PROBLEM 1 : (*Myrmecophagous?* 24 points)

Write Python statement(s)/expression(s) to solve each of the problems below.

Consider the list `lista` below used to illustrate each problem.

```
lista = ["sloth", "aardvark", "pangolin", "pangolin", \
         "aardvark", "sloth", "sloth", "numbat","anteater"]
```

This list is used to illustrate the problems, but the code you write should work with any values stored in `lista`, don't write code that depends on any particular values stored in the list.

Part A (4 points)

Write Python code that stores in variable `uniq` the number of different values in `lista` — this is five in the example above since the five different strings in `lista` are 'sloth', 'aardvark', 'pangolin', 'numbat', 'anteater'.

Part B (4 points)

Write Python code that stores in variable `smalls` a list of of the strings in `lista` that have fewer than six letters in them. This would be `["sloth", "sloth", "sloth"]` in the example above. The words in `smalls` should be in the same order they appear in `lista`.

Part C (4 points)

Write Python code that stores in variable `most` the number of times the most frequently occurring string in `lista` occurs — this is three in the example above (for `"sloth"`).

Part D (12 points)

Write Python code that stores in variable `ordered` a list of the unique strings in `lista` in order from most frequently occuring to least frequently occuring. Ties should be broken alphabetically, e.g., `"aardvark"` appears before `"pangolin"` in `ordered` below (using `lista` as above) because they both occur twice but `"aardvark"` comes before `"pangolin"` alphabetically. Using `lista` above the values stored in `ordered` are:

```
ordered == ["sloth", "aardvark", "pangolin", "anteater", "numbat"]
```

because the number of occurrences of each of these is 3, 2, 2, 1, and 1, respectively. Note that `"anteater"` is alphabetically before `"numbat"` and both occur one time. (You'll earn more than half-credit if strings are ordered correctly by number of occurrences, but you don't break ties alphabetically.)

Part C (8 points)

In the Jotto program the function `get_guess` has the documentation shown below when the module `jottoModel.py` is snarfed/copied for the assignment.

```
def get_guess():
    """
    Choose a random word from _possiblewords, remove it from
    _possiblewords so it won't be guessed again, and return it.
    Update all state needed to indicate a guess has been made.
    """
```

Many students wrote lines of code similar to the following for `get_guess`:

```
global _guessed, _possiblewords, _gcount
_guessed = random.choice(_possiblewords)
_gcount += 1
return _guessed
```

First, briefly explain the purpose of each of the four lines of code in the context of playing Jotto with this code. Be sure you explain the purpose of each global variable and how it is used.

Then, briefly explain why the function shown above may often play the game correctly, but why it does **not** satisfy the documentation/comment. In doing so you should explain why this code could result in the computer guessing "break" several times in a row when the player is thinking of a secret word "baker" that the computer is trying to guess.

Part D (8 points)

Program

The program below generates the output on the right when run, showing that each simulated dice roll (a,b) occurs roughly the same number of times. However, the number of times each *sum* is rolled is different since there is only one way to roll a two: (1,1), but six ways to roll a seven: (1,6), (2,5), (3,4), (4,3), (5,2), (6,1). Write the function `get_totals` that returns a dictionary in which the key is a number from 2-12, inclusive, representing the sum of rolling two simulated dice; the corresponding value is the number of times the total occurs. The parameter to `get_totals` is the dictionary returned by `track_rolls`.

```
import random

def get_roll():
    return (random.randint(1,6), random.randint(1,6))

def track_rolls(repeats):
    d = {}
    for x in range(0,repeats):
        roll = get_roll()
        if roll not in d:
            d[roll] = 0
        d[roll] += 1

    for key in sorted(d.keys()):
        print key,d[key]

    return d

def main():
    d = track_rolls(10000)

if __name__ == "__main__":
    main()
```

For example, adding the line `p = get_totals(d)` in the function `main` and printing the contents of `p` should result in the output below given a dictionary storing information as shown on the right (the output won't necessarily be sorted by sum):

```
2 263
3 575
4 875
5 1121
6 1386
7 1627
8 1299
9 1152
10 888
11 531
12 283
```

(write code on next page)

Output from Running Program

```
(1, 1) 263
(1, 2) 283
(1, 3) 289
(1, 4) 261
(1, 5) 293
(1, 6) 270
(2, 1) 292
(2, 2) 297
(2, 3) 269
(2, 4) 297
(2, 5) 270
(2, 6) 254
(3, 1) 289
(3, 2) 284
(3, 3) 272
(3, 4) 245
(3, 5) 242
(3, 6) 295
(4, 1) 307
(4, 2) 246
(4, 3) 262
(4, 4) 273
(4, 5) 308
(4, 6) 302
(5, 1) 278
(5, 2) 301
(5, 3) 261
(5, 4) 254
(5, 5) 285
(5, 6) 278
(6, 1) 279
(6, 2) 269
(6, 3) 295
(6, 4) 301
(6, 5) 253
(6, 6) 283
```

```
def get_totals(rolld):
    """
    rolld is a dictionary in which (a,b) tuples are
    the keys, the corresponding value is the number of times
    (a,b) was rolled in a dice simulation. Return dictionary
    in which keys are unique values of a+b for (a,b) in
    rolld and value is number of times sum a+b occurs for
    each key
    """
```

