Test 2 : Compsci 102/Neuroscience 104

Owen Astrachan
Shani Daily
Jennifer Groh

April 6, 2020

Name: _____ (1 point)

NetID/Login: _____ (1 point)

Community standard acknowledgment (signature) _____

	value	grade
Problem 1	30 pts.	
Problem 2	20 pts.	
Problem 3	10 pts.	
TOTAL:	60 pts.	

This test has 11 pages and a two-page APT-handout, be sure your test has them all. You'll use either this document, or a separate document to record answers. Make sure the first page has your netid and name stored in the indicated area on the first page when you turn in your test in Sakai.

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

PROBLEM 1 : (*Stun Chaos Tipi (24 points)*)

You'll be asked to write code that references the list `nuts` below. The Python code you write should work with any values stored in the list `nuts`. *You can write one line of code or many for each of the tasks below.* In particular, you can often write a list comprehension that can be used to solve the problem in one line, but you can also write many lines for each problem.

```
nuts = ["cashew", "filbert", "chestnut", "coconut", "macadamia",
        "peanut", "pecan", "peanut", "peanut", "macadamia", "cashew"]
```

Recall that `len(nuts)` has the value 11, and that `len(set(nuts))` has the value 7.

Part A (4 points)

Write code to store in variable `unic` the number of *different* values stored in `nuts` that begin with the letter 'c' (in the example this value is 3 since the unique strings that start with 'c' are `"cashew"`, `"chestnut"`, `"coconut"`).

Part B (4 points)

Write code to store in `solo` the number of strings in `nuts` that occur exactly once, this would be four in the example above: `"filbert"`, `"chestnut"`, `"coconut"`, and `"pecan"` each occur once. Recall that `nuts.count("pecan")` has the value 1.

```
nuts = ["cashew", "filbert", "chestnut", "coconut", "macadamia",
        "peanut", "pecan", "peanut", "peanut", "macadamia", "cashew"]
```

Part C (4 points)

Write code to store in `truenut` a list of the (unique) strings in `nuts` that end with `'nut'`. In the example above this is `[chestnut,coconut,peanut]` (order of words doesn't matter)

```
nuts = ["cashew", "filbert", "chestnut", "coconut", "macadamia",
        "peanut", "pecan", "peanut", "peanut", "macadamia", "cashew"]
```

Part D (6 points)

Write code to store in `nuttiest` the string that occurs most often in `nuts`. This is `"peanut"` in the example above. Assume the string that occurs most often is unique, don't worry about breaking ties.

```
nuts = ["cashew", "filbert", "chestnut", "coconut", "macadamia",
        "peanut", "pecan", "peanut", "peanut", "macadamia", "cashew"]
```

Part E (6 points challenge, not required)

Write code to store in `freqs` the unique strings in `nuts` sorted by number of times each string occurs with the least frequently occuring string first. For strings that occur the same number of times, break ties alphabetically. For example, for the list `nuts` above the value stored in `freqs` is `["chestnut", "coconut", "filbert", "pecan", "cashew", "macadamia", "peanut"]`.

```
nuts = ["cashew", "filbert", "chestnut", "coconut", "macadamia",
        "peanut", "pecan", "peanut", "peanut", "macadamia", "cashew"]
```

PROBLEM 2 : (*Songs in the Key of Life (20 points)*)

A data files stores purchase from an online music store like iTunes, part of such a data file, `purchases.txt`, is shown below.

```
ola@cs.duke.edu,Light My Fire,The Doors,0.99
rcd@yahoo.com,Ice Ice Baby,Vanilla Ice,0.99
ola@cs.duke.edu,Your Smiling Face,James Taylor,0.99
rbo@gmail.com,The Cave,Mumford and Sons,1.29
rcd@yahoo.com,Raise Your Glass,P!ink,1.29
rbo@gmail.com,Sleepyhead,Passion Pit,0.99
rbo@gmail.com,Landfill,Daughter,0.99
ola@cs.duke.edu,Raise Your Glass,P!ink,1.29
rbo@gmail.com,Raise Your Glass,P!ink,1.29
```

Each line has four strings separated by commas as shown: email address, track purchased, artist/group name, and price.

The code below, when run on the data file above, generates the output shown if the data file is named `"purchases.txt"`.

```
41  def file2list(fname):
42      f = open(fname)
43      ret = []
44      for line in f:
45          line = line.strip().split(",")
46          ret.append(line)
47      f.close()
48      return ret
49
50  if __name__ == '__main__':
51      fname = "purchases.txt"
52      data = file2list(fname)
53      for each in data:
54          print(each)
```

Output generated:

```
['ola@cs.duke.edu', 'Light My Fire', 'The Doors', '0.99']
['rcd@yahoo.com', 'Ice Ice Baby', 'Vanilla Ice', '0.99']
['ola@cs.duke.edu', 'Your Smiling Face', 'James Taylor', '0.99']
['rbo@gmail.com', 'The Cave', 'Mumford and Sons', '1.29']
['rcd@yahoo.com', 'Raise Your Glass', 'P!ink', '1.29']
['rbo@gmail.com', 'Sleepyhead', 'Passion Pit', '0.99']
['rbo@gmail.com', 'Landfill', 'Daughter', '0.99']
['ola@cs.duke.edu', 'Raise Your Glass', 'P!ink', '1.29']
['rbo@gmail.com', 'Raise Your Glass', 'P!ink', '1.29']
```

(questions on next pages)

Part A (2 points)

Explain why quotes are shown for the price in each list printed, e.g., why the string '0.99' appears in the first list and not the float 0.99.

