**Computer Science Principles and Programming - 1**

**Project - Hangman**

For this problem, you will implement a variation of the classic wordgame Hangman. For those of you who are unfamiliar with the rules, you may read all about it [here](https://en.wikipedia.org/wiki/Hangman_(game)). In this problem, the second player will always be the computer, who will be picking a word at random.

In this problem, you will implement a function, called hangman, that will start up and 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 to get you going.

For this problem, you will need the file words.txt. This file contains the list of words that can be used in this game.

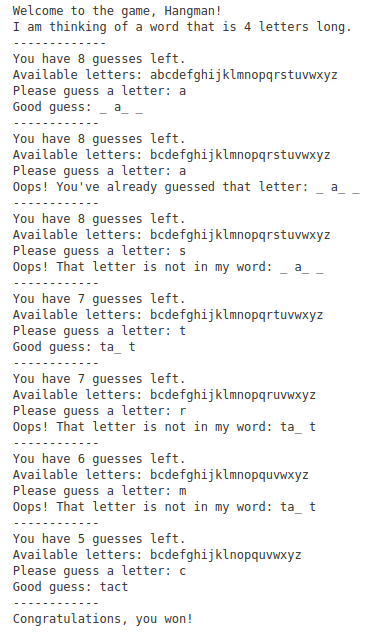
**Requirements**

Here are the requirements for your game:

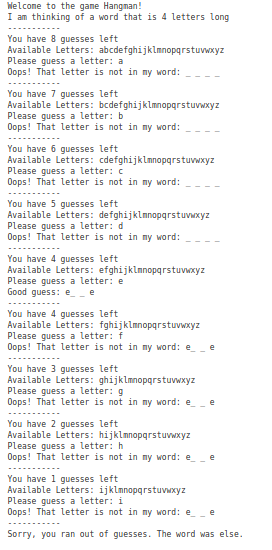
1. The computer must select a word at random from the list of available words that was provided in words.txt. The functions for loading the word list and selecting a random word have to be created.
2. The game must be interactive; the flow of the game should go as follows:
   1. At the start of the game, let the user know how many letters the computer's word contains.
   2. Ask the user to supply one guess (i.e. letter) per round.
   3. The user should receive feedback immediately after each guess about whether their guess appears in the computer's word.
   4. After each round, you should also display to the user the partially guessed word so far, as well as letters that the user has not yet guessed.
3. Some additional rules of the game:
4. A user is allowed 8 guesses. Make sure to remind the user of how many guesses s/he has left after each round. Assume that players will only ever submit one character at a time (A-Z).
5. A user loses a guess only when s/he guesses incorrectly.
6. If the user 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.
7. The game should end when the user constructs the full word or runs out of guesses. If the player runs out of guesses (s/he "loses"), reveal the word to the user when the game ends.

**Sample Output**

1. The output of a winning game should look like this…



1. And the output of a losing game should look like this...



**Steps :**

We'll start by writing 3 simple functions that will help us easily code the Hangman problem.

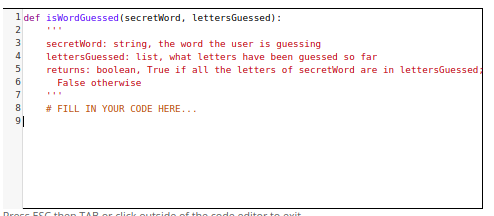
**Step 1 - Is the Word Guessed**

First, implement the function isWordGuessed that takes in two parameters - a string, secretWord, and a list of letters, lettersGuessed. This function returns a boolean - True if secretWord has been guessed (ie, all the letters of secretWord are in lettersGuessed) and False otherwise.

Example Usage:

|  |
| --- |
| >>> secretWord = 'apple'  >>> lettersGuessed = ['e', 'i', 'k', 'p', 'r', 's']  >>> print(isWordGuessed(secretWord, lettersGuessed))  False |

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



**Step 2 - Printing Out the User's Guess**

Next, implement the function getGuessedWord that takes in two parameters - a string, secretWord, and a list of letters, lettersGuessed. This function returns a string that is comprised of letters and underscores, based on what letters in lettersGuessed are in secretWord. This shouldn't be too different from isWordGuessed!

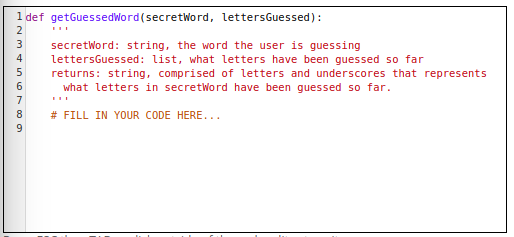
Example Usage:

|  |
| --- |
| >>> secretWord = 'apple'  >>> lettersGuessed = ['e', 'i', 'k', 'p', 'r', 's']  >>> print(getGuessedWord(secretWord, lettersGuessed))  '\_ pp\_ e' |

When inserting underscores into your string, it's a good idea to add at least a space after each one, so it's clear to the user how many unguessed letters are left in the string (compare the readability of \_\_\_\_ with \_ \_ \_ \_ ). 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!

For this problem, you are free to use spacing in any way you wish - our grader will only check that the letters and underscores are in the proper order; it will not look at spacing. We do encourage you to think about usability when designing.

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



**Step 3 - Printing Out all Available Letters**

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

Example Usage:

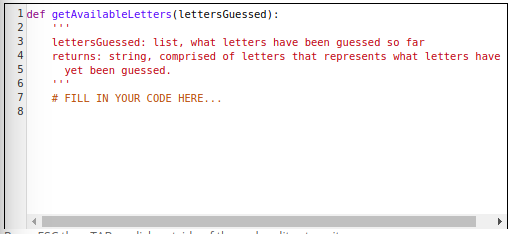
|  |
| --- |
| >>> lettersGuessed = ['e', 'i', 'k', 'p', 'r', 's']  >>> print(getAvailableLetters(lettersGuessed))  abcdfghjlmnoqtuvwxyz |

Note that this function should return the letters in alphabetical order, as in the example above.

For this function, you may assume that all the letters in lettersGuessed 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 |



**Step 4 - The Game**

Now you will implement the function hangman, which takes one parameter - the secretWord the user is to guess. This starts up an interactive game of Hangman between the user and the computer. Be sure you take advantage of the three helper functions, isWordGuessed, getGuessedWord, and getAvailableLetters, that you've defined in the previous part.

Hints:

You should start by loading the words and pick a random one. Write the functions loadWords and chooseWord. For loading the words.txt file in colab, upload the file in the program by using the following code.

|  |
| --- |
| from google.colab import files  uploaded = files.upload() |

Consider using lower() to convert user input to lower case. For example:

guess = 'A'

guessInLowerCase = guess.lower()

Consider writing additional helper functions if you need them!

There are four important pieces of information you may wish to store:

1. secretWord: The word to guess.
2. lettersGuessed: The letters that have been guessed so far.
3. mistakesMade: The number of incorrect guesses made so far.
4. availableLetters: The letters that may still be guessed. Every time a player guesses a letter, the guessed letter must be removed from availableLetters (and if they guess a letter that is not in availableLetters, you should print a message telling them they've already guessed that - so try again!).