Assignment 1 Solution

Matthew Braden bradenm

January 25, 2019

Before proceeding with the report, I just want to address a missed comment in the code for the file of testCalc.py. There was meant to be a comment in that file showing the source of the Equal function; lines 8-12 were found from https://tinyurl.com/yavn3sao. As well as in CalcModule.py for the sorted algorithm; line 10 was found from https://tinyurl.com/y9xdx4ep.

1 Indroduction

This assignment consisted of three modules, ReadAllocationData, CalcModule and testCalc. ReadAllocationData consisted of three functions; readStdnts which reads all the students data, including their macid, first name, last name, gender, gpa and three second year choices and creates a dictionary. ReadFreeChoice reads the students who have free choice. ReadDeptCapacity reads how many students can be allocated to each department and creates a dictionary with that information. CalcModule contains functions that returned the average of a certain gender, sorted the students in terms of gpa, and allocates the students for their choices. Finally testCalc involved testing each function in different ways.

2 Testing of the Original Program

For when it came to testing the original program, it was a priority to ensure that all the code modules were working and running with a variety of different inputs. In order to use a variety of test cases, there was a decision to choose 10 different cases, 7 of which were used as Normal Test Cases. These test cases ensure that all modules run smoothly no matter how many inputs used. The other 3 test cases used were Boundary Test Cases. During these tests, the goal is to try to use different inputs to crash the program to see what does and doesn't work.

The first Normal Test Case looked at was to test the sort function. This included an input of a dictionary of students in a random order of gpa. This test will sort the dictionary in order of highest gpa first to lowest gpa. The results for this test case went as expected, as the students were correctly sorted in order of highest to lowest gpa. The second Normal Test Case was for calculating the male students average gpa. A dictionary of students with random gpa and gender is included as the input. This will test the ability of the function to read and calculate the average of the gender inputed. The program correctly calculated the male students gpa, meaning the results were successful. The third, fourth and fifth Normal Test Cases were used to check the code for the ReadAllocationData files to ensure that the code works and runs efficiently. The outcome was a success as a list of students data, a list of free choice students and a list of the departments data were outputted for each function respectively. The sixth Normal Test Case involved trying to test for the average gpa of females, when all the inputs were male. This test will see if the program will know to output 0.0 as the correct output after reading that there are no females in the input. The program correctly outputted the float value of 0.0, thus passing the test. The seventh Normal Test Case includes a students dictionary, free choice students and the departments dictionary as inputs and tests the allocate function to see if each student should get into the correct program of their choice. There was a problem with my original code allocating students to more than one program and therefore the test case failed.

The first Boundary Test Case involved sorting students in decreasing order of gpa when two or more students have the exact same gpa. This test is to check what order the program will put the students in when two or more students has the same gpa. The students were allocated in correct order and when it came to the students with the same gpa the student who was first on the list got sorted first. The second Boundary Test Case involves calculating the female genders average gpa when the input used is spelt Female instead of female. This will test what the code will do when a student has an incorrect spelling in the gender category of their dictionary. This test ignored the student with the misspelling and ended up outputting an average gpa of 0.0 as expected. The third Boundary Test Case is to test what the program does with a student who does not have the minimum 4.0 gpa as it uses the inputs that includes the students dictionary, free choice students and the departments dictionary. The code recognized the student under the 4.0 gpa and did not allocate them. However the same problem occurred from the seventh Normal Test Case and allocated students to more than one program.

When testing my code, a few assumptions were made when deciding about what the programs inputs should be. For the inputs of the readStdnts function there was no need to include the part about saying what each item in the dictionary corresponds to. Other

assumptions made include only using the students macid for the readFirstChoice input, the readDeptCapacity input will be in dictionary format, and the Sort function must contain the exact same inputs as the readStdnts function.

3 Results of Testing Partner's Code

The partners code was able to run with the same test cases that were used in the testCalc function, as both codes followed a very similar format when calling the items of the dictionary. The Normal Test Cases ran smoothly, the results of the test cases were correct as their program was able to achieve the correct outputs. For the Boundary Test Cases their code contained error traps, allowing the program to stop and raise an error for each test case when needed, causing the results of the original code against the partners to have different outputs on these Boundary Test Cases. The partners allocate function was also working unlike the original code allowing all test cases to execute correct outputs.

