Problem Set 2: Hangman

The questions below are due on Wednesday October 19, 2022; 09:00:00 PM.

Halfway due date: Oct 12 at 09:00PM
Checkoff start: Oct 20 at 11:00AM
Checkoff due: Oct 27 at 09:00PM

Download Files

Pset Buddy

You do not have a buddy assigned for this pset.

Introduction

Objectives

- Writing and calling functions in Python
- Using loop mechanisms to repeat a computational process until a condition is reached

Collaboration

- Students may work together, but each student should write up and hand in their assignment separately. Students may not submit the exact same code.
- Students are not permitted to look at or copy each other's code or code structure.
- Include the names of your collaborators in a comment at the start of each file.
- **Please refer to the collaboration policy in the** Course Information **for more details.**

Although this handout is long, the information is here to provide you with context, useful examples, and hints, so be sure to read carefully.

Getting Started

In this problem set, you will implement a variation of the classic word game, Hangman.

A) File Setup

Download the files **hangman.py**, **test_ps2_student.py**, and **words.txt**, and **save them all in the same directory.**

Run **hangman.py** before writing any code to ensure your files are saved correctly. The starter code loads words from the **words.txt** file. You should see the following output in your shell:

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

B) Hangman Game Overview

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.

The general behavior we want to implement is described below. We will break this down into steps and provide further functional specifications later on in the pset.

1. The computer selects a word at random from the list of words 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 user inputs their guess, and the computer either:
   - reveals the letter if it exists in the secret word,

- informs the user if their guess is invalid (i.e. longer than 1 character, not a letter, or has already been guessed) and does not penalize or reveal anything, or
- penalizes the user and updates the number of guesses remaining if the guess is valid and does not exist in the secret word.

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

We will also introduce a feature to the game to make the game easier for the user. The user will be able to input a special 'help' character, `!`, that reveals an unguessed letter at the expense of losing more guesses.

1) The Three Helper Functions

To approach the game, we will break down the problem into logical subtasks, creating three helper functions. This is a common approach to computational problem solving.

The starter code in **hangman.py** has implementations for the helper functions **`load_words`** and **`choose_word`**. You may read the docstrings associated with each function to understand what they do.

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.

1.1) Determine if the Player has Won

Implement the function **`has_player_won`** according to its docstring. This function will be useful in determining when the hangman game has been won (i.e., the user has guessed all the letters in the secret word).

Example Usage

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

Testing: Navigate to the **test_ps2_student.py** file and run it in Spyder. This will run a series of unit tests on your code. Note that this file contains tests for functions you will implement later on in this pset, so not all of them will pass right away. Examine the tests that start with **`test_has_player_won`**. If your function is correct, you should see the following printout:

```
test_has_player_won (__main__.TestPS2) ... ok
test_has_player_won_empty_list (__main__.TestPS2) ... ok
test_has_player_won_empty_string (__main__.TestPS2) ... ok
test_has_player_won_repeated_letters (__main__.TestPS2) ... ok
```

1.2) Revealing Letters

Next, implement the function **`get_word_progress`** according to its docstring. This should be fairly similar to **`has_player_won`**.

Hint: Think about...

- if you need to store information as you loop over a data structure
- 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_word_progress(secret_word, letters_guessed))
*pp*e
```

Testing: Run **test_ps2_student.py**. Examine the tests that start with **`test_get_word_progress`**. If your function is correct, the test printout should read:

```
test_get_word_progress (__main__.TestPS2) ... ok
test_get_word_progress_empty_list (__main__.TestPS2) ... ok
test_get_word_progress_empty_string (__main__.TestPS2) ... ok
test_get_word_progress_repeated_letters (__main__.TestPS2) ... ok
```

1.3) Get Available Letters

Next, implement the function **`get_available_letters`** according to its docstring. This function should return the letters in alphabetical order.

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
```

Testing: Run **test_ps2_student.py**. Examine the tests that start with **`test_get_available_letters`**. If your function is correct, the test printout should read:

```
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
```

2) The Game

Now that you have written some useful helper functions, you can implement the **`hangman`** function, which takes two parameters: (1) **`secret_word`**, the secret word the user is to guess; and (2) **`with_help`**, a boolean representing whether or not the game is to be played with the 'help' functionality.

