Assignment 1 Solution

James Zhang zhany111

January 25, 2019

1 Testing of the Original Program

```
from ReadAllocationData import *
from CalcModule import *

def fine(a,b):
    if (abs(a-b)/a)<(1.0/100.0):
        return True
    return False

print(fine(average(readStdnts('student.txt'),'female'),5.16))

ans = {'civil': [{'macid': 'zhany111', 'fname': 'james', 'lname': 'zhang',
'gender': 'male', 'gpa': 8.0, 'choices': ['civil', 'electrical', 'materials']
'chemical': [], 'electrical': [{'macid': 'liy200', 'fname': 'iris', 'lname':
 'gender': 'female', 'gpa': 10.0, 'choices': ['electrical', 'materials', 'che
 'mechanical': [], 'software': [{'macid': 'jiaoz666', 'fname': 'dump',
 'lname': 'li', 'gender': 'female', 'gpa': 4.3, 'choices': ['software',
 'engphys', 'chemical']}], 'materials': [{'macid': 'fnndp123', 'fname':
 'tie', 'lname': 'dan', 'gender': 'male', 'gpa': 8.2, 'choices': ['electrical
 'materials', 'chemical']}, {'macid': 'ojbk123', 'fname': 'an', 'lname': 'li'
 'gender': 'male', 'gpa': 6.5, 'choices': ['electrical', 'materials', 'chemic
, 'engphys': []}

print(allocate(readStdnts('student.txt'),readFreeChoice('free.txt'),
  readDeptCapacity('dept.txt'))==ans)
```

The above code shall print the following result:

True
True

The first True represents the correctness of the average(L, G) function and also the readStdnts(s) fuction. I test the result by comparing the output of the funciton with the actualy average calculated by the calculator. I used a new function fine(a,b) to compare only up to two decimal places of the numbers since the float variable returned from average(L, G) has almost infinity decimal places.

The second True represents the correstness of the alloacate(S, F, C), readStdnts(s), readFreeChoice(s) and readDeptCapacity(s). I tested the alloacate function using the result of the rest other fuctions so that I do not have to test them individually. I used a variable called ans as the result of the correct answer which found by myself using the test files I wrote.

Therefore, I used the least space to test the correctness of all the function included in the code.

Description of approach to testing. Rationale for test case selection. Summary of results. Any problems uncovered through testing.

Under testing you should list any assumptions you needed to make about the program's inputs or expected behaviour.

2 Results of Testing Partner's Code

The result of testing partner's code is:

True
True

The result is exactly same as the result I got from running my code. The code goes through all my tests. This means that my partner's code is good.

3 Discussion of Test Results

3.1 Problems with Original Code

After reading my partner's code, I realize that I should include the case that the file is not readable. I can raise an IOError when the file is not readable and ask the user to motify the input file. Then, the project can handle more cases.

3.2 Problems with Partner's Code

My partner has a really nice code and it is working perfect. However, I realize that in the function average(L,g), it calculates the average gpa for both male and female no matter what the input is. As an example, the user asks for the average gpa for male, the code will still calculate the average for female and male, but it will return the average for male. This makes the average running time increased, which is time wasting when calculating the large number of students, such as calculating the average gpa of all the male students from Canada.

4 Critique of Design Specification

The results of the functions are not readable for humanbeing. Therefore, I recommand to create a class call student and make the printed output looked easier to read for people using `__str__`.

5 Answers to Questions

(a) The average(L,g) function can be more versatile by adding the new input parameter m representing the student's first prefered options for their second year engineer type. So that we can calculate the average of the student's who chooses any specific major. Also, we can motify the code so that when we set either of the input parameter g or m to null, the function calculates all the students. Such as when we input g as null, the function return the average gpa of both male and female.

We can change the sort(s) function in the similar way by adding extra input parameter m and g to return the sorted male or female student who chooses specific major.

(b) In my opinion the aliasing means that some element exists in the wrong position such as the gpa of the student was stored as the last name. The aliasing usually is not a concern of dictionary because the order of the elements inside the dictionary is not nesseary. However the key of the element is important for the information to be stored in the correct postition. Therefore I recommend to set the condition check of the input values to make sure that the information is stored in the correct place. For example, checking if the gpa is float and is in the range (0,12), is helpful to ckeck the input value is not in wrong order.

