Introduction to Computer Programming
Program 2

Due: Friday, March 26th, 2021 at 11:59pm

This program will introduce you to the topic of creating functions in Python, as well as looping mechanisms for repeating a computational process until a condition is reached.

Phase 1: Basic Hangman

You will implement a variation of the classic word game Hangman. If you are unfamiliar with the rules of the game, read http://en.wikipedia.org/wiki/Hangman_(game). Don't be intimidated by this problem - it's actually easier than it looks! I will guide you through the functions needed to implement this game.

1A) Getting Started

Download the files in common_area/program2 folder and copy them to your csce160 program2 folder on your computer.

Run the hangman.py program and you should see the following output:

```
Loading word list from file...
   55900 words loaded.

Let's play: None
```

If you see the above text, continue on to Hangman Game Requirements. If you don't, double check that both files are saved in the same place!

1B) Hangman Game Requirements

You will implement a function called hangman that will allow the user to play hangman against the computer. The computer picks the word, and the player tries to guess letters in the word.

Here is the general behavior you need to implement. Don't be intimidated! This is just a description; I will break this down into steps and provide further functional specs later on so keep reading!

1. The computer must select a word at random from the list of available words that was provided in words.txt **Note that words.txt contains words in all lowercase letters.**
2. The user is given a certain number of guesses at the beginning.
3. 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. penalize the user and updates the number of guesses remaining

4. The game ends when either the user guesses the secret word, or the user runs out of guesses.

Phase 2 Hangman: Three helper functions

Before I have you write code to organize the hangman game, I am going to break down the problem into logical subtasks, creating three helper functions you will need to have in order for this game to work. This is a common approach to computational problem solving.

The file hangman.py has a number of already implemented functions you can use while writing up your solution. You can ignore the code in the two functions at the top of the file that have already been implemented for you, though you should understand how to use each helper function by reading the docstrings.

2A) Determine whether the word has been guessed

First, implement the function is_word_guessed that takes in two parameters a string, secret_word, and a list of letters (strings), letters_guessed. This function returns a boolean True if secret_word has been guessed (i.e., all the letters of secret_word are in letters_guessed), and False otherwise.

This function will be useful in helping you decide when the hangman game has been successfully completed, and becomes an end test for any iterative loop that checks letters against the secret word.

For this function, you may assume that all the letters in secret_word and letters_guessed are lowercase.

Example Usage:
```
>>> secret_word = 'apple'
>>> letters_guessed = ['e', 'i', 'k', 'p', 'r', 's']
>>> print(is_word_guessed(secret_word, letters_guessed))
False
```

2B) Getting the user's guess

Next, implement the function get_guessed_word that takes in two parameters a string, secret_word, and a list of letters, letters_guessed. This function returns a string that is comprised of letters and underscores, based on what letters in letters_guessed are in secret_word. This shouldn't be too different from is_word_guessed!

We are going to use an underscore followed by a space `'_ '` to represent unknown letters. We could have chosen other symbols, but the combination of underscore and space is visible and easily discerned. Note that the space is super important, as otherwise it hard to distinguish whether `'____'` is four elements long or three. This is called usability it's very important, when programming, to consider the usability of your program. If users find your program difficult to understand or operate, they won't use it! We encourage you to think about usability when designing your program.

Hint: In designing your function, think about what information you want to return when done, whether you need a place to store that information as you loop over a data structure, and how you want to add information to your accumulated result.

Example Usage:

```
>>> secret_word = 'apple'
>>> letters_guessed = ['e', 'i', 'k', 'p', 'r', 's' ]
>>> print(get_guessed_word(secret_word, letters_guessed) )
'_ pp_ e'
```

2C) Getting all available letters

Next, implement the function get_available_letters that takes in one parameter a list of letters, letters_guessed. This function returns a string that is comprised of lowercase English letters all lowercase English letters that are not in letters_guessed.

This function should return the letters in alphabetical order. For this function, you may assume that all the letters in letters_guessed are lowercase.

Hint: You might consider using string.ascii_lowercase , which is a string comprised of all lowercase letters:

```
>>> import string
>>> print(string.ascii_lowercase)
abcdefghijklmnopqrstuvwxyz
```

Example Usage:

```
>>> letters_guessed = ['e', 'i', 'k', 'p', 'r', 's']
>>> print get_available_letters(letters_guessed)
abcdfghjlmnoqtuvwxyz
```

Phase 3 Hangman: The Game

Now that you have built some useful functions, you can turn to implementing the function hangman, which takes one parameter the secret_word the user is to guess. Initially, you can (and should!) manually set this secret word when you run this function – this will make it easier to test your code. But in the end, you will want the computer to select this secret word at random before inviting you or some other user to play the game by running this function.

