Assignment 1 Solution

Joshua Guinness, guinnesj

January 25, 2019

In this assignment, a python program was created which automatically reads in data from various files, then allocates first year engineering students into their stream in order of free choice, then GPA. The program consists of three modules; one to read in the student, departement, and free choice data, one to allocate the students and perform operations, and one which acts as a test driver, consisting of test cases.

The rest of this document will outline the methodology behind the testing and serve as a reflection on the assignment.

1 Testing of the Original Program

In order to talk about testing results and the methodology used to design test cases, it is important to first talk about what was assumed while coding the program.

The reason assumptions had to be taken is because the specification was not formal or specific enough on how the interface was meant to work. For example, the specification gave no instructions on program behaviour when the file does not exist. Either an assumption had to be made that the file will always exists, or the potential error had to be dealt with in the code, the later of which was done in this program.

The assumptions used while programming are as follows.

1. All the names and macid of the students are correct and match those in other files such as freeChoice.
2. All students have a gender that is either 'male' or 'female' and the gender is spelt correctly.
3. All students have a GPA that is between 1.0 and 12.0.
4. All students have three proper engineering streams that they would like to go into, spelt correctly.

5. There is enough capacity in each department for every student with free choice to be allocated to their first choice

6. There is enough capacity in each department for every student to be allocated to at least one of their top three choices.

7. Every student with free choice passed first year engineering (GPA greater than or equal to 4.0).

Based on the assumptions outline above, there were two types of test cases designed.

The first type of test cases checked for correctness of the functions. This type included the second, third, fourth, fifth, sixth, and eighth test cases. These cases were chosen to test the main functionality of each of the functions in CalcModule at least once. After these test cases, it can be confirmed that the implementation of the functions are correct. A description and rationale of each one will be breifly discussed. All these test cases passed.

The second test case checked to see whether the sort function sorted the students in order of GPA correctly. It just compared the output of the function to a dictionary of sorted students which was known to be correct.

The third, fourth, fifth, and sixth test cases check to see whether a correct average was returned for males and females from two different data sets. The test case returned false when the exact float value was compared with the output so to solve the issue, a float value in a very small range was compared with the output.

The eighth test case confirmed that the allocate properly placed students into their engineering streams first by free choice, then in order of GPA.

The second type of test cases checked for correct behaviour of program when weird inputs, like missing files, or incomplete data was passed to the functions. This type included the first, seventh, and ninth test cases. The number of test cases of this type required to build a modest amount of confidence in the program depends on the amount of assumptions. Since quite a few things were assumed, less test cases of this type were needed. All these test cases passed after modification to the the code. The rationale behind them, problems uncovered while testing, and a brief description of each test case will be discussed.

In the first test case, the functions in ReadAllocationData were tested instead of those in CalcModule. It tested for program response when all the files were empty. The functions responded correctly by returnning empty dictionaries or lists.

In the seventh test case, the average function was checked to see behaviour when tried to get the average of all the females, but there are no females in the data. Based on this test case failing for the first time, the code had to be modified to return 'None', as well as correctly deal with the fact that there is no one of that gender in the data.

Finally, the ninth test case checked to see whether the allocation function worked correctly when it was given student data and none of them had free choice. This was tested to ensure that the function still operated correctly when one of the files was empty.

All this list of nine test cases covered quite a bit of program functionality and possible errors, building confidence in the robustness and correctless of it, there is still more testing that could have been done to verify these things. This was what could be done in the timing given.

2 Results of Testing Partner's Code

A table outlining the summary of results after running the test script on the partners code is seen below.

Test Case	Result
1	Pass
2	Pass
3	Pass
4	Pass
5	Pass
6	Pass
7	Fail - ZeroDivisionError
8	Pass
9	Pass

Table 1: Table outlining the results of testing partner code

The results of this table are further discussed in section 3.2

3 Discussion of Test Results

The following two subsections will discuss and analyze the test results of the code written by the programmer, and the partners code.

