Assignment 1 Solution

Harshil Modi Modih1

February 25, 2019

The purpose of Part 1 of this software design exercise was to write a Python program that matches the natural language specification provided. In part 2, design critique was done based on experience with a classmate's implementation.

1 Testing of the Original Program

For testing my program I decided on using the unit testing framework pytest. For the test cases I ensured that I had enough cases to test each and every line of my program hence the sixteen test cases.

SUMMARY OF RESULTS:
test_sort: PASSED
test_sort_1: PASSED
test_sort_2: PASSED
test_sort_3: PASSED
test_average: PASSED
test_average1: PASSED
test_average3: PASSED
test_average4: PASSED
test_average5: PASSED
test_allocate: PASSED
test_allocate1: PASSED
test_allocate2: PASSED
test_allocate3: PASSED
test_allocate4: PASSED
test_allocate5: PASSED
test_allocate6: PASSED
test_allocate7: PASSED

Rational for test cases:

test_sort(): This test case contains 24 different students. The goal of this test case is to test the accuracy of the function sort for large datasets. Having a larger dataset then provided would not be efficient and would be tiresome to manually input the compare dictionary.

test_sort_1(): This test case contains no students. The goal of this test case is to test if the function sort can handle a situation where no student are present.

test_sort_2(): This test case contains a few students with a GPA of zero. The goal of this test case is to test if the function sort can handle a situation where a few students have a GPA of zero.

test_sort_3(): This test case contains all students with the same GPA. The goal of this test case is to test if the function sort can handle a situation where all the students have the same GPA.

test_average(): This test case contains 24 different students. The goal of this test case is to test the accuracy of the function average for large datasets when the second input parameter 'g' is female. Having a larger dataset would be very tedious and tiresome for testing purposes.

test_average1(): This test case also contains 24 different students. The goal of this test case is to test the accuracy of the function average for large datasets when the second input parameter 'g' is male. Again, Having a larger dataset would be very tedious and tiresome for testing purposes.

test_average3(): This test case contains no female students. The goal of this test case is to test if the function average can handle a situation where the denominator(number of female students) is zero.

test_average4(): This test case contains no male students. The goal of this test case is to test if the function average can handle a situation where the denominator(number of male students) is zero.

test_average5(): This test case contains all students with the same GPA. The goal of this test case is to test if the function average can handle a situation where all the student

GPA are the same.

test_allocate(): This test case contains a situation where the free choice file contains names of students not present in textfile. The goal of this function is to ensure the program allocate ignore those students and only work with students that are in the textfile.

test_allocate1(): This test case contains students without free choice and a GPA of less then 4.0. The goal of this test case is to ensure that the function allocate does not allocate these students to a second-year department.

test_allocate2(): This test case contains a large dataset which represents a random typical situation. A few students have free choice, a few students will need to be allocated to their second choice which a few to their third. The goal of this function is to ensure that everything works correctly when put together.

test_allocate3(): This test case contains students with free choice and a GPA of less then 4.0. The goal of this test case is to ensure that the function allocate does not allocate these students to a second-year department.

test_allocate4(): The goal of this test case is to ensure that students without free choice are allocated into their second choice if the first choice is full and second choice is not.

test_allocate5(): The goal of this test case is to ensure that students without free choice are allocated into their third choice if both the first and second choices are full.

test_allocate6(): The goal of this test case is to ensure that students without free choice are allocated into their third choice even when all three of their choices are full. This approach was choose because these students need to be put into a department. This does not represent the real situation as student generally choose 12 choices and not 3.

test_allocate7(): The goal of this test case is to ensure that students with free choice are allocated into their first choice even when anyone of their choices are full. This approach was choose because a commitment was made by the department that these students have the right of choice and can not be taken away from them. For this situation, a special accommodation is made in this situation.

2 Results of Testing Partner's Code

```
SUMMARY OF RESULTS:
test_sort: PASSED
test_sort_1: PASSED
test_sort_2: PASSED
test_sort_3: PASSED
test_average: FAILED
test_average1: FAILED
test_average3: PASSED
test_average4: PASSED
test_average5: PASSED
test_allocate: FAILED
test_allocate1: FAILED
test_allocate2: FAILED
test_allocate3: FAILED
test_allocate4: FAILED
test_allocate5: FAILED
test_allocate6: FAILED
test_allocate7: FAILED
```

3 Discussion of Test Results

Assumptions made:

(1) All .txt files must be in the correct format provided.

(2) GPA must be a non-negative float or int.

