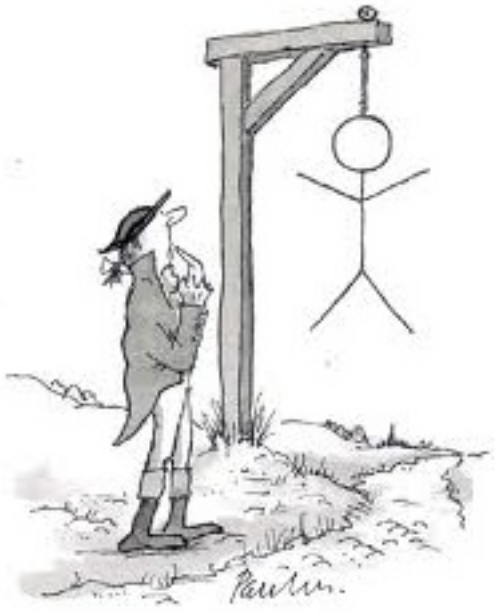
# CS150 - Assignment 7
# Hangman

*Due: Wednesday April 9 at the beginning of class*

For this lab, you will be implementing a text-based version of hangman. There are many variants of the program, so make sure to follow the specifications closely. You can also play a demo version of the program you will be implementing. See the lab prep for details on how to do this.

Make sure you read through this whole handout first before getting started. I've given an overview of how to proceed and some hints near the end.

If you want, you may work with a partner on this lab. If you do, you must both be there whenever you're working on the lab. Only one of you needs to submit the assignment, but make sure both your names on in the comments.



# 1 Playing the game

When the program starts it reads a file containing a list of possible words. I've placed a text file with CS words at:

http://www.cs.middlebury.edu/~dkauchak/classes/cs150/assignments/assign7/

(Right click on the file and select "Save link as..." to download. Remember, this file needs to be in the same directory as your program.)

After reading the file, the program randomly selects one of these words as the word that the user is trying to guess and then displays the following:

```
-----------------
Guessed letters:
Incorrect guesses left: *******
Word: _______
```

This display has three main components:

1. `Guessed letters:` Shows the letters the player has guessed so far. The letters will be separated by a space and should be *all capitalized*. When the game starts, no letters have been guessed, so this area is blank.
2. `Incorrect guesses left:` Shows the remaining number of wrong/incorrect guesses the player has before the game is over. Each remaining guess is represented as an '*'. In this version of the game, the player starts out with 7 incorrect guesses.
3. `Word:` This shows the word that the player is trying to guess. Each underscore (_) represents a letter in the word. The word above is 7 letters long. As the user plays, the correctly guessed letters will be filled in.

The program runs, by continually prompting the user for the next letter. When the user enters a letter, there are three possible cases:

- If the letter has been previously guessed, then the program lets the user know with a message like "Letter already guessed!" and prompts the user for another letter.
- If the letter has not been guessed before, but does not occur in the word to be guessed, then the number of incorrect guesses available is decremented by one and then the display is shown again, prompting the user for another letter.
- If the letter has not been guessed before and does occur in the word, then the underscored word is updated, with all occurrences of that letter in the appropriate place.

The following transcript shows all three of these different situations happening as the game is being played:

```
-----------------
Guessed letters:
Incorrect guesses left: *******
Word: ____

Guess a letter: e
-----------------
Guessed letters: E
Incorrect guesses left: *******
Word: _e__

Guess a letter: a
-----------------
Guessed letters: A E
Incorrect guesses left: *******
Word: _ea_

Guess a letter: r
-----------------
Guessed letters: A R E
Incorrect guesses left: ******
Word: _ea_

Guess a letter: s
-----------------
Guessed letters: A S R E
Incorrect guesses left: *****
Word: _ea_

Guess a letter: a
Letter already guessed!
Guess a letter: s
Letter already guessed!
Guess a letter: t
-----------------
Guessed letters: A S R E T
Incorrect guesses left: ****
Word: _ea_

Guess a letter:
```

The game ends when either the user runs out of guesses, in which case the word is shown:

```
The word was: heap
Better luck next time!
```

or if the user fills in all of the underscores in the word, then the user wins and the number of incorrect guesses is printed out:

```
Guessed letters: A H S R E
Incorrect guesses left: *****
Word: hea_

Guess a letter: p
You win!
The word was: heap
You guessed it with 2 incorrect guesses
```

## 2 Implementation requirements


http://www.cravingtech.com/best-cyanide-and-happiness-comic-strips.html

Besides following the game specification described above, your program **MUST** meet the following requirements:

- Your program must contain at least the following four functions:

- `read_words`: which takes a filename with words in it (one per line) as a parameter and returns a list of the words in the file.
- `set_to_string`: which takes a set as a parameter and returns the letters in the set as a string, with each letter capitalized and separated by a space.
- `insert_letter`: which takes three parameters: a letter, the current underscored word and the word the user is trying to guess. This function should return a new version of the underscored letter with all occurrences of the letter filled in based on the letters occurrence in the word to be guessed. For example:

  ```
  >>> insert_letter("a", "__n_n_", "banana")
  '_anana'
  ```

  Note that I've represented the underscored word and the word to be guessed as a string, but you may have some other representation, which is fine.
- `play_hangman`: which takes two parameters, the filename that contains the candidate words and the number of incorrect guesses the player gets. This function should then initiate the hangman game and continue to run until the game finishes.

