Assignment 1 Solution

Shivam Taneja - tanejs4

January 24, 2019

This is the report for Assignment 1 CS2ME3, providing the tests cases i checked, reasoning for my answers, assumptions i made, and parter's tests and answers to the questions specified in the Assignment.

1 Testing of the Original Program

Description of approach to testing. Rationale for test case selection. Summary of results. Any problems uncovered through testing. Under testing you should list any assumptions you needed to make about the program's inputs or expected behaviour.

While all testing phases, i assumed that the file is not empty and in proper json format.

To test my files,I used test driven development(TDD) and used "unittest" framework to make sure the ease of use. I also made a sample json files prior to writing code and checked my output based on the files. This helped a lot with CalcModule as the input is the same as the output for ReadAllocationData, which made it crucial for me to make it works properly at least on my sample file. In addition, TDD make ensured the modularity of the code.

I created 6 main functions, each testing output for each of the functions from CalcModule and ReadAllocationData with pre made expected output based on the sample files. While testing, i uncovered multiple bugs for example, with function 'allocate' as every time it displayed error but when i run the function independently, i.e. not inside testCalc.py, it worked perfectly. Using the sample files, and running both my files in testCalc.py displayed that all my test cases were working. (no errors).

2 Results of Testing Partner's Code

At start, calcModule was unable to run. My partner added variables and functions to his files. "sort": The function like a charm and matched with my test cases. - no errors

found "average": The function did not work with my test cases, as it required two constant and one function that is stored in different files(I did not receive any other files). All the test cases failed. error found: "NameError: name "MIN_GPA" is not defined" & "NameError: name 'MAX_GPA' is not defined" Assumptions made: In order to run the function, i assumed the two constants named above. MIN_GPA = 0 and MAX_GPA = 12 After that, it satisfied all my test cases. "allocate":The function did not work with my test cases, as it required two constant that is stored in different file(I did not receive any other files). All the test cases failed. error found : "NameError: name 'MIN_GPA' is not defined" & "NameError: name 'remove_duplicates' is not defined Assumptions made: In order to run the function, i assumed the constant named above. MIN_GPA = 0 and used my own version of "remove_duplicate" After that, it satisfied all my test cases.

3 Discussion of Test Results

With my parter's code, I found some of the bad practices that i personally avoid. Mainly, his dependence of other files even for constants, which are, in reference to this assignment, not a lot. If i use my assumptions and my code (for remove_duplicate function), every function was fine.

3.1 Problems with Original Code

The issues I found in my code is that my readAllocationModule works just with certain text style (content). And i did not add any base cases or any exceptions in my code that could dramatically increase the flexibility of the code.

3.2 Problems with Partner's Code

One of the main issues i encountered with my partner's code was to it imports multiple global variables and functions from two different text files that does not exist. To fix this issue, i assumed these variables and function. Without these variables and a certain function, all my test cases failed.The allocate function was made unnecessarily complex, even though it passed all my test cases, the efficiency drops with high number of data set.

4 Critique of Design Specification

I have some ideas that could enhance the readability and to some extent efficiency of the code. the allocate function was the one that took me a lot of time to read and understand the working of it. My partner made one separate function to allocate "topmost available

choice if they have a passing gpa, defined in a_constants.py", which could be implemented in the main function itself easily. Other than that, I found the assignment specifications to be alright.

5 Answers to Questions

(a) we can start by making sure the input to the function is always correct, like not in upper case, we can solve this by using g.lower() function. Next step, we can also implement average for the whole class instead of just of particular gender, we can do this by comparing the string input with "all", and calculate accordingly For function Sort, to make it more general, we can add another input that would sort it accordingly, the input would be the keys (like macid, last name or first name).

(b) According to this question, Aliasing refers the a variable holding reference of another variables.

```
def alias():
        x = [1,12,4]
        y = x
        y[1] = 17
        assert x[1] == 17
```

There will no assertion error, hence y is alias of x, and therefor modifying y will modify x

(c) To build some confidence, you can have made couple of small sample json files, for example, one empty file, and rest that answers your three main function's output CalcModule was selected as this py file is more complex and gives functional results, hence have more chances of errors. Another reason could be because the general input of the CalcModule's functions were similar to the outputs of ReadAllocationData's Functions

(d) Here are some alternatives that we can use instead of string in some of the elements of dictionary 1. We can start with .lower() function is all the keys and the corresponding values to decrease the chances for any miss is comparison 2. Use of enum instead of strings in the department and gender keys to reduce any incorrect spellings 3. instead of macid we can use the mac number.