Calling the `hangman` function starts up an interactive game between the user and the computer. In designing your code, be sure you take advantage of the three helper functions, `has_player_won`, `get_word_progress`, and `get_available_letters` from Problem 1! **You may write additional helper functions if you need them**.

To test your `hangman` function, you should call it from within the `if __name__ == "__main__":` condition found at the bottom of **hangman.py**. Manually set the secret word to make it easier to test your code. However, in the end, you should also test your program with **the secret word selected at random**.

Example code to manually test your `hangman` function (at the bottom of **hangman.py**):

```
if __name__ == "__main__":
    secret_word = "tact"
    with_help = False
    hangman(secret_word, with_help)
```

Important Notes:

1. **Make your print statements as close to the example games as possible.** The appendix also contains helpful examples of gameplay.
2. Please refer to Helpful Debugging Tips if you have trouble debugging.

2.1) Game Setup

1. The **`secret_word`** along with the boolean **`with_help`** are passed into the **`hangman`** function as parameters.
2. At the start of the game, display how many letters `secret_word` contains.
3. Users start with **10** guesses.

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.
```

2.2) User-computer interaction

1. Before each guess, you should display to the user:
   - At least **three (3)** dashes (e.g. `--------------`) to separate individual guesses from each other. **Leaving out the row of dashes will cause the tester to fail**.
   - How many guesses they have remaining
   - All the letters that have not yet been guessed
2. Ask the user to supply one guess at a time.
   - The user can type any number, symbol, or letter. Your code should only accept capital and lowercase single letters as valid guesses!
   - If the game is played with help, your code should also accept the help character (`!`)
3. Immediately after each guess, you should display:
   - Whether or not the letter is in the secret word (see the example implementation below)
   - The word with guessed letters revealed and unguessed letters as asterisks (`*`)

Example Game Implementation 1:

```
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  # This is the user input
Good guess: *a**
--------------
You have 10 guesses left.
Available letters: bcdefghijklmnopqrstuvwxyz
Please guess a letter: b  # This is the user input
Oops! That letter is not in my word: *a**
--------------
You have 9 guesses left.
Available letters: cdefghijklmnopqrstuvwxyz
Please guess a letter: 2  # This is the user input
Oops! That is not a valid letter. Please input a letter from
the alphabet: *a**
--------------
You have 9 guesses left.
Available letters: cdefghijklmnopqrstuvwxyz
Please guess a letter: foo  # This is the user input
Oops! That is not a valid letter. Please input a letter from
the alphabet: *a**
--------------
You have 9 guesses left.
Available letters: cdefghijklmnopqrstuvwxyz
Please guess a letter: +  # This is the user input
Oops! That is not a valid letter. Please input a letter from
the alphabet: *a**
```

NOTE: `# This is the user input` is a **comment** and will not be displayed in your output.

Hints:

1. Use calls to the **`input()`** function to get the user's guess.
   - Check that the user input is an alphabet letter (or the help character if the game is played with help).
   - If the user does not input a valid letter/character, tell them they can only input a letter from the alphabet.
2. Since the words in **words.txt** are lowercase, we suggest converting user input to lowercase so program only needs to handle lowercase characters.
3. You may find the string functions **`str.isalpha()`** and **`str.lower()`** helpful! You can type `help(str.isalpha)` or `help(str.lower)` in the Spyder shell to see documentation for the functions.

Example usage of `str.isalpha()` and `str.lower()`:

