Assignment 1 Solution

Shunbo Cui 400141410

January 25, 2019

In this assignment 2 modules are created. The first one ReadAllocationData.py includes 3 functions for reading data from 3 different files to get the student list, student with free choice and the department capacity list. The other module CalcModule.py includes 3 functions for calculations of the data, including the average gpa, sorting students according to gpa, and allocating the students to different departments.

1 Testing of the Original Program

(a) First of all, 3 functions for comparing the lists, floats and dictionaries are created for the following test cases.
The first test case is to test how sort woring on empty lists. After sorting, the empty list should still be empty. The case can be tested by comparing if the sorted list remains the same. The original program passed.
The second test case for sorting is a normal process on a simple list. The list only includes macid, gender and gpa of the students. The function can sort the dictionaries of students according to the key 'gpa' in descending order. Then the test function compare the output list with the expected test to tell if the test passed.

(b) For the average function I made 3 test cases. The first 2 ones have the same input list. For the first case the input argument is 'male' while for the second case it is 'female'. For both cases, by comparing if the output float is similar to the expected one, we can tell if the function passes the test. Because two double numbers cannot be exactly the same so I implement this by calculating the difference between the output value and the expected value. If the difference is smaller than 0.001, the result is considered to be correct. The function passed with both male and female.
The third test can tell how the average function works on the empty list. If there is no special case for empty list in the average function, there will be an error caused

by dividing by zero, because the count of the students is zero. In the function there is a section which can return 0 directly if the input list is empty.

(c) The first test case for the allocate function is testing a normal input list. There is no student with free choice in this case so everyone is sorted by gpa and allocated. In the next case the input list is empty. The output is still an empty list as expected.
In the third test case all students in the list have free choice. I did not set the rule for allocating such students so the order in the output depends on the order in the input list.
The purpose of the 4th test case is to test how the function will allocate the students when their first choice department is full. I set the capacity of civil department to 3 and filled it with 3 students with free choice. Then the other students are sorted and allocated to their second choice.
In the 5th case, the civil department is full and all other department capacities are set to 0. That means some students are not able to be allocated to any of their choices, and an error message should be printed. As expected the macid of the students who are not allocated were printed out.

2 Results of Testing Partner's Code

Test passed, lists are equal for sorting empty list
Test passed, lists are equal for sorting added empty list
Test passed, floats are equal for calculating average gpa for male
Test passed, floats are equal for calculating average gpa for female
Test failed, floats are not equal for calculating average gpa for empty list
Test passed, directories are equal for allocating
Test passed, directories are equal for allocating empty list
WARNING: Capacity below 0 for civil
Test failed, directories are not equal for allocating (all freechoice)
Test failed, directories are not equal for allocating (first choice full)
ERROR: Could not allocate macid: F, all 3 program choices are full!
ERROR: Could not allocate macid: E, all 3 program choices are full!
ERROR: Could not allocate macid: D, all 3 program choices are full!
Test failed, directories are not equal for allocating (capacities full)

3 Discussion of Test Results

3.1 Problems with Original Code

Order of allocating: in the instructions students with free choice are not sorted with gpa so in the allocated list they are not in order of gpa. If I compare the output with the expected list, it will not pass the test because of the order. So I can put a sort function inside of the test driver and sort the allocated list again. After that I can comare it with the expected one.

3.2 Problems with Partner's Code

The partner's file does not pass the third test on the average function. In the function when the length of the list is equal or smaller than 0, the function will return -1, which is not 0. However in my test case, the expected output is 0 which is not -1 so an error message is showed.
The allocate function passed the first 2 test cases, and failed in the last 3 ones. In the 3rd test, there are 6 students with the same first choice and I set the capacity to 5, which means 1 student with free choice will not be allocated to his first choice. The partner announced that the function will not consider this condition, which can make the students in the department will exceed the capacity. In some special conditions, when many students with free choice choose the same department as their 1st choice, the capacity will be overflow. However this is not going to really happen since the capacity will not be that small.
For the 4th and 5th test of the allocate function, the 3 students without free choice are allocated to their 2nd choice because the 1st choice is full, which is the same as expected. However in the test case, the department is filled with students with free choice. In the test case students with free choice are not expected to be allocated according to their gpa. The order of those students in allocated list is expected to be the same as the input list. However in the function, all students including those with free choice are allocated according to the gpa. The order of students with free choice made the result different from expected.