(3) parameter g must be either 'male' or 'female' only.

(4) Average GPA is rounded to two decimal places

(5) The number of students with free choice and first choice to a particular department does not exceed the department capacity.

(6) If the first choice and second choice of a student is full, third choice must not be full or an exception will be made.

(7) students with GPA of 4.0 are not allocated to second-year.

(8) A case where 2 students having the same GPA and one space left will not be present.

Original file: All 16 of the test cases passed flawlessly.

Partner's file: 7 out of 16 test cases passed. All the test cases regarding sort function passed. 3 out of 5 of the test cases regarding the average function passed. 0 out of 8 of the test cases regarding the allocate function passed.

3.1 Problems with Original Code

There were no problems with the original code as all the test cases passed.

3.2 Problems with Partner's Code

For the sort function, there was no problem with my Partner's code and my implementation as all the test cases passed. However for the average function only 3 out of 5 cases passed. The reason that the first 2 cases failed was do to a rounding point error. In my implementation, I rounded the average to two decimal points while my partner's implementation did not. For the allocate function, my partner's program was not fully functional as it contained local variables before assignment (Even after changing this, my partner's code broke because it was not complete). Since my partner's implementation was not correct, all the test cases for the allocate function failed. Overall, due to non specific design implementation criteria, I and my partner's implementation varied a little.

4 Critique of Design Specification

Although the assignment was mostly open ended as we had the choice of choosing most of our design decisions, a few of these specification were forced upon us. These included the data structure of students, free choice, and department capacity. I liked how this was forced upon us as it provided me with a guideline of how to build the functions. Although this was not necessary, having a guideline made my work easier as I had to do less work figuring out the types of data structures to use and implement. I also liked the open ended design specification for the textfile as it allowed us to create the textfiles to my convinces. What I did not like was the fact that free choice was not a key value pair in the student dictionary but was returned in a separate list by the readFreeChoices function. Having it along side the other information would make it more organized. Although the design was mostly open ended, when it came to testing the partner's files this caused major problems. This was because there was no structure to follow and as a result each of us had our own implementation. Next time, if comparisons between individual's implementation was going to be made, a less open ended design specification would be preferred.

5 Answers to Questions

(a) To make the average function more general, I would eliminate the need for inputting the 'g' parameter. I would instead generalize the new average function to return the average of all students, all males, and all females. This eliminates the need to provide a second parameter making the function more versatile while also providing the ability of the function to generate the overall student GPA which is something the original average function lacks. On a similar note, to make the sort function more general I would change it to sort based on any characteristics and not only GPA (e.g alphanumeric characters). It would be the user's preference to choose the characteristic to sort by.

(b) In this context aliasing is when one variable's values are also values of another variable and can be mutated from either variables. This is especially true in python with mutable data structures as variables are just names that store references to values. Since dictionaries in python is a mutable data structure, aliasing can be a major problem if care is not taken. For example if variable a is assigned a dictionary, and b is assigned a, any change made to b would directly result in a change in a. To guard against potential problems due to aliasing, I would use the copy function for copying a list or dictionary instead of assignment. This way, I will always have 2 distinct copies of information and aliasing would be avoided it.

(c) Some potential test cases I could have used to test ReadAllocationData.py would to check if the functions can read large datasets, medium sized datasets and no datasets. Testing of large datasets is important to ensure that read files does not have upper bound. The medium sized dataset would be used to just mimic an average dataset hence test the general functionality of the functions. The no dataset modules would be used to test for edge cases. The reason that CalcModule.py was used to test the functionality of the program was because the module CalcModule relies heavy on the ReadAllocationData module but is also more complex then the ReadAllocationData module. As a result, if the test cases pass of the CalcModule we can infer that ReadAllocationData module must also be correct. However, if we were to test just ReadAllocationData, we would not be able to infer that CalcModule is correct as ReadAllocationData does not rely on the CalcModule.

(d) The problems with using strings in this case to represent keys for a dictionary or a finite set is that strings are case sensitive. For example a person may input "female" while another person inputs ”Female”. Although the intent was the same from both users, The end result is different hence they are 2 distinct keys with different sets. A better way would be to use numerical values to represent strings such as 1 = male and 2 = female. This way ambiguity and errors will be reduced. Another problem with using strings is that comparisons are expensive. For example, to compare a string, you will need to compare each character individually and this can take longer then numerical values.

