6.0001/6.00 Fall 2017

Problem Set 2

Handed out: Thursday, September 14th, 2017.

Due: Friday, September 22th, 2017 at 4:59pm

This problem set 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.

Note on Collaboration:
You may work with other students. However, each student should write up and hand in his or her assignment separately. *Be sure to indicate with whom you have worked in the comments of your submission.*

Problem 1: Basic Hangman

You will implement a variation of the classic word game Hangman. Don't be intimidated by this problem - it's actually easier than it looks! We will 'scaffold' this problem, guiding you through the creation of helper functions before you implement the actual game.

A) Getting Started

Download the files "hangman.py", "test_ps2_student.py", and "words.txt", and save them all in the same directory. Run the file hangman.py before writing any code to ensure your files are saved correctly. The code we have given you loads in words from a file. You should see the following output in your shell:

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

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

B) 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 we want to implement. We will break this down into steps and provide further functional specs later on in the pset 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.

Problem 2
Hangman Part 1: Three helper functions

Before we have you write code to organize the hangman game, we are 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, and one we want you to begin experiencing.

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. Important: Do NOT change the name, input parameters, or specifications of any of the provided functions! You may add helper functions, but changing the definitions of the given functions will cause the unit tests to fail!

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
```

In order to test your functions, and this pset, please run the test_ps2_student.py file in Spyder. This will run a series of unit tests on your code. Note that these tests contain tests for functions you will implement in the code, so not all of them will pass right away. Examine the tests that start with test_is_word_guessed. If your function is correct, you

should see the following in the test printout:

```
test_is_word_guessed (__main__.TestPS2) ... ok
test_is_word_guessed_empty_list (__main__.TestPS2) ... ok
test_is_word_guessed_empty_string (__main__.TestPS2) ... ok
test_is_word_guessed_repeated_letters (__main__.TestPS2) ... ok
```

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!

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'
```

In order to test your functions, and this pset, please run the test_ps2_student.py in Spyder. This will run a series of unit tests on your code. Note that these tests contain tests for functions you will implement in the code, so not all of them will pass right away. **Examine the tests that start with test_get_guessed_word. If your function is correct, you should see the following in the test printout:**

```
test_get_guessed_word (__main__.TestPS2) ... ok
test_get_guessed_word_empty_list (__main__.TestPS2) ... ok
test_get_guessed_word_empty_string (__main__.TestPS2) ... ok
test_get_guessed_word_repeated_letters (__main__.TestPS2) ... ok
```

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
```

In order to test your functions, and this pset, please run the test_ps2_student.py file in Spyder. This will run a series of unit tests on your code. Note that these tests contain tests for functions you will implement in the code, so not all of them will pass right away. Examine the tests that start with test_get_guessed_word. If your function is correct, you should see the following in the test printout:

```
test_get_available_letters (__main__.TestPS2) ... ok
test_get_available_letters_empty_list (__main__.TestPS2) ... ok
test_get_available_letters_empty_string (__main__.TestPS2) ... ok
```

Problem 3
Hangman Part 2: 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 10 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.
Welcome to Hangman!
I am thinking of a word that is 4 letters long.
-------------
You have 10 guesses left.
Available letters: abcdefghijklmnopqrstuvwxyz
```

B. User-Computer Interaction:

The game must be interactive and flow as follows:

1. **Before each guess, you should display to the user:**
   a. **Remind the user of how many guesses s/he has left after each guess. Your code must print a string that contains the number of guesses the user has followed by the word "guess" or "guesses", for example "10 guesses", "3 guesses", "1 guess". If it does not follow this format, it will fail the test cases.**
   b. **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 must also display to the user the computer's word, with guessed letters displayed and unguessed letters replaced with an underscore (_)**
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.)

```
You have 10 guesses left.
Available letters: abcdefghijklmnopqrstuvwxyz
Please guess a letter: a
Good guess: _a__
------------
You have 10 guesses left.
Available letters: bcdefghijklmnopqrstuvwxyz
Please guess a letter: b
Oops! That letter is not in my word: _a__
```

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 a letter from the alphabet. Your code must print a message that contains the string "That is not a valid input".**

Hint #1: Use calls to the input function to get the user's guess.

  a. **Check that the user input is an alphabet**
  b. **If the user does not input an uppercase or lowercase alphabet letter, tell them they can only input a letter from the alphabet.**

Hint #2: you may find the string functions str.isalpha('your string') and str.lower('Your String') helpful! If you don't know what these functions are you could try typing help(str.isalpha) or help(str.lower) in your Spyder shell to see the documentation for the functions.