4 Critique of Design Specification

I think there are not enough rules and Specifications about how the functions should work. For example when allocating the students, if the capacity is full, should the students with free choice be allocated to their second choice? Such problem make the requirement kind of ambiguous, but it also encourage us to make these conditions clear by ourselves.

5 Answers to Questions

(a) The current function take the parameter g and tells if it is male or female, and calculate the gpa according to it. To make the function more general, the parameter does not need to be limited in the 2 string. It can be any other string for student information, as long as the string is in the input list. First we can put a line testing if the parameter string is a information key in the dictionary. Then use a line to set g as the rule to judge if the student should be taken into calculation. There can also be one more parameter for the key of the data be calculated. In this case it is 'gpa' but in the function it can be set to any other key. For example there is a key 'testgrade' in the student dictionary and the content is a double, the function can take a parameter 'testgrade' and calculate the average of those data.
For the sort function it can also take a string parameter as the rule of sorting. According to any other keys in the dictionary with a double or integer as value, the function can sort it.

(b) There is aliasing when we have two variable names with the same value. The dictionary is mutable in Python. To pretend this, there should be no same keys in a dictioanry.

(c) I can make empty files to test if the functions can return empty lists and dictionaries. I can take off some keys and their values to test if the functions can read the rest normally. For the readStdnts I can change the data type of the numbers. There should be an error message when the data of gpa is not float. Also I can change the capacity data to string or float to test if there is any error message.
I think the reason why we are not testing the ReadAllocationData.py is the functions in it are not flexible. The functions are written based on the format of the input file. If the input format is changed, the functions won't work. For example if I switch the gpa and macid in the file, the output list will be wrong. In CalcModule.py, all functions take the parameter and knows which data to calculate.

(d) There will not be repeating element in a set. If we want to put multiple same strings in the set, there will be only one left and all others are deleted.
If the strings are used as keys in the set, the value of them will not be passed into the set. We can use a list or dictionary instead. We can also use other types instead of string.

(e) Tuple is simialr to list in Python. It can store data and can also be accessed. The difference is we cannot modify the element in it. So it is not as convenient as dictionary in this case. If we want more functions in the module, we can use class.

(f) It does not need to be modified since we can access the elements in tuple in the same way as a list.
We do not need to modify the CalcModule.py when using the class. The output will remain the same.

F Code for ReadAllocationData.py

```
## @file ReadAllocationData.py
#  @title ReadAllocationData
#  @author Shunbo Cui
#  @date 1/17/2019

## @brief This function reads detailed information of students.
#  @param s Name of list file
#  @return Return the list of students
def readStdnts(s):
    f = open('1.txt', 'r', -1, 'utf-8-sig')
    n = 0
    S = []
    for line in f.readlines():
        v = line.strip().split(':')
        result = {}
        choices = []
        result['macid'] = v[0]
        result['fname'] = v[1]
        result['lname'] = v[2]
        result['gender'] = v[3]
        result['gpa'] = float(v[4])
        choices.append(v[5])
        choices.append(v[6])
        choices.append(v[7])
        result['choices'] = choices
        S.append(result)
    f.close()
    return(S)

## @brief This function reads macid of students with freechoice.
#  @param s Name of list file
#  @return Return the list of students with free choice
def readFreeChoice(s):
    f = open(s, 'r', -1, 'utf-8-sig')

    F = []
    for line in f.readlines():
        v = line.strip()
        F.append(v)
    f.close()
    return (F)

## @brief This function reads capacity of each department from the file
#  @param s Name of list file
#  @return Return list of departments and their capacity
def readDeptCapacity(s):
    f = open(s, 'r', -1, 'utf-8-sig')
    C = {}
    count = 0
    for line in f.readlines():
        v = line.strip().split(':')
        C[v[0]] = int(v[1])
    return (C)
```

