Practice Midterm 2

PROBLEM 1 : (***What is the output? (18 points)***)

Part A. What is the output of the following code segments?

Write the output to the right. Note that there is only output for the print statements.

OUTPUT

```
lst = [1,3,6]
print(min(lst))
lst.append(10)
print(max(lst))
lst.append(10)
lst.append(8)
lst.append(1)
print(lst.count(10))
print(lst.count(1))
# -------------------------
seta = set([2,1,2,2,6,1,6])
seta.add(2)
print(sorted(list(seta)))
seta.add(3)
print(sorted(list(seta)))
seta = set([2,1,2,2,6,1,6])
seta.remove(2)
print(sorted(list(seta)))
# -------------------------
seta = set([8, 3, 1])
setb = set([9, 8, 6])
print(seta^setb)
print(setb.union(seta))
print(seta&setb)
# -------------------------
d = {'J':2, 'M':7, 'Y':3, 'A':2}
print(sorted(d.keys()))
d['H'] = 5
d['Y'] = 7
print(sorted(d.keys()))
print(sorted(d.values()))
print(sorted(d.items()))
```

Part B. What is the output of the following code segment? **Write the output after the code segment**. Note that there is only output for the print statements.

```
nums = [60, 40, 60, 20, 20, 20, 30, 20, 40]
dictnums = {}
for n in nums:
    if n not in dictnums:
        dictnums[n] = 0
    dictnums[n] += 1

print dictnums.keys()
print dictnums.values()
print max(dictnums.values())
```

What is the output?

PROBLEM 1 : (*Basket Case (24 points)*)

You'll be asked to write code that references the list `fruits` below. The Python code you write should work with any values stored in the list `fruits`. *You can write one line of code or many for each of the tasks below.*

```
fruits = ["apple", "cherry", "kiwi", "tangerine", "grape", "mango",
          "nectarine", "lemon", "date", "pear", "plum", "papaya", "grape",
          "pear", "banana", "apple", "cantaloupe", "pear", "plum"]
```

Part A (4 points)

Write code to store in **int** variable `ecount` the number of *different* values stored in `fruits` that end with the letter 'e' (in the example this value is 6: `"apple"`, `"tangerine"`, `"grape"` `"nectarine"`, `"date"`, `"cantaloupe"`).

Part B (4 points)

Write code to store in **list** variable `bigfruit` the *different* values stored in `fruits` that are more than seven letters long (in the example this value is `['tangerine', 'nectarine', 'cantaloupe']`). The order of strings in `bigfruit` does not matter.

Part C (4 points)

Write code to store in **list** variable `ulist` those elements of `fruit` that contain the letter 'u'. For the list above this is `['plum', 'cantaloupe', 'plum']`. The same word containing a 'u' should be in `ulist` more than once if it appears more than once in `fruits`. The order of strings in `ulist` does not matter.

Part D (4 points)

Write code to store in **string** variable `last` the fruit that is alphabetically last in `fruit`. For the list above this is `'tangerine'`.

Part E (8 points)

Write code to store in **string** variable `mostfruit` the string that occurs most often in `fruits`. This is `"pear"` in the example above since it occurs three times, and no other fruit occurs more than twice. Assume the string that occurs most often is unique, don't worry about breaking ties.

PROBLEM 3 : (*A Clear APTitude (20 points)*)

Students are solving a new APT, in which the function to be written should return the netid of the student with the most APT points. APT data for each student is represented by the student's netid and the points earned on each APT as a single string, where name and points are separated by a colon character ':' — for example, the APT data could be represented as the list `scores` below.

```
scores = ["ola:10", "rcd:9", "ola:10", "rodger:8", "rodger:10",
          "ola:8", "rcd:8", "rodger:10"]
```

This shows that 'ola' and 'rodger' have submitted three APTs, but 'rcd' has submitted two APTs. Write function `maxPoints` which has a parameter `scores` that is a list of strings in the format shown here. Function `maxPoints` returns the netid of the student with the most APT points. If more than one student has the same maximal number of points, return the netid that is first alphabetically among the students with the maximal number of points.

For the data above, the student with netid 'rcd' has earned 17 points, 'rodger' has earned 28 points, and 'ola' has earned 28 points. Since 'ola' comes before 'rodger' alphabetically, the string 'ola' should be returned when `maxPoints` is called with the strings shown.

You will compare two solutions to this, you will not write your own solution to this APT.

```
def maxPoints(scores):
```

Two all-green solutions to this APT are given below. You'll be asked about the code in these solutions.

