 guesses left.
Available letters: bcdfghijklmnopqrstuvwxyz
Please guess a letter: e
Oops! You've already guessed that letter: a_ e
------------
You have 6 guesses left.
Available letters: bcdfghijklmnopqrstuvwxyz
Please guess a letter: r
Oops! That letter is not in my word: a_ e
------------
You have 5 guesses left.
Available letters: bcdfghijklmnopqstuvwxyz
Please guess a letter: o
Oops! That letter is not in my word: a_ e
------------
You have 3 guesses left.
Available letters: bcdfghijklmnpqstuvwxyz
Please guess a letter: g
Good guess: age
------------
Congratulations, you won!
Your total score for this game is: 9
```

Example Loss:

```
------------
Sorry, you ran out of guesses. The word was: age
```

Structure and Documentation

The structure of your code and documentation will be analysed and assessed.

This will be done using a static analysis tool called **Pylint**. This software checks the code in your program, ensures that it follows Python conventions and that all functions, classes and modules have been documented. You can read more about it here: https://www.pylint.org/.

Python has an official style guide named PEP8, which is where most Python conventions/coding standards originate from. Example checks that Pylint carries out to ensure that the PEP8 coding standard is followed include things such as:

- checking line-code's length
- checking if variable names are well-formed (snake case)
- checking if imported modules/functions are used
- checking if variables/function parameters are used

It is a good idea to run these checks on your code at regular intervals and before submitting. There are several free websites that allow you to do this e.g. https://pythonbuddy.com/.

Note: Marks will be deducted for warning and errors detected in your code.

Challenge (Additional Credit):

Currently, there isn't much incentive for players to keep coming back to the game. There is no real progression system as players scores aren't stored on a persistent basis.

Implementation Details:

For this challenge, you will extend the existing program so that players scores are saved to and read from a text file e.g. *scores.txt*. You should ask for the player's name at the start of each new game and use it to load their previous high score. If they manage to beat their personal best while playing, ask them if they would like to update the leaderboard and do so if prompted.

To do this, you should create two new functions:

- A function to retrieve the existing high score for the current player called: `get_score`. This should take a single argument, the players name and return their high score. If they don't have a high score return 0 or None. Players without a previous high score should always be asked whether they would like to store their score.

- A function to save a new high score for the current player called: `save_score`. This should take two arguments (the players name and score). The function does not need to have a return value, but should handle writing the latest score to the text file. If a player already has a lower score saved in the file, update it instead.

Your game will also need a more robust menu, it should contain options to play the game, load the leaderboard or quit. Players should return here after playing or viewing the leaderboard. Selecting the option to quit should close the program after printing a goodbye message. The logic for this should be contained in your main function, update it as required.

Hints:

- When the player loads the leaderboard, you will need to read the text file used to store the data and present the information in a tabular format so that it's easy to read.

Example Implementation (input in red):

```
Loading word list from file...
55900 words loaded.
Welcome to Hangman Ultimate Edition
Do you want to Play (p) view the leaderboard (l) or quit (q): p

Score          Name
-----------------------------
7              Matthew Howell
5              Herbert Daly
3              Hiran Patel

Would you like to play (p) or view the leaderboard (l): p
What is your name: Matthew Howell
I am thinking of a word that is 2 letters long
-------------
You have 6 guesses left.
Available letters: abcdefghijklmnopqrstuvwxyz
Please guess a letter: a
Good guess: a_
------------
You have 6 guesses left.
Available letters: bcdefghijklmnopqrstuvwxyz
Please guess a letter: n
Good guess: an
------------
Congratulations, you won!
Your total score for this game is: 9
A new personal best! Would you like to save your score(y/n): y
Ok, your score has been saved.
Do you want to Play (p) view the leaderboard (l) or quit (q): q
Thanks for playing, goodbye!
```

Submission and Marking

You should upload your submission to Canvas. If you fail to do so, you will receive a grade of 0 NS (Non-Submission). The deadline for doing so is the 29th November 2020 at 14:00.

Your work will be automatically marked using a program that tests each of the individual functions you have implemented. Therefore, it is very important that you do not alter any of the function signatures in the template and implement everything as instructed.

You will receive an individualised test report, showing which of the tests passed and failed. Spot checks will be carried out to ensure that the marking is both fair and consistent. However, if you feel that you have been unfairly penalised, please do not hesitate to get in touch.