```
>> my_string = "HeLLoWoRlD"
>> my_string.isalpha()
True
>> my_string.lower()
'helloworld'
```

2.3) Guesses Remaining

If the user inputs:

1. Anything besides a letter in the alphabet (e.g. symbols or numbers), tell the user that they can only input an alphabet letter. The user loses **no** guesses. **Note**: When the game is being played with help, `!` is also a valid input.
2. A letter that has already been guessed, print a message telling the user the letter has already been guessed before. The user loses **no** guesses.
3. Any 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.
5. **Vowels:** If the user inputs a vowel that 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`. The letter `y` does not count as a vowel. Note: if a user inputs an incorrect vowel that hasn't been guessed and there is only one guess remaining, the user loses and the game is over.

Example Game Implementation 1 (continued):

```
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: bcdfghijklmnopqruvwxyz
Please guess a letter: e
Oops! You've already guessed that letter: ta*t
```

2.4) The Game with Help

It isn't always easy to beat the computer, especially when it selects an esoteric word. It might be nice if you could ask for some help.

To do this you will create a feature of the game that works as follows:

- If you type the special character `!`, the computer will provide you with one of the missing letters in the secret word at a cost of **three** guesses. This character should be the **only** non-letter input that your game accepts as a guess.
- **If you do not have at least three guesses remaining**, the computer will warn you of this and let you try again. You lose **no** guesses.

Note: The user can play the game with this feature only when the `with_help` parameter is **`True`**.

As a starting point, we suggest writing **a helper function that chooses a letter to reveal**. It should take two arguments: the secret word and the string of available letters (from `get_available_letters`). This helper function should create a string **`choose_from`** containing the unique letters that are in both the secret word and the available letters. You can then use the following statements to pick a random character **`revealed_letter`** from that string:

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

Your helper function should then return this **`revealed_letter`**. Back in your original game logic, you'll need to add a conditional statement to catch the case of the user inputting `!`. This case, if triggered, can add the letter returned by your helper function to `letters_guessed`, show the new word progress, decrement the remaining guesses by 3, and continue the gameplay.

Example Game Implementation 2:

```
Welcome to Hangman!
I am thinking of a word that is 7 letters long.
--------------
You currently have 10 guesses left.
Available letters: abcdefghijklmnopqrstu
Please guess a letter: !
Letter revealed: r
r*****r
--------------
You currently have 7 guesses left.
Available letters: abcdefghijklmnopqrstu
Please guess a letter: !
Letter revealed: a
ra***ar
--------------
You currently have 4 guesses left.
Available letters: abdefghijklmnopqrstu
Please guess a letter: !
Letter revealed: e
ra*e*ar
--------------
You currently have 1 guess left.
Available letters: abdefghijklmnopqstu
Please guess a letter: !
Oops! Not enough guesses left: ra*e*ar
```

Please refer to the appendix at the end of this handout for an example of a complete game of hangman with help.

2.5) Game Termination

1. The game ends when the user guesses all the letters in **`secret_word`** or has 0 guesses remaining.
2. If the user wins, print a congratulatory message, and tell the user their score.
   - $\texttt{total\_score} = (\texttt{guesses\_remaining} + 4 \times \text{number of unique letters in } \texttt{secret\_word}) + (3 \times \text{length of } \texttt{secret\_word})$
   - Example: For a game with secret word "asleep" with 6 guesses remaining, there are a total of 5 unique letters (`a`, `s`, `l`, `e`, and `p`). Thus, the final score is: $(6 + 4 \times 5) + (3 \times 6) = 44$.
3. 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.

Example Implementation (win):

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

NOTE: `# ...snip...` is not part of the output; it indicates that only part of the example game implementation is shown.

Example Implementation (Lose):

```
# ... snip ...
You have 1 guess left.
Available Letters: ghijklmnopqrstuvwxyz
Please guess a letter: i
Oops! That letter is not in my word: e**e
--------------
Sorry, you ran out of guesses. The word was else.
```

NOTE: `# ...snip...` is not part of the output; it indicates that only part of the example game implementation is shown.

2.6) Testing your code

Look carefully at the example hangman games in the handout appendix and make your print statements as close to the example games as possible! If you run into issues, try consulting the debugging hints.

If you scroll to the bottom of hangman.py, you will see the lines below:

```
if __name__ ==  "__main__":
    # To test your game, uncomment the following three lines.

