Assignment 2 Solution

Shivam Taneja

February 18, 2019

This is the report for Assignment 2 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.

To test my code, I used pytest. The test cases included extreme cases i.e. providing empty or null inputs, normal cases (expected some value), and cases where i expect certain errors. During testing phase, i found couple of bugs predominantly with *DCapALst.py* which was not giving the error $KeyError$ where expected. After fixing the code, other cases failed because of output it was providing $NoneType$. I creates 32 test cases in total, and my code passes all of them.

2 Results of Testing Partner's Code

Consequences of running partner's code. Success, or lack of success, running test cases. Explanation of why it worked, or didn't.

On testing my partner's file, I found that my code still have a bug with *DCapALst.py* where it was not still giving $KeyError$ but instead some int value. Other than that, my partner's code seemed to pass all the requirements and my test cases.

3 Critique of Given Design Specification

Advantages and disadvantages of the given design specification. My partner's code passed all the test cases, but after running $flake8$, there were 219 violations with 79 indentation errors. This is a bad practice, as in future we will work with codes of hundreds of line, making code, at times, unreadable and too messy.

4 Answers

1. In contrast to natural language in A1, i prefer having formal specification of A2 as understanding what each function does, what error it should raise, and what inputs are apt for it becomes really easy. Also the fact Natural language can be interpreted in different ways.

2. By just having an if condition, it would go something like this

```
def add(m, i):
    if isinstance(i, SInfoT):
            if (i.gpa < 0 or i.gpa >12):
                    raise ValueError
            else:
                    SALst.s.add(tuple((m, i)))
    else:
        raise KeyError
```

   and No, we don't need to replace record type with new ADT.

3. The documentation can be modified by making a base class, and sub classes and access the values The similarities between *DCapALst* and *SALst* can be inherited to the new class and pass rest of the functions as arguments.

4. In A1, $FreeChoice$ is based on GPA only i.e. $4 \leq x$ In A2, you have more power to control this factor by a simple true or false value. The same can be said with other functions, names *sort* and *allocate*, In A2, input as lambda function makes it really versatile to control the output.

5. In contrast to List, *SeqADT* is a generator i.e. it creates item in an instance instead of loading it in the memory. Also List could be complicated at times, for example having a list inside a dictionary inside a list.

6. Enums not only prevent any form of lingual incorrection such as capitalisation but also it directly maps to anything you want to use. It also help to easily check if the element exists or not.

   Enums were not used in *macid* as they as unique to each student.

E Code for StdntAllocTypes.py

```
## @file StdntAllocTypes.py
#  @author Shivam Taneja
#  @brief Creates custom datatypes
#  @date 01/02/2019
from SeqADT import *
from typing import NamedTuple
from enum import Enum

## @brief An abstract data type that represents gender


class GenT(Enum):

    male = 0
    female = 1

## @brief An abstract data type that represents department name


class DeptT(Enum):

    civil = 0
    chemical = 1
    electrical = 2
    mechanical = 3
    software = 4
    materials = 5
    engphys = 6

## @brief An abstract data type that represents all student info except macid


class SInfoT(NamedTuple):

    fname: str
    lname: str
    gender: GenT
    gpa: float
    choices: SeqADT
    freechoice: bool
```

F Code for SeqADT.py

```
## @file SeqADt.py
#  @author Shivam Taneja
#  @brief SeqADt
#  @date 01/02/2019

## @brief An abstract data type that represents a sequence (in the set)


class SeqADT:

    i = 0

    ## @brief SeqADT constructor
    #  @details takes a sequence and initializes index
    #  @param x Sequence to be used
    def __init__(self, x):
        self.s = list(x)
        self.i = 0
    ## @brief start initializes the variable i which represents index

    def start(self):
        SeqADT.i = 0
    ## @brief next moves the index to the next by just incrementing
    #  @return the value of the index
    #  @exception StopIteration - if the index goes above the length of the sequence

    def next(self):
        if(self.i >= len(self.s)):
            raise StopIteration
        res = (self.s)[self.i]
        self.i = self.i + 1
        return res
    ## @brief end ends the iteration
    #  @return true or false, if the index is the last or if the index is
    #          not the last of the sequence, respectively

    def end(self):
        return self.i >= len(self.s)
```