Calling the hangman function starts up an interactive game of Hangman between the user and the computer. In designing your code, be sure you take advantage of the three helper functions, is_word_guessed , get_guessed_word , and get_available_letters , that you've defined in the previous part!

Below are the game requirements broken down in different categories. Make sure your implementation fits all the requirements!

Game Requirements

A. Game Architecture:

1. The computer must select a word at random from the 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 hangman.py.
2. Users start with 6 guesses.
3. At the start of the game, let the user know how many letters the computer's word contains and how many guesses s/he starts with.
4. The computer keeps track of all the letters the user has not guessed so far and before each turn shows the user the “remaining letters”

Example Game Implementation:

```
Loading word list from file...
   55900 words loaded.

Let's play: _ _ _ _ _ _ _ _ _ _ _
You have 6 guesses left.
You have 3 warnings left.
Available letters: abcdefghijklmnopqrstuvwxyz
Enter a letter:
```

B. User-Computer Interaction:

The game must be interactive and flow as follows:

1. Before each guess, you should display to the user:
   - i Remind the user of how many guesses s/he has left after each guess.
   - ii all the letters the user has not yet guessed
2. Ask the user to supply one guess at a time. (Look at the user input requirements below to see what types of inputs you can expect from the user)
3. Immediately after each guess, the user should be told whether the letter is in the computer's word.
4. After each guess, you should also display to the user the computer's word, with guessed letters displayed and unguessed letters replaced with an underscore and space (_)
5. At the end of the guess, print some dashes (-----) to help separate individual guesses from each other

Example Game Implementation:
(The blue color below is only there to show you what the user typed in, as opposed to what the computer output.)

```
Let's play: _ _ _ _ _ _ _ _ _ _
You have 6 guesses left.
You have 3 warnings left.
Available letters: abcdefghijklmnopqrstuvwxyz
Enter a letter: t
Oops! That letter is not in my word: _ _ _ _ _ _ _ _ _ _
--------------------
You have 5 guesses left.
You have 3 warnings left.
Available letters: abcdefghijklmnopqrsuvwxyz
Enter a letter:
```

C. User Input Requirements:

1. You may assume that the user will only guess one character at a time, but the user can choose any number, symbol or letter. Your code should accept capital and lowercase letters as valid guesses!

2. If the user inputs anything besides an alphabet (symbols, numbers), tell the user that they can only input an alphabet. Because the user might do this by accident, they should get 3 warnings at the beginning of the game. Each time they enter an invalid input, or a letter they have already guessed, they should lose a warning. If the user has no warnings left and enters an invalid input, they should lose a guess.

Hint #1: Use calls to the input function to get the user's guess.
1. Check that the user input is an alphabet
2. If the user does not input an uppercase or lowercase alphabet letter, subtract one warning or one guess.

Hint #2: you may find the string functions `str.isalpha() and str.lower()` helpful!

Hint #3: Since the words in words.txt are lowercase, it might be easier to convert the user input to lowercase at all times and have your game only handle lowercase.

Example Game Implementation:

```
--------------------
You have 5 guesses left.
You have 3 warnings left.
Available letters: abcdefghijklmnopqrsuvwxyz
Enter a letter: $
Oops! That is not a valid letter.
--------------------
You have 5 guesses left.
You have 2 warnings left.
Available letters: abcdefghijklmnopqrsuvwxyz
Enter a letter:
```

D. Game Rules:

1. The user starts with 3 warnings.
2. If the user inputs anything besides an alphabet (symbols, numbers), tell the user that they can only input an alphabet.
    a. If the user has one or more warning left, the user should lose one warning. Tell the user the number of remaining warnings.
    b. If the user has no remaining warnings, they should lose one guess.
3. If the user inputs a letter that has already been guessed, print a message telling the user the letter has already been guessed before.
    a. If the user has one or more warning left, the user should lose one warning. Tell the user the number of remaining warnings.
    b. If the user has no warnings, they should lose one guess.
4. If the user inputs a letter that hasn't been guessed before and the letter is in the secret word, the user loses **no** guesses.
5. **Consonants:** If the user inputs a consonant that hasn't been guessed and the consonant is not in the secret word, the user loses **one** guess if it's a consonant.
6. **Vowels:** If the vowel hasn't been guessed and the vowel is not in the secret word, the user loses **two** guesses. Vowels are *a*, *e*, *i*, *o*, and *u*. *y* does not count as a vowel.