G Code for CalcModule.py

```
## @file CalcModule.py
#  @title CalcModule
#  @author Shunbo Cui
#  @date 1/18/2019

## @brief This function sorts the list of students according to their gpa.
#  @param S List of students
#  @return Return the sorted list of students
def sort(S):
    def gpa(s):
        return s['gpa']
    sortedlist = sorted(S,key = gpa,reverse = True)
    return (sortedlist)

## @brief This function calculates the average gpa in a list according to the gender.
#  @param L List of students
#  @param g The determined gender
#  @return Return the average gpa
def average(L, g):
    genderlist = []
    sum = 0
    count = 0
    if L == []:
        return 0
    for result in L:
        if result['gender'] == g:
            sum += result['gpa']
            count += 1
    average = sum / count
    return average

## @brief This function allocated students to departments according to the gpa and the capacities.
#  @param S List of all students
#  @param F List of students with free choice
#  @return Return the allocated list
def allocate(S, F, C):
    #Creating empty lists for each department
    civillist = []
    civilcount = 0

    chemicallist = []
    chemicalcount = 0

    electricallist = []
    electricalcount = 0

    mechanicallist = []
    mechanicalcount = 0

    softwarelist = []
    softwarecount = 0

    materialslist = []
    materialscount = 0

    engphyslist = []
    engphyscount = 0

    #Createing a list for students with free choice
    freelist = []

    t = S.copy()
    for allstudent in t:
        if allstudent['macid'] in F:
            S.remove(allstudent)
            if allstudent['gpa'] > 4:
                freelist.append(allstudent)



    #Allocate the students with free choice according to their first choice
    for freestudent in freelist:
        if freestudent['choices'][0] == 'civil' and civilcount < C['civil']:
            civillist.append(freestudent)
            civilcount += 1
        elif freestudent['choices'][0] == 'chemical'and chemicalcount < C['chemical']:
```

```
            chemicallist.append(freestudent)
            chemicalcount +=1
        elif freestudent['choices'][0] == 'electrical'and electricalcount < C['electrical']:
            electricallist.append(freestudent)
            electricalcount += 1
        elif freestudent['choices'][0] == 'mechanical'and mechanicalcount < C['mechanical']:
            mechanicallist.append(freestudent)
            mechanicalcount += 1
        elif freestudent['choices'][0] == 'software'and softwarecount < C['software']:
            softwarelist.append(freestudent)
            softwarecount += 1
        elif freestudent['choices'][0] == 'materials'and materialscount < C['materials']:
            materialslist.append(freestudent)
            materialscount += 1
        elif freestudent['choices'][0] == 'engphys'and engphyscount < C['engphys']:
            engphyslist.append(freestudent)
            engphyscount += 1
        #If the capacity of any department is not enough for students with free choice
        else:
            print('Student %s with freechoice is not allocated bacause the first choice department is full' %
                  (freestudent['macid']))

sort(S)

#Allocate the students without free choice according to their first choice when the department is not full
for unfreestudent in S:
    if unfreestudent['gpa'] > 4.0:
        if unfreestudent['choices'][0] == 'civil'and civilcount < C['civil']:
            civillist.append(unfreestudent)
            civilcount += 1
        elif unfreestudent['choices'][0] == 'chemical'and chemicalcount < C['chemical']:
            chemicallist.append(unfreestudent)
            chemicalcount +=1
        elif unfreestudent['choices'][0] == 'electrical'and electricalcount < C['electrical']:
            electricallist.append(unfreestudent)