(e) A mathematical tuple is a data structure which contain a collection of elements of different types. To implement a mathematical tuple in python similar to a dictionary, we can use namedtuples. The main difference between dictionaries and namedtuples is that namedtuples are immutable. I would recommend, that we a change the implemntation of a dictionary to named tuples because once we create our dictionaries, we are not modifying them. By change the structure to a namedtuple, it would reduce the changes of someone accidentally modifying the data structure. To change this, I would add the unique identifier or the key and the value to the tuple while reading the file. Another option to do this is to use class. We can define an abstract data structure or a student class which contains the required information as instance variables. This would be a good idea to implement as it can make the program easier to organize. To implement this, I would create a new class called student which contains its information as instance variables. Then in the main class I would create a new object of type student for each student.

(f) For my implementation, if the list of choices were changed to a tuple, I would not need to change CalcModule.py. This is because I do not modify the list or the order of the choices but only access the information. If I did, then I would need to change up CalcModule since in a tuple this would not be possible as they are immutable. If we after add an abstract data type to our code which returns the value of the next choice when the current choice is full, I would need to modify my code. This is because I currently don’t have a counter to count if the department is full but instead keep getting the length of the string to determine the number of people. In the abstract data type, I would need to add another variable to determine if the department is full or some other way keep a track of the department capacities. Lets say that the software implements this, I would change my current code to not check if department is full as I currently check that per student. For my implementation, even if the lists was changed to a tuple, I would still not need to change my code as I still would be able to access the information and get the length of the tuple. This was because I made my code in a way to not require me to use new variables or modify the current list.

F Code for ReadAllocationData.py

```
"""
## @File: ReadAllocationData.py
#  @Title: ReadAllocationData
#  @Author: Harshil Modi
#  @Date:January 17th 2019
"""
'''
Format of line of readstdnts testfile:
    macid fname lname gender gpa choice1 choice2 choice3
    eg. modih3 harshil2 modi male 7.0 civil electrical chemical

Format of line of readFreeChoice textfile:
    macid
    eg. modih1

format of line of readDeptCapacity textfile:
    department_name department_capacity
    eg. civil 10

'''

## @brief This function takes a string corresponding to a filename and returns a list of dictionary
     of student information
#  which includes: macid, first_name, last_name, gender, GPA, department_choices.
#  Assumptions: 1) The file must be in the format specified above,
#               2) GPA must be a non-negative float or int.
#  @param S type (String): A filename with student information
#  @return A list of dictionaries of student information

def readStdnts(s):

    mac_student = []
    file = open(s, "r")

    for each_line in file:
        dictionary = {}
        student = each_line.split(" ")
        dictionary['macid'] = student[0]
        dictionary['fname'] = student[1]
        dictionary['lname'] = student[2]
        dictionary['gender'] = student[3]
        dictionary['gpa'] = float(student[4])
        dictionary['choices'] = [student[5], student[6], student[7].strip()]
        mac_student.append(dictionary)

    file.close()
    return mac_student

## @brief This function takes a filename and returns a list of students with free choice.
#  Assumptions: 1) The file must be in the format specified above.
#  @param S type (String): A filename with macid of students with free choice
#  @return A list student's macid with free choice

def readFreeChoice(s):

    free_choice = []
    file = open(s, "r")

    for each_line in file:
        free_choice.append(each_line.strip())
    file.close()
    return free_choice

## @brief takes a filename and returns a dictionary of departments and their max capacity.
#  @details Within the dictionary the keys are the department names and their corressponding values
     are
#  the max department capacity.
#  Assumptions
#            1) The file must be in the format specified above.
#  @param s type (String): A filename with department names and corressponding max capacities
#  @return A dictionary of department names with their corressponding max capacities

def readDeptCapacity(s):

    department_capacity = {}
    file = open(s, "r")
```

```
for each_line in file:
    department = each_line.split(" ")
    department_capacity[department[0]] = int(department[1])

file.close
return department_capacity
```