    # secret_word = choose_word(wordlist)
    # with_help = False
    # hangman(secret_word, with_help)
```

Uncomment the bottom three lines to choose a random secret word and play hangman with the provided secret word. Feel free to pass in your own secret word when testing your program.

2.6.1) Student Tester

In order to test if your game runs properly, please run **test_ps2_student.py**. If your function is correct, you should see the following in the test printout:

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

You might see some additional messages printed out between the `...` and the `ok`. For example, you might see the following:

```
Problem Set 2 Unit Test Results:
All correct!
Points for these tests: 5/5
(Please note that this is not your final pset score, additional test cases will be run on submissions)
ok
```

This is fine.

3) Hand-in Procedure

3.1) Naming Files

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

3.2) Time and Collaboration Info

At the start of each file, in a comment, write down the names of your collaborators. For example:

```
  # Problem Set 2, hangman.py
  # Name: Jane Lee
  # Collaborators: John Doe
```

Please estimate the number of hours you spent on the Problem Set in the question box below.

A Python Error Occurred:

```
Error on line 30 of question tag.ImportError: cannot import name 'Sequence' from 'collections' (/home/ca
```

3.3) Half-way Submission

All students should submit their progress by the half-way due date (1 week before the final due date).
This submission will be worth 1 point out of the problem set grade and will not be graded for correctness. The intention is to make sure that you are making steady progress on the problem set as opposed to working on it in the final days before the due date.

You may upload new versions of each file until Oct 12 at 09:00PM. You cannot use extensions or late days on this submission.

Download Most Recent Submission

Select File No file selected

Submit Halfway Handin Submitted

You have infinitely many submissions remaining.

3.4) Final Submission

Be sure to run the student tester and make sure all the tests pass. However, the student tester contains only a subset of the tests that will be run to determine the problem set grade. Passing all of the provided test cases does not guarantee full credit on the pset.

You may upload new versions until Oct 19 at 09:00PM, 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.

When you upload a new file with the same name, your old one will be overwritten.

Please refresh the page before submitting a new file. If you do not, your latest submission won't be updated.

Download Most Recent Submission

Select File No file selected

Submit **Code Submitted Successfully!**

You have infinitely many submissions remaining.

4) Appendix

4.1) Hangman Example (Winning Game)

```
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. Please input a letter from the alphabet: *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: 31
```

4.2) Hangman Example (Losing Game)