4 Discussion of Test Results

4.1 Problems with Original Code

Through testing the original code, the testing brought up a few problems. One of these issues was with the allocate function. When this function runs it doesn't assign the students in the proper manner. This was due to the code not having the ability to break the for loop when it assigns the students to their selected programs.

4.2 Problems with Partner's Code

The only problems with the partners code was that they had error traps in it as mentioned above. This only caused a slight issue when testing as the expected results in the testCalc module had to be changed as the error traps will stop the code and get a different output when ran.

5 Critique of Design Specification

The Design Specification has certain areas that could be improved such as the exact format for the input of the readStdnts file. This caused some problems when starting the readStdnts function as the instructions were vague in what we did and didn't need when formatting the students dictionary. Other than that minor issue the rest of the Design Specification explained in great detail what each step must contain and the inputs used

as mentioned above in the results of the code. For next time consider changing the Design Specification in the manner of specifically stating what each input for each function should look like in order for there to be less ambiguity when creating the test cases inputs. This should therefore allow any students code to work in the same manner and not cause any problems when it comes to testing the partners code.

6 Answers to Questions

(a) The average function could be made more general by calculating the average for all students as well as the average for each gender. Sort can also be made to sort each gender in decending order of gpa instead of just the student population as a whole.

(b) Aliasing is when we have two variables that refer to the same dictionary object. This can be a concern with dictionaries as we would have to ensure that if two variables are for a similar object, we must make two objects in order to store the two variables.

(c) For ReadAllocationData. The readStdnts function test cases would've included a file in dictionary format that includes, in order, each students macid, first name, last name, gender, gpa and choices. For readFreeChoice the test case would've included a file with a list of students that are in free choice from the readStdnts dictionary. Finally readDeptCapacity would've used a test case of a file with the departments and their capacity in a dictionary format.

(d) Strings are a case sensitive data type requiring the formatting of the strings to be correct in order to run the code properly. The use of sets would be a more accurate data type as they do not depend on upper or lower case formatting.

(e) Instead of using a dictionary we could also have implemented the records in a Custom Class format. Another way we could have done this is through the typing.namedTuple Class. If I were to change the data structure used in the code modules I would change it to a Custom Class format. This is because a Custom Class is able to store the records in a similar way a dictionary does, but in my opinion a Custom Class is easier to call and append the records at any time.

(f) If the list of strings for choices was changed to be a tuple instead, there will be no need for modifications in the CalcModule module as we would only be slicing a tuple to access a certain value as there is no need for tuple modifications. This is done in the exact format as a list. If a custom class for students was changed to a tuple, there would be a necessary modification to modify the CalcModule module as accessing a custom class is done differently than a list. This involves having to call different functions inside of the class.

G Code for ReadAllocationData.py

```
## @file ReadAllocationData.py
#  @brief Provides read opperations on files
#  @author Matthew Braden
#  @date 1/17/2019

## @brief Reads students data
#  @details Creates a dictionary of students from a file
#  @param s File of students data
def readStdnts(s):
    file = open(s, "r")
    students = file.read()
    students = students.replace("{", "")
    students = students.replace("\'", "")
    students = students.replace("}", "")
    students = students.replace(",", "")
    students = students.replace("[", "")
    students = students.replace("]", "")
    students = students.splitlines()
    file.close()
    studentlst = []
    for i in students:
        studenti = i.split()
        studentdict = {}
        for j in range(len(studenti)):
            if (j == 0):
                studentdict["macid"] = studenti[0]
            elif (j == 1):
                studentdict["fname"] = studenti[1]
            elif (j == 2):
                studentdict["lname"] = studenti[2]
            elif (j == 3):
                studentdict["gender"] = studenti[3]
            elif (j == 4):
                studentdict["gpa"] = float(studenti[4])
            elif (j == 5):
                choices = [studenti[5], studenti[6], studenti[7]]
                studentdict["choices"] = choices

        studentlst.append(studentdict)
    return (studentlst)

## @brief Reads students data
#  @details Creates a dictionary of students from a file for free choice
#  @param s File of students data

def readFreeChoice(s):
    file = open(s, "r")
    fstudent = file.read()
    fstudent = fstudent.replace("\'", "")
    file.close()
    return fstudent.splitlines()
## @brief Reads students data
#  @details Creates a dictionary from students in from a file
#  @param s File of departments data
def readDeptCapacity(s):
    file = open(s, "r")
    departments = file.read()
    departments = departments.replace("{", "")
    departments = departments.replace("\'", "")
    departments = departments.replace("}", "")
    departments = departments.replace(",", "")
    departments = departments.splitlines()
    file.close()
    for i in departments:
        departmenti = i.split()
        department = {}
        for j in range(len(departmenti)):
            if (j == 0):
                department["Civil"] = int(departmenti[1])
            elif (j == 1):
                department["Chemical"] = int(departmenti[3])
            elif (j == 2):
                department["Electrical"] = int(departmenti[5])
            elif (j == 3):
                department["Mechanical"] = int(departmenti[7])
            elif (j == 4):
```

```
            department["Software"] = int(departmenti[9])
        elif (j == 5):
            department["Materials"] = int(departmenti[11])
        elif (j == 6):
            department["EngPhys"] = int(departmenti[13])
return department
```

