Objective: Practice designing a program, using files, and using built-in Python list and dictionary.

To start the homework: Download and extract the file hw4.zip. The hw4.zip file contains only a single file: dictionary.txt - a fairly complete dictionary file

For this homework you are to design and implement two programs:
- Part A Program: the “standard” word game Hangman, and
- Part B Program: an alternate version of Hangman where you can choose between Statistical or Evil Hangman.

I’ll describe the rules for these alternate versions later, but I want to stress that you only need to do one.

Part A: Standard Hangman Rules:
You are probably familiar with the game Hangman, but my rules are as follows:
1. One player (the computer) chooses a secret word, then writes out a number of dashes equal to the word length.
2. The other player (the human) begins guessing letters. Whenever she guesses a letter contained in the hidden word, the first player (the computer) reveals all instances 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.

Your assignment is to design and write a computer program to play Hangman. In particular, your program should do the following:
0. Read the file dictionary.txt, which contains over 120,000 words. You should use a Python dictionary to store the words such that each key is the word length with its corresponding value being a list of all word of that length (for example key 3 would have a value like ['dog', 'cat', 'the', ...])
1. Prompt the user for a word length. Reprompting as necessary until she enters a valid choice. For example, enters a number such that there's at least one word that's exactly that long, i.e., don't accept word lengths of -42 or 137, since no English words are that long. Plus, watch out for gaps in the dictionary. When the user specifies a word length, you will need to check that there are indeed words of that length in the dictionary. You might initially assume that if the requested word length is less than the length of the longest word in the dictionary, there must be some word of that length. Unfortunately, the dictionary contains a few “gaps.” The longest word in the dictionary has length 29, but there are no words of length 27 or 26. Be sure to take this into account when checking if a word length is valid.
2. Choose a secret word of the correct length at random
3. Prompt the user for a number of guesses, which must be an integer greater than zero. Don't worry about unusually large numbers of guesses – after all, having more than 26 guesses is clearly not going to help your opponent!
4. Play a game of Hangman using the standard Hangman rules, involves the following steps:
a) Print out how many guesses the user has remaining, along with any letters the player has guessed and the current blanked-out version of the word.
b) Prompt the user for a single letter guess, reprompting until the user enters a letter that she hasn't guessed yet. Make sure that the input is exactly one character long and that it's a letter of the alphabet.
c) Reveal the position of all occurrences of the guessed letter (if any) to the user.
d) If the word doesn't contain any instances of the guessed letter, subtract one from the remaining guess for the user.
e) If the player has run out of guesses, display the word that the computer “picked”.
f) If the player correctly guesses the word, congratulate her.

NOTE: Part of your grade will depend on the robustness and usability of your software, i.e., doesn’t crash, well-formatted prompts and quality of user interaction.

Design First: Since you're building this program from scratch, you'll need to do a bit of planning to figure out what the best data structures (Python lists and dictionaries, etc.) to use in the program, and how to functionally decompose the program. I want a brief design document for Part A hangman version. Your design document should include: structure chart and a couple sentences describing each functions -- see lab 1 description for an example.
Data Structures (CS 1520)    Homework #4      Due: Oct. 19 (Sat.) at 11:59 PM

**Part B:** Implement an alternate version of either Statistical or Evil Hangman (NOTE: You only need to do one)

**Statistical Hangman Rules:** (I made up this version for this assignment)
Same as standard Hangman, except the computer tries to select a statistically difficult word. The rules are as follows:

0. The human player selects the length of word they want to guess.
1. The computer analyzes all the words of that length to determine the overall frequencies for all 26 letter. The computer chooses the secret word of the selected length with the lowest sum of frequencies, then writes out a number of dashes equal to the word length. Suppose that an analysis of all four-letter words found frequencies of a’s to be 8.167%, h’s to be 6.094%, and t’s to be 9.056%, then the word “that” would have a sum of frequencies of 32.373 (= 9.056 + 6.094 + 8.167 + 9.056).
2. The other player (the human) begins guessing letters. Whenever she guesses a letter contained in the hidden word, the computer reveals all 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.

**Evil Hangman Rules:** (modified from Niffy Assignment description of Keith Schwarz at Stanford):
Evil Hangman “bends the rules” of Hangman (i.e., it cheats) to trounce its human opponent time and time again by deferring its choice of the actual word as long as possible. 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: “double” and “doable.” If the computer 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 the computer 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, the computer can claim that it picked the other word, and you will lose the game. That is, if you guess that the word is “double” the computer can pretend that it committed to “doable” the whole time, or vice-versa.

