Bookmarks

- Overview
- Entrance Survey
- Week 1
- Week 2
- Week 3

Lecture 5 - Recursion - Time 55:06
Lecture Sequence

Lecture 6 - Objects - Time 46:20
Lecture Sequence

Complete Programming Experience: ndigits

Problem Set 3
Problem Set due Jul 04, 2016 at 23:30 UTC

- Sandbox

Week 3 > Problem Set 3 > Introduction to Hangman

Bookmark

A WORDGAME: HANGMAN

Note: Do not 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.

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. 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 code files ps3_hangman.py and words.txt. Right-click on each and hit "Save Link As". **Be sure to save them in same directory.** Open and run the file ps3_hangman.py without making any modifications to it, in order to ensure that everything is set up correctly. By "open and run" we mean do the following:

- Go to Canopy. From the File menu, choose "Open".
- Find the file ps3_hangman.py and choose it.
- The template ps3_hangman.py file should now be open in Canopy. Click on it. From the Run menu, choose "Run File" (or simply hit Ctrl + R).

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 printed out:

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

If you see an IOError instead (e.g., "No such file or directory"), you should 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). Windows users, change the backslashes to forward slashes, like below.

For example, if you saved ps3_hangman.py and words.txt in the directory "C:/Users/Ana/" change the line:

WORDLIST_FILENAME = "words.txt" to something like

WORDLIST_FILENAME = "C:/Users/Ana/words.txt"

This folder will vary depending on where you saved the files.

The file ps3_hangman.py has a number of already implemented functions you can use while writing up your solution. You can ignore the code between the following comments, though you should read and understand how to use each helper function by reading the docstrings:

```
# -----------------------------------
# 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 will want to do all of your coding for this problem within this file as well because you will be writing a program that depends on each function you write.

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 already been provided for you in ps3_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 user know how many letters the computer's word contains.

   - Ask the user to supply one guess (i.e. letter) per round.

   - The user should receive feedback immediately after each guess about whether their guess appears in the computer's word.

   - 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:

   - 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).

   - A user loses a guess **only** when s/he guesses incorrectly.

   - 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.

   - 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

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

On the next page, we'll break down the problem into logical subtasks, creating helper functions you will need to have in order for this game to work.