(e) answer If we change dictionary to tuple, it will either have to be a double tuple or a tuple with just values and no keys, in this we need to ensure the order of the values in some cases. I would not recommend changing data structure as using dictionary is

really easy, and the ease of implementing in functions is somewhat superior to tuples and other data structure, in this scenario.

(f) answer If we choose to use tuple instead of given data structure, it would not make much of a hassle in calcModule as both list and tuple supports concatenation, repetition, indexing, slicing and some of the inbuilt functions, the only key difference being tuple are immutable. If a custom class is used where it returns next choice and a method to return true if no choice are left, calcModule does not need to change.

F Code for ReadAllocationData.py

```
## @file ReadAllocationData.py
#  @author Shivam Taneja
#  @brief reads data from json files and return required results
#  @date 17/01/2019

import json

## @brief A function to return list of dictionaries
# @details uses the package "json" to output list of dictionaries from the json file
# @param s The filepath of concerned json file
# @return The list of dictionaries with students information
def readStdnts(s):
    with open(s,'r') as fh:
        x = json.load(fh)

    return x

## @brief A function to return list of students macid
# @details uses the package "json" to output list of strings(macid) from the json file
# @param s The filepath of concerned json file
# @return The list of strings with students with free choices
def readFreeChoice(s):
    with open(s,'r') as fh:
        x = json.load(fh)

    fh.close()
    return x

        # for everyDic in x:
        #     curValue = everyDic.get("gpa")
        #     print(type(curValue))
        #     if (curValue >= 4.0 and curValue <= 12.0):
        #         lst.append(everyDic['macid'])

## @brief A function to return dictionary of dept with the number of seats available
# @details uses the package "json" to output dictionary from the json file
# @param s The filepath of concerned json file
# @return The dictionary with department names as keys and number of available
def readDeptCapacity(s):
    # choice = { 'civil' : 0 , 'chemical' : 0 , 'electrical' : 0, 'mechanical' :0 , 'software' : 0,
        'materials' : 0, 'engphys' :0 }
    with open(s,'r') as fh:
        x = json.load(fh)
        # for i in x:
        #     which3 = i['dept']
        #     for x in choice:
        #         for j in range(3):
        #             if (x == which3[j]):
        #                 choice[x] = choice[x] + 1
    fh.close()
    return x
```

G Code for CalcModule.py

```
## @file CalcModule.py
#  @author Shivam Taneja
#  @brief Calculates and provide required results based on the output from ReadAllocationData
#  @date 17/01/2019


## @brief a function to sort list
# @details The function sorts in descending order on "gpa"
# @param S The unordered list of dictionaries
# @return sorted list of dictionaries using 2 inbuilt functions
def sort(S):
    newS = sorted(S, key=lambda k: k['gpa'])
    newS.reverse()
    return newS

## @brief a function to get the average of male/female 's GPA
# @details The function uses the for loop to check if the string matches
#           and adds the gpa and increment variable "tot" so as to perform
#           average algorithm correctly
# @param L The list of dictionaries
# @param g The string input -> "male" or "female"
# @return average of gender's gpa
def average(L,g):
    tot = 0
    avgB = 0
    for i in L:
        if(i['gender'] == g):
            avgB +=i['gpa']
            tot +=1
    average = avgB/tot
    return average

## @brief a function to allocate student to a dept
# @details The function allocate student to a dept based on their gpa
#           and their preferenece
#           function also gives preferenece to students with free choice
#           The function also makes sure dept in not over filled
# @param S The list of dictionaries
# @param F The list of "macid" of students with free choice
# @param C The dictionary of each dept and their seats available
# @return list of dictionaries where key is the dept and values
#           are the dictionaries of students alloted the dept
def allocate(S,F,C):
    newS = sorted(S, key=lambda k: k['gpa'])
    newS.reverse()

    resDict = { 'civil' : [] , 'chemical' : [] , 'electrical' : [], 'mechanical' :[] , 'software' :
        [], 'materials' : [], 'engphys' :[] }
    for everyDic in newS: #newS is the list of dic and i in every dic
        if everyDic['macid'] in F:
            stream = everyDic['dept'][0]
            for allStreams in resDict:
                if (stream == allStreams):
                    resDict[stream].append(everyDic) #all free choice
                    C[stream] -=1

        elif everyDic['macid'] not in F and everyDic['gpa'] > 4.0:
            for checkLength in resDict:
                for iterate in range(3):
                    stream = everyDic['dept'][iterate]
                    if (len(resDict[checkLength]) < C[checkLength]):
                        resDict[stream].append(everyDic)
                        C[stream] -=1
                        break
    return resDict
```