3.1 Problems with Original Code

While doing the testing, two problems with the code became evident. The first is that if there are either no males, or no females, then the average function would divide by 0 and would crash. This was solved by implementing a counter and checking to see if

there were none of that particular gender. The second problem discovered is that in the dictionary of streams and the students in each stream, each student is supposed to represent a dictionary of a student, not just the macid. This was again solved by altering the code.

3.2 Problems with Partner's Code

As can be seen from **??** all the test cases were passed except for the seventh one. The seventh test case check to see how the program behaved when asked the caluclate the average GPA of all the females, yet there were no females in the data.

The error that was gotten was a ZeroDivisionError at line 42 in the partners CalcModule program. This line of code is:

$$aveFemale = sumFemale \ / \ numFemale$$

This line of code divides the total sum by the number of females. However, if there are zero females, the program tries to divide by zero, throwing an error, leading the this particular test case to fail.

4 Critique of Design Specification

The design specification was good in some ways, but poor in others.

Since it had been the first time programming in python in a while, having restrictions on the types of data structures to use, as well as a laid out module interface made it simpler to begin programming. Also, since the specification was in natural language, it made it easy to understand, then implement the basic idea of what the modules were supposed to do.

The specification was poor in that because it was given in natural language, the programmer had to make a lot of assumptions about things like operations on the data and how to deal with errors. The specification should have been made more specific and explicit in how exactly what the interace was so less assumptions would have to be made.

A proposed change for the design is to give the specification both in natural language and in a formal way. This was the basic idea of the program can be understood easily, and the specifics can also be understood how to be dealt with. Specifying the implentation, especially for a first assignment is good and should be kept that way.

5 Answers to Questions

(a) The average function can be made more general by instead calculating the average GPA of students that possess a specific characteristic in common. Right now, the

average function just gets the average of either the males or females, but a more general function would be able to return the average of all students with the same first stream in common for example.

Along the same line of thinking, a more general sort function would be able to sort students by any specified characteristic like last name, macid, or first stream choice.

(b) In this context aliasing means you can have multiple names for the same dictionary. In general, aliasing could be a concern with dictionaries because the dictionary could be accidently modified or changed due to not knowing the same data structure is pointed to by two different names. This can be guarded against in two ways. The first is by using a non-mutable data stucture, like a tuple. This way it cannot be modified, only read from. The second is by creating copies of the dictionary and only modifiying the copies as needed.

(c) Even though the assignment did not require testing of ReadAllocationData, the first test case tested how the function within that module respond to empty files. It was tested because it was thought that it would be important to ensure that the functions responsible for getting the data work properly and perform as intended

However some more test cases that could be done to build confidence in ReadAllocationData are:

- No filenames are passed to function.
- Filenames that are passed to function are incorrect.
- Files that are passed to the function are empty.
- Files that contain the correct data, but in a format that is not expected by the function, e.g. tabs instead of newline characters, or missing the students macid.
- Files that are in the correct format, but contain incorrect info, e.g. two stream choices instead of three, or GPA greater than 12.0.

Of the two modudles, CalcModule was chosen over ReadAllocationData to test because CalcModule requires ReadAllocationData to be correct. By testing just CalcModule, ReadAllocationData is also being tested at the same time because if the first doesn't work, the second one wouldn't function correctly either.

Also, CalcModule does more interesting things with the data and performs operations on it making the test cases that would come out of it more unique and less obvious.

(d) The problem with using strings for a set of finite elements is that it leaves more room for error. Although the same functionality could be easily implemented with strings,

using integers or something related would reduces the margin of error in the program. This new implementation would help catch errors early, rather than miss data when performing operations later. For example, having 0 = male, and 1 = female, means that when the data is read from the file, if it is not either a 1 or 0, an error is thrown and the mistake could be corrected. If they were instead represented as before, and the mistake was allowed to propagate, when the average function is called, it would miss the person with the spelling mistake representing their gender.