(c) I can check if the returned output value is same as the value that calculated using the input value. However, test the functions from ReadAllocationData.py is not neccesary

because the funtions from CalcModule.py use all the functions from ReadAllocationData.py. Therefore, by testing CalcModule.py we also test ReadAllocationData.py.

(d) Using strings to represent the gender of student is not the best because typing "male" or "female" can easily lead typo. Also there are only two elements in {male, female}, so using string is time consuming. The better way is use integers to represent the gender, by letting 0 be the first postion male in [male, female] and 1 be the second position in [male, female]. So that there is less possibility for typos to occur and it is easier to input the information.

(e) The collections.namedtuple Class is also a good way to store information. I recommand to use the collections.namedtuple class instead of dictionary beacause the collections.namedtuple class is immutable, which means it can not be changed after created. It locks down the field to avoid typos and it is easy to accessing the stored information.

(f) If the 'choices' is changed from list to tuple, it is not neccesary to change CalcModule.py. Because the tuple works the same way as list when accessing the information stored in it. If a custom class was written for students, and there is a method that returns the next choice and another method that returns True when there are no more choices, also the lists are changed to tuples, the CalcModule.py will need to be motified bacause tuples are immutable. The method that check if there is any choices left will not work properly, if it uses something like pop and push concepts. You cannot change the elements in the tuple after the tuple is created.

F Code for ReadAllocationData.py

```
## @file ReadAllocationData.py
#  @title ReadAllocationData.py
#  @author James Zhang
#  @date 1/16/2018


## @brief Read a file and output the list with student info in it
#  @param a file s with the format of the input file is:
#      first student: macID fname lname gender gpa choise1 choise2 choise3
#      second student: macID fname lname gender gpa choise1 choise2 choise3
#      ...
#      where all different terms are seprated by space and there is no space within any term
def readStdnts(s):

    f = open(s,"r")
    f1 = f.readlines()
    output = []
    for i in f1:
        word = i.split()
        d = {'macid':word[0], 'fname': word[1], 'lname': word[2], 'gender': word[3], 'gpa':
            float(word[4]), 'choices': [word[5], word[6], word[7]]}
        output += [d]
    f.close()
    return output

## @brief Read a file and output the list with students that have free choices
#  @param a file s with the format of the input file is:
#          first student: macID grade(high school)
#          second student: macID grade(high school)
#      also assume the students with 95+ high school grade are granted free chooces
def readFreeChoice(s):
    f = open(s, "r")
    f1 = f.readlines()
    output = []
    for i in f1:
        word = i.split()
        if float(word[1]) > 95:
            output += [word[0]]

    f.close()
    return output

## @brief Read a file and output the dictionary that shows the capacity of the depts
#  @param a file s with the format of the input file is:
#          first department: dept integer
#          second department: dept integer
def readDeptCapacity(s):

    f = open(s, "r")
    f1 = f.readlines()
    output = {}
    for i in f1:
        word = i.split()
        ## check whether the input is valid
        if word[0] in ['civil', 'chemical', 'electrical', 'mechanical', 'software', 'materials',
            'engphys']:
            output[word[0]] = int(word[1])
        else:
            raise IOError

    f.close()
    return output
```