H Code for testCalc.py

```
## @file testCalc.py
#  @author Shivam taneja
#  @brief Provides cases for the CalcModule and ReadAllocationData
#  @date 17/01/2019

# @details for more documentation with package i used
#           https://docs.python-guide.org/writing/tests/

import unittest

import json
import CalcModule
import ReadAllocationData


## @brief A class required by "unittest" package to test cases

class testCase(unittest.TestCase):
## @brief ReadAllocationData - test

  ## @brief tests readStdnts function from ReadAllocationData.py

  #  @details uses provided json file to use get its own answer and checks for the output
  #           provided by readStdnts
  #  @exception is the outputs doesnt match, one error is added and whole test file fails
    def test_readStdnts(self):
        file = 'allstud.json'
        fh = open(file ,'r')

        res1 = json.load(fh)
        res2 = ReadAllocationData.readStdnts(file)
        self.assertEqual(res1, res2)

        fh.close()

 ## @brief tests readFreeChoice function from ReadAllocationData.py

 #  @details uses provided json file to use get its own answer and checks for the output
 #           provided by readFreeChoice
 #  @exception is the outputs doesnt match, one error is added and whole test file fails
    def test_readFreeChoice(self):
        file = 'freechoice.json'

        res1 = ['dog1', 'via1', 'gav', 'diiv', 'taneja', 'bdsfds']
        res2 = ReadAllocationData.readFreeChoice(file)
        self.assertEqual(res1, res2)

 ## @brief tests readDeptCapacity function from ReadAllocationData.py

 #  @details uses provided json file to use get its own answer and checks for the output
 #           provided by readDeptCapacity
 #  @exception is the outputs doesnt match, one error is added and whole test file fails
    def test_readDeptCapacity(self):
        file = 'department.json'

        res1 = {"civil": 20,"chemical": 30, "electrical": 40, "mechanical": 100, "software": 10,
             "materials": 100,"engphys": 40}

        res2 = ReadAllocationData.readDeptCapacity(file)

        self.assertEqual(res1, res2)

## @brief CalcModule - test

 ## @brief tests sort function from CalcModule.py

 #  @details uses provided json file to use get its own answer and checks for the output
 #           provided by CalcModule
 #  @exception is the outputs doesnt match, one error is added and whole test file fails

    def test_sort(self):
        requestedFile = [{'macid': 'via1', 'fname': 'cols', 'lname': 'Gas', 'gender': 'female', 'gpa':
             9, 'dept': ['mechanical', 'engphys', 'software']}, {'macid': 'diiv', 'fname': 'dovija',
             'lname': 'gupt', 'gender': 'female', 'gpa': 9, 'dept': ['software', 'mechanical',
             'materials']}, {'macid': 'taneja', 'fname': 'shiva', 'lname': 'helloworld', 'gender':
```

```
            'female', 'gpa': 9, 'dept': ['chemical', 'civil', 'electrical']}, {'macid': 'bdsfds',
            'fname': 'hnfdgb', 'lname': 'vdsffs', 'gender': 'female', 'gpa': 9, 'dept': ['software',
            'engphys', 'electrical']}, {'macid': 'gav', 'fname': 'ry', 'lname': 'chuk', 'gender':
            'male', 'gpa': 6, 'dept': ['chemical', 'electrical', 'software']}, {'macid': 'dog1',
            'fname': 'gie', 'lname': 'ge', 'gender': 'male', 'gpa': 5, 'dept': ['engphys',
            'software', 'mechanical']}]

        res1 = CalcModule.sort(ReadAllocationData.readStdnts('allstud.json'))
        res2 = [{'macid': 'bdsfds', 'fname': 'hnfdgb', 'lname': 'vdsffs', 'gender': 'female', 'gpa':
            9, 'dept': ['software', 'engphys', 'electrical']}, {'macid': 'taneja', 'fname': 'shiva',
            'lname': 'helloworld', 'gender': 'female', 'gpa': 9, 'dept': ['chemical', 'civil',