(e) Dictionaries aren't the only option to implement records and structs in Python, some other options include tuples, a custom class, collections.namedtuple class, typing.NamedTuple class, and the struct.Struct class. Based on the information I learned from: https://dbader.org/blog/records-structs-and-data-transfer-objects-in-python I would recomend changing the data structure to either collections.namedtuple or typing.NamedTuple. The reason for this is because similar to classes, both data structures allow for using the same format and ensuring that the fields and their data are correct. The two structures are also immutable and can be accessed through unique identifiers. They have all the properties we want of dictionaries but are safer to use. In terms of implementation in the code, each student would be a collections.namedtuple, or typing.NamedTuple, and each student would be stored in a list.

(f) If the list of strings was changed to a tuple, the CalcModule would not to be changed as the implentation would work the same. If it were changed to another data structure that is indexed by values, it may not need to be changed, but changing it to one that isn't would most likely require the code to be altered.

If students were implemented as a custom class or Abstract Data Type (ADT), if the data structure in the class was changed, the CalcModule code would not need to be altered because of abstraction. The client, which is the program running, does not need to know how the data structure is implemented, rather it just calls the method in the class and uses the value returned by it.

F Code for ReadAllocationData.py

```
## @file ReadAllocationData.py
#  @Joshua Guinness, guinnesj
#  @A module to open files, process, then store the data
#  @January 14-18, 2019

# For these functions I am making the assumption that the the contents in the file are accurate and
    the formatting is correct.
# I am doing error checking on the opening files, but am assuming the rest is correct

# Import for error raising and exiting
import sys


## @brief A function which reads in student data from a file and stores it in a list of dictionaries
#  @param p1 A string corresponding to a filename
#  @return Returns a list of dictionaries of student information
def readStdnts(s):

    # Creates a new list which will store student dictionaries
    student_info = []

    # Try-Except statement for opening the file
    try:

        # Opens the file for reading
        # How to open a file taken from:
            https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files
        f = open(s, 'r')

    except:

        # Exit if the file does not exist or can't be opened.
        # Method seen here: https://python-forum.io/Thread-How-to-exit-after-a-try-exception
        sys.exit("Error opening the file of student data!")

    # Reads in each line, converts to a dictionary, then adds the dictionay to the list
    for line in f:

        # Strips the newline character at the end of the string.
        # Method taken from:
            https://stackoverflow.com/questions/275018/how-can-i-remove-a-trailing-newline-in-python
        line = line.rstrip()

        # Splits each line at each tab into a list
        # How the split strings taken from here:
            #https://www.pythoncentral.io/cutting-and-slicing-strings-in-python/
        student_string = line.split('\t')

        # Splits the choices at commas and spaces
        choices = student_string[5].split(', ')

        # Transforms the current line student to a dictionary
        student_dictionary = {'macid': student_string[0], 'fname': student_string[1], 'lname':
            student_string[2],
                              'gender': student_string[3], 'gpa': float(student_string[4]), 'choices':
                                  choices}

        # Adds the current dictionary to a list of student info
        student_info.append(student_dictionary)

    # Closes the file
    f.close()

    #print(student_info)

    # Return statement
    return student_info


## @brief A function which reads in data about students with free choice from a file and stores it in
    a list
#  @param p1 A string corresponding to a filename
#  @return Returns a list of macids of students with free choice
def readFreeChoice(s):

    # Creates an empty list
```

```
    students_free_choice = []

    # Try-Except block for opening the file
    try:

        # Opens the file
        f = open(s, 'r')

    except:

        sys.exit("Error opening the file of students with free choice!")

    # Iterates through the file, stripping the new line character, and appending each macid to the list
    for line in f:
        line = line.rstrip()
        students_free_choice.append(line)

    # Closes the file
    f.close()

    #print(students_free_choice)

    # Returns the list
    return students_free_choice


## @brief A function which reads in data about department capacities from a file and stores it in a
     dictionary
#  @param p1 A string corresponding to a filename
#  @return Returns a dictionary of departments and their capacity
def readDeptCapacity(s):

    # Create an empty dictionary
    department_capacity = {}

    # Try-Except for opening the file
    try:

        # Opens the file
        f = open(s, 'r')

    except:

        sys.exit("Error opening the file of department capacities!")

    # Iterates through the file line by line, splitting at spaces, and adding the department and its
         capacity to a dictionary
    for line in f:
        line = line.rstrip()
        list = line.split(' ')

        # How to add key's and values to a dictionary got from:
            https://docs.python.org/3/tutorial/datastructures.html#dictionaries
        department_capacity[list[0]] = int(list[1])

    # Closes the file
    f.close()

    #print(department_capacity)

    # Returns a dictionary of the departments and their capacity
    return department_capacity
```