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 10 guesses left.
Available letters: bcdefghijklmnopqrstuvwxyz
Please guess a letter: s
Oops! That letter is not in my word: _a__
------------
You have 9 guesses left.
Available letters: bcdefghijklmnopqrtuvwxyz
Please guess a letter: $
Oops! That is not a valid letter. Please input a letter from the alphabet:  _a__
```

D. Game Rules:

1. **If the user inputs anything besides an alphabet (symbols, numbers), tell the user that they can only input an alphabet.**
2. **If the user inputs a letter that has already been guessed, print a message telling the user the letter has already been guessed before.**
3. **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.**
4. **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.**
5. **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.**

Example Implementation:

```
You have 7 guesses left.
Available letters: bcdefghijklmnopqrtuvwxyz
Please guess a letter: t
Good guess: ta_t
------------
You have 7 guesses left.
Available letters: bcdefghijklmnopqruvwxyz
Please guess a letter: e
```

```
Oops! That letter is not in my word: ta_t
------------
You have 5 guesses left.
Available letters: bcdfghijklmnopqruvwxyz
Please guess a letter: e
Oops! You've already guessed that letter: ta_t
```

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 plus the number of unique letters in secret_word times the length of the word. For example, the word "cake" has four unique letters, and is four letters long, thus the score for guessing this word would be guesses_remaining + (4*4). The word "call" has three unique letters, and is four letters long, so its score would be guesses_remaining + (3*4) Your code must print a string that contains the numerical score along with the word 'score'. For example "Your score is 48", "24 is your score!" or something along those lines, as long as it contains "score" and the number.**

Total score = guesses_remaining + (number unique letters in secret_word x length of secret word)

Example Implementation:

```
You have 5 guesses left.
Available letters: bcdfghijklnopquvwxyz
Please guess a letter: c
Good guess: tact
------------
Congratulations, you won!
Your total score for this game is: 17
```

Example Implementation:

```
You have 5 guesses left.
Available letters: bcfgjknquvwxyz
Please guess a letter: n
Good guess: dolphin
------------
Congratulations, you won!
Your total score for this game is: 54
```

F. General Hints:

1. **Consider writing additional helper functions if you need them.**
2. **There are three important pieces of information you may wish to store:**
   a. **secret_word: The word to guess. This is already used as the parameter name for the hangman function.**

b. **letters_guessed: The letters that have been guessed so far. If they guess a letter that is already in letters_guessed, you should print a message telling them they've already guessed it, but they do not lose a guess.**
c. **guesses_remaining: The number of guesses the user has left. Note that in our example game, the penalty for choosing an incorrect vowel is different than the penalty for choosing an incorrect consonant.**

G. Example Game:

Look carefully at the examples given above of running hangman, as that suggests examples of information you will want to print out after each guess of a letter.

Note: Try to make your print statements as close to the example game as possible!

The output of a winning game should look like this. (The blue color below is only there to show you what the user typed in, as opposed to what the computer output.)

```
Loading word list from file...
55900 words loaded.
Welcome to Hangman!
I am thinking of a word that is 4 letters long.
-------------
You have 10 guesses left.
Available letters: abcdefghijklmnopqrstuvwxyz
Please guess a letter: a
Good guess: _a__
------------
You have 10 guesses left.
Available letters: bcdefghijklmnopqrstuvwxyz
Please guess a letter: a
Oops! You've already guessed that letter: _a__
------------
You have 10 guesses left.
Available letters: bcdefghijklmnopqrstuvwxyz
Please guess a letter: s
Oops! That letter is not in my word: _a__
------------
You have 9 guesses left.
Available letters: bcdefghijklmnopqrtuvwxyz
Please guess a letter: $
Oops! That is not a valid letter: _a__
------------
You have 9 guesses left.
Available letters: bcdefghijklmnopqrtuvwxyz
Please guess a letter: t
Good guess: ta_t
------------
You have 9 guesses left.
Available letters: bcdefghijklmnopqruvwxyz
Please guess a letter: e
```

```
Oops! That letter is not in my word: ta_t
------------
You have 7 guesses left.
Available letters: bcdfghijklnopquvwxyz
Please guess a letter: c
Good guess: tact
------------
Congratulations, you won!
Your total score for this game is: 19
```

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

```
Loading word list from file...
55900 words loaded.
Welcome to Hangman!
I am thinking of a word that is 4 letters long
-----------
You have 10 guesses left
Available Letters: abcdefghijklmnopqrstuvwxyz
Please guess a letter: a
Oops! That letter is not in my word: ____
-----------
You have 8 guesses left
Available Letters: bcdefghijklmnopqrstuvwxyz
Please guess a letter: b
Oops! That letter is not in my word: ____
-----------
You have 7 guesses left
Available Letters: cdefghijklmnopqrstuvwxyz
Please guess a letter: c
Oops! That letter is not in my word: ____
-----------
You have 6 guesses left
Available Letters: defghijklmnopqrstuvwxyz
Please guess a letter: 2
Oops! That is not a valid letter: ____
-----------
You have 6 guesses left
Available Letters: defghijklmnopqrstuvwxyz
Please guess a letter: d
Oops! That letter is not in my word: ____
-----------
You have 5 guesses left
Available Letters: efghijklmnopqrstuvwxyz
Please guess a letter: u
Oops! That letter is not in my word: ____
-----------
You have 3 guesses left
Available Letters: efghijklmnopqrstvwxyz
Please guess a letter: e
```