G Code for CalcModule.py

```
"""
##@File: CalcModule.py
#@Title: Calcmodule
#@Author: Harshil Modi
#@Date:January 17th 2019
"""
import ReadAllocationData # importing file ReadAllocationData.py

## @brief This function sorts the students in non ascending GPA.
#  @Details The function uses a self-made sorting algorithm to sort the dictionary created in
    readStdnts(s).
#  @param S A list of dictionaries of student information
#  @return A list of dictionaries of student information in non ascending GPA order

def sort(S):

    mac_student = S
    new_list = [] #output list

    for student in mac_student:
        insert = False  #To keep track if the student was inputted

        for i in range(len(new_list)):
            new_student = new_list[i]

            if new_student['gpa'] < student['gpa']:
                new_list.insert(i,student)
                insert = True
                break

        if not insert:
            new_list.append(student)

    return new_list

## @brief This function finds the average GPA of females or males.
#  @details If parameter g is 'male' the average GPA of all males is calculated and vice versa.
#  Assumption: 1) parameter g must be either 'male' or 'female'.
#  @param L: A list of dictionaries of student information created by readFreeChoice(s)
#  @param g type (String): A string representing the gender male or female
#  @return Average GPA of females or males rounded to 2 decimal places

def average(L, g):

    sum = 0  #counter to track sum GPA
    count = 0  #number of students counted

    for student in L:

        if student['gender'] == g:
            sum += student['gpa']
            count +=1
        else:
            continue

    if count == 0: #error check for students equals zero
        return 0

    return round(sum/count, 2)

## @brief This function allocates students to second-year programs with GPA OVER 4.0
#  @details The function first allocates all the students with free choice to their respected first
    choice programs.
#  Next the function begins allocating students from highest GPA to lowest GPA to their respected
    choices. If the
#  students first choice department is full, they will be allocated to their second choice. If that
    is also full, they
#  will be allocated to their last choice.
#  Assumptions: 1) The number of students with free choice and first choice to a particular
    department does not exceed the department capacity,
#               2) If the first choice and second choice of a student is full, third choice must not
    be full,
#               3) students with GPA of 4.0 are not allocated to second-year,
#               4) A case where 2 students having the same gpa and one space left will not be present.
#  @param S: A list of dictionaries of student information created in readStdnts()
#  @param F: A list of students with free choice created in readFreeChoice(s)
#  @param C: A dictionary of department capacity created in readDeptCapacity(s)
#  @return Dictionary of which students have been allocated to which department

def allocate(S, F, C):

    department_dictionary  = {'civil':[], 'chemical':[], 'electrical':[], 'mechanical':[], 'software':
        [], 'materials': [], 'engphys':[]}

    #allocates students with freechoice first
    for student in S:
        if student['macid'] in F and student['gpa'] > 4.0:
            department_students = department_dictionary[student['choices'][0]]
            department_students.append(student)
            department_dictionary[student['choices'][0]] = department_students

    S = sort(S) # sorts the students based on GPA

    # allocates the students without free choice
    for student in S:
        if student['macid'] not in F and student['gpa'] > 4.0:

            if len(department_dictionary[student['choices'][0]]) < C[student['choices'][0]]:
                department_students = department_dictionary[student['choices'][0]]
                department_students.append(student)
                department_dictionary[student['choices'][0]] = department_students

            elif len(department_dictionary[student['choices'][1]]) < C[student['choices'][1]]:
                department_students = department_dictionary[student['choices'][1]]
                department_students.append(student)
                department_dictionary[student['choices'][1]] = department_students

            else:
                department_students = department_dictionary[student['choices'][2]]
                department_students.append(student)
                department_dictionary[student['choices'][2]] = department_students

        else:
            continue

    return (department_dictionary)
```