H Code for CalcModule.py

```
## @file CalcModule.py
#  @brief Preforms calculations on modules
#  @author Matthew Braden
#  @date 1/17/2019
from ReadAllocationData import *
## @brief Sorts students data
#  @details Sorts the students in reverse order
#  @param S List of the dictionaries from readStdnts
def sort(S):
    sortList = sorted(S, key = lambda i: i["gpa"], reverse = True)
    return sortList
#sorted line 10 was found from https://tinyurl.com/y9xdx4ep

## @brief Average of students data
#  @details Calculates the average of each genders gpa's
#  @param L List of the dictionaries from readStdnts
#  @param g Students gender
def average(L, g):
    maleStudents = 0
    femaleStudents = 0
    maleSum = 0
    femaleSum = 0
    for i in L:
        if (i["gender"] == "male"):
            maleStudents += 1
            maleSum += i["gpa"]
        else:
            femaleStudents += 1
            femaleSum += i["gpa"]
    if (g == "male"):
        return maleSum / maleStudents
    else:
        return femaleSum / femaleStudents



## @brief Allocates students
#  @details Sorts students into their selected choice of programs
#  @param S List of the dictionaries from readStdnts
#  @param F List of the dictionaries from readFreeChoice
#  @param C List of the dictionaries from readDeptCapacity
def allocate(S, F, C):
    allocations = {
    }
    Civil = 0
    Chemical = 0
    Electrical = 0
    Mechanical = 0
    Software = 0
    Materials = 0
    EngPhys = 0

    CivLst = []
    ChemLst = []
    ElecLst = []
    MechLst = []
    SoftLst = []
    MatLst = []
    PhysLst = []

    totalCiv = C["Civil"]
    totalChem = C["Chemical"]
    totalElec = C["Electrical"]
    totalMech = C["Mechanical"]
    totalSoft = C["Software"]
    totalMat = C["Materials"]
    totalPhys = C["EngPhys"]

    for i in S:
        if (i["gpa"] >= 4.0):
            for j in F:
                if (i["macid"] == j):
                    if (i["choices"][0] == "Civil"):
                        CivLst.append(i)
                        Civil += 1
                    elif (i["choices"][0] == "Chemical"):
```

```
                    ChemLst.append(i)
                    Chemical += 1
                elif (i["choices"][0] == "Electrical"):
                    ElecLst.append(i)
                    Electrical += 1
                elif (i["choices"][0] == "Mechanical"):
                    MechLst.append(i)
                    Mechanical += 1
                elif (i["choices"][0] == "Software"):
                    SoftLst.append(i)
                    Software += 1
                elif (i["choices"][0] == "Materials"):
                    MatLst.append(i)
                    Materials += 1
                elif (i["choices"][0] == "EngPhys"):
                    PhysLst.append(i)
                    EngPhys += 1
        for k in F:
            if (i["macid"] != k):
                for choice in i["choices"]:
                    if ((choice == "Civil") and (Civil != totalCiv)):
                        CivLst.append(i)
                        Civil += 1
                        break
                    elif ((choice == "Chemical") and (Chemical != totalChem)):
                        ChemLst.append(i)
                        Chemical += 1
                        break
                    elif ((choice == "Electrical") and (Electrical != totalElec)):
                        ElecLst.append(i)
                        Electrical += 1
                        break
                    elif ((choice == "Mechanical") and (Mechanical != totalMech)):
                        MechLst.append(i)
                        Mechanical += 1
                        break
                    elif ((choice == "Software") and (Software != totalSoft)):
                        SoftLst.append(i)
                        Software += 1
                        break
                    elif ((choice == "Materials") and (Materials != totalMat)):
                        MatLst.append(i)
                        Materials += 1
                        break
                    elif ((choice == "EngPhys") and (EngPhys != totalPhys)):
                        PhysLst.append(i)
                        EngPhys += 1
                        break

allocations["Civil"] = CivLst
allocations["Chemical"] = ChemLst
allocations["Electrical"] = ElecLst
allocations["Mechanical"] = MechLst
allocations["Software"] = SoftLst
allocations["Materials"] = MatLst
allocations["EngPhys"] = PhysLst
return allocations
```