G Code for CalcModule.py

```
## @file CalcModule.py
#  @Joshua Guinness
#  @A to preform operations on the data got from the ReadAllocationData.py
#  @January 14-18, 2019

# Imports
from ReadAllocationData import *
import sys


# For this function I am assuming that the information contained in S is corrected and formatted
    properly

## @brief A function which sorts the students based on gpa from highest to lowest
#  @param p1 A list of dictionaries of students
#  @return Returns a list of dictionaries of students sorted by gpa
def sort(S):

    # Sorts students by GPA from highest to lowest using Bubble Sort
    # Bubble sort implementation gotten from:
        https://www.geeksforgeeks.org/python-program-for-bubble-sort/
    for i in range (len(S)):
        for j in range (0, len(S)-1-i):

            # How to access stuff from dictionaries gotten from:
                https://www.pythonforbeginners.com/dictionary/how-to-use-dictionaries-in-python
            if (S[j].get('gpa') < S[j+1].get('gpa')):
                swap(S, j, j+1)

    return S


# For this function I am assuming that the information contained in L is correct and formatted properly
# and the value passed for g is either 'male' or 'female.'

## @brief A function which gets the average gpa of either all the male or female students
#  @param p1 A list of dictionaries of students
#  @param p2 A string which is either 'male' or 'female'
#  @return Returns the average gpa
def average(L, g):

    total_sum = 0
    counter = 0

    # Iterates through each student, checks the gender, then adds the gpa to a sum if the gender
        matches
    for i in range(len(L)):
        if (L[i].get('gender') == g):
            total_sum += L[i].get('gpa')
            counter += 1

    # Returns none if there are no students with that gender
    if (counter == 0):
        return None

    # Returns the total sum gpa divided by the number of students of that gender
    else:
        average_gpa = total_sum / counter
        return average_gpa


# For this function I am assuming that the contents of S, F, and C are correct and formatted properly.
# I am also assuming that their is enough department capacity to handle all the people with free choice
# Finally I am assuming that their is enough department capacity to grant everyone at least their
    third choice

## @brief A function which allocates the students into their streams.
#  @details It first allocates those with free choice, then it allocates those without free choice in
    order of gpa
#  @param p1 A list of dictionaries of students
#  @param p2 A list of students with free choice
#  @param p3 A dictionary of department capacities
#  @return Returns a dictionary of allocated students
def allocate(S, F, C):

    # Creation of the allocation dictionary
```

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

    # Students are now sorted from highest GPA to lowest
    sorted_students = sort(S)

    # Allocate students with free choice

    # Iterates through all the students with free choice, first checking to make sure their gpa is >=
        4.0,
    # if not then it does not allocate them. If it meets the gpa requirements, it allocates the
        student to
    # the program of their first choice by adding their dictionary to the allocation dictionary
    for i in F:
        student_choice = ""
        for j in sorted_students:
            if (i == j.get('macid') and j.get('gpa') >= 4.0):
                student_choice = j.get('choices')[0]
                allocation_dictionary.get(student_choice).append(j)

                # Reduces the department capacity of the most recent choice by one
                C[student_choice] = C[student_choice]-1