Solution A

```
    def maxPoints(scores):
1       mostNet=""
2       mostScore=0
3       d={}
4       for element in scores:
5           (net,score)=tuple(element.split(":"))
6           amount_int=int(score)
7           if net in d.keys():
8               d[net]+=amount_int
9           if net not in d.keys():
10              d[net]=amount_int
11      for akey in d.keys():
12          if d[akey]>mostScore:
13              mostScore=d[akey]
14              mostNet=akey
15          if d[akey]==mostScore and akey<mostNet:
16              mostNet=akey
17      return mostNet
```

Solution B

```
    def maxPoints(scores):
1       dict1 = {}
2       for item in scores:
3           data = item.split(":")
4           net = data[0]
5           score = int(data[1])
6           if net not in dict1:
7               dict1[net] = score
8           else:
9               dict1[net] += score
10      largest = max(dict1.values())
11      for key in sorted(dict1):
12          if dict1[key] == largest:
13              return key
```

Part A: (4 points)

Lines 1-10 in Solution A correspond to lines 1-9 in Solution B in terms of their purpose and what they create in variables `d` and `dict1`, respectively. In words, but making specific references to keys and values in dictionaries, describe the purpose of these sections and the dictionaries created. Your description should apply to both Solution A and to Solution B.

Part B: (6 points)

Lines 10-13 of Solution B correspond to lines 11-17 (which references lines 1 and 2) in Solution A. In solution A there are two if statements in the for-loop of lines 11-17, but there is only one if statement in the loop of lines 10-13 of Solution B. Explain how the approach taken in solution B makes it possible to have only one if statement compared to the approach taken in Solution A where two if statements are required. You should make explicit references to **both** lines 10 and 11 of Solution B in your description.

Part C: (6 points)

Suppose that rather than using strings of the format `"ola:10"` and `"rcd:8"`, the strings were formatted as `"10,ola"` and `"8,rcd"`, so that a comma separates name from score and the score comes before the name.

C.1 Identify the one line that must change in Solution A, and what that change is with this new format.

C.2 Identify which three lines must change in Solution B and what those changes are in each of the three lines.

Part D: (4 points)

Both solutions break ties for the maximal netid and return the netid whose name is first alphabetically if there is more than one netid with the maximal score. Suppose the netid whose name is last should be used to break ties. The code in Solution B can be changed by changing line 11 to: `for key in reversed(sorted(dict1))`

Explain what single line must change in Solution A and what that change is to break ties by returning the alphabetically last netid.

PROBLEM 4 : (*Visiting the Internet of Things (18 points)*)

A webserver log tracks which IP address visits each webpage on a site and the time of the visit. For example the file below shows several lines in the log file. Each line contains three items. In order these are the IP address, the URL visited, and the time (in this problem you can ignore the time). Each line of the log file represents an IP address visiting a specific webpage at the specific time recorded on the line.

```
152.3.140.1 foobar.org/wig 1446336000
152.3.140.1 foobar.org/food 1446336002
152.3.140.1 foobar.org/wig 1446336010
```

```
153.180.173.132 foobar.com/pig 1446336000
152.3.140.1 foobar.org/wig 1446336020
153.180.173.132 foobar.com/dog 1446336100
67.180.241.1 foobar.com/cat 1446336000
152.3.140.1 foobar.org/wig 1446336200
153.180.173.132 foobar.com/cat 1446336202
```

The IP address `153.180.173.132` visits three unique webpages, these have URLs `foobar.com/pig`, `foobar.com/dog`, and `foobar.com/cat`. The IP address `152.3.140.1` visits only two unique web pages, but makes five total visits as recorded in this log since it appears on five lines.

Part A (4 points)

Complete function `logData` that reads a data file in the format above and that returns a dictionary in which each unique IP address is a key and the total number of webpages visited is the value associated with the key. For the file shown above, the dictionary returned is

```
{'153.180.173.132': 3, '152.3.140.1': 5, '67.180.241.1': 1}
```

Complete the function below:

```
def logData(filename):
    f = open(filename)
    d = {}
    for line in f:
        data = line.strip().split()
        # write code here









    f.close()
    return d
```

Part B (6 points)

Write a function `getUniqueVisits` that takes a filename of a log file as a parameter, just as the function `logData` does. Function `getUniqueVisits` returns a dictionary in which each distinct URL in the log file is a key, and the value associated with the key is a list of unique IP addresses that visited the URL (*unique means that each value in the list appears once*).

For the logfile shown above the dictionary returned is shown below. The order of the IP addresses in each list doesn't matter. Note that IP address `152.3.140.1` appears once in the list associated with key `foobar.org/wig` although that IP address visited the URL four times as recorded in the log file.

```
{
 'foobar.com/pig': ['153.180.173.132'],
 'foobar.com/dog': ['153.180.173.132'],
 'foobar.org/food': ['152.3.140.1'],
 'foobar.com/cat': ['153.180.173.132', '67.180.241.1'],
 'foobar.org/wig': ['152.3.140.1']
}
```

Complete the function below.

```
def getUniqueVisits(filename):
```

Part C: (8 points)

Complete function `mostDifferent` that returns the IP address that visits the most different webpages as recorded in a webserver logfile such as the example shown at the beginning of this problem. For that file, `mostDifferent` should return 153.180.173.132, as that IP address visits 3 unique web pages, which is more than any other IP address. The parameter to `mostDifferent` is the name of the logfile.

```
def mostDifferent(filename):
```