H Code for testCalc.py

```
"""
## @File: testCalc.py
# @Title: testCalc
# @Author: Harshil Modi
# @Date:January 17th 2019
"""

import CalcModule
import ReadAllocationData

#Test case to check if the sort function can sort large datasets
def test_sort():
    S = ReadAllocationData.readStdnts("textfile.txt")
    sort = CalcModule.sort(S)
    assert sort == [{'macid': 'modih1', 'fname': 'harshil', 'lname': 'modi', 'gender': 'male', 'gpa':
        12.0, 'choice': ['software', 'electrical', 'mechanical']},
                    {'macid': 'modih23', 'fname': 'harshil22', 'lname': 'modi', 'gender': 'male',
                        'gpa': 11.5, 'choice': ['software', 'electrical', 'chemical']},
                    {'macid': 'modih18', 'fname': 'harshil17', 'lname': 'modi1', 'gender': 'female',
                        'gpa': 11.0, 'choice': ['software', 'electrical', 'mechanical']},
                    {'macid': 'modih17', 'fname': 'harshil16', 'lname': 'modi', 'gender': 'male',
                        'gpa': 10.0, 'choice': ['software', 'electrical', 'mechanical']},
                    {'macid': 'modih21', 'fname': 'harshil20', 'lname': 'modi', 'gender': 'male',
                        'gpa': 9.3, 'choice': ['software', 'electrical', 'mechanical']},
                    {'macid': 'modih9', 'fname': 'harshil8', 'lname': 'modi', 'gender': 'male',
                        'gpa': 9.2, 'choice': ['chemical', 'engphys', 'software']},
                    {'macid': 'modih16', 'fname': 'harshil15', 'lname': 'modi', 'gender': 'female',
                        'gpa': 9.0, 'choice': ['chemical', 'engphys', 'software']},
                    {'macid': 'modih5', 'fname': 'harshil4', 'lname': 'modi', 'gender': 'male',
                        'gpa': 8.5, 'choice': ['software', 'electrical', 'mechanical']},
                    {'macid': 'modih22', 'fname': 'harshil21', 'lname': 'modi1', 'gender': 'female',
                        'gpa': 8.2, 'choice': ['electrical', 'software', 'mechanical']},
                    {'macid': 'modih15', 'fname': 'harshil14', 'lname': 'modi', 'gender': 'male',
                        'gpa': 8.0, 'choice': ['civil', 'electrical', 'chemical']},
                    {'macid': 'modih20', 'fname': 'harshil19', 'lname': 'modi', 'gender': 'male',
                        'gpa': 7.9, 'choice': ['engphys', 'electrical', 'mechanical']},
                    {'macid': 'modih6', 'fname': 'harshil5', 'lname': 'modi1', 'gender': 'female',
                        'gpa': 7.5, 'choice': ['electrical', 'sofware', 'mechanical']},
                    {'macid': 'modih19', 'fname': 'harshil18', 'lname': 'modi', 'gender': 'female',
                        'gpa': 7.4, 'choice': ['civil', 'electrical', 'chemical']},
                    {'macid': 'modih3', 'fname': 'harshil2', 'lname': 'modi', 'gender': 'male',
                        'gpa': 7.0, 'choice': ['civil', 'electrical', 'chemical']},
                    {'macid': 'modih12', 'fname': 'harshil11', 'lname': 'modi', 'gender': 'female',
                        'gpa': 7.0, 'choice': ['software', 'electrical', 'mechanical']},
                    {'macid': 'modih10', 'fname': 'harshil9', 'lname': 'modi1', 'gender': 'female',
                        'gpa': 6.8, 'choice': ['electrical', 'software', 'mechanical']},
                    {'macid': 'modih2', 'fname': 'harshil1', 'lname': 'modi1', 'gender': 'female',
                        'gpa': 6.0, 'choice': ['electrical', 'software', 'mechanical']},
                    {'macid': 'modih7', 'fname': 'harshil6', 'lname': 'modi', 'gender': 'male',
                        'gpa': 5.3, 'choice': ['software', 'electrical', 'mechanical']},
                    {'macid': 'modih4', 'fname': 'harshil3', 'lname': 'modi', 'gender': 'male',
                        'gpa': 5.0, 'choice': ['chemical', 'engphys', 'software']},
                    {'macid': 'modih8', 'fname': 'harshil7', 'lname': 'modi', 'gender': 'male',
                        'gpa': 5.0, 'choice': ['software', 'electrical', 'mechanical']},
                    {'macid': 'modih24', 'fname': 'harshil23', 'lname': 'modi', 'gender': 'female',
                        'gpa': 5.0, 'choice': ['chemical', 'engphys', 'software']},
                    {'macid': 'modih14', 'fname': 'harshil13', 'lname': 'modi1', 'gender': 'female',
                        'gpa': 4.0, 'choice': ['electrical', 'software', 'mechanical']},
                    {'macid': 'modih11', 'fname': 'harshil10', 'lname': 'modi', 'gender': 'male',
                        'gpa': 3.9, 'choice': ['civil', 'electrical', 'mechanical']},
                    {'macid': 'modih13', 'fname': 'harshil12', 'lname': 'modi', 'gender': 'male',
