Hangman

CompSci 101

February 19, 2013

1 Introduction/Goals

The goal of this assignment is to write a program that implements the well known *Hangman* game on the python terminal.

In the context of this assignment you must:

- Use a text file that contains, which your program will use to pick a word at random and then allow the user to play *Hangman* with that word.
- Write the functions that are needed to accept user input, check if the letter is in the word, show the current progress of the player, etc.
- Combine the functions that you wrote in a clear manner to construct the full Hangman game.

In order to do so, a project is snarfable via Ambient where you must write your code in the Hangman.py file.

The code can also be found here: `http://www.cs.duke.edu/courses/spring13/compsci101/assign/hangman/code/Hangdemo.py`

Among others you will need a text file, which contains all the various words which from your main program will choose from. This file is already provided for you and can be found here: `http://www.cs.duke.edu/courses/spring13/compsci101/assign/hangman/code/Hangdemo.py`. It is strongly recommended that you download both these files and try to run them in order to understand *what* they do.

Example code is given to you in the Hangdemo.py file and the Guess-Number.py. Feel free to copy and paste that code.

2 Specifications

This section presents the specifications of a *Hangman game*.

- Valid words contain lower case characters and hyphens. This means that all the following are **valid words**:
  - alpha
  - home-brew
  - telecommunications

  textbfNon-valid words include:
  - Chris
  - NBA
  - 1st
  - m.v.p.

  This means that players of your game (users) will only be able to enter as an input lower case letters and hyphens. As a trick you can allow the user to input any letters and then transform them to lower case.
- Before the game starts the user should be able to specify how many tries he has before he loses, if he doesn't specify then the default will be to give him 15 tries. The user shouldn't be able to give as an input *more than 26 tries* or *less than 1.*
- After each player's move the program should print out the progress of the game. This includes:
  - Printing the word in the hangman way - i.e. a special character at the position of each letter that the player hasn't found and the letters that the player has found in their positions. An example would be: **Ex*mp*e**.
  - Printing the list of the letters that the player has already tried without any success.
  - Printing the number of tries the player has left.
- When the player uses all his tries, or he finds the word then the game should end and print either a winning message or a losing message.
- In your code, you should not use global variables (i.e. variables that are accessible from all functions.

The rest of this document gives you a specific guidance on fulfilling the above specifications, which **is not required to follow**, but is provided for your own help.

3 Guidelines

In this section you are provided with some general guidelines for the various functions that you will need to implement in order to accomplish all specifications as described in the previous section. This is optional as you can code the game any way you want as long as it works.

3.1 Functions

Letter Validation: As already stated, valid words contain only lower case characters and the hyphen character. This means that you need to implement a function that checks if the user's input is compliant with the aforementioned rule.

Your function should either return a valid letter that corresponds to the user's input, or return an error code which your main program will recognize and inform the user that his input was wrong and he needs to enter a letter again.

The functionality that we want to achieve is that whenever a user enters a non-valid character from his keyboard (e.g. a number, a special character etc.) the function should recognize that and the program should ask for another letter from the user.

Saving Inputs: Whenever the user enters a new **valid** letter, your program should decide whether that letter appears in the word or not, and accordingly save it to the corresponding list.

This means that in the internal state of your program you will have 2 different lists, one for the correct letters and one for the wrong letters.

This specific task can be implemented either in the main function of the program, or as a separate function.

Printing State: You should implement a function which reads the state of the game (i.e. the lists of the user's inputs, the number of tries and the word-to-find) and accordingly prints the relevant information - as discussed in section 2.

This function should be called after each player's move.

Winning Condition: You should implement a function that given the state of the game either:

- Lets the game continue (i.e. when the player hasn't found the word, and he still have some tries left).
- Informs the player that he has won (i.e. when a player has found **all the letters** of the word).
- Informs the player that he has lost (i.e. when he is out of tries and he hasn't found the word).

3.2 Main Program

Your main program should keep an internal state of the game as described earlier, and combine the functions that you implemented in section 3 in order to implement the game of hangman.

You should focus on building a main program that is lightweight and easy to read. Building functions for smaller tasks that are performed often will help you in that aspect.

Section 3 just provides general guidelines on how to make your code more robust and lightweight. Solutions that implement the specifications of section 2 but do not follow the guidelines of section 3 will be graded equally.

4 Extra Credit

For extra credit, you can offer the user a choice of playing hangman with words belonging to 3 or more categories. You *must supply* different text files for each category (i.e. having multiple text files which contain words only from one category).

5 Deliverables

The following are expected:

- The python module that implements the hangman application.
- A README plain text file that contains valuable information like additional comments, the resources you used to finish the assignment etc.

Working together is permitted, what is not permitted is working together without stating so in the README file. Plagiarism will not be tolerated and we expect everybody to comply with the Duke Community Standard.

Important Dates

- Turn in by March 4: Full points
- Turn in by March 7: 10% down
- Turn in by March 18: 50% down

All deadlines are at 11:55pm on that day. **Anything turned in after 11:55 on March 18 will not be graded**.

How to submit: Either via Eclipse Ambient or the web submit system. Links to these two methods can be found on the Large Assignment tab on the course website.

6 Appendix

Input Files In this url `http://www.cs.duke.edu/courses/spring13/compsci101/assign/hangman/code/lowerwords.txt` you will find a plain-text file which contains one lower case word per line. You are expected to download this file into your working directory and use it as the input file of your program - i.e. the program will extract words out of this file.

Hints **Code refactoring** is a very common and useful tactic amongst programmers. A simple definition for code refactoring is the process of 'isolating' complex functionalities of your main program into separate functions, with the intend of calling this functions from your main program. More about code refactoring you can find here: `http://en.wikipedia.org/wiki/Code_refactoring`