420-LCU-05 Programming in Python - Assignment 2

October 2, 2017

Here is a reminder of the general requirements for these assignments:

1. **Identification section** Do this for *every* Python file in every assignment in this course. This section must be either in a comment, with a '#' preceding each line, or enclosed within triple quotes ("'). The grader and I need this section for the *accurate processing of your assignment*. Assignments missing this may lose up to 5% of the total mark.

   Please note that you should also include your ID number after your name!

   Example:

   ```
   """
   Justin Trudeau, 1122334
   420-LCU Computer Programming, Section 2
   Sunday, February 31
   R. Vincent, instructor
   Assignment 1
   """
   ```

2. Your submission for this assignment will be a single Python file. There is no need to create a ZIP archive for this assignment!

3. Be sure to respect other instructions specified in the assignment. Part of each assignment is to correctly follow the instructions as closely as possible.

Introduction[1]

This assignment is not as hard or long as it may seem, so do not despair. The description will be much longer than your code. We have provided you with some of the basic parts of the program, and you have to finish some incomplete functions.

You will create a variation of the classic word game "Hangman". For those of you who are unfamiliar with the rules, they are explained at `http://en.wikipedia.org/wiki/Hangman%20%28game%29`. In the version we will implement, the computer always picks a word at random, and the human player will try to guess the word.

You will implement a function, named `hangman`, that will carry out an interactive Hangman game between a player and the computer. Before we get to this function, we'll first implement a few helper functions you will need.

For this problem, you will need the code file `a2_hangman.py` and the dictionary of words `words.txt`. Be sure to save the files in the same directory. Open and run the file `a2_hangman.py` without making any modifications to it, in order to ensure that everything is set up correctly. In other words, you should do the following:

Load IDLE. From the File menu, choose "Open". Find the file `a2_hangman.py` and choose it. The template `a2_hangman.py` should now be open. Run the file. The code we have given you loads in a list of words from a file. If everything is working okay, after a small delay, you should see the following messages:

[1]This assignment is adapted from one used in the MITx 6.00.1x course.

```
Loading word list from file...
  55909 words loaded.
```

If you see an error instead (such as "No such file or directory"), you may need to change the value of the `WORDLIST_FILENAME` constant (defined near the top of the file) to the complete pathname for the file `words.txt` (This will vary based on where you saved the file).

The file `a2_hangman.py` already contains three functions functions you will use in your solution. You can ignore the details of the code between the following comments, though you should read the docstrings so that you understand how to use each function.

```
# -----------------------------------
# Helper code
# You don't need to understand this helper code,
# but you will have to know how to use the functions
# (so be sure to read the docstrings!)
    .
    .
    .
# (end of helper code)
# -----------------------------------
```

You must do all of your coding for this problem entirely within this file, because you will be writing a program that depends on each function you write.

Requirements

Here are the requirements for the game:

1. The computer will select a word at random from a list of available words that was provided in `words.txt`. The functions for loading the word list and selecting a random word have already been provided for you in `a2_hangman.py`.

2. The game must be interactive; the flow of the game should go as follows:

   - At the start of the game, let the player know how many letters the computer's word contains.
   - Ask the player to supply one guess (i.e. letter) per turn.
   - The player should receive feedback immediately after each guess about whether their guess appears in the computer's word.
   - After each turn, you should also print the partially guessed word as well as the list of letters that the player has not yet guessed.

3. We have also provided two files, `sample-success.txt` and `sample-failure.txt`, that show how the output of your program should appear for both a player victory and a player loss.

4. Some additional rules of the game:

   - A player is allowed 8 guesses. Make sure to remind the player of how many guesses she has left after each turn. Assume that players will only ever submit one character at a time (A-Z).

- A player loses a guess only when she guesses incorrectly.
- If the player guesses the same letter twice, do not take away a guess - instead, print a message letting them know they've already guessed that letter and ask them to try again.
- The game should end when the player constructs the full word or runs out of guesses. If the player runs out of guesses (she "loses"), reveal the word to the player when the game ends.

Exercise 1

(4 points) In this part, we will complete the first function that we need for our game. We have already decomposed the problem into smaller components, and the first component we need is a function `isWordGuessed`. This is the function that answers the question, "Has the player won yet?".

In Hangman, the player has won if all of the words in the `secretWord` have been guessed.

Considering this, complete the function `isWordGuessed`. You will almost certainly want to use a `for` loop to do this. The logic of the algorithm would be expressed like this in "pseudocode":

```
def isWordGuessed(secretWord, lettersGuessed):
    For each letter in the secretWord:
        If the letter is not in lettersGuessed:
            return False
    Return True
```

Exercise 2

(4 points) In this part, we will complete the other function we need for the game. This function, `getAvailableLetters`, returns a string we can print out. This string will contain all of the letters the player has not yet guessed. So the logic of this function is something like this:

```
def getAvailableLetters(lettersGuessed):
    Set result to an empty string.
    For each letter in the alphabet:
        If the letter has not been guessed:
            Add letter to result
    Return result
```

Exercise 3

(12 points) Now we will use the existing functions to implement our Hangman program. To do this, we'll write a single function `hangman(secretWord)` that takes a single argument, a string containing the word to be guessed, all in lower case.

There are three pieces of information you will need to store. Suggestions for the types of those values in parentheses:

- `secretWord` - a parameter, the word to guess (`str`).
- `lettersGuessed` - the list of letters guessed so far (`list` or `str`).

- `remainingGuesses` - the number of incorrect guesses left (`int`).

The logic here is of course a bit more complicated than the two previous functions. Here is a "pseudocode" description of what this function will need to do:

```
def hangman(secretWord):
    Initialize lettersGuessed and remainingGuesses
    Print a welcome message
    while the game is not over:
        if the player has won:
            Congratulate the player
            Exit the function
        else if the player has run out of guesses:
            Print a message telling the player has lost
            Exit the function
        else:
            Print the number of remaining guesses
            Print the available letters
            Get a new letter from the player
            if the letter has already been guessed:
                Tell the player they already guessed that letter
            else:
                Add the new letter to lettersGuessed
                if the new letter is in the secretWord:
                    Tell the player they guessed correctly
                else:
                    Tell the player they guessed wrong
                    Take away one from remainingGuesses
```

For this part you will implement this function in Python.

When you have completed all three parts of the assignment, you should test your program several times, and verify that it gives results similar to those shown in the sample output files `sample-success.txt` and `sample-failure.txt`.

Once you have tested your code and verified your identification section, you can then submit the Python file `a2_hangman.py` to Omnivox.