I Code for testCalc.py

```
## @file testCalc.py
#  @brief Preforms tests on previous modules
#  @author Matthew Braden
#  @date 1/17/2019
from ReadAllocationData import *
from CalcModule import *

def Equal(test, result, name):
    if test == result:
        print("Test passed, %s == %s, %s " % (test, result, name))
    else:
        print("Test failed, %s != %s, %s " % (test, result, name))
#Line 8-12 for Equal found from https://tinyurl.com/yavn3sao

def SortSameGpa():
    readFile = readStdnts({'bradenm', 'Matthew', 'Braden', 'male', 9.0, ['Software', 'Mechanical',
        'Civil'] },{'patelh' , 'Harsh', 'Patel', 'male', 10.4, ['Software', 'Chemical',
        'EngPhys']},{'barriosm', 'Michael', 'Barrios', 'male', 4.8, ['Software', 'Materials',
        'Civil']})
    Equal(sort(readFile),({'patelh' , 'Harsh', 'Patel', 'male', 10.4, ['Software', 'Chemical',
        'EngPhys']},{'bradenm', 'Matthew', 'Braden', 'male', 9.0, ['Software', 'Mechanical',
        'Civil'] },{'barriosm', 'Michael', 'Barrios', 'male', 4.8, ['Software', 'Materials',
        'Civil']}),
    "Sorting in descending order")
SortSameGpa()

def CalculatingMaleAvg():
    readFile = readStdnts({'bradenm', 'Matthew', 'Braden', 'male', 9.0, ['Software', 'Mechanical',
        'Civil'] },{'patelh' , 'Harsh', 'Patel', 'male', 10.0, ['Software', 'Chemical',
        'EngPhys']},{'barriosm', 'Michael', 'Barrios', 'male', 5.0, ['Software', 'Materials',
        'Civil']})
    Equal(average(readFile, "male"), 8.0, "Calculating male Average")
CalculatingMaleAvg()

def readStdntsTest():
    readFile = readStdnts({'bradenm', 'Matthew', 'Braden', 'male', 9.0, ['Software', 'Mechanical',
        'Civil'] },{'patelh' , 'Harsh', 'Patel', 'male', 10.0, ['Software', 'Chemical',
        'EngPhys']},{'barriosm', 'Michael', 'Barrios', 'male', 5.0, ['Software', 'Materials',
        'Civil']})
    Equal(readFile, [{'bradenm', 'Matthew', 'Braden', 'male', 9.0, ['Software', 'Mechanical',
        'Civil'] },{'patelh' , 'Harsh', 'Patel', 'male', 10.0, ['Software', 'Chemical',
        'EngPhys']},{'barriosm', 'Michael', 'Barrios', 'male', 5.0, ['Software', 'Materials',
        'Civil']}], "Checking if readStdnts works")
readStdntsTest()

def readFreeChoiceTest():
    readFile = readFreeChoice('bradenm',
                              'patelh')
    Equal(readFile, ['bradenm', 'patelh'] , "This is to test if the read free choice function works")
readFreeChoiceTest()

def readDepartmentsTest():
    readFile = readDeptCapacity({'Civil', '50', 'Chemical', '60', 'Electrical', '80', 'Mechanical',
        '75', 'Software', '90', 'Materials', '50', 'EngPhys', '60'})
    Equal(readFile, {'Civil': 50, 'Chemical': 60, 'Electrical': 80, 'Mechanical': 75, 'Software': 90,
        'Materials': 50, 'EngPhys': 60}, "This is to test the readDeptCapacity")
readDepartmentsTest()

def noAverage():
    readFile = readStdnts({'bradenm', 'Matthew', 'Braden', 'male', 9.0, ['Software', 'Mechanical',
        'Civil'] },{'patelh' , 'Harsh', 'Patel', 'male', 10.0, ['Software', 'Chemical',
        'EngPhys']},{'barriosm', 'Michael', 'Barrios', 'male', 5.0, ['Software', 'Materials',
        'Civil']})
    Equal(average(readFile, "female"), 0, "Calculating female average with no females")
noAverage()

def Allocating():