            electricalcount += 1
        elif unfreestudent['choices'][0] == 'mechanical'and mechanicalcount < C['mechanical']:
            mechanicallist.append(unfreestudent)
            mechanicalcount += 1
        elif unfreestudent['choices'][0] == 'software'and softwarecount < C['software']:
            softwarelist.append(unfreestudent)
            softwarecount += 1
        elif unfreestudent['choices'][0] == 'materials'and materialscount < C['materials']:
            materialslist.append(unfreestudent)
            materialscount += 1
        elif unfreestudent['choices'][0] == 'engphys'and engphyscount < C['engphys']:
            engphyslist.append(unfreestudent)
            engphyscount += 1
        #If the department is full, allocate them according to their second choice
        elif unfreestudent['choices'][1] == 'civil'and civilcount < C['civil']:
            civillist.append(unfreestudent)
            civilcount += 1
        elif unfreestudent['choices'][1] == 'chemical'and chemicalcount < C['chemical']:
            chemicallist.append(unfreestudent)
            chemicalcount +=1
        elif unfreestudent['choices'][1] == 'electrical'and electricalcount < C['electrical']:
            electricallist.append(unfreestudent)
            electricalcount += 1
        elif unfreestudent['choices'][1] == 'mechanical'and mechanicalcount < C['mechanical']:
            mechanicallist.append(unfreestudent)
            mechanicalcount += 1
        elif unfreestudent['choices'][1] == 'software'and softwarecount < C['software']:
            softwarelist.append(unfreestudent)
            softwarecount += 1
        elif unfreestudent['choices'][1] == 'materials'and materialscount < C['materials']:
            materialslist.append(unfreestudent)
            materialscount += 1
        elif unfreestudent['choices'][1] == 'engphys'and engphyscount < C['engphys']:
            engphyslist.append(unfreestudent)
            engphyscount += 1
        #If the department is full, allocate them according to their third choice
        elif unfreestudent['choices'][2] == 'civil'and civilcount < C['civil']:
            civillist.append(unfreestudent)
            civilcount += 1
        elif unfreestudent['choices'][2] == 'chemical'and chemicalcount < C['chemical']:
            chemicallist.append(unfreestudent)
            chemicalcount +=1
        elif unfreestudent['choices'][2] == 'electrical'and electricalcount < C['electrical']:
            electricallist.append(unfreestudent)
            electricalcount += 1
```

```
            elif unfreestudent['choices'][2] == 'mechanical'and mechanicalcount < C['mechanical']:
                mechanicallist.append(unfreestudent)
                mechanicalcount += 1
            elif unfreestudent['choices'][2] == 'software'and softwarecount < C['software']:
                softwarelist.append(unfreestudent)
                softwarecount += 1
            elif unfreestudent['choices'][2] == 'materials'and materialscount < C['materials']:
                materialslist.append(unfreestudent)
                materialscount += 1
            elif unfreestudent['choices'][2] == 'engphys'and engphyscount < C['engphys']:
                engphyslist.append(unfreestudent)
                engphyscount += 1
            #If the department of every department in the choice list is full
            else:
                print('Student %s  is not allocated bacause the departments are full' % (unfreestudent['macid']))

    allocated = {}
    allocated['civil'] = civillist
    allocated['chemical'] = chemicallist
    allocated['electrical'] = electricallist
    allocated['mechanical'] = mechanicallist
    allocated['software'] = softwarelist
    allocated['materials'] = materialslist
    allocated['engphys'] = engphyslist
    return allocated
```

