Assignment 1 Solution

Yunbing Weng, 400158853

January 30, 2019

Introductory blurb.

1 Testing of the Original Program

Sadly, I missed a lot of beggining of the year and participated in Code to win competition in Waterloo and also Hack Concordia so didnt have time to finish my own programm

2 Results of Testing Partner's Code

In order to test the program, I created a ordered list of 1000 strudents, where each next student has a little bit less GPA, using formula: next students GPA = random number from 99.82gender and choise is also assigned randomly also, during initialization of the list, we count the number of students of each gender and sum of their GPA to get the average one.

To test sort, we created clone of test list and random shuffle it after sorting, we should get the same list, wich is easily to check

For average GPA test, we call the function for male and female and compare with our generated result. We also can see that the difference is smaller then 1E-10, wich is normal error for float

For allocation we assigned capacity for each faculty as 150 and get the last student in my list the right of free choice, then, we just need to check that: 1. We didnt exeeded the 150 capacity 2. We have assigned special student somewhere 3. In each faculty all students are sorted (because original list was sorted)

Allocation didnt work on my computer because of the feature of dict (On my computer we cant directly use result of get() function and gives error: TypeError: 'NoneType' object is not subscriptable)

3 Discussion of Test Results

It would probably work in some enviroment without type restriction

3.1 Problems with Original Code

3.2 Problems with Partner's Code

Described in above sections

4 Critique of Design Specification

It is pretty well described for me

5 Answers to Questions

(a) I would instead of g for gender, I would make it more general, then we can find average not only by gender, also by prefered faculty or by the name. Just instead of two argument, we would need one mor, but thats okay. For sort we can do something like sort in c++, so have choice to give a function with ording criteria

(b) answer

(c) The ReadAllocationData is just program for input, and also in your requirement we can use our own input file, so in this case is hard to make general test for it. Also, if I understand it right, to test something, we need the right answer first. Here the only thing we can check is the structure of our dictionaries.

F Code for ReadAllocationData.py

```
## @file ReadAllocationData.py
#  @wengy12
#  @brief
#  @01/25/2019, I'm really sorry for late code :(

def readStdnts(s):
    file = open(s, "r") #input format will be "macid,fname,lname,gender,gpa,choice,choice,choice/n"
    inp = file.readlines()
    lst = []
    for i in inp:
        elm = i.split(',')
        elm[7].pop() #delete the last \n element
        newstd = {"macid": elm[0], "fname": elm[1], "lname": elm[2], "gender": elm[3], "gpa":
            float(elm[4]), "choice": [elm[5], elm[6], elm[7]]}
        lst.append(newstd);
    return lst

def readFreeChoice(s):
    file = open(s, "r") #input format will be "macid/n"
    inp = file.readlines()
    return inp

def readDeptCapacity(s):
    file = open(s, "r") #input format will be "faculty_name,capacity/n"
    inp = file.readlines()
    lst = []
    for i in inp:
        elm = i.split(',')
        elm[1].pop() #delete the last \n element
        newelm = {"dept": elm[1]}
        lst.append(newelm);
    return lst
```

