Computer Science 135 (Spring 2016)
Duane A. Bailey
Laboratory 2
Building a Python Toolbox (due 11pm, next Tuesday)

Objective. To construct a small, well-constructed module.

This week we'll write our first non-trivial program. First, we'll construct a small module of tools for manipulating words from the dictionary. Each function will be fairly simple to write. When we're finished, we'll be able to support the development of a small game, called Bulls & Cows, a challenging game similar to Hangman or Mastermind.

Getting the Lab Starter. Before you begin, you need to check out a repository that contains a few starter files. This is best done when you're sitting in your `cs135` folder:

```
cd ~/cs135
```

then, check out the repository (replace `18xyz` with your CS username):

```
git clone ssh://18xyz@gala.cs.williams.edu/~cs135/18xyz/lab2.git lab2
```

When the repository has been checked out, you'll find this handout and several other informational files in the lab2 folder. Now, you're ready to begin.

Required Tasks.

1. Add the following small collection of functions to `wordTools.py`:
   (a) Your `canonical(s)` function from `lab1` that returns the lower-cased, sorted, space-free arrangement of the letters of `s`.
   (b) Write a function, `words(dfile)`, that returns a `list` of words found in a dictionary file, `dfile`. If you wish, you can have your `dfile` parameter default to `'/usr/share/dict/words'`. Make sure that you strip out any unnecessary whitespace (newlines, etc.) from the words as you read them in.
   (c) Write a function, `sized(l,n=4)`, that takes a word list `l` and a word length, `n`. It returns the words in the list that are exactly length `n`.
   (d) Write a function, `within(w,cs)`. This function takes a word `w` and a string of characters, `cs`. The result returns `True` if each letter of `w` occurs within `cs`. (Hint: you can check to see if a 1 character string, `c`, is in `cs` with the boolean test `c in cs`).
   (e) Write a function, `chars(l,cs)`, that returns the words of `l` whose letters are limited to `cs`. If you wish, you can have `cs` default to the lowercase letters of the alphabet.
   (f) Write a function, `rev(s)`, that returns string `s`, reversed.
   (g) Write a function, `pal(s)`, that returns `True` if a word `s` is the same written backwards or forwards.
   (h) Write a function, `isogram(w)`, that returns `True` if a word `w` is an *isogram*: it's composed of unique letters. (Hint: the size of `w` and the set composed of the letters of `w` are the same.)
   (i) Write a function, `isograms(l)`, that returns only those words of list `l` are isograms.

(j) Write a function, `match(a,b)`. This function takes two strings, `a` and `b`, that are the same length. It returns a pair of value (a *tuple*): the first of the pair is the number of characters of `a` that match characters of `b` *in the same position* (called *bulls*). The second integer is the number of non-bull letters of `a` that occur within `b`. For example,

```
>>> match('amherst','hamster')
(0,7)
>>> match('rail','rial')
(2,2)
```

2. Make sure your `wordTools` toolkit is a solidly built module:

   (a) The module starts with appropriate comments.

   (b) There is a triple-quoted docstring that appears at the top of the file that helps the user understand the purpose of this module. You can check this out by asking Python to show your documentation:

   ```
   pydoc3 wordTools
   ```

   (c) It contains a global variable `__all__` that is a list of strings that identify the functions that should be imported when you write

   ```
   from wordTools import *
   ```

   (d) Make sure that every function is documented with a docstring.

   (e) Thoroughly test each of the functions.

3. Now, let's write a word guessing game that makes use of these tools, called `guess.py`. The computer picks a random four letter isogram word and a human takes turns trying to guess the word. Each guess is scored for "bulls and cows". If there are 4 bulls, the human wins. Otherwise, the number of bulls and cows are reported and the game continues:

```
% python3 guess.py
Ok.  I'm thinking of a word.
What is your guess? mule
Matches: 2
Matched, wrong position: 0
What is your guess? muse
Matches: 3
Matched, wrong position: 0
What is your guess? fuse
You guessed it!
```

This is the basic functionality. For a bit more credit, you can make the game a little more advanced. Limit the number of turns. Give hints. Allow for words that are other lengths. Make sure you document your new features in the comments to `guess.py`.

When you're finished with `wordTools.py` and `guess.py`, commit and push your submission.