PROBLEM 4 : (*Top Songs (10 points)*)

Rolling Stone magazine published a list of the top 500 songs of all time in 2004 and updated the list in 2010. A file stores the song, the artist, and the year the song was released as shown below.

```
Like a Rolling Stone:Bob Dylan:1965
(I Can't Get No) Satisfaction:The Rolling Stones:1965
Imagine:John Lennon:1971
What's Going On:Marvin Gaye:1971
...
Born to Run:Bruce Springsteen:1975
Help!:The Beatles:1965
```

Write the function `artists` that returns a dictionary in which the key is an artist (group, singer) and the corresponding value is a list of the song titles from that artist. For example, both of these entries would appear in the dictionary returned:

```
"The Beatles" : ["Hey Jude", "Yesterday", "I Want to Hold Your Hand", "Help!", ...]
"The Rolling Stones" : ["(I Can't Get No) Satisfaction", "Sympathy for the Devil", ...]
```

The parameter `filename` is the name of a file as shown above. Return the dictionary described.

```
def artists(filename):
    """
    return dictionary in proper format given parameter
    filename which has song information in proper format
    """


    f = open(filename)









    f.close()
```

Compsci 06, Spring 2011 Test Redux: Dictionary

You're given code that creates a dictionary of students for lab attendance. The keys are student names, the value for each student is a list of labs the student has attended, with the labs labeled by "*lab1*", "*lab2*", "*lab3*", and so on. For example:

```
labs = {
   "Fred"   : ["lab1", "lab2", "lab4", "lab6"],
   "Ethel"  : ["lab1", "lab2", "lab3", "lab4", "lab5", "lab6"],
   "Alex"   : ["lab3"],
   "Mary"   : ["lab4", "lab3", "lab1", "lab6"]
}
```

Write the function `labify` that has as a parameter a dictionary in the format above and which returns a dictionary in which each possible lab that appears in some value list in `attend` is the key. The value associated with this key is a list of strings: the people who attended that lab. Each list of strings should be sorted alphabetically.

For example, for the data above the call `labify(labs)` should return the dictionary diagrammed below:

```
{
    "lab1" : ["Ethel", "Fred", "Mary"],
    "lab2" : ["Ethel", "Fred"],
    "lab3" : ["Alex", "Ethel", "Mary"],
    "lab4" : ["Ethel", "Fred", "Mary"],
    "lab5" : ["Ethel"],
    "lab6" : ["Ethel", "Fred", "Mary"]
}
```

Write your code below:

```
def labify(attend):
    """
    attend is a dictionary in format name : [strings of labs attended],
    return a dictionary in which key is a string "labX" for all named
    labs that appear in values of attend, value for "labX" is
    alphabetized list of people attending that lab
    """
```

Compsci 06, Spring 2011, Files/Dictionaries, April 24

Name_____ net-id _____

Name_____ net-id _____

Name_____ net-id _____

The data from sites that rate faculty members might look like what's shown below: Each line shows a school, the name of a faculty member, the department, a rating on a 0-5 scale, and the IP address from which the rating was made (last names are shown for brevity, the full-name would be recorded).

```
Duke:Astrachan:Computer Science:4:152.3.250.1
Duke:Forbes:Computer Science:4:152.3.250.1
Duke:Astrachan:Computer Science:5:153.39.0.22
Stanford:Parlante:Computer Science:3:152.3.250.1
Stanford:Sahami:Computer Science:4:152.3.150.1
Stanford:Lattin:Business:5:153.39.0.22
Duke:Fullenkamp:Economics:4:152.3.250.1
Duke:Leachman:Econonmics:3:152.3.250.1
Duke:Bonk:Chemistry:4:152.3.250.1
```

Write a function that takes a filename for a file in the format shown and the name of a school (both strings) as parameters and which returns a list of `(name,department,rating)` three-tuples that are rank-ordered by average-rating, highest-to-lowest, for all professors in the school. If there are ties, break them by the alphabetic/lexicographical ordering of professor name.

```
def rankings(filename,school):
    """
    filename is name of file in format shown (string)
    school is a string

    return list of 3-tuples ordered by faculty rating, high-to-low
    """
```