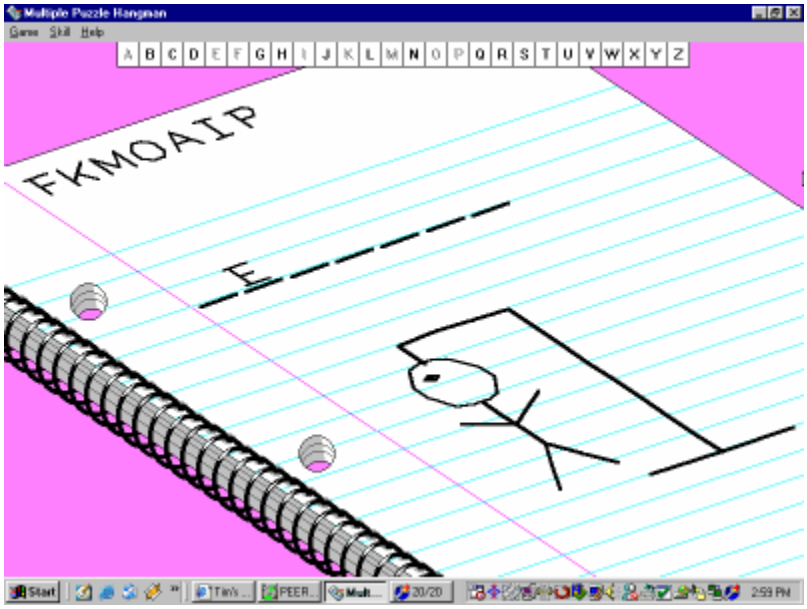
Evil Hangman



Picture from: http://www.timshen.truepath.com/bwg.html

Objectives:

- To review python classes
- To review python lists
- To practice algorithm development

Assignment:

This assignment is based off a Nifty Assignment by Keith Schwarz and presented at SIGCSE in 2011. From the original assignment:

It's hard to write computer programs to play games. When we as humans sit down to play a game, we can draw on past experience, adapt to our opponents' strategies, and learn from our mistakes. Computers, on the other hand, blindly follow a preset algorithm that (hopefully) causes it to act somewhat intelligently. Though computers have bested their human masters in some games, most notably checkers and chess, the programs that do so often draw on hundreds of years of human game experience and use extraordinarily complex algorithms and optimizations to out-calculate their opponents.

While there are many viable strategies for building competitive computer game players, there is one approach that has been fairly neglected in modern research – cheating. Why spend all the effort trying to teach a computer the nuances of strategy when you can simply write a program to play dirty and win handily all the time? In this assignment, you will build a mischievous program that bends the rules of *Hangman* to trounce its human opponent time and time again. In doing so, you'll cement your skills with abstract data types and iterators, and will hone your

general programming savvy. Plus, you'll end up with a piece of software which will be highly entertaining. At least, from your perspective. ☺

In case you aren't familiar with the game *Hangman*, the rules are as follows:
1. One player chooses a secret word, then writes out a number of dashes equal to the word length.
2. The other player begins guessing letters. Whenever she guesses a letter contained in the hidden word, the first player reveals each instance of that letter in the word. Otherwise, the guess is wrong.
3. The game ends either when all the letters in the word have been revealed or when the guesser has run out of guesses.

Fundamental to the game is the fact the first player accurately represents the word she has chosen. That way, when the other players guess letters, she can reveal whether that letter is in the word. But what happens if the player doesn't do this? This gives the player who chooses the hidden word an enormous advantage. For example, suppose that you're the player trying to guess the word, and at some point you end up revealing letters until you arrive at this point with only one guess remaining:

D O – B L E

There are only two words in the English language that match this pattern: “doable” and “double.” If the player who chose the hidden word is playing fairly, then you have a fifty-fifty chance of winning this game if you guess 'A' or 'U' as the missing letter. However, if your opponent is cheating and hasn't actually committed to either word, then there is no possible way you can win this game. No matter what letter you guess, your opponent can claim that she had picked the other word, and you will lose the game. That is, if you guess that the word is “doable,” she can pretend that she committed to “double” the whole time, and vice-versa.

In the original assignment, he goes on to explain how to break everything down into word families and to always choose the word family with the largest number of words in it.

We’re going to do something simpler. We will start by asking the user for the length of the word and verify that there are words of that length in the dictionary. Then we will have a “pool” of possible words – all of the words in the dictionary of that length. Then as the user guesses letters, we’ll throw out all of the words that contain that letter so long as we have at least one word in our pool of possible words.

