Test 2 Analysis: Compsci 06

Owen Astrachan

April 25, 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 is about **analyzing** code that was used on the second test in spring 2011. For each question several responses are given. You are to judge the responses as correct or incorrect. You must give an explanation in addition to your response, explaining in words why you think the answer given is correct or why it is incorrect. Don't write more than a few sentences.

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'.

1.
```
uniq  = 0
repeats = []
for word in lista:
    if word not in repeats:
        uniq += 1
        repeats.append(word)
```

2.
```
uniq = 0
slist = sorted(lista)
for i in range(1,len(slist)):
    if slist[i] != slist[i-1]:
        uniq += 1
```

3.
```
uniq = len(set(lista))
```

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`.

1.
```
smalls = [x for x in list(a) if len(x) <= 6]
```

2.
```
smalls = []
for elt in lista:
    if len(elt) < 6:
        smalls.append(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"`).

1.
```
most = max([list.count(x) for x in lista])
```

2.
```
dc = {}
for a in lista:
    dc[a] = dc.get(a,0)+1
pairs = sorted(dc.items(), key=itemgetter(1), reverse=True)
most =  pairs[0][1]
```

3.
```
c = {}
for elt in lista:
   if elt not in c:
       c[elt] = 0
   c[elt] += 1
most = max(c.values())
```

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.)

1.
```
counts = sorted([(elt,lista.count(elt)) for elt in set(lista)])
sc = sorted(counts, key=itemgetter(1), reverse=True)
ordered = [e[0] for e in sc]
```

2.
```
d = {}
for elt in lista:
    d[elt] = d.get(elt,0) + 1
alph = sorted(d.items())
num = sorted(alph, key=itemgetter(1),reverse=True)
ordered = [elt[1] for elt in num]
```

3.
```
uni = sorted(set(lista))
data = sorted([ista.count(x) for x in uni], reverse=True)
ordered = []
for num in data:
    for word in uni:
        if lista.count(word)  == num:
            ordered.append(word)
```

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

Part A (3 points)

No analysis on this part

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.

Consider these three responses: **Analyze each one as a viable response. Be brief.**

1. The first choice, using a string, is better because printing a string will look better than printing a list – the latter has brackets, commas, etc.

2. The second choice is better because lists are mutable whereas strings are not, so changing one element of a list is easier than changing one element of a string — the latter isn't possible, instead a new string must be created.

3. Strings look better, but are immutable. Lists are mutable, but look ugly. The code to print a list in a nice way is much simpler to write than the code to create a new string with the letter 'e' in each blank spot it's supposed to be in depending on the word being checked. So use lists instead.

Part C (8 points)

No analytical part.

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
    """
```

1.
```
    d = {}
    for roll in rolld:
        s = sum(roll)
        if s not in d:
            d[s] = 0
        d[s] += rolld[roll]
    return d
```

2.
```
    d = {}
    info = [ [ key,rolld[key] ] for key in rolld]
    nlist = [ [r[0][0] + r[0][1], r[1]] for r in info]
    for sub in nlist:
        d[sub[0]] = d.get(sub[0],0) + sub[1]
    return d
```

3.
```
    d = {}
    for tup in d.items():
        s = sum(tup[0])
        tot = tup[1]
        if s not in d:
            d[s] = tot
        else:
            d[s] += tot
    return d
```

4.
```
    diction = {}
    for n in range(2,13):
        count = sum([rolld[key] for key in rolld if key[0]+key[1] == n])
        diction[n] = count
    return diction
```

5.
```
d = {}
for dat in rolld.items():
    r = sum(dat[0])
    d[r] = d.get(r,0) + dat[1]
return d
```

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.

Nothing to analyze

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`.

Nothing to analyze

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)
    """
```

1.
```
ret = []
names = [p[0] for p in contribs]
for name in names:
    s = 0
    for datum in donors[name]:
        if datum[2] < 1000:
            s += 1
    ret.append( (name,s) )
return ret
```

2.
```
d = {}
for can,nations in donors.items():
    smalls = [x for x in nations if x[2] < 1000]
    d[can] = len(smalls)
return d.items()
```