    # Allocate all the rest of the students.

    # First check to make sure the student has a gpa >= 4.0, and they were not allocated in previously
        by free choice,
    # not allocating them if they don't. If they do, it allocates them by order of gpa, by placing
        them into
    # their first, second, or third choice by department capacity. It then decreases the department
        capacity by one.
    for i in sorted_students:

        # Checking if a value is in a list from:
            https://thispointer.com/python-how-to-check-if-an-item-exists-in-list-search-by-value-or-condition/
        if ((i.get('gpa') >= 4.0) and (i.get('macid') not in F)):
            first_choice = i.get('choices')[0]
            second_choice = i.get('choices')[1]
            third_choice = i.get('choices')[2]
            if (C.get(first_choice) >= 1):
                allocation_dictionary.get(first_choice).append(i)
                C[first_choice] = C[first_choice]-1
            elif (C.get(second_choice) >= 1):
                allocation_dictionary.get(second_choice).append(i)
                C[second_choice] = C[second_choice]-1
            elif (C.get(third_choice) >= 1):
                allocation_dictionary.get(third_choice).append(i)
                C[third_choice] = C[third_choice]-1

    # Returns the dicionary of allocated students
    return allocation_dictionary

# Function to swap two elements in a list. This function is used to implement
# bubble sort in the sort() function above

## @brief A function which swaps two elements in a list
#  @param a list
#  @param p2 An element
#  @param p3 An element
#  @return The list
def swap(list, elem1, elem2):

    temp = list[elem1]
    list[elem1] = list[elem2]
    list[elem2] = temp

    return list
```

H Code for testCalc.py

```
## @file testCalc.py
#  @Joshua Guinness, guinnesj
#  @A module to test CalcModule.py
#  @January 14-18, 2019

# Imports
from CalcModule import *
from ReadAllocationData import *
import unittest

# Test Case 1:
# Checks response of functions when the files are empty.

if (readStdnts('TestingData/rawStudentData2') == []
    and readFreeChoice('TestingData/freeChoice2') == []
    and readDeptCapacity('TestingData/rawDepartmentData2') == {}):
    print("Test Case 1: Pass")
else:
    print("Test Case 1: Fail")


# Test Case 2:
# Testing whether sort function sorts students by gpa properly
student_dictionaries = readStdnts('TestingData/rawStudentData')
students_with_free_choice = readFreeChoice('TestingData/freeChoice')
department_capacity = readDeptCapacity('TestingData/rawDepartmentData')

sorted_student_dictionaries = sort(student_dictionaries)

if (sorted_student_dictionaries == [{'gender': 'male', 'gpa': 12.0, 'choices': ['software',
     'electrical', 'mechanical'], 'lname': 'Guinness', 'fname': 'Joshua', 'macid': 'guinnesj'},
                                    {'gender': 'male', 'gpa': 11.9, 'choices': ['materials',
                                        'engphys', 'software'], 'lname': 'Guinness', 'fname':
                                        'Daniel', 'macid': 'danielg'},
                                    {'gender': 'female', 'gpa': 11.2, 'choices': ['chemical',
                                        'electrical', 'engphys'], 'lname': 'Khani', 'fname':
                                        'Sophia', 'macid': 'khanis1'},
                                    {'gender': 'male', 'gpa': 9.8, 'choices': ['civil', 'mechanical',
                                        'materials'], 'lname': 'Tara', 'fname': 'Mickey', 'macid':
                                        'taram1'},
                                    {'gender': 'female', 'gpa': 8.6, 'choices': ['mechanical',
                                        'civil', 'materials'], 'lname': 'Brown', 'fname': 'Grace',
                                        'macid': 'browng'},
                                    {'gender': 'female', 'gpa': 7.1, 'choices': ['engphys',
                                        'chemical', 'civil'], 'lname': 'Kass', 'fname': 'Cassie',
                                        'macid': 'cassyk'},
                                    {'gender': 'male', 'gpa': 6.4, 'choices': ['software', 'civil',
                                        'electrical'], 'lname': 'So', 'fname': 'Leon', 'macid':
                                        'leon2'},
                                    {'gender': 'female', 'gpa': 6.3, 'choices': ['mechanical',
                                        'electrical', 'software'], 'lname': 'Smith', 'fname':
                                        'Roxanne', 'macid': 'smithr'},
                                    {'gender': 'female', 'gpa': 4.4, 'choices': ['mechanical',
                                        'electrical', 'chemical'], 'lname': 'Xiao', 'fname': 'Joy',
                                        'macid': 'joyxiao'},
                                    {'gender': 'male', 'gpa': 2.2, 'choices': ['software',
                                        'materials', 'civil'], 'lname': 'Choy', 'fname': 'Timothy',
                                        'macid': 'choyt2'}]):
    print("Test Case 2: Pass")
else:
    print("Test Case 2: Fail")