H Code for testCalc.py

```
import CalcModule
def comparelist(list1, list2, name):
    if list1 == list2:
        print("Test passed, lists are equal for %s" % (name))
    else:
        print("Test failed, lists are not equal for %s" % (name))

def comparefloat(x, y, name):
    temp = x - y;
    if abs(temp) < 0.001:
        print("Test passed, floats are equal for %s" % (name))
    else:
        print("Test failed, floats are not equal for %s" % (name))

def comparedirectory(dir1, dir2, name):
    if dir1 == dir2:
        print("Test passed, directories are equal for %s" % (name))
    else:
        print("Test failed, directories are not equal for %s" % (name))

def sorttest1():
    emptylist = []
    sortedemptylist = CalcModule.sort(emptylist)
    comparelist(sortedemptylist, emptylist, 'sorting empty list')

def sorttest2():
    unsortedlist = [{'macid': 'A', 'gender': 'male', 'gpa': 7.4},
                    {'macid': 'B', 'gender': 'female', 'gpa': 3.7},
                    {'macid': 'C', 'gender': 'female', 'gpa': 8.2},
                    {'macid': 'D', 'gender': 'male', 'gpa': 6.7},
                    {'macid': 'E', 'gender': 'male', 'gpa': 9.1}]
    sortedlist = CalcModule.sort(unsortedlist)
    expected = [{'macid': 'E', 'gender': 'male', 'gpa': 9.1},
                {'macid': 'C', 'gender': 'female', 'gpa': 8.2},
                {'macid': 'A', 'gender': 'male', 'gpa': 7.4},
                {'macid': 'D', 'gender': 'male', 'gpa': 6.7},
                {'macid': 'B', 'gender': 'female', 'gpa': 3.7}]
    comparelist(sortedlist, expected, 'sorting added empty list')

def averagetest1():
    studentlist = [{'macid': 'A', 'gender': 'male', 'gpa': 7.4},
                    {'macid': 'B', 'gender': 'female', 'gpa': 3.7},
                    {'macid': 'C', 'gender': 'female', 'gpa': 8.2},
                    {'macid': 'D', 'gender': 'male', 'gpa': 6.7},
                    {'macid': 'E', 'gender': 'male', 'gpa': 9.1}]
    averageresult = CalcModule.average(studentlist, 'male')
    expected = 7.733
    comparefloat(expected, averageresult, 'calculating average gpa for male')

def averagetest2():
    studentlist = [{'macid': 'A', 'gender': 'male', 'gpa': 7.4},
                    {'macid': 'B', 'gender': 'female', 'gpa': 3.7},
                    {'macid': 'C', 'gender': 'female', 'gpa': 8.2},
                    {'macid': 'D', 'gender': 'male', 'gpa': 6.7},
                    {'macid': 'E', 'gender': 'male', 'gpa': 9.1}]
    averageresult = CalcModule.average(studentlist, 'female')
    expected = 5.95
    comparefloat(expected, averageresult, 'calculating average gpa for female')

def averagetest3():
    studentlist = []
    averageresult = CalcModule.average(studentlist, 'female')
    expected = 0
    comparefloat(expected, averageresult, 'calculating average gpa for empty list')

def allocatetest1():
    studentlist = [{'macid': 'A', 'gender': 'male', 'gpa': 7.4, 'choices': ['electrical', 'software', 'chemical']},
                    {'macid': 'B', 'gender': 'female', 'gpa': 3.7, 'choices': ['civil', 'materials', 'chemical']},
                    {'macid': 'C', 'gender': 'female', 'gpa': 8.2, 'choices': ['engphys', 'materials', 'electrical']},
                    {'macid': 'D', 'gender': 'male', 'gpa': 6.7, 'choices': ['software', 'civil', 'mechanical']},
                    {'macid': 'E', 'gender': 'male', 'gpa': 9.1, 'choices': ['chemical', 'materials', 'engphys']}]
    freelist = []
    capacity = {'civil': 5,
                'chemical': 5,
                'electrical': 5,
                'mechanical': 5,
```