                        'gpa': 2.0, 'choice': ['software', 'engphys', 'software']}]


#Test case to check if the sort function can handle empty file
def test_sort_1():
    S = ReadAllocationData.readStdnts("textfile1.txt")
    sort = CalcModule.sort(S)
    assert sort == []

#Test case to check if it can handle GPA of 0 while sorting
def test_sort_2():
    S = ReadAllocationData.readStdnts("textfile2.txt")
    sort = CalcModule.sort(S)
    assert sort == [{'macid': 'modih1', 'fname': 'harshil', 'lname': 'modi', 'gender': 'male', 'gpa':
        12.0, 'choice': ['software', 'electrical', 'mechanical']},
                        {'macid': 'modih2', 'fname': 'harshil1', 'lname': 'modi1', 'gender': 'female',
                            'gpa': 6.0, 'choice': ['electrical', 'software', 'mechanical']},
                        {'macid': 'modih3', 'fname': 'harshil2', 'lname': 'modi', 'gender': 'male',
                            'gpa': 0.0, 'choice': ['civil', 'electrical', 'chemical']}]

#Test case to check if it can sort all files with same GPA
def test_sort_3():
    S = ReadAllocationData.readStdnts("textfile4.txt")
    sort = CalcModule.sort(S)
    assert sort == [{'macid': 'modih1', 'fname': 'harshil', 'lname': 'modi', 'gender': 'male', 'gpa':
        1.0, 'choice': ['software', 'electrical', 'mechanical']},
                        {'macid': 'modih2', 'fname': 'harshil1', 'lname': 'modi1', 'gender': 'male',
                            'gpa': 1.0, 'choice': ['electrical', 'software', 'mechanical']},
                        {'macid': 'modih3', 'fname': 'harshil2', 'lname': 'modi', 'gender': 'male',
                            'gpa': 1.0, 'choice': ['software', 'electrical', 'chemical']}]

#test case to check if average can handle large dataset for female
def test_average():
    S = ReadAllocationData.readStdnts("textfile.txt")
    average = CalcModule.average(S, "female")
    assert average == 7.19

#test case to check if average can handle large dataset for male
def test_average1():
    S = ReadAllocationData.readStdnts("textfile.txt")
    average = CalcModule.average(S, "male")
    assert average == 7.47

#test case to check if average can handle no data for female
def test_average3():
    S = ReadAllocationData.readStdnts("textfile1.txt")
    average = CalcModule.average(S, "female")
    assert average == 0.0

#test case to check if average can handle no data for male
def test_average4():
    S = ReadAllocationData.readStdnts("textfile1.txt")
    average = CalcModule.average(S, "male")
    assert average == 0.0

#test case to check if average can handle same GPA for all
def test_average5():
    S = ReadAllocationData.readStdnts("textfile4.txt")
    average = CalcModule.average(S, "male")
    assert average == 1.0

#test case when free choice file contains names but file is empty
def test_allocate():
    S = ReadAllocationData.readStdnts("textfile1.txt")
    F = ReadAllocationData.readFreeChoice("freeChoice.txt")
    C = ReadAllocationData.readDeptCapacity("department.txt")
    department = CalcModule.allocate(S, F, C)
    department1 = {'civil': [], 'chemical': [], 'electrical': [], 'mechanical': [], 'software': [],
        'materials': [], 'engphys': []}
    assert department == department1

#test case to check when a person has a GPA of zero (<4.0)
def test_allocate1():
    S = ReadAllocationData.readStdnts("textfile2.txt")
    F = ReadAllocationData.readFreeChoice("freeChoice.txt")
    C = ReadAllocationData.readDeptCapacity("department.txt")
    department = CalcModule.allocate(S, F, C)
    department1 = {'civil': [], 'chemical': [], 'electrical': [{'macid': 'modih2', 'fname':
        'harshil1', 'lname': 'modi1', 'gender': 'female', 'gpa': 6.0,
                    'choice': ['electrical', 'software', 'mechanical']}], 'mechanical': [], 'software':
                        [{'macid': 'modih1', 'fname': 'harshil', 'lname': 'modi', 'gender': 'male',
                            'gpa': 12.0, 'choice': ['software', 'electrical', 'mechanical']}],
                                'materials': [], 'engphys': []}
    assert department == department1

# test case to check if allocate can handle large data with a few free choice students and full first
    and second choices
def test_allocate2():
    S = ReadAllocationData.readStdnts("textfile.txt")
    F = ReadAllocationData.readFreeChoice("freeChoice.txt")
    C = ReadAllocationData.readDeptCapacity("department.txt")
    department = CalcModule.allocate(S, F, C)
    department1 = {'civil': [{'macid': 'modih15', 'fname': 'harshil14', 'lname': 'modi', 'gender':
        'male', 'gpa': 8.0, 'choice': ['civil', 'electrical', 'chemical']}],