E. Game Termination:

1. The game should end when the user constructs the full word or runs out of guesses.
2. If the player runs out of guesses before completing the word, tell them they lost and reveal the word to the user when the game ends.
3. If the user wins, print a congratulatory message and tell the user their score.
4. The total score is the number of `guesses_remaining` once the user has guessed the `secret_word` times the number of unique letters in `secret_word`.

Total score = guesses_remaining x number unique letters in secret_word

Example Implementation:
```
Loading word list from file...
   55900 words loaded.

Let's play: _ _ _ _ _ _ _ _ _ _
You have 6 guesses left.
You have 3 warnings left.
Available letters: abcdefghijklmnopqrstuvwxyz
Enter a letter: t
Oops! That letter is not in my word: _ _ _ _ _ _ _ _ _ _
--------------------
You have 5 guesses left.
You have 3 warnings left.
Available letters: abcdefghijklmnopqrsuvwxyz
Enter a letter: $
Oops! That is not a valid letter.
--------------------
You have 5 guesses left.
You have 2 warnings left.
Available letters: abcdefghijklmnopqrsuvwxyz
Enter a letter: r
Good guess: _ _ _ _ _ _ _ r_ _
--------------------
You have 5 guesses left.
You have 2 warnings left.
Available letters: abcdefghijklmnopqsuvwxyz
Enter a letter: e
Good guess: _ e_ _ _ _ _ re_
--------------------
You have 5 guesses left.
You have 2 warnings left.
Available letters: abcdfghijklmnopqsuvwxyz
Enter a letter: b
Good guess: _ e_ _ _ b_ re_
--------------------
You have 5 guesses left.
You have 2 warnings left.
Available letters: acdfghijklmnopqsuvwxyz
Enter a letter: n
Good guess: ne_ _ _ b_ re_
--------------------
You have 5 guesses left.
You have 2 warnings left.
Available letters: acdfghijklmopqsuvwxyz
Enter a letter: i
Good guess: nei_ _ b_ re_
--------------------
You have 5 guesses left.
You have 2 warnings left.
Available letters: acdfghjklmopqsuvwxyz
Enter a letter: g
Good guess: neig_ b_ re_
--------------------
You have 5 guesses left.
You have 2 warnings left.
Available letters: acdfhjklmopqsuvwxyz
Enter a letter: h
Good guess: neighb_ re_
--------------------
You have 5 guesses left.
You have 2 warnings left.
Available letters: acdfjklmopqsuvwxyz
Enter a letter: o
Good guess: neighbore_
--------------------
You have 5 guesses left.
You have 2 warnings left.
Available letters: acdfjklmpqsuvwxyz
Enter a letter: s
Oops! That letter is not in my word: neighbore_
--------------------
You have 4 guesses left.
You have 2 warnings left.
Available letters: acdfjklmpquvwxyz
Enter a letter: d
Good guess: neighbored
--------------------
Congratulations, you won!
Your total score for this game is: 36
```

Example Implementation:

```
Loading word list from file...
   55900 words loaded.

Let's play: _ _ _ _ _ _ _
You have 6 guesses left.
You have 3 warnings left.
Available letters: abcdefghijklmnopqrstuvwxyz
Enter a letter: f
Oops! That letter is not in my word: _ _ _ _ _ _ _
--------------------
You have 5 guesses left.
You have 3 warnings left.
Available letters: abcdeghijklmnopqrstuvwxyz
Enter a letter: &
Oops! That is not a valid letter.
--------------------
You have 5 guesses left.
You have 2 warnings left.
Available letters: abcdeghijklmnopqrstuvwxyz
Enter a letter: e
Good guess: _ _ _ _ _ e_
--------------------
You have 5 guesses left.
You have 2 warnings left.
Available letters: abcdghijklmnopqrstuvwxyz
Enter a letter: o
Oops! That letter is not in my word: _ _ _ _ _ e_
--------------------
You have 3 guesses left.
You have 2 warnings left.
Available letters: abcdghijklmnpqrstuvwxyz
Enter a letter: a
Oops! That letter is not in my word: _ _ _ _ _ e_
--------------------
You have 1 guesses left.
You have 2 warnings left.
Available letters: bcdghijklmnpqrstuvwxyz
Enter a letter: h
Good guess: _ _ _ _ he_
--------------------
You have 1 guesses left.
You have 2 warnings left.
Available letters: bcdgijklmnpqrstuvwxyz
Enter a letter: s
Good guess: s_ _ _ he_
--------------------
You have 1 guesses left.
You have 2 warnings left.
Available letters: bcdgijklmnpqrtuvwxyz
Enter a letter: b
Oops! That letter is not in my word: s_ _ _ he_
--------------------
Sorry, you ran out of guesses.
The word was scythed
```