Let's illustrate this technique with a more complete example. Suppose that you are playing Evil Hangman against the computer and specify you want to guess a word of length four. The computer initially displays ‘-----’, but rather than committing to a secret word it instead compiles a list of every four-letter words in the English language. For simplicity, let's assume that English only has a few four-letter words, all of which are listed here:

```
ALLY BETA COOL DEAL ELSE FLEW GOOD HOPE IBEX
```

Now, suppose that you guess the letter 'E.' The computer now needs to tell you which letters in the word it “picked” are E's. Of course, it has not picked a word, and so it has multiple options about where to reveal the E's. Here's the above word list, with E's highlighted in each word:

```
ALLY BETA COOL DEAL ELSE FLEW GOOD HOPE IBEX
```

If you'll notice, every word in its word list falls into one of five “families:”

- -----, containing the words ALLY, COOL, and GOOD.
- -E--E, containing BETA and DEAL.
- --E-E, containing FLEW and IBEX.
- E--E, containing ELSE.
- -----E, containing HOPE.

Since the letters the computer reveals has to correspond to some word in its word list, it can choose to reveal any one of the above five families. There are many ways to pick which family to reveal – perhaps you want to steer your opponent toward a smaller family with more obscure words, or toward a larger family in the hopes of keeping your options open. In this assignment, in the interests of simplicity, I suggest the latter approach and always choose the largest of the remaining word families. In the above example, it means that the computer should pick the family -----E. This reduces your word list down to:

```
ALLY COOL GOOD
```

and since you didn't reveal any letters, you would tell your opponent that his guess of an ‘E’ was wrong. Continuing with this example, if you guess the letter O, then you would break your word list down into two families:

- -00-, containing COOL and GOOD.
- -----, containing ALLY.

The first of these families is larger than the second, and so you choose it, revealing two O's in the word, -00-, and reducing your list down to:

```
COOL GOOD
```

But what happens if your opponent guesses a letter that doesn't appear anywhere in your word list? For example, what happens if your opponent now guesses 'T'? This isn't a problem. If you try splitting these words apart into word families,
you'll find that there's only one family – the family ---- in which T appears nowhere and which contains both COOL and GOOD. Since there is only one word family here, it's trivially the largest family, and by picking it you'd maintain the word list you already had. There are two possible outcomes of this game. First, your opponent might be smart enough to pare the word list down to one word and then guess what that word is. In this case, you should congratulate him – that's an impressive feat considering the scheming the computer was up to! Second, and by far the most common case, your opponent will be completely stumped and will run out of guesses. When this happens, you can pick any word you'd like from your list and say it's the word that you had chosen all along. The beauty of this setup is that your opponent will have no way of knowing that you were dodging guesses the whole time – it looks like you simply picked an unusual word and stuck with it the whole way.

Tips, and Tricks
Here are some general tips and tricks that might be useful:

1. Letter position matters just as much as letter frequency. When computing word families, it's not enough to count the number of times a particular letter appears in a word; you also have to consider their positions. For example, “-EE-” and “-E-E” are in two different families even though they both have two E's in them. Consequently, representing word families as numbers representing the frequency of the letter in the word will get you into trouble.

2. Don't explicitly enumerate word families. If you are working with a word of length \( n \), then there are \( 2^n \) possible word families for each letter. However, most of these families don't actually appear in the English language. For example, no English words contain three consecutive U’s, and no word matches the pattern E-EE-EE--E. Rather than explicitly generating every word family whenever the user enters a guess, see if you can generate word families only for words that actually appear in the word list. One way to do this would be to scan over the word list, storing each word in a table mapping word families to words in that family.

Possible (Extra credit) Extensions to Evil Hangman:
The algorithm outlined in this handout is by no means optimal, and there are several cases in which it will make bad decisions. For example, suppose that the human has exactly one guess remaining and that computer has the following word list: DEAL TEAR MONK. If the human guesses the letter 'E' here, the computer will notice that the word family -E-- has two elements and the word family ---- has just one. Consequently, it will pick the family containing DEAL and TEAR, revealing an E and giving the human another chance to guess. However, since the human has only one guess left, a much better decision would be to pick the family ---- containing MONK, causing the human to lose the game.

There are several other places in which the algorithm does not function ideally. For example, suppose that after the player guesses a letter, you find that there are two word families, the family --E- containing 10,000 words and the family ---- containing 9,000 words. Which family should the computer pick? If the computer picks the first family, it will end up with more words, but because it revealed a letter the user will have more chances to guess the words that are left. On the other hand, if the computer picks the family ----, the computer will have fewer words left but the human will have fewer guesses as well. More generally, picking the largest word family is not necessarily the best way to cause the human to lose. Often, picking a smaller family will be better.

After you implement this assignment, take some time to think over possible improvements to the algorithm. You might weight the word families using some metric other than size. You might consider having the computer “ahead” a step or two by considering what actions it might take in the future. (This idea of looking ahead generalizes to an AI strategy called minimax, which can be shown to play a theoretically perfect game. Researching and implementing a minimax search can make your player substantially more powerful.)

If you implement something interesting, feel free to include it with your solution... I'd love to see what you've cooked up!

Submit all necessary files (dictionary.txt, designPartA.doc (or .pdf, .txt, ...), hangmanPartA.py and hangmanPartB.py) as a single zipped file (called hw4.zip) electronically at https://www.cs.uni.edu/~schafer/submit/which_course.cgi