G Code for CalcModule.py

```
## @file CalcModule.py
#  @author janzej2
#  @brief Three functions for calculating/allocating students to their chosen programs.
#  @date 01/17/19

#import Read AllocationData as well as operator (used for sorting list of dictionaries)
import operator
from ReadAllocationData import *

## @brief This function sorts a list of dictionaries from lowest to highest GPAs
# @param S A list of students, each represented as a dictionary.
# @return A list of students, in order from highest to lowest GPAs.
#sort function inspired by
    https://www.geeksforgeeks.org/ways-sort-list-dictionaries-values-python-using-lambda-function/
def sort(S):
    return sorted(S, key = lambda i: i['gpa'], reverse=True)

## @brief This function finds the average GPA of a set of students (based on gender).
# @param L A list of dictionaries created by readStdnts(s).
# @param g A string representing the gender ("male" or "female").
# @return The average GPA of male or female students as a float.
def average(L, g):
    average = 0
    count = 0
    for i in range(len(L)):
        if L[i].get("gender") == g:
            count += 1
            average += L[i].get("gpa")
    if count != 0:
        return average/count
    else:
        return 0

## @brief This function allocates students to their program choices based on a number of
# factors.
# @details The function takes a list of all students, students with free choice and a
# dictionary containing the capacities of each department. If the student has free choice,
# they are automatically added to their chosen program (it is assumed that there will always
# be space in a given program for free choice students, see ReadAllocationData.py for full
# assumption list) while otherwise, they are allocated based on capacity.
# @param S A list of dictionaries of students, created by readStdnts().
# @param F A list of students with free choice (created by readFreeChoice()).
# @param C A dictionary of department capacities (created by readDeptCapacity()).
# @return A dictionary formatted with the format 'program' : [student, student...]
# for each potential program.
def allocate(S, F, C):
    civ, chem, elec, mech, soft, matls, engphys = ([] for i in range(7))
    #create final dictionary to return/index in the future
    final = {'civil' : civ, 'chemical' : chem, 'electrical' : elec, 'mechanical' : mech, 'software' :
        soft, 'materials' : matls, 'engphys' : engphys}
    #allocate free choice students first
    for i in range(len(F)):
        for j in range(len(S)):
            #find macid in student list and take index position
            if F[i] == S[j].get("macid"):
                break
        #add student to first choice program
        (final.get((S[j].get("choices"))[0])).append(S[i])
        #remove student from main student list so they aren't allocated twice
        del S[j]
    S = sort(S)
    #define previous GPA in case of duplicates
    previous = -1
    for i in range(len(S)):
        #only allocate those with gpa greater than 4.0. see assumptions in ReadAllocationData.py
        if S[i].get("gpa") < 4.0:
            break
        first = (S[i].get("choices"))[0]
        second = (S[i].get("choices"))[1]
        third = (S[i].get("choices"))[2]
        #check capacities and allocate accordingly
        if len(final.get(first)) < C.get(first) or (len(final.get(first)) == C.get(first) and previous
            == S[i].get('gpa')):
            (final.get(first)).append(S[i])
        elif len(final.get(second)) < C.get(second) or (len(final.get(second)) == C.get(second) and
            previous == S[i].get('gpa')):
```

```
            (final.get(second)).append(S[i])
        else:
            (final.get(third)).append(S[i])
        previous = S[i].get('gpa')
    return final
```

H Code for testCalc.py

```
## @file testCalc.py
#  @wengy12
#  @brief
#  @01/29/2019

import random
import CalcModule

gen = ["male", "female"]
choices = ["software", "civil", "chemical", "electrical", "mechanical", "materials", "engphys"]
testlst = [{"macid": 400158850, "fname": "Peter", "lname": "The Best", "gender": "male", "gpa": 12,
    "choice": [choices[0], choices[1], choices[2]]}]
male_gpa = 12
male_num = 1
female_gpa = 0
female_num = 0

for i in range(1, 1000):
    random.shuffle(choices)
    prev_gpa = testlst[i-1].get("gpa")
    if prev_gpa is not None:
        prev_gpa = float(prev_gpa)
        newstd = {"macid": 400158850+i, "fname": "liz("+str(i)+")", "lname": "goodman("+str(i)+")",
            "gender": gen[random.randint(0,1)], "gpa": (random.random()/500+0.998)*prev_gpa,
            "choice": [choices[0], choices[1], choices[2]]}
        g = newstd.get("gender")
        gp = newstd.get("gpa")
        if g is not None and gp is not None:
            if g == "male":
                male_gpa += gp
                male_num += 1
            else:
                female_gpa += gp
                female_num += 1
        testlst.append(newstd)

male_gpa /= male_num
female_gpa /= female_num

newlst = testlst
random.shuffle(newlst)
CalcModule.sort(newlst)
if newlst != testlst:
    print("error in sorting")
else:
    print("sorting is good")

random.shuffle(newlst)
print("male average gpa is ", male_gpa)
print("calculated male agpa is ", CalcModule.average(newlst, "male"))
print("female average gpa is ", female_gpa)
print("calculated female agpa is ", CalcModule.average(newlst, "female"))

capacity = {"software": 150, "civil": 150, "chemical": 150, "electrical": 150, "mechanical": 150,
    "materials": 150, "engphys": 150}
free_choice = [400159849]
res = CalcModule.allocate(newlst, free_choice, capacity)
fc = 0
now = 0
for i in res:
    for j in range(i):
        if i[j].get("macid") == 400159849:
            fc += 1
            if (j != 0):
                print("error")
            continue
        if now > i[j]:
            print("error")
        else:
            now = i[j]
    if len(i) > 150:
        print("error")
    now = 0
```