```
                'software': 5,
                'materials': 5,
                'engphys': 5}
    expected = {'civil': [],
                'chemical': [{'macid': 'E', 'gender': 'male', 'gpa': 9.1, 'choices': ['chemical', 'materials', 'engphys']}],
                'electrical': [{'macid': 'A', 'gender': 'male', 'gpa': 7.4, 'choices': ['electrical', 'software',
                     'chemical']}],
                'mechanical': [],
                'software': [{'macid': 'D', 'gender': 'male', 'gpa': 6.7, 'choices': ['software', 'civil', 'mechanical']}],
                'materials': [],
                'engphys': [{'macid': 'C', 'gender': 'female', 'gpa': 8.2, 'choices': ['engphys', 'materials',
                     'electrical']}]}
    allocated = CalcModule.allocate(studentlist, freelist, capacity)
    comparedirectory(allocated, expected, 'allocating')

def allocatetest2():
    studentlist = []
    freelist = []
    capacity = {'civil': 5,
                'chemical': 5,
                'electrical': 5,
                'mechanical': 5,
                'software': 5,
                'materials': 5,
                'engphys': 5}
    expected = {'civil': [],
                'chemical': [],
                'electrical': [],
                'mechanical': [],
                'software': [],
                'materials': [],
                'engphys': []}
    allocated = CalcModule.allocate(studentlist, freelist, capacity)
    comparedirectory(allocated, expected, 'allocating empty list')

def allocatetest3():
    Studentlist = [{'macid': 'A', 'gender': 'male', 'gpa': 7.4, 'choices': ['civil', 'software', 'chemical']},
                   {'macid': 'B', 'gender': 'female', 'gpa': 7.3, 'choices': ['civil', 'materials', 'chemical']},
                   {'macid': 'C', 'gender': 'female', 'gpa': 8.2, 'choices': ['civil', 'materials', 'electrical']},
                   {'macid': 'D', 'gender': 'male', 'gpa': 6.7, 'choices': ['civil', 'software', 'mechanical']},
                   {'macid': 'E', 'gender': 'male', 'gpa': 9.1, 'choices': ['civil', 'materials', 'engphys']},
                   {'macid': 'F', 'gender': 'female', 'gpa': 10.1, 'choices': ['civil', 'mechanical', 'materials']}]
    Freelist = ['A', 'B', 'C', 'D', 'E', 'F']
    Capacity = {'civil': 5,
                'chemical': 5,
                'electrical': 5,
                'mechanical': 5,
                'software': 5,
                'materials': 5,
                'engphys': 5}
    expected = {'civil': [{'macid': 'A', 'gender': 'male', 'gpa': 7.4, 'choices': ['civil', 'software', 'chemical']},
                   {'macid': 'B', 'gender': 'female', 'gpa': 7.3, 'choices': ['civil', 'materials', 'chemical']},
                   {'macid': 'C', 'gender': 'female', 'gpa': 8.2, 'choices': ['civil', 'materials', 'electrical']},
                   {'macid': 'D', 'gender': 'male', 'gpa': 6.7, 'choices': ['civil', 'software', 'mechanical']},
                   {'macid': 'E', 'gender': 'male', 'gpa': 9.1, 'choices': ['civil', 'materials', 'engphys']},
                   ],
                'chemical': [],
                'electrical': [],
                'mechanical': [],
                'software': [],
                'materials': [],
                'engphys': []}
    allocated = CalcModule.allocate(Studentlist, Freelist, Capacity)
    comparedirectory(allocated, expected, 'allocating (all freechoice)')

def allocatetest4():
    Studentlist = [{'macid': 'A', 'gender': 'male', 'gpa': 7.4, 'choices': ['civil', 'software', 'chemical']},