Part B (2 points)

Explain why a CSV reader, e.g., from the csv module/library, might be a better choice than using `.split` as shown in the code. In your answer consider songs with titles like `"Peace, Love, and Understanding"`.

Part C (8 points)

Write code in the function `purchases` to return a dictionary so that the code below generates the output shown (the order of the lines may be different, the information on each line should be the same, representating the information in the file shown on the previous page).

```
ola@cs.duke.edu $3.27
rcd@yahoo.com   $2.28
rbo@gmail.com   $4.56
```

Code generating this output:

```
58 ▶ if __name__ == '__main__':
59       fname = "purchases.txt"
60       data = file2list(fname)
61       info = purchases(data)
62       for key in info:
63           print("{}\t${:.2f}".format(key,info[key]))
```

```
def purchases(data):
    """
    data is a list of lists,
    each list is a list of strings representing
    one line of information "email", "song", "artist", "price"

    returns dictionary: key is email address, value total spent
    """
    #you write code here
```

Part D (8 points)

Suppose a function `playlists` takes a list returned by `file2list` as a parameter and returns a dictionary in which the key is an email address and in which the corresponding value is a list of songs purchased by the person with the given email address. For the data file shown the call `playlists(file2plist("purchases.txt"))` would return this dictionary:

```
{'rcd@yahoo.com': ['Ice Ice Baby', 'Raise Your Glass'],
 'ola@cs.duke.edu': ['Light My Fire', 'Landfill', 'Raise Your Glass'],
 'rbo@gmail.com': ['The Cave', 'Sleepyhead', 'Landfill', 'Raise Your Glass']
}
```

Given the existence of this function `playlists`, which you **do not write**, write the function `topsong` whose *parameter is a dictionary in the format described, that is as as returned by* `playlists`. Function `topsong` returns a string: the song purchased by more people than any other song. Don't worry about ties.

```
def topsong(songd):
    """
    songd is dictionary: key is email address
    corresponding value is list of song titles purchased
    by person with email address

    returns: song bought by most people
    """
```

PROBLEM 3 : (*Adenosine Triphosphate (20 points)*)

Part A (10 points)

The write-up for the APT *ContestWinner* is at the end of this test. You're given code below that uses the following idea to solve the problem, the code is only partially complete. You'll loop over each value in the list, each value representing a contestant, as the code processes the elements of `events` in order the code will:

1. Maintain a dictionary named `d` in which the key is an `int` value representing a contestant. The corresponding value for each key is the number of times the contestant has occurred in the list (as the loop progresses).
2. As the dictionary `d` is updated, update a variable named `winner` that is the value of the contestent with the most problems solved. Initially `winner` is the first value in the list. If the value just updated in the dictionary is greater than `d[winner]`, then update `winner`.

For example, in the third example of the APT writeup, both 123 and 456 occur three times, so the maximal value is three. As the list `[123,123,456,456,456,123]` is processed the count of 456 becomes three when the count of 123 is still two. So 456 will be the value of `winner` when the loop processing all elements finishes executing.

Complete the code below by adding fewer than eleven lines of code so that the function is correct (all green) and the idea above is used to get all-green.

```
def getWinner(events):
    """
    events is a list of int values
    returns the int that occurs the most frequently
    and return the first int if more than one
    """
    d = {}
    winner = events[0]
    for elt in events:









    return winner
```

APT: ContestWinner

Problem Statement

Specification

```
filename: ContestWinner.py

def getWinner(events):
    """
    return int based on events,
    a list of int values
    """

    # you write code here
    return 0
```

Exactly one million contestants, numbered 1 through 1,000,000, took part in a programming contest. The rules of the contest are simple: the winner is the contestant who solves the most tasks. If there are more contestants tied for most tasks solved, the winner is the one who was the first to solve the most tasks.

During the contest the judges keep a log of all accepted solutions. You are given this log as `events`, a list of int values. The i-th element of events is the number of the contestant who submitted the i-th accepted solution (both indices are 0-based).

For example, if `events = [4, 7, 4, 1]`, this is what happened during the contest:

- Contestant 4 solved her first task.
- Contestant 7 solved his first task.
- Contestant 4 solved her second task.
- Contestant 1 solved his first task.

Compute and return the number of the contestant who won the contest.

Constraints

- `events` will contain between 1 and 50 elements, inclusive.
- Each element of `events` will be between 1 and 1,000,000, inclusive.

Examples

1.
```
events = [4,7,4,1]

Returns: 4
```

   Example from the problem statement.

2.
```
events = [10,20,30,40,50]
```

```
Returns: 10
```

3. ```
   events = [123,123,456,456,456,123]

   Returns: 456
   ```

4. ```
   events = [1,2,2,3,3,3,4,4,4,4]

   Returns: 4
   ```

This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. ©2010, TopCoder, Inc. All rights reserved.