                             {'macid': 'modih19', 'fname': 'harshil18', 'lname': 'modi', 'gender':
                                 'female', 'gpa': 7.4, 'choice': ['civil', 'electrical',
                                 'chemical']},
                             {'macid': 'modih3', 'fname': 'harshil2', 'lname': 'modi', 'gender':
                                 'male', 'gpa': 7.0, 'choice': ['civil', 'electrical',
                                 'chemical']}], 'chemical': [{'macid': 'modih16', 'fname':
                                 'harshil15', 'lname': 'modi', 'gender': 'female', 'gpa': 9.0,
                                 'choice': ['chemical', 'engphys', 'software']},
                             {'macid': 'modih9', 'fname': 'harshil8', 'lname': 'modi', 'gender':
                                 'male', 'gpa': 9.2, 'choice': ['chemical', 'engphys', 'software']},
                             {'macid': 'modih4', 'fname': 'harshil3', 'lname': 'modi', 'gender':
                                 'male', 'gpa': 5.0, 'choice': ['chemical', 'engphys', 'software']},
                             {'macid': 'modih24', 'fname': 'harshil23', 'lname': 'modi', 'gender':
                                 'female', 'gpa': 5.0, 'choice': ['chemical', 'engphys',
                                 'software']}], 'electrical': [{'macid': 'modih2', 'fname':
                                 'harshil1', 'lname': 'modi1', 'gender': 'female', 'gpa': 6.0,
                                 'choice': ['electrical', 'software', 'mechanical']},
                             {'macid': 'modih22', 'fname': 'harshil21', 'lname': 'modi1', 'gender':
                                 'female', 'gpa': 8.2, 'choice': ['electrical', 'software',
                                 'mechanical']},
                             {'macid': 'modih5', 'fname': 'harshil4', 'lname': 'modi', 'gender':
                                 'male', 'gpa': 8.5, 'choice': ['software', 'electrical',
                                 'mechanical']},
                             {'macid': 'modih6', 'fname': 'harshil5', 'lname': 'modi1', 'gender':
                                 'female', 'gpa': 7.5, 'choice': ['electrical', 'sofware',
                                 'mechanical']},
                             {'macid': 'modih10', 'fname': 'harshil9', 'lname': 'modi1', 'gender':
                                 'female', 'gpa': 6.8, 'choice': ['electrical', 'software',
                                 'mechanical']},
                             {'macid': 'modih7', 'fname': 'harshil6', 'lname': 'modi', 'gender':
                                 'male', 'gpa': 5.3, 'choice': ['software', 'electrical',
                                 'mechanical']}], 'mechanical': [], 'software': [{'macid': 'modih1',
                                 'fname': 'harshil', 'lname': 'modi', 'gender': 'male', 'gpa': 12.0,
                                 'choice': ['software', 'electrical', 'mechanical']},
                             {'macid': 'modih8', 'fname': 'harshil7', 'lname': 'modi', 'gender':
                                 'male', 'gpa': 5.0, 'choice': ['software', 'electrical',
                                 'mechanical']}, {'macid': 'modih12', 'fname': 'harshil11', 'lname':
                                 'modi', 'gender': 'female', 'gpa': 7.0, 'choice': ['software',
                                 'electrical', 'mechanical']},
                             {'macid': 'modih21', 'fname': 'harshil20', 'lname': 'modi', 'gender':
                                 'male', 'gpa': 9.3, 'choice': ['software', 'electrical',
                                 'mechanical']}, {'macid': 'modih23', 'fname': 'harshil22', 'lname':
                                 'modi', 'gender': 'male', 'gpa': 11.5, 'choice': ['software',
                                 'electrical', 'chemical']},
                             {'macid': 'modih18', 'fname': 'harshil17', 'lname': 'modi1', 'gender':
                                 'female', 'gpa': 11.0, 'choice': ['software', 'electrical',
                                 'mechanical']}, {'macid': 'modih17', 'fname': 'harshil16', 'lname':
                                 'modi', 'gender': 'male', 'gpa': 10.0, 'choice': ['software',
                                 'electrical', 'mechanical']}], 'materials': [], 'engphys':
                                 [{'macid': 'modih20', 'fname': 'harshil19', 'lname': 'modi',
                                 'gender': 'male', 'gpa': 7.9, 'choice': ['engphys', 'electrical',
                                 'mechanical']}]}
    assert department == department1

#test case to check when a person has a GPA of zero (<4.0) but also free choice
def test_allocate3():