Once we have to give them the letter, we’ll choose a random word from our pool of words. At this point, it becomes a regular game of hangman. We will use the traditional 7 guesses (one for each body part drawn) as the maximum number of allowed guesses.

In each round of the game, the user should be shown the state of the word so far using an underscore to represent letters that haven’t been guessed, but filling in the positions with the correct letter if they were guessed. The user should be shown a list of letters that have already been guessed during the game and the number of incorrect guesses

that they have remaining. Finally, the hangman should be drawn (use the Hangman class method). The user should be asked to guess a letter or try to guess the entire word (use the Word class methods).

The game ends when the user either guesses the word correctly or the hangman is hung.

To make this program easier (hopefully), two classes have been provided:

- Word
- Hangman

In the Word class, the following methods are provided:

- constructor (actual name is __init__): It requires that an integer that represents the length of the word to be guessed. It must greater than 1 be sent. It creates a list of all possible words of that length.
- guess_letter: It requires a single letter be sent. It returns a list of integers that represent the positions (the first letter in the word would be position 0) of the letter in the word that is being guessed. If the letter isn't found in the word, it returns a list that contains only a -1. This method is the method that makes it an *Evil* Hangman, rather than just plain Hangman. If it can, it will throw out all words that contain the letter that is sent. Only if it must, does it decide on a single word and give the positions of the letter in it.
- guess_word: It requires a string be sent. If the Evil Hangman hasn't selected a single word from the list of possibilities, or if the guess doesn't match the single word that has been selected, it returns false. Otherwise, it returns true.
- get_word: Returns the first word in the list of words still being considered for the word to be guessed or the single word that actually was selected. It's useful for debugging purposes. It's also needed to be able to tell the user what the word was if they lost the game.

In the Hangman class, the following methods are provided:

- constructor (actual name is __init__): It constructs the gallows. It also initializes the number of guesses to 0.
- guessed_wrong: This method should be called ~~if~~ when the user guesses incorrectly. It updates the number of wrong guesses.
- print_hangman: It not only prints the gallows with the hangman, but updates the picture (adds body parts) based on the number of incorrect guesses.

Program Requirements:

For the code:

- Your program should have the correct comment block at the top (see last assignment)

- Use appropriate comments throughout the code. Be sure to include comments before each function definition explaining the interface (what must be sent, what is returned, and the purpose of the function).

- Make good use of whitespace. Be sure to include a blank line of space before each comment. Two lines of space between functions is nice, but one will suffice – whichever you choose, be consistent. Follow the Python programming style guide: https://www.python.org/dev/peps/pep-0008/#introduction

- Use the two classes provided

For the lab report, follow the Lab Report Format Guide and complete the following sections:

- Title Page
- Design – I recommend hand drawing the design. Make sure that it is neatly drawn and staple the hand-drawn document into your printed lab report. It does not need to be included in the electronic copy.
- Analysis and Conclusions
- Extra Analysis Questions:
  This section isn't shown in the Lab Report Format Guide. It is named Extra Analysis Questions, and in it, you should answer the following questions:
  - Did the Word class make your program easier to write or harder? How?
  - If you were designing the Word class, would you change anything? What would you change anything? What would you change about it?
  - Did the Hangman class make your program easier to write or harder? How?
  - If you were designing the Hangman class, would you change anything? What would you change about it?
  - What did you learn about classes, if anything, from completing this lab?
- Appendix B - code (Be sure that you copy and paste the code into your word document, don't take a screen shot of it and paste that – it ends up being too hard to read.)

Deliverables:

Electronic submission in myCourses:

- Code (due by the end of the class period)
- Lab Report (due by the start of the next time that your lab meets) -- either a Microsoft Word document or a pdf

Paper submission in class:

- Lab Report (due at the start of lab the next time that it meets)

Grading:

Task	**Points**
Lab Report	35 points
Title Page	5 points
Design	10 points
Analysis and Conclusions	10 points
Extra Analysis Questions	10 points
Code	65 points
Show the state of the word so far (and keep it updated)	10 points
Show the number of guesses left (and keep it updated)	10 points
Show the letters that have been guessed so far (and keep it updated)	10 points
Allow the user to guess a letter or word and update everything accordingly	10 points
Draw the hangman	5 points
Determine when (if) the user wins, congratulate him or her, and end the game	10 points
Determine when (if) the user loses, tell him or her so, and end the game	10 points