I Code for Partner's CalcModule.py

```
## @file CalcModule.py
#  @author janzej2
#  @brief Three functions for calculating/allocating students to their chosen programs.
#  @date 01/17/19

#import Read AllocationData as well as operator (used for sorting list of dictionaries)
import operator
from ReadAllocationData import *

## @brief This function sorts a list of dictionaries from lowest to highest GPAs
# @param S A list of students, each represented as a dictionary.
# @return A list of students, in order from highest to lowest GPAs.
#sort function inspired by
    https://www.geeksforgeeks.org/ways-sort-list-dictionaries-values-python-using-lambda-function/
def sort(S):
    return sorted(S, key = lambda i: i['gpa'], reverse=True)

## @brief This function finds the average GPA of a set of students (based on gender).
# @param L A list of dictionaries created by readStdnts(s).
# @param g A string representing the gender ("male" or "female").
# @return The average GPA of male or female students as a float.
def average(L, g):
    average = 0
    count = 0
    for i in range(len(L)):
        if L[i].get("gender") == g:
            count += 1
            average += L[i].get("gpa")
    if count != 0:
        return average/count
    else:
        return 0

## @brief This function allocates students to their program choices based on a number of
# factors.
# @details The function takes a list of all students, students with free choice and a
# dictionary containing the capacities of each department. If the student has free choice,
# they are automatically added to their chosen program (it is assumed that there will always
# be space in a given program for free choice students, see ReadAllocationData.py for full
# assumption list) while otherwise, they are allocated based on capacity.
# @param S A list of dictionaries of students, created by readStdnts().
# @param F A list of students with free choice (created by readFreeChoice()).
# @param C A dictionary of department capacities (created by readDeptCapacity()).
# @return A dictionary formatted with the format 'program' : [student, student...]
# for each potential program.
def allocate(S, F, C):
    civ, chem, elec, mech, soft, matls, engphys = ([] for i in range(7))
    #create final dictionary to return/index in the future
    final = {'civil' : civ, 'chemical' : chem, 'electrical' : elec, 'mechanical' : mech, 'software' :
        soft, 'materials' : matls, 'engphys' : engphys}
    #allocate free choice students first
    for i in range(len(F)):
        for j in range(len(S)):
            #find macid in student list and take index position
            if F[i] == S[j].get("macid"):
                break
        #add student to first choice program
        (final.get((S[j].get("choices"))[0])).append(S[i])
        #remove student from main student list so they aren't allocated twice
        del S[j]
    S = sort(S)
    #define previous GPA in case of duplicates
    previous = -1
    for i in range(len(S)):
        #only allocate those with gpa greater than 4.0. see assumptions in ReadAllocationData.py
        if S[i].get("gpa") < 4.0:
            break
        first = (S[i].get("choices"))[0]
        second = (S[i].get("choices"))[1]
        third = (S[i].get("choices"))[2]
        #check capacities and allocate accordingly
        if len(final.get(first)) < C.get(first) or (len(final.get(first)) == C.get(first) and previous
            == S[i].get('gpa')):
            (final.get(first)).append(S[i])
        elif len(final.get(second)) < C.get(second) or (len(final.get(second)) == C.get(second) and
            previous == S[i].get('gpa')):
```

```
            (final.get(second)).append(S[i])
        else:
            (final.get(third)).append(S[i])
        previous = S[i].get('gpa')
    return final
```

J Makefile

```
PY = python
PYFLAGS =
DOC = doxygen
DOCFLAGS =
DOCCONFIG = docConfig

SRC = src/testCalc.py

.PHONY: all test doc clean

test:
	$(PY) $(PYFLAGS) $(SRC)

doc:
	$(DOC) $(DOCFLAGS) $(DOCCONFIG)
	cd latex && $(MAKE)

all: test doc

clean:
	rm -rf html
	rm -rf latex
```