```
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. Please input a letter from the alphabet: ****
--------------
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
--------------
```

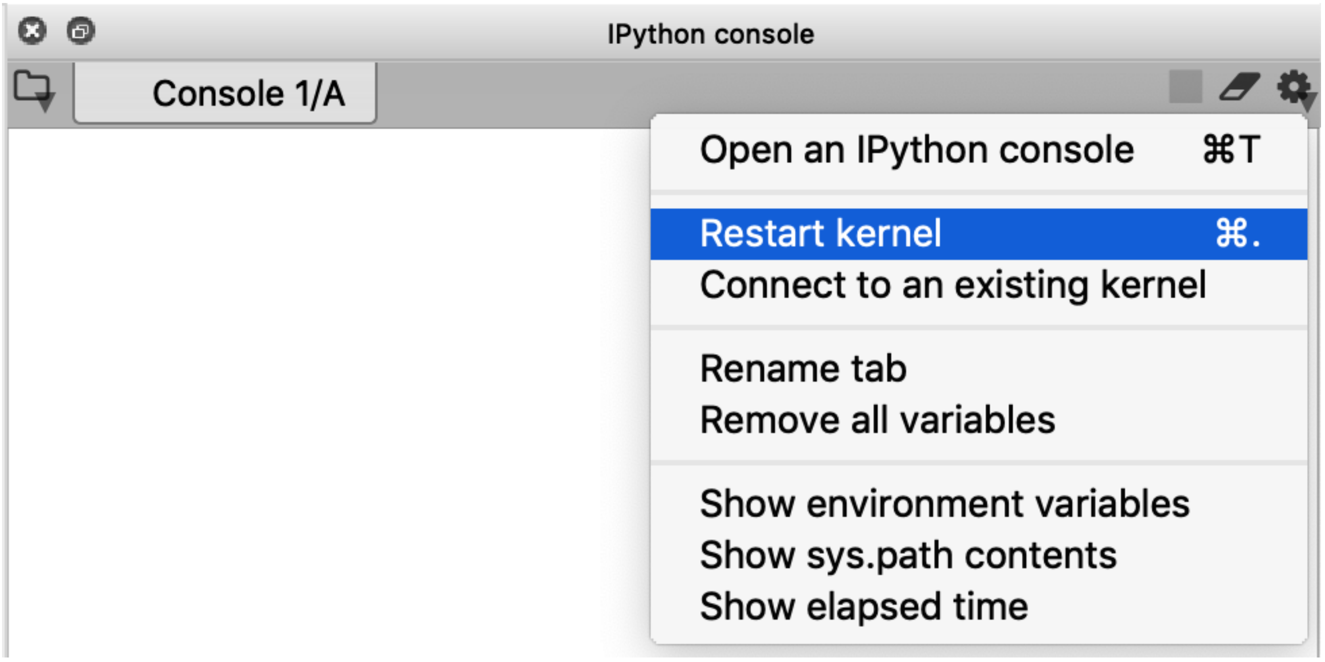
```
You have 2 guesses left.
Available Letters: ghijklmnopqrstuvwxyz
Please guess a letter: o
Oops! That letter is not in my word: e**e
--------------
Sorry, you ran out of guesses. The word was else.
```

4.3) Hangman With Help

```
Loading word list from file...
  55900 words loaded.
Welcome to Hangman!
I am thinking of a word that is 7 letters long
--------------
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 7 guesses left
Available letters: abdeghijklmnopqstuvwxyz
Please guess a letter: !
Letter revealed: a
rac*car
--------------
You currently have 4 guesses left
Available letters: bdeghijklmnopqstuvwxyz
Please guess a letter: e
Good guess: racecar
--------------
Congratulations, you won!
Your total score for this game is: 41
```

5) Helpful Debugging Tips

- Remember, the auto grader is testing your printed output against the expected printed output. That makes it necessary that you design your game to match as closely as possible. For instance, **your output needs to have the same number of lines as the expected output.** Therefore, be careful to include everything that our examples print out, including the dashes (`--------------`) at the end of a turn!
- If you fail any of the **`test_play_game`** tests, a file called **run_game_test_results.txt** should appear in the same directory as **test_ps2_student.py**. The text file should contain the expected output and your actual output.
    - **Do not be concerned if your input prompt messages (i.e. `Please guess a letter:`) do not show up in the text file.** Unfortunately, input prompts are not redirected to the file output. This should not affect the tester.
    - Make sure you provide a **non-empty string parameter to input**. That is, you should be calling `input("Please guess a letter: ")`. The tester cannot properly feed input into your program if you call `input("")` instead.
    - If you come across `NameError: name 'outputstr' is not defined`, please **restart your Spyder kernel**.



- If you are using the print function, you may be adding extra spaces unintentionally. For example, the following statement inserts a space between the two strings.

```
>>> print('foo', 'bar')
foo bar
```

You can change what print inserts between strings by passing in an optional `sep` parameter, which is by default one space. Python by default inserts a `\n`, the newline character, at the end of the string to be printed, which is why you normally see each print statement on a new line. You can change this using the optional `end` parameter. You don't need this for the pset, but it's good to be aware of it.

```
>>> print('foo', 'bar', 'baz', sep=" and ", end='\n')
foo and bar and baz
```

- The Spyder console might insert an extra blank line in the output even though there's nothing in the code that tells it to. This should not affect the tester.
- Are you scoring things properly? You might want to write a function to help you calculate the score given a word and the number of guesses remaining. Be sure to account for guesses lost using hints as well.

As always, strategically **printing the values of variables** is often a helpful way of checking that the correct values are stored in those variables.