- Your program should properly use the above four functions.
- You must use a `set` to store the guessed letters.
- Your program should immediately start playing the game with a default of 7 incorrect guesses when it is run.
- Your program should be able to handle upper-case or lower-case letters as input.
- The guessed letters shows should all be capitalized.

## 3 One path to implementation

There are many different ways to implement your program, but here is one suggestion:

- Write the `read_words` function and make sure that it correctly reads in the words in the file to a list. Don't forget to remove the end of line when reading in words.
- Write the `set_to_string` function. This should be very similar to the function you wrote for lab prep. *Hint:* `set`s are `iterable`, so you can iterate over the items in a `set` just like you can a `list`. Test to make sure it works properly.
- Write the `insert_letter` function. There are many ways to write this function, but one way is to build up the new string from scratch in a similar way we did for the encryption functions. For each letter in the new underscored word you're creating, you will either just be copying over what was in the underscores word before *or*, if that position corresponds to the letter you're looking for in the word to be guessed, you'll be using the correct letter. This isn't a lot of code, but think about how you can write this.

- Write `play_hangman` incrementally, testing each step before moving on to the next:
  - Start by having it read in the file, pick a random word and then display the underscored word. Note, when you're developing, feel free to print out other information (like the word you're supposed to be guessing) to help you debug your program. Just make sure to remove this information before you hand it in.
  - Add the user input and add the letter that the user guessed into the underscored word appropriately (your functions from above should be useful).
  - Add in functionality to keep track of the letters the user has guessed and to print out the letters guessed.
  - Add the functionality to make sure that the user continues to be prompted to enter a new letter as long as the letter input has already been guessed.
  - End the game when the user gets the right answer.
  - Keep track of the number of incorrect guesses and also end the game when the user runs out of guesses.
  - Print out the appropriate win/lose information depending on whether the user won or lost.

**Hints:** Here are some things to think about:

- How do the four functions fit into the program? Each of them should play a role in your program and will make your life easier and the program simpler.
- What information does your program need to keep track of (e.g. letters guessed, word to be guessed, underscored word, etc.)? Make sure you think about how you're storing each of these pieces of information.
- If you're confused about the behavior of your program, put in more print statements to print out variables, etc.
- Remember, if your program hangs when you run it, you probably have an infinite loop. If your run Wing in debug mode (two icons over from the green run arrow), you will see more information and it will often help you figure out where the infinite loop is.
- This program should not be a lot of code, but some of it can be tricky. Make sure you think about how you want to accomplish each step. Sometimes it helps to write down what you want to do in English (i.e. pseudocode), focusing on the logic, and then translate this into code.

## 4 Extra credit

As always, you may add any additions to the program for extra credit as long as it doesn't make the program easier. Make sure to put in comments at the top of the file indicating any extra credit that you implement. Here are some ideas that you might consider:

- Currently, we don't do any error checking to make sure that the user is entering valid entries, in particular a single character or a character a-z. Add these additional checks and reprompt the user to enter another character when this happens.
- Rather than using asterisks ('*') to represent how many guesses the person has left, add an ASCII generated stick figure based on how many they have left. You'll have to play with it but something like:

```
|-----|
|     O
|    \|/
|     |
|    / \
```

  is a good place to start (I'm sure you can do better, though :). In the real game, each time you get it wrong, you add a body part.

# 5 When you're done

Make sure that your program is properly commented:

- You should have comments at the very beginning of the file stating your name, course (including section number), assignment number and the date.
- Each function should have an appropriate *docstring*
- Other miscellaneous comments to make things clear

In addition, make sure that you've used good ***style***.

### Submission procedure

Submit your .py file online using the digital submission link on the course web page. You must have submitted it online before the beginning of class on Wed.

**Grading**

| | points |
|---|---|
| all words are read from the file | 1 |
| each time a random word is picked from the file | 1 |
| initial underscored word shown correctly | 1 |
| each guess is inserted appropriately in underscored word | 3 |
| guessed letters correctly displayed | 2 |
| guess letters are updated appropriately | 1 |
| guess letters uses a set | 2 |
| repeated requests from user on repeated letter | 3 |
| incorrect guesses updated appropriately | 2 |
| game ends correctly on win | 2 |
| game ends correctly on lose | 2 |
| handles lower and uppercase letters | 1 |
| program starts automatically on run | 1 |
| followed function specifications | 3 |
| Comments, style | 5 |
| lab prep | 2 |
| extra credit | 2 |
| total | 32 (+2) |


http://xkcd.com/804/
(All the other hangman comics seemed a bit too morbid...)