Fall 2013 CSCI2961: Programming in Python Homework 2

General Instructions

This homework assignment is to be completed individually. Do not share code or review anyone else's code. Work on this assignment is to be your own.

Submit your homework via RPILMS before midnight on the due date. Put all of your code into exactly one Python file and name it with the homework number followed by an underscore, followed by your RCS userid. For example, if your RCS userid is mehtaa2, then your Python file name for this homework assignment must be hw2_mehtaa2.py.

Be sure to comment your code and include your name at the top of each file submitted.

Hangman

Write a python program to play hangman. If you are not familiar with this game, details can be found at http://en.wikipedia.org/wiki/Hangman_(game).

Sample output

```
+---------
|      |       Word/Phrase: han  an
|      O                       ^^
|     \|/
|     /        Misses: eiost
|
+=========

Guess? r

+---------
|      |       Word/Phrase: han  an
|      O                       ^^
|     \|/
|     / \      Misses: eiorst
|
+=========

Guesser loses - the answer was:
hangman
```

Details

You will work with a word or phrase to guess, and a dictionary of valid words or phrases.

The game should be case insensitive. During game play, only the ascii letters are to be guessed (the characters that are not ascii letters should just be printed). If the user incorrectly guesses six letters, they lose the game.

The functions below may specify positional or keyword arguments. Unless an argument order is explicitly specified or an argument is listed as a positional argument, you can assume that it will be invoked by keyword during the evaluation of your assignment. Because of this, it is important that your your argument names match the specifications. Note that there may be multiple correct ways to define your function.

Function get_dictionary

Write a function, `get_dictionary`. The function has three arguments.

• `source`	Can be specified by caller as the first positional argument or keyword argument.
• `type_`	Keyword only argument, required.
• `single_word`	Keyword only argument, optional (default True).

It creates and returns a dictionary that represents the valid entries. Note: the use of the term dictionary here means the universe of valid entries that the game will be played against, not the Python `dict` data type. The return value can be, but does not have to be, an object of type `dict`, or it can be a `list`, `set`, or any other type of object you choose.

If `single_word` is True, then the dictionary is to contain single words (split the source on whitespace as necessary, and then remove any non-ascii letters from the beginning or end). If `single_word` is False, then each entry may contain a word or a phrase, and you should split the source on newlines to find the entries.

Valid values for `type_` are:

• file	Source is the name of a file that contains text to be used for the dictionary.
• url	Source is a URL that is to be retrieved/processed which contains text to be used.
• text	Source is a String that contains the text to be used.

Function display

Write a function, `display`, with any parameters you need, to display the current state of the game. Your display should convey the following information:

- The word/phrase being solved. You should show all non-ascii letters, and all correctly guessed characters. You should hide the ascii letters that still need to be solved by the guesser.
- The set of characters that were guessed, but are not in the word / phrase.

Function play_against_human_guesser

Write a function, `play_against_human_guesser`. This must take one of two arguments, specified by keyword only: `solution` or `dictionary`. If the caller provides the `solution`, your function presents the solution, masking the ascii letters, and interactively prompts the user until they have solved it (or have taken too many incorrect guesses). If the caller provides a dictionary (an object returned by the `get_dictionary` function), then the function begins by picking an entry from the dictionary randomly as the solution. When the game is over (solution is complete, or a sixth letter was guessed incorrectly), the function returns a tuple of two values. The first is the solution (useful if the one was chosen randomly from a dictionary); the second is the list of letters guessed, in the order in which the guesses were made (note: this second parameter includes correct as well as incorrect guesses).

Function play_against_computer_guesser

Write a function, `play_against_computer_guesser`, that takes the same two arguments as play_against_human_guesser and has the same return value. Additionally, it takes a third optional argument, `guesses`, that is a sequence of letters to be used to guess the solution.

For this function, a `dictionary` must be specified, and it must NOT be directly modified by your function. Make a copy of it, so that the original dictionary is not modified. The `solution` argument is optional (if it is not provided, a solution is chosen randomly from the dictionary).

Instead of prompting the user for letters, however, choose the letters from the sequence of `guesses`, in order, until the game ends, or until the solution is obtained. Note that the intent for this optional parameter is to have a way to validate the basic mechanics and structure of your solution. If this optional parameter it is not specified, then your function should calculate the best letter to use as the next guess, as described below.

The typical way that this function is meant to be invoked is to NOT specify the sequence of `guesses`. When the caller omits this parameter and does not supply the `guesses`, your program should pick the best guess at each step.

To do this, the program must keep track of the subset of entries from the dictionary that are still viable. At each iteration, from the list of remaining / viable solutions, you should choose as your guess the letter in the alphabet which

1. Has NOT yet been already guessed.
2. Is present in the largest number of solutions that are still viable. You should count a letter once if it appears at all in a potential solution. If a letter appears multiple times in a solution, it still only counts once. For example, if the viable solutions contain "teepee", and "turnip", "t" and "p" are better guesses than "e".
3. Breaking ties: If you have multiple letters with the same frequency, choose the first letter of the alphabet. In the above example, "p" would be chosen before "t".

Additional functions

You can add other functions as needed to organize your solution, but make sure you include the functions above, to facilitate grading.

The program should be submitted as a module that can be imported (when imported as a module, only the functions should be defined; there should be no output). You can optionally have the module automatically start up in a certain way when it is invoked as the main module.

Make sure that your functions are “well-behaved” to avoid side effects that would prevent them from being invoked repeatedly (the TA should be able to run your program with multiple test cases without having to exit or reload your module).