```
Good guess: e__e
-----------
You have 3 guesses left
Available Letters: fghijklmnopqrstuvwxyz
Please guess a letter: f
Oops! That letter is not in my word: e__e
-----------
You have 2 guesses left
Available Letters: ghijklmnopqrstuvwxyz
Please guess a letter: o
Oops! That letter is not in my word: e__e
-----------
Sorry, you ran out of guesses. The word was else.
```

Once you have completed and tested your code (where you have manually provided the "secret" word, since knowing it helps you debug your code), you may want to try running against the computer. If you scroll down to the bottom of the file we provided, you will see two commented lines underneath the text if __name__ == "__main__":

```
#secret_word = choose_word(wordlist)
#hangman(secret_word)
```

These lines use functions we have provided (near the top of hangman.py), which you may want to examine. Try uncommenting these lines, and rerunning your code. This will give you a chance to try your skill against the computer, which uses our functions to load a large set of words and then pick one at random.

In order to test if your game runs properly, please run the test_ps2_student.py in Spyder. Examine the tests that start with test_play_game. If your function is correct, all of them should pass except the test labeled test_play_game_wildcard which you will implement later. You should see the following in the test printout:

```
test_play_game_short (__main__.TestPS2) ... ok
test_play_game_short_fail (__main__.TestPS2) ... ok
```

Problem 4
Hangman Part 3: The Game with Help

As you try playing Hangman against the computer, you probably have noticed that it isn't always easy to get the computer, especially when it selects and esoteric word (like "esoteric"!). It might be nice if you could occasionally ask the computer for some help.

We are going to have you create a variation of Hangman (we call this hangman_with_help and have provided an initial scaffold for writing it), with two new properties:

- If you guess the special character # the computer will provide you with one of the missing letters in the secret word, at a cost of two guesses. If you don't have two guesses still remaining, the computer will warn you of this and let you try again. To do this, we suggest you write a helper function that takes two arguments: the secret word and the set of available letters (which you can compute using get_available_letters; and which finds the set of unique letters that are in both the secret word and the available letters, and returns that as a string. Assume that you call this string choose_from. You can then use the following expressions to pick a character from that string at random:

```
new = random.randint(0, len(choose_from) -1)

exposed_letter = choose_from[new]
```

and thus you can add this letter to your set of correctly guessed letters, show the new guessed word, and continue. Your code must print the letter that was revealed as well as the word “revealed”. If it does not, it will not pass the test case!

Here is some sample output of using hangman_with_help:

Example Implementation:

```
------
You currently have 1 guess left
Available letters: abdefghijklmnopqstuvwxyz
Please guess a letter: #
Oops! Not enough guesses left: r_c_c_r
------
```

Example Implementation:

```
The secret word contains 7 letters
------
You currently have 10 guesses left
Available letters: abcdefghijklmnopqrstuvwxyz
Please guess a letter: r
Good guess: r_____r
------
You currently have 10 guesses left
Available letters: abcdefghijklmnopqstuvwxyz
Please guess a letter: #
Letter revealed: c
r_c_c_r
------
You currently have 8 guesses left
Available letters: abdeghijklmnopqstuvwxyz
Please guess a letter: #
Letter revealed: a
rac_car
------
You currently have 6 guesses left
Available letters: bdeghijklmnopqstuvwxyz
Please guess a letter: e
Good guess: racecar
------
Congratulations, you won!
Your total score for this game is: 34
```

In order to test if your game runs properly, please run the test_ps2_student.py file in Spyder. Examine the test_play_game_wildcard test. If your function is correct, all of the tests should now pass. You should see the following in the test printout:

```
test_play_game_wildcard (__main__.TestPS2) ... ok
```

(Please note that passing all of these tests does not necessarily mean you will receive a 100 on the problem set. The staff will run additional tests on your code to check for correctness.)

This completes the problem set!

Hand-in Procedure

1. Naming Files

Save your solutions with the original file name: hangman.py. Do not ignore this step or save your files with a different name!

2. Time and Collaboration Info

At the start of your file, in a comment, write down the number of hours (roughly) you spent on the problems in that part, and the names of the people with whom you collaborated.

For example:

```
        # Problem Set 2
        # Name: Jane Lee
        # Collaborators: John Doe
        # Time Spent: 3:30
        # Late Days Used: 1 (only if you are using any)
        # … your code goes here …
```

3. Submit

To submit hangman.py, upload it to the problem set website linked from Stellar. You may upload new versions of each file until the 4:59 PM deadline, but anything uploaded after that time will be counted towards your late days, if you have any remaining. If you have no remaining late days, you will receive no credit for a late submission.

After you submit, please be sure to view your submitted file and double-check you submitted the right thing.