# Test Case 3:
# Testing whether average function returns the correct average for males
student_dictionaries = readStdnts('TestingData/rawStudentData')

average_gpa = average(student_dictionaries, 'male')

if (average_gpa > 8.45 and average_gpa < 8.47):
    print("Test Case 3: Pass")
else:
    print("Test Case 3: Fail")

# Test Case 4:
# Testing whether average function returns the correct average for females
student_dictionaries = readStdnts('TestingData/rawStudentData')
```

```
average_gpa = average(student_dictionaries, 'female')

if (average_gpa > 7.51 and average_gpa < 7.53):
    print("Test Case 4: Pass")
else:
    print("Test Case 4: Fail")

# Test Case 5:
# Testing whether average function returns the correct average for males on a different data set
student_dictionaries = readStdnts('TestingData/rawStudentData3')

average_gpa = average(student_dictionaries, 'male')

if (average_gpa > 6.73 and average_gpa < 6.75):
    print("Test Case 5: Pass")
else:
    print("Test Case 5: Fail")

# Test Case 6:
# Testing whether average function returns the correct average for females on a different data set
student_dictionaries = readStdnts('TestingData/rawStudentData3')

average_gpa = average(student_dictionaries, 'female')

if (average_gpa > 9.21 and average_gpa < 9.23):
    print("Test Case 6: Pass")
else:
    print("Test Case 6: Fail")

# Test Case 7:
# Testing whether average function returns the correct average when there are no students of the
    gender called
student_dictionaries = readStdnts('TestingData/rawStudentData4')

average_gpa = average(student_dictionaries, 'female')

if (average_gpa == None):
    print("Test Case 7: Pass")
else:
    print("Test Case 7: Fail")


# Test Case 8:
# Testing whether allocate function allocates the students properly
student_dictionaries = readStdnts('TestingData/rawStudentData')
students_with_free_choice = readFreeChoice('TestingData/freeChoice')
department_capacity = readDeptCapacity('TestingData/rawDepartmentData')

allocation_dictionary = allocate(student_dictionaries, students_with_free_choice, department_capacity)