G Code for CalcModule.py

```
## @file CalcModule.py
#  @title Calcmodule.py
#  @author James Zhang
#  @date 1/16/2018

## @brief Read a list of students and output the sorted students in descending order
#  @detail based on the quicksort learned from algorithm course. modified for this case
def sort(S):
    if len(S) <= 1:
        return S
    ## a for less gpa and b for greater gpa so that we can sort the student in descending order
    a = []
    b = []
    pivot = S.pop()
    for item in S:
        if item['gpa'] < pivot['gpa']:
            a.append(item)
        else:
            b.append(item)
    S.append(pivot)

    return sort(b) + [pivot] + sort(a)


## @brief Read a list of students and calculate the average based on their gender
def average(L,g):
    ## check if the input is valid
    if not(g in ['male', 'female']):
        raise IOError
    s = 0.0
    c = 0
    for i in L:
        if i['gender'] == g:
            s += i['gpa']
            c += 1
    return s/c

## @brief the algorithm that allocate the students to the correspoding major base on their gpa,
     choices and having free choices or not
#  @param S from readStdnts, F from readFreeChoice, C from readDeptCapacity
def allocate(S, F, C):
    ## sort the students in the beginning so that we always arrange the student with higher gpa first
    SS = sort(S)
    free = []
    notfree = []

    result = {'civil':[], 'chemical':[], 'electrical':[], 'mechanical':[], 'software':[],
         'materials':[], 'engphys':[]}
    ## check if the student have free choices or not
    for i in SS:
        ## assmuing only the students with a gpa that is higher or equyal to 4.0 get allocate
        if i['gpa'] >= 4.0:
            if i['macid'] in F:
                free += [i]
            else:
                notfree += [i]
    ## arrange free choices first
    for j in free:
        ## if there is no more space, go to the next choice
        if C[j['choices'][0]] > len(result[j['choices'][0]]):
            result[j['choices'][0]] += [j]
        else:
            if C[j['choices'][1]] > len(result[j['choices'][1]]):
                result[j['choices'][1]] += [j]
            else:
                if C[j['choices'][2]] > len(result[j['choices'][2]]):
                    result[j['choices'][2]] += [j]
    ## using same idea as for free choices, arrange notfree students.
    for k in notfree:
        if C[k['choices'][0]] > len(result[k['choices'][0]]):
            result[k['choices'][0]] += [k]
        else:
            if C[k['choices'][1]] > len(result[k['choices'][1]]):
                result[k['choices'][1]] += [k]
            else:
                if C[k['choices'][2]] > len(result[k['choices'][2]]):
```

```
                result[k['choices'][2]] += [k]
    return result
```

H Code for testCalc.py

```
## @file testCalc.py
#  @title testCalc.py
#  @author James Zhang
#  @date 1/16/2018


## @brief An easy function that calculate if two numbers are almost equal or not(look at equality with
    two decimal places).
def fine(a,b):
    if (abs(a-b)/a)<(1.0/100.0):
        return True
    return False

## test calculation of the average gpa for male
print(fine(average(readStdnts('student.txt'),'female'),5.16))

## test the allocate function
ans = {'civil': [{'macid': 'zhany111', 'fname': 'james', 'lname': 'zhang', 'gender': 'male', 'gpa':
    8.0, 'choices': ['civil', 'electrical', 'materials']}], 'chemical': [], 'electrical': [{'macid':
    'liy200', 'fname': 'iris', 'lname': 'li', 'gender': 'female', 'gpa': 10.0, 'choices':
    ['electrical', 'materials', 'chemical']}], 'mechanical': [], 'software': [{'macid': 'jiaoz666',
    'fname': 'dump', 'lname': 'li', 'gender': 'female', 'gpa': 4.3, 'choices': ['software',
    'engphys', 'chemical']}], 'materials': [{'macid': 'fnndp123', 'fname': 'tie', 'lname': 'dan',
    'gender': 'male', 'gpa': 8.2, 'choices': ['electrical', 'materials', 'chemical']}, {'macid':
    'ojbk123', 'fname': 'an', 'lname': 'li', 'gender': 'male', 'gpa': 6.5, 'choices': ['electrical',
    'materials', 'chemical']}], 'engphys': []}

## only test allocate function since it includes sort() and all the functions from
    ReadAllocatetionData.py
print(allocate(readStdnts('student.txt'),readFreeChoice('free.txt'),readDeptCapacity('dept.txt'))==ans)
```

I Code for Partner's CalcModule.py

```
## @file CalcModule.py
#  @author Behzad Khamneli
#  @brief Sorts the students info based on their GPA, calculates their average and assigns them to a
     department.
#  @date 1/18/2019

## @brief Function sort takes a list of the dictionaries of students from readStdnts in
     ReadAllocationData.
#  @return A list of dictonaries in sorted order.
#  Citation: https://www.saltycrane.com/blog/2007/09/how-to-sort-python-dictionary-by-keys/
def sort(S):
    if S == "File Error":
        return "File Error"
    else:
        return sorted(S, key=lambda grade: grade['gpa'], reverse=True)


## @brief This function calcuates the average gpa of male or female students.
#  @details If param g is 'male', then it returns the average gpa of male students and if g is
     'female' then it returns the average gpa for female students.If non of them has been entered,
     then it returns an error.
#  @param L Is a list of dictionaries of students.
#  @param g Can be either male or female.
#  @return The average gpa of male or female students, with the choice depending on the param g.
def average(L, g):
    if L == "File Error":
        return "File Error"
    else:
        sumMale = 0
        numofMale = 0
        aveMale = 0

        sumFemale = 0
        numFemale = 0
        aveFemale = 0
        for dicty in L:
            if dicty['gender'] == 'male':
                sumMale = sumMale + dicty['gpa']
                numofMale += 1
            if dicty['gender'] == 'female':
                sumFemale = sumFemale + dicty['gpa']
                numFemale += 1
        if g == 'male':
            aveMale = sumMale / numofMale
            return aveMale
        if g == 'female':
            aveFemale = sumFemale / numFemale
            return aveFemale
        else:
            return "g can be either male or female"


## @brief This function assigns the students to a department depending on their gpa and freechoice.
#  @details After each iteration, this function stores the macid of the students who have been
     assigned to a department to a list so that the same student will not be assigned to another
     department.
#  @param S Is a list of the dictionaries of students created by readStdnts.
#  @param F Is a list of students with free choice.
#  @param C Is a dictionary of department capacities.
#  @return A dictionary with the following format {'civil': [student, student,...], 'chemical':
     [student, student,...],...}.Student corresponds to the students' information.
def allocate(S, F, C):
    if (S == "File Error") or (F == "File Error") or (C == "File Error"):
        return "File Error"
    else:
        alldic = {}
        numofFree = len(F)
        assigned = []
        civil = []
        chemical = []
        electrical = []
        mechanical = []
        software = []
        materials = []
        engphys = []
```

```
for std in sort(S):
    for i in range(numofFree):
        if std['macid'] == F[i]:
            if std['choices'][0] == 'civil' and C['civil'] != 0:
                civil.append(std)

                C['civil'] = C['civil'] - 1

            elif std['choices'][0] == 'chemical' and C['chemical'] != 0:
                chemical.append(std)
                C['chemical'] = C['chemical'] - 1

            elif std['choices'][0] == 'electrical' and C['electrical'] != 0:
                electrical.append(std)
                C['electrical'] = C['electrical'] - 1

            elif std['choices'][0] == 'mechanical' and C['mechanical'] != 0:
                mechanical.append(std)
                C['mechanical'] = C['mechanical'] - 1

            elif std['choices'][0] == 'software' and C['software'] != 0:
                software.append(std)
                C['software'] = C['software'] - 1

            elif std['choices'][0] == 'materials' and C['materials'] != 0:
                materials.append(std)
                C['materials'] = C['materials'] - 1

            elif std['choices'][0] == 'engphys' and C['engphys'] != 0:
                engphys.append(std)
                C['engphys'] = C['engphys'] - 1

for std in sort(S):
    for i in range(3):
        if std['macid'] not in F and std['macid'] not in assigned and std['gpa'] >= 4.0:

            if std['choices'][i] == 'civil':
                if C['civil'] != 0:
                    civil.append(std)
                    C['civil'] = C['civil'] - 1
                    assigned.append(std['macid'])

            if std['choices'][i] == 'chemical':
                if C['chemical'] != 0:
                    chemical.append(std)
                    C['chemical'] = C['chemical'] - 1
                    assigned.append(std['macid'])

            if std['choices'][i] == 'electrical':
                if C['electrical'] != 0:
                    electrical.append(std)
                    C['electrical'] = C['electrical'] - 1
                    assigned.append(std['macid'])

            if std['choices'][i] == 'mechanical':
                if C['mechanical'] != 0:
                    mechanical.append(std)
                    C['mechanical'] = C['mechanical'] - 1
                    assigned.append(std['macid'])

            if std['choices'][i] == 'software':
                if C['software'] != 0:
                    software.append(std)
                    C['software'] = C['software'] - 1
                    assigned.append(std['macid'])

            if std['choices'][i] == 'materials':
                if C['materials'] != 0:
                    materials.append(std)
                    C['materials'] = C['materials'] - 1
                    assigned.append(std['macid'])

            if std['choices'][i] == 'engphys':
                if C['engphys'] != 0:
                    engphys.append(std)
                    C['engphys'] = C['engphys'] - 1
                    assigned.append(std['macid'])

alldic['civil'] = civil
alldic['chemical'] = chemical
```

```
alldic['electrical'] = electrical
alldic['mechanical'] = mechanical
alldic['software'] = software
alldic['materials'] = materials
alldic['engphys'] = engphys
return alldic
```

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
```