    S = ReadAllocationData.readStdnts("textfile2.txt")
    F = ReadAllocationData.readFreeChoice("freeChoice1.txt")
    C = ReadAllocationData.readDeptCapacity("department.txt")
    department = CalcModule.allocate(S, F, C)
    department1 = {'civil': [], 'chemical': [], 'electrical': [{'macid': 'modih2', 'fname':
        'harshil1', 'lname': 'modi1', 'gender': 'female', 'gpa': 6.0,
                  'choice': ['electrical', 'software', 'mechanical']}], 'mechanical': [], 'software':
                       [{'macid': 'modih1', 'fname': 'harshil', 'lname': 'modi', 'gender': 'male',
                            'gpa': 12.0, 'choice': ['software', 'electrical', 'mechanical']}],
                                'materials': [], 'engphys': []}
    assert department == department1

#test case to check when the first choice is full for a person without free choice
def test_allocate4():
    S = ReadAllocationData.readStdnts("textfile2.txt")
    F = ReadAllocationData.readFreeChoice("freeChoice1.txt")
    C = ReadAllocationData.readDeptCapacity("department1.txt")
    department = CalcModule.allocate(S, F, C)
    department1 = {'civil': [], 'chemical': [], 'electrical': [{'macid': 'modih1', 'fname': 'harshil',
        'lname': 'modi', 'gender': 'male', 'gpa': 12.0, 'choice': ['software', 'electrical',
        'mechanical']},
                            {'macid': 'modih2', 'fname': 'harshil1', 'lname': 'modi1', 'gender':
                                'female', 'gpa': 6.0, 'choice': ['electrical', 'software',
                                'mechanical']}], 'mechanical': [], 'software': [], 'materials': [],
                                'engphys': []}
    assert department == department1

#test case to check when the first choice is full and second choice is full for a person without free
    choice
def test_allocate5():
    S = ReadAllocationData.readStdnts("textfile2.txt")
    F = ReadAllocationData.readFreeChoice("freeChoice1.txt")
    C = ReadAllocationData.readDeptCapacity("department2.txt")
    department = CalcModule.allocate(S, F, C)
    department1 = {'civil': [], 'chemical': [], 'electrical': [], 'mechanical': [{'macid': 'modih1',
        'fname': 'harshil', 'lname': 'modi', 'gender': 'male', 'gpa': 12.0, 'choice': ['software',
        'electrical', 'mechanical']},
                            {'macid': 'modih2', 'fname': 'harshil1', 'lname': 'modi1', 'gender':
                                'female', 'gpa': 6.0, 'choice': ['electrical', 'software',
                                'mechanical']}], 'software': [], 'materials': [], 'engphys': []}
    assert department == department1

#test to see when all 3 of the choices are full for someone without free choice
def test_allocate6():
    S = ReadAllocationData.readStdnts("textfile2.txt")
    F = ReadAllocationData.readFreeChoice("freeChoice1.txt")
    C = ReadAllocationData.readDeptCapacity("department3.txt")
    department = CalcModule.allocate(S, F, C)
    department1 = {'civil': [], 'chemical': [], 'electrical': [], 'mechanical': [{'macid': 'modih1',
        'fname': 'harshil', 'lname': 'modi', 'gender': 'male', 'gpa': 12.0, 'choice': ['software',
        'electrical', 'mechanical']},
                            {'macid': 'modih2', 'fname': 'harshil1', 'lname': 'modi1', 'gender':
                                'female', 'gpa': 6.0, 'choice': ['electrical', 'software',
                                'mechanical']}], 'software': [], 'materials': [], 'engphys': []}
    assert department == department1

#test case to check when the first choice is full for a person with freechoice
def test_allocate7():
    S = ReadAllocationData.readStdnts("textfile2.txt")
    F = ReadAllocationData.readFreeChoice("freeChoice2.txt")
    C = ReadAllocationData.readDeptCapacity("department3.txt")
    department = CalcModule.allocate(S, F, C)
    department1 = {'civil': [], 'chemical': [], 'electrical': [{'macid': 'modih2', 'fname':
        'harshil1', 'lname': 'modi1', 'gender': 'female', 'gpa': 6.0, 'choice': ['electrical',
        'software', 'mechanical']}],
                'mechanical': [{'macid': 'modih1', 'fname': 'harshil', 'lname': 'modi', 'gender':
                    'male', 'gpa': 12.0, 'choice': ['software', 'electrical', 'mechanical']}],
                    'software': [], 'materials': [], 'engphys': []}
    assert department == department1
```

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 = pytest -v
DOC = doxygen
DOCFLAGS =
DOCCONFIG = docConfig

SRC = src/test_Calc.py

.PHONY: all test doc clean

test:
	$(PYFLAGS) $(SRC)

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

all: test doc

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