            'electrical']}, {'macid': 'diiv', 'fname': 'dovija', 'lname': 'gupt', 'gender': 'female',
            'gpa': 9, 'dept': ['software', 'mechanical', 'materials']}, {'macid': 'via1', 'fname':
            'cols', 'lname': 'Gas', 'gender': 'female', 'gpa': 9, 'dept': ['mechanical', 'engphys',
            'software']}, {'macid': 'gav', 'fname': 'ry', 'lname': 'chuk', 'gender': 'male', 'gpa':
            6, 'dept': ['chemical', 'electrical', 'software']}, {'macid': 'dog1', 'fname': 'gie',
            'lname': 'ge', 'gender': 'male', 'gpa': 5, 'dept': ['engphys', 'software', 'mechanical']}]

        self.assertEqual(res1,res2)
## @brief tests average function from CalcModule.py

#  @details uses provided json file to use get its own answer and checks for the output
#           provided by CalcModule
#  @exception is the outputs doesnt match, one error is added and whole test file fails

    def test_average(self):
        requestedFile = [{'macid': 'dog1', 'fname': 'gie', 'lname': 'ge', 'gender': 'male', 'gpa': 5,
            'dept': [ 'engphys','software', 'mechanical']}, {'macid': 'via1', 'fname': 'cols',
            'lname': 'Gas', 'gender': 'female', 'gpa': 9, 'dept': [ 'mechanical',
            'engphys','software']}, {'macid': 'gav', 'fname': 'ry', 'lname': 'chuk', 'gender':
            'male', 'gpa': 6, 'dept': [ 'chemical', 'electrical','software']}, {'macid': 'diiv',
            'fname': 'dovija', 'lname': 'gupt', 'gender': 'female', 'gpa': 9, 'dept': [
            'software','mechanical', 'materials']}, {'macid': 'taneja', 'fname': 'shiva', 'lname':
            'helloworld', 'gender': 'female', 'gpa': 9, 'dept': ['chemical','civil', 'electrical']},
            {'macid': 'bdsfds', 'fname': 'hnfdgb', 'lname': 'vdsffs', 'gender': 'female', 'gpa': 9,
            'dept': ['software', 'engphys', 'electrical']}]

        res1M = CalcModule.average(requestedFile,'male')
        res1F = CalcModule.average(requestedFile,'female')

        self.assertEqual(res1M,5.5)
        self.assertEqual(res1F,9)
## @brief tests allocate function from CalcModule.py

#  @details uses provided json file to use get its own answer and checks for the output
#           provided by CalcModule
#  @exception is the outputs doesnt match, one error is added and whole test file fails

    def test_allocate(self):
        file1 = ReadAllocationData.readStdnts("allstud.json")
        file2 = ReadAllocationData.readFreeChoice("freechoice.json")
        file3 = ReadAllocationData.readDeptCapacity("department.json")

        res1 = {'civil': [], 'chemical': [{'macid': 'taneja', 'fname': 'shiva', 'lname': 'helloworld',
            'gender': 'female', 'gpa': 9, 'dept': ['chemical', 'civil', 'electrical']}, {'macid':
            'gav', 'fname': 'ry', 'lname': 'chuk', 'gender': 'male', 'gpa': 6, 'dept': ['chemical',
            'electrical', 'software']}], 'electrical': [], 'mechanical': [{'macid': 'via1', 'fname':
            'cols', 'lname': 'Gas', 'gender': 'female', 'gpa': 9, 'dept': ['mechanical', 'engphys',
            'software']}], 'software': [{'macid': 'bdsfds', 'fname': 'hnfdgb', 'lname': 'vdsffs',
            'gender': 'female', 'gpa': 9, 'dept': ['software', 'engphys', 'electrical']}, {'macid':
            'diiv', 'fname': 'dovija', 'lname': 'gupt', 'gender': 'female', 'gpa': 9, 'dept':
            ['software', 'mechanical', 'materials']}], 'materials': [], 'engphys': [{'macid': 'dog1',
            'fname': 'gie', 'lname': 'ge', 'gender': 'male', 'gpa': 5, 'dept': ['engphys',
            'software', 'mechanical']}]}


        res2 = CalcModule.allocate(file1,file2,file3)

        self.assertEqual(res1,res2)

## @brief runs all the test cases
if __name__ == '__main__':
    unittest.main()
```