G Code for DCapALst.py

```
## @file DCapALst.py
#  @author Shivam Taneja
#  @brief DCapALst
#  @date 01/02/2019
from StdntAllocTypes import *

## @brief An abstract data type that stores departments and it's capacity


class DCapALst:
    s = set()

    ## @brief init initial data structure
    def init():
        DCapALst.s = set()

    ## @brief add adds element to the set data structure
    #  @param d department name of DeptT
    #  @param n capacity of the corresponding department
    #  @exception throws KeyError if datatypes of n doesn't matches
    @staticmethod
    def add(d, n):
        for i in DCapALst.s:
            if d == i[0]:
                raise KeyError

        DCapALst.s.add((d, n))


    ## @brief remove deletes element from the set data structure
    #  @param d department name of DeptT
    #  @exception throws KeyError if department is not found
    @staticmethod
    def remove(d):
        flag = None
        for i in DCapALst.s:
            if i[0] == d:
                flag = i
        if not flag:
            raise KeyError
        else:
            DCapALst.s.remove(flag)

    ## @brief elm checks if the department exists in the set
    #  @param d department name of DeptT
    #  @return True if department exists else False
    @staticmethod
    def elm(d):
        for i in DCapALst.s:
            if i[0] == d:
                return True
        return False

    ## @brief capacity checks the department size
    #  @param d department name of DeptT
    #  @return number of seats available in department
    #  @exception throws KeyError if department is not found
    @staticmethod
    def capacity(d):
        try:
            for i in DCapALst.s:
                if d == i[0]:
                    return i[1]
        except:
            raise KeyError
```

H Code for AALst.py

```
## @file AALst.py
#  @author Shivam Taneja
#  @brief AALst
#  @date 01/02/2019

from StdntAllocTypes import *

## @brief An abstract data type that stores the allocated departments and students


class AALst:
    s = list()

    ## @brief init initial data structure
    @staticmethod
    def init():
        AALst.s = [(DeptT.materials, []), (DeptT.civil, []),
         (DeptT.chemical, []), (DeptT.electrical, []), (DeptT.mechanical, []),
          (DeptT.engphys, []), (DeptT.software, []), (DeptT.electrical, [])]


    ## @brief add_stdnt adds elements in the form of tuple to the set data structure
    #  @param dep department name of DeptT
    #  @param m list of all the macids in that department (string)
    @staticmethod
    def add_stdnt(dep, m):
        for i in AALst.s:
            if i[0] == dep:
                i[1].append(m)

    ## @brief lst_alloc provides with the list of macids in the department
    #  @param d department name of DeptT
    #  @return List of macids (string)
    @staticmethod
    def lst_alloc(d):
        for i in AALst.s:
            if i[0] == d:
                return i[1]

    ## @brief num_alloc provides with the number of allocated macids in the department
    #  @param d department name of DeptT
    #  @return  Number of allocated students (Natural number)
    @staticmethod
    def num_alloc(d):
        for i in AALst.s:
            if i[0] == d:
                return len(i[1])
```

I Code for SALst.py

```
## @file SALst.py
#  @author Shivam Taneja
#  @brief SALst
#  @date 01/02/2019

from StdntAllocTypes import *
from AALst import *
from DCapALst import *

## @brief An abstract data type that stores student information


class SALst:

    s = set()
    ## @brief init initial data structure

    def init():
        SALst.s = set()

    ## @brief add adds the tuple of macid and student info in the set
    #  @param m MacId of the student (string)
    #  @param i information of the student(SInfoT)
    #  @exception throws KeyError if input i is not of the SInfoT type

    @staticmