Test 2: Compsci 06

Owen Astrachan

April 13, 2011

Name: ____

NetID/Login: ______________

Honor code acknowledgment (signature) __________________________________

	value	grade
Problem 1	24 pts.	
Problem 2	22 pts.	
Problem 3	18 pts.	
Problem 4	10 pts.	
TOTAL:	74 pts.	

This test has 10 pages, be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts 75 minutes. You should spend roughly one minute per point.

In writing code you do not need to worry about specifying the proper `import statements`. Don't worry about getting function or method names exactly right. Assume that all libraries and packages we've discussed are imported in any code you write.

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.)

PROBLEM 2 : (***Playing Dice With the Universe (22 points)***)

Part A (3 points)

This line appeared in the Hangman programs submitted by nearly every student where `words` is a list of strings:

```
secret = random.choice(words)
```

In a sentence or two explain what the purpose of this line was in the Hangman program.

Part B (3 points)

In the Hangman program a variable `display` was used by most students to represent what the user is shown before each guess, e.g., something like `_ a _ _ a _ _` for *pancake* if the user has guessed an 'a' correctly.

Two different initializations were used by students, both are shown below (students used one or the other, but not both in the same program).

```
slen = len(secret)
display = "_"*slen

# alternative

slen = len(secret)
display = ["_"]*slen
```

The first creates a string of underscores, the second creates a list of underscores, in each case of the appropriate length. Give a reason to prefer one initialization over the other. There is no correct answer here *per se*, but you will be evaluated on your justification/reasoning. Be brief.

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 3 : (***Follow the Money (18 points)***)

A list of political contributions in 2010 is stored in a Python list of tuples named `contribs` where each tuple stores four values: a string, the politician's name; a two-letter string, a US State abbreviation; an integer, the total of all donations to the politician; and a single letter 'R', 'D', or 'I' for Republican, Democrat, or Independent, respectively.

For example, the second line below shows that *Barbara Boxer* from CA (California) received $20,314,189 in donations and she is a Democrat.

```
contribs = [
   ("Jeff Greene","FL",23807119,"D")
   ("Barbara Boxer","CA",20314189,"D")
   ("Charles E Schumer","NY",17302006,"D")
   ("Harry Reid","NV",17213358,"D")
   ("Kirsten Elizabeth Gillibrand","NY",12900217,"D")
   ("Joseph A Sestak Jr","PA",11842844,"D")
   ("Linda Mcmahon","CT",46682270,"R")
   ("Sharron E Angle","NV",21470516,"R")
   ("John S Mccain","AZ",20077490,"R")
   ("Marco Rubio","FL",18251722,"R")
   ("Carly Fiorina","CA",17935605,"R")
   ("Scott P Brown","MA",17527893,"R")
]
```

As an example, to find the total of all contributions to all politicians we could use this Python expression:

```
   total = sum([c[2] for c in contribs])
```

Part A (4 points)

Write a Python expression or code to store in variable `rtotal` the total of all donations for all politicians who are Republicans.

Part B (4 points)

Write a Python expression or code that creates a list of strings: the names of all politicians who have received more than $15 million in donations, store this list in variable `heavies`.

Part C (10 points)

For this part of the problem new data is provided in addition to the list `contribs` in the previous parts.

Each of the politician's for whom data is found in the list `contribs` is the key in a dictionary `donors` whose value is a list of tuples, the tuples giving the contributions for each donor to that politician. For example, for Senator Rob Portman of Ohio part of his dictionary entry is shown below:

```
"Rob Portman" : [("L. Abbott", "4/22/10", 1900), ("V. Alpaugh", "12/29/09", 1000), ...
                 ("C. Klein", "11/23/10", 500)]
```

The tuples in the list of donors have three values: the name of the donor, the date of the donation, and the amount of money donated. In the data shown above, C. Klein gave $500.00 (to Rob Portman) on November 23, 2010 (`"11/23/10"`).

Write the function `small_donors` that returns a list of two-tuples. The first element of each two-tuple is the name of a politician that is in the list `contribs` (the first element of the tuples in list `contribs`, see previous page) the second element in the two-tuple is an integer, the number of *small* donors, those who gave less than $1000 to the candidate. For example, the list returned might look like this depending on the data passed to `small_donors`:

```
[("Rob Portman", 52), ("Carly Fiorina", 107), ... ("Barbara Boxer", 972)]
```

The tuples in the list can be in any order. Every politician in `contribs` is a key in `donors`.

```
def small_donors(contribs, donors):
   """
   contribs is a list described earlier containing 4-tuples,
   in which first element is politician's name

   donors is a dictionary: key is politician's name and value
   is list of 3-tuples as above (name,date,money)

   return list of 2-tuples (politician's name, #small donors)
   """
```

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()
```