    readFile = readStdnts({'bradenm', 'Matthew', 'Braden', 'male', 9.0, ['Software', 'Mechanical',
        'Civil'] },{'patelh' , 'Harsh', 'Patel', 'male', 10.0, ['Software', 'Chemical',
        'EngPhys']},{'barriosm', 'Michael', 'Barrios', 'male', 5.0, ['Software', 'Materials',
        'Civil']})
    readFree = readFreeChoice('bradenm','patelh')
    readDept = readDeptCapacity({'Civil', '3', 'Chemical', '2', 'Electrical', '3', 'Mechanical', '1',
         'Software', '2', 'Materials', '2', 'EngPhys', '1'})
    Equal(allocate(readFile, readFree, readDept), {'Civil': [], 'Chemical': [], 'Electrical': [],
        'Mechanical': [], 'Software': [{'macid': 'bradenm', 'fname': 'Matthew', 'lname': 'Braden',
```

```
        'gender': 'male', 'gpa': 9.0, 'choices': ['Software', 'Mechanical', 'Civil']}, {'macid':
        'patelh', 'fname': 'Harsh', 'lname': 'Patel', 'gender': 'male', 'gpa': 10.0, 'choices':
        ['Software', 'Chemical', 'EngPhys']},'Materials': [{'macid': 'barriosm', 'fname': 'Michael',
        'lname': 'Barrios', 'gender': 'male', 'gpa': 5.0, 'choices': ['Software', 'Materials',
        'Civil']}], 'EngPhys': []}, "Allocates the data")
Allocating()

def SortTwoEquals():
    readFile = readStdnts({'bradenm', 'Matthew', 'Braden', 'male', 10.0, ['Software', 'Mechanical',
        'Civil'] },{'patelh' , 'Harsh', 'Patel', 'male', 10.0, ['Software', 'Chemical',
        'EngPhys']},{'barriosm', 'Michael', 'Barrios', 'male', 5.0, ['Software', 'Materials',
        'Civil']})
    Equal(sort(readFile),{'bradenm', 'Matthew', 'Braden', 'male', 10.0, ['Software', 'Mechanical',
        'Civil'] },{'patelh' , 'Harsh', 'Patel', 'male', 10.0, ['Software', 'Chemical',
        'EngPhys']},{'barriosm', 'Michael', 'Barrios', 'male', 5.0, ['Software', 'Materials',
        'Civil']}, "This sorts two students with equal gpa")
SortTwoEquals()

def FemaleWithWrongSpell():
    readFile = readStdnts({'bradenm', 'Matthew', 'Braden', 'male', 10.0, ['Software', 'Mechanical',
        'Civil'] },
    {'browniea' , 'Alexandra', 'Brownie', 'Female', 10.0, ['Software', 'Chemical', 'EngPhys']},
    {'barriosm', 'Michael', 'Barrios', 'male', 5.0, ['Software', 'Materials', 'Civil']})
    Equal(average(readFile, "female"), 0, "This test is for wrong spelling in genders")
FemaleWithWrongSpell()

def AllocateWith3GPA():
    readFile = readStdnts({'bradenm', 'Matthew', 'Braden', 'male', 10.0, ['Software', 'Mechanical',
        'Civil'] },{'patelh' , 'Harsh', 'Patel', 'male', 3.0, ['Software', 'Chemical',
        'EngPhys']},{'barriosm', 'Michael', 'Barrios', 'male', 5.0, ['Software', 'Materials',
        'Civil']})
    readFree = readFreeChoice('bradenm', 'patelh')
    readDept = readDeptCapacity({'Civil', '3', 'Chemical', '2', 'Electrical', '3', 'Mechanical', '1',
         'Software', '2', 'Materials', '2', 'EngPhys', '1'})
    Equal(allocate(readFile, readFree, readDept), {'Civil': [], 'Chemical': [], 'Electrical': [],
        'Mechanical': [], 'Software': [{'macid': 'bradenm', 'fname': 'Matthew', 'lname': 'Braden',
        'gender': 'male', 'gpa': 10.0, 'choices': ['Software', 'Mechanical', 'Civil']},'Software':
        [{'macid': 'barriosm', 'fname': 'Michael', 'lname': 'Barrios', 'gender': 'male', 'gpa': 5.0,
        'choices': ['Software', 'Materials', 'Civil']}], 'EngPhys': []}, "Allocates the data")
AllocateWith3GPA()
```

J Code for Partner's CalcModule.py

```
## @file CalcModule.py
#  @author Samuel Crawford
#  @brief Completes calculations on the list of students from ReadAllocationData.
#  @date 1/16/2019

from warnings import warn

## @brief Defines parameter error for when parameters are incorrect.
class ParamError(Exception):
    pass

## @brief Sorts a list of student dictionaries in order of descending GPA.
#  @param[in] S A list of dictionaries of students.
#  @return A list of student dictionaries in sorted order.
def sort(S):

    # usage of sorted() borrowed from:
    #
        https://stackoverflow.com/questions/72899/how-do-i-sort-a-list-of-dictionaries-by-a-value-of-the-dictionary
    return sorted(S, key = lambda k: k["gpa"], reverse = True)

## @brief Calculates the average GPA of the specified gender of students.
#  @param[in] L A list of dictionaries of students.
#  @param[in] g A string (either "male" or "female").
#  @return The average GPA of the specified gender of students.
#  @throw ParamError if the string passed is not "male" or "female".
def average(L, g):

    # define error string
    wrongGender = "Gender must be 'male' or 'female'"

    # raise exception
    if g not in ["male", "female"]:
        raise ParamError(wrongGender)

    gpaSum  = 0
    counter = 0

    for i in L:
        if i["gender"] == g:
            gpaSum  += i["gpa"]
            counter += 1
    if counter == 0:
        return 0.0
    return gpaSum / counter

## @brief Allocates students to departments.
#  @param[in] S A list of dictionaries of students.
#  @param[in] F A list of students with free choice.
#  @param[in] C A dictionary of department capacities.
#  @return A dictionary of departments with enrolled students.
def allocate(S, F, C):

    notFree    = []
    deptAssign = {
        "civil"      : [],
        "chemical"   : [],
        "electrical" : [],
        "mechanical" : [],
        "software"   : [],
        "materials"  : [],
        "engphys"    : []
    }

    S = sort(S)

    for name in F:
        notPresent = True
        for student in S:
            if student["macid"] == name:
                notPresent = False
        if notPresent:
            warn("{} is in free choice list but not in student list".format(name))

    # allocate students with free choice
    for student in S:
```

```
        if student["macid"] not in F:
            notFree.append(student)
        else:
            deptAssign = assign(student, deptAssign, C)

    # allocate students without free choice
    for student in notFree:
        deptAssign = assign(student, deptAssign, C)

    return deptAssign

## @brief Helper function for allocate(S, F, C) that assigns a single student to a department.
#  @param[in] s The student to be assigned.
#  @param[in] D The department assignement list.
#  @param[in] C A dictionary of department capacities.
#  @return The updated department assignment list.
def assign(s, D, C):

    if s["gpa"] >= 4.0:
        num = 0
        while num != 3:
            choice = s["choices"][num]
            # check for capacity
            if len(D[choice]) < C[choice]:
                D[choice].append(s)
                break
            else:
                num += 1
        if num == 3:
            warn("Third choice department full; {} not allocated".format(s["macid"]))
    return D
```

K 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
```