if (allocation_dictionary == {'engphys': [{'gender': 'female', 'gpa': 7.1, 'choices': ['engphys',
    'chemical', 'civil'], 'lname': 'Kass', 'fname': 'Cassie', 'macid': 'cassyk'}],
                              'civil': [{'gender': 'male', 'gpa': 9.8, 'choices': ['civil',
                                  'mechanical', 'materials'], 'lname': 'Tara', 'fname': 'Mickey',
                                  'macid': 'taram1'}],
                              'chemical': [{'gender': 'female', 'gpa': 11.2, 'choices': ['chemical',
                                  'electrical', 'engphys'], 'lname': 'Khani', 'fname': 'Sophia',
                                  'macid': 'khanis1'}],
                              'materials': [{'gender': 'male', 'gpa': 11.9, 'choices': ['materials',
                                  'engphys', 'software'], 'lname': 'Guinness', 'fname': 'Daniel',
                                  'macid': 'danielg'}],
                              'electrical': [{'gender': 'female', 'gpa': 4.4, 'choices':
                                  ['mechanical', 'electrical', 'chemical'], 'lname': 'Xiao', 'fname':
                                  'Joy', 'macid': 'joyxiao'}],
                              'mechanical': [{'gender': 'female', 'gpa': 8.6, 'choices':
                                  ['mechanical', 'civil', 'materials'], 'lname': 'Brown', 'fname':
                                  'Grace', 'macid': 'browng'},
                                             {'gender': 'female', 'gpa': 6.3, 'choices':
                                                 ['mechanical', 'electrical', 'software'], 'lname':
                                                 'Smith', 'fname': 'Roxanne', 'macid': 'smithr'}],
                              'software': [{'gender': 'male', 'gpa': 12.0, 'choices': ['software',
                                  'electrical', 'mechanical'], 'lname': 'Guinness', 'fname':
                                  'Joshua', 'macid': 'guinnesj'},
                                           {'gender': 'male', 'gpa': 6.4, 'choices': ['software',
                                               'civil', 'electrical'], 'lname': 'So', 'fname':
                                               'Leon', 'macid': 'leon2'}]}):
    print("Test Case 8: Pass")
else:
    print("Test Case 8: Fail")
```

```
# Test Case 9:
# Testing whether allocate function allocates the students properly when no one has free choice
student_dictionaries = readStdnts('TestingData/rawStudentData')
students_with_free_choice = readFreeChoice('TestingData/freeChoice2')
department_capacity = readDeptCapacity('TestingData/rawDepartmentData')

allocation_dictionary = allocate(student_dictionaries, students_with_free_choice, department_capacity)

if (allocation_dictionary == {'engphys': [{'gender': 'female', 'gpa': 7.1, 'choices': ['engphys',
     'chemical', 'civil'], 'lname': 'Kass', 'fname': 'Cassie', 'macid': 'cassyk'}],
                              'civil': [{'gender': 'male', 'gpa': 9.8, 'choices': ['civil',
                                   'mechanical', 'materials'], 'lname': 'Tara', 'fname': 'Mickey',
                                   'macid': 'taram1'}],
                              'chemical': [{'gender': 'female', 'gpa': 11.2, 'choices': ['chemical',
                                   'electrical', 'engphys'], 'lname': 'Khani', 'fname': 'Sophia',
                                   'macid': 'khanis1'}],
                              'materials': [{'gender': 'male', 'gpa': 11.9, 'choices': ['materials',
                                   'engphys', 'software'], 'lname': 'Guinness', 'fname': 'Daniel',
                                   'macid': 'danielg'}],
                              'electrical': [{'gender': 'female', 'gpa': 4.4, 'choices':
                                   ['mechanical', 'electrical', 'chemical'], 'lname': 'Xiao', 'fname':
                                   'Joy', 'macid': 'joyxiao'}],
                              'mechanical': [{'gender': 'female', 'gpa': 8.6, 'choices':
                                   ['mechanical', 'civil', 'materials'], 'lname': 'Brown', 'fname':
                                   'Grace', 'macid': 'browng'},
                                             {'gender': 'female', 'gpa': 6.3, 'choices':
                                                  ['mechanical', 'electrical', 'software'], 'lname':
                                                  'Smith', 'fname': 'Roxanne', 'macid': 'smithr'}],
                              'software': [{'gender': 'male', 'gpa': 12.0, 'choices': ['software',
                                   'electrical', 'mechanical'], 'lname': 'Guinness', 'fname':
                                   'Joshua', 'macid': 'guinnesj'},
                                           {'gender': 'male', 'gpa': 6.4, 'choices': ['software',
                                                'civil', 'electrical'], 'lname': 'So', 'fname':
                                                'Leon', 'macid': 'leon2'}]}):
    print("Test Case 9: Pass")
else:
    print("Test Case 9: Fail")

print("All Test Cases Complete")
```

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
```