                   {'macid': 'B', 'gender': 'female', 'gpa': 7.3, 'choices': ['civil', 'materials', 'chemical']},
                   {'macid': 'C', 'gender': 'female', 'gpa': 8.2, 'choices': ['civil', 'materials', 'electrical']},
                   {'macid': 'D', 'gender': 'male', 'gpa': 6.7, 'choices': ['civil', 'software', 'mechanical']},
                   {'macid': 'E', 'gender': 'male', 'gpa': 9.1, 'choices': ['civil', 'materials', 'engphys']},
                   {'macid': 'F', 'gender': 'female', 'gpa': 10.1, 'choices': ['civil', 'mechanical', 'materials']}]
    Freelist = ['A', 'B', 'C']
    Capacity = {'civil': 3,
                'chemical': 5,
                'electrical': 5,
                'mechanical': 5,
                'software': 5,
                'materials': 5,
                'engphys': 5}
```

```
    expected = {'civil': [{'macid': 'A', 'gender': 'male', 'gpa': 7.4, 'choices': ['civil', 'software', 'chemical']},
                    {'macid': 'B', 'gender': 'female', 'gpa': 7.3, 'choices': ['civil', 'materials', 'chemical']},
                    {'macid': 'C', 'gender': 'female', 'gpa': 8.2, 'choices': ['civil', 'materials', 'electrical']}],
                'chemical': [],
                'electrical': [],
                'mechanical': [{'macid': 'F', 'gender': 'female', 'gpa': 10.1, 'choices': ['civil', 'mechanical',
                     'materials']}],
                'software': [{'macid': 'D', 'gender': 'male', 'gpa': 6.7, 'choices': ['civil', 'software', 'mechanical']}],
                'materials': [{'macid': 'E', 'gender': 'male', 'gpa': 9.1, 'choices': ['civil', 'materials', 'engphys']}],
                'engphys': []}
    allocated = CalcModule.allocate(Studentlist, Freelist, Capacity)

    comparedirectory(allocated, expected, 'allocating (first choice full)')

def allocatetest5():
    Studentlist = [{'macid': 'A', 'gender': 'male', 'gpa': 7.4, 'choices': ['civil', 'software', 'chemical']},
                    {'macid': 'B', 'gender': 'female', 'gpa': 7.3, 'choices': ['civil', 'materials', 'chemical']},
                    {'macid': 'C', 'gender': 'female', 'gpa': 8.2, 'choices': ['civil', 'materials', 'electrical']},
                    {'macid': 'D', 'gender': 'male', 'gpa': 6.7, 'choices': ['civil', 'software', 'mechanical']},
                    {'macid': 'E', 'gender': 'male', 'gpa': 9.1, 'choices': ['civil', 'materials', 'engphys']},
                    {'macid': 'F', 'gender': 'female', 'gpa': 10.1, 'choices': ['civil', 'mechanical', 'materials']}]
    Freelist = ['A', 'B', 'C']
    Capacity = {'civil': 3,
                'chemical': 0,
                'electrical': 0,
                'mechanical': 0,
                'software': 0,
                'materials': 0,
                'engphys': 0}
    expected = {'civil': [{'macid': 'A', 'gender': 'male', 'gpa': 7.4, 'choices': ['civil', 'software', 'chemical']},
                    {'macid': 'B', 'gender': 'female', 'gpa': 7.3, 'choices': ['civil', 'materials', 'chemical']},
                    {'macid': 'C', 'gender': 'female', 'gpa': 8.2, 'choices': ['civil', 'materials', 'electrical']}],
                'chemical': [],
                'electrical': [],
                'mechanical': [],
                'software': [],
                'materials': [],
                'engphys': []}
    allocated = CalcModule.allocate(Studentlist, Freelist, Capacity)

    comparedirectory(allocated, expected, 'allocating (capacities full)')

sorttest1()
sorttest2()
averagetest1()
averagetest2()
averagetest3()
allocatetest1()
allocatetest2()
allocatetest3()
allocatetest4()
allocatetest5()
```