I Code for Partner's CalcModule.py

```
## @file CalcModule.py
#  @author Rohit Saily
#  @brief Module used to process student and department data.
#  @date 2019-01-18

from a1_constants import MIN_GPA, MAX_GPA, MIN_PASSING_GPA, GENDERS
from a1_utility import remove_duplicates

""" Helper Functions """
## @brief Allocates a student to their topmost available choice if they have a passing gpa, defined in
    a1_constants.py.
#  @details If a student cannot be allocated to any of their choices, they simply are not allocated.
    If their choice is not found in allocations or spots_available, it is ignored.
#  @param student The student to be allocated to a department, represented by a dictionary that maps
    data categories to data.
#  @param allocations A dictionary that maps a department to a list of student dictionaries already
    allocated.
#  @param spots_available A dictionary that maps a department to the number of students that can be
    allocated to it.
def allocate_topmost_available_choice(student, allocations, spots_available):
    if not (MIN_PASSING_GPA <= student['gpa'] <= MAX_GPA):
        return #Do not allocate the student, they are not passing or have an invalid GPA
    for choice in student['choices']:
        if 0 < spots_available[choice]:
            try:
                allocations[choice].append(student)
            except KeyError:
                continue #The choice is not allocateable, continue to try the next one
            else:
                try:
                    spots_available[choice] -= 1
                except KeyError:
                    allocations[choice].remove(student) #We cannot gurantee the department exists with
                        space, so we cannot allocate the student without it being in the
                        spots_available dictionary
                    continue
                else:
                    return

""" API """
## @brief Sorts students by descending GPA.
#  @details
#       Each student is represented by a dictionary that maps data categories to data.
#       This function does not mutate the original list.
#       Sorts using Tim Sort via Python's sorted function.
#  @param S The list of students represented by dictionaries that map data categories to corresponding
    data.
#  @return The list of dictionaries representing students organized by descending GPA.
def sort(S):
    if S == []:
        return S
    return sorted(S, key=lambda student: student['gpa'], reverse=True)

## @brief Averages the GPA of students, filtered by gender.
#  @details Any student found to have a gpa below the minimum or above the maximum, each defined in
    a1_constants.py, will be ignored in the computation.
#  @param L The list of students, each represented by dictionaries that map data categories to data.
#  @param g The gender to filter by, case insensitive.
#  @return None if no students were found to average otherwise it is the average computed.
def average(L, g):
    if not L:
        return None
    gpas = []
    for student in L:
        if not (MIN_GPA <= student['gpa'] <= MAX_GPA):
            continue
        try:
            if student['gender'] == g.lower():
                gpas.append(student['gpa'])
        except KeyError:
            continue
        else:
            pass
    try:
        average = sum(gpas) / len(gpas)
    except ZeroDivisionError:
```

```
        return None #There where no values to average hence there is no average
    else:
        return average

## @brief Allocates students to departments based on code defined allocation scheme.
#  @details
#       The code defined allocation scheme is as follows:
#           1. Free choice students are allocated before students without free choice.
#           2. Students with higher GPAs are allocated first.
#           3. Students with failing GPAs or GPAs above the maximum are not allocated.
#           4. Students who cannot fit into any of departments they chose are not allocated to any
     department.
#  @param S The list of students represented by dictionaries that map data categories to corresponding
     data.
#  @param F The list of mac ids of students who have free choice.
#  @param C A dictionary that maps department names to their capacity.
#  @return A dictionary mapping departments to a list of students allocated to that department.
def allocate(S, F, C):
    if not C:
        return {} #No allocations can be made without departments!
    allocations = {}
    spots_available = {department:capacity for department, capacity in C.items()}
    for department in spots_available:
        allocations[department] = []
    if not S:
        return allocations #No students to allocate
    students = []
    students.extend(sort(S))
    remove_duplicates(students) #To prevent accidentally allocating a student twice if they are
         defined multiple times in the list.
    free_choice_students = []
    for student in students:
        if student['macid'] in F:
            free_choice_students.append(student)
            students.remove(student)
    non_free_choice_students = students #Just to 'rename' the variable and increase code readability
         since students now holds only students without free choice.
    for student in free_choice_students:
        allocate_topmost_available_choice(student, allocations, spots_available)
    for student in non_free_choice_students:
        allocate_topmost_available_choice(student, allocations, spots_available)
    return allocations
```

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
```