I Code for Partner's CalcModule.py

```
## @file CalcModule.py
#  @author Dominik Buszowiecki
#  @brief Functions that can manipulate data given by the functions in ReadAllocataionData.py
#  @date 1/18/2019

from ReadAllocationData import *

## @brief Sorts students by there GPA (Descending).
#  @details The function is given a list of students. This list is created by the readStdnts(s) function in
#           ReadAllocationData.py. The function will sort the students
#           by there GPA, but in descending order and then return it. \n \n
#           REFERENCES: \n
#           Sorted function implementation: \n
#           https://stackoverflow.com/questions/72899/how-do-i-sort-a-list-of-dictionaries-by-a-value-of-the-dictionary
# @param S A list of dictionaries where each Dictionary represents a student.
# @return  The original list of students, but sorted by GPA.
def sort(S):
    newList = sorted(S, key=lambda gpa: float(gpa['gpa']), reverse=True)
    return newList


## @brief Calculates the average of either male or female students.
#  @details The function is given a list of students as well as a gender. Based on the gender given, the function
#           computes the average GPA of that gender in the list. If the gender is neither 'male' or 'female'
#           the function will provide an error message and return -1. If there are 0 students of a particular gender
#           and you compute the average of that gender, the function returns -1.
#  @param L A list of dictionaries. Each Dictionary represents a student.
#  @param g A string, must be either 'male' or 'female'.
#  @return Returns the average of the gender as a float.
def average(L,g):
    Flag = False
    if len(L) <= 0:
        return -1
    if g != 'male' and g!= 'female':
        print("ERROR: Gender should be 'male' or 'female' (Note: Case Sensitive)")
        return -1
    else:
        sum=0.0
        total=0
        for i in range(len(L)):
            if L[i]['gender'] == g:
                sum += L[i]['gpa']
                total += 1
                Flag = True
        if Flag == False:
            return -1
        return sum/total


## @brief Allocates students into a 2nd year Engineering program.
#  @details All students that have a GPA below 4.0, are not given a program. Students who were given free choice and
#           have a GPA above 4.0 are automatically assigned there first choice. The function does NOT check the program
#           capacity for free choice students. The rest of the students are given there choice based on there
#           GPA. If there first choice is full, they are given there second, if that is full, there third.
#           If all choices are full for a student, they are not given a program and a error is outputted.
#  @param S A list of dictionaries. Each Dictionary represents a student.
#  @param F A list of strings. Each string represents a macid of a student with Free choice.
#  @param C A dictionary. The dictionary contains the program and its capacity.
#  @return A dictionary that contains the programs and each student in that program.
def allocate (S, F, C):
    temp = sort(S).copy()
    tempC = C.copy()
    programs = {'civil': [], 'chemical': [], 'electrical': [], 'mechanical': [], 'software': [], 'materials': [], 'engphys':
        []}

    i=0
    while i < len(temp):
        if temp[i]['gpa'] < 4.0:
            del temp[i]
        elif temp[i]['macid'] in F:
            programs[temp[i]['choices'][0]].append(temp[i])
            if (tempC[temp[i]['choices'][0]]) <= 0:
                print("WARNING: Capacity below 0 for ", temp[i]['choices'][0])
            tempC[temp[i]['choices'][0]] -= 1
            del temp[i]
        else:
```

```
        i+=1

while (len(temp) > 0):
    if tempC[temp[0]['choices'][0]] > 0:
        programs[temp[0]['choices'][0]].append(temp[0])
        tempC[temp[0]['choices'][0]] -= 1
        del temp[0]
    elif tempC[temp[0]['choices'][1]] > 0:
        programs[temp[0]['choices'][1]].append(temp[0])
        tempC[temp[0]['choices'][1]] -= 1
        del temp[0]
    elif tempC[temp[0]['choices'][2]] > 0:
        programs[temp[0]['choices'][2]].append(temp[0])
        tempC[temp[0]['choices'][2]] -= 1
        del temp[0]
    else:
        print("ERROR: Could not allocate macid:", temp[0]['macid'], ", all 3 program choices are full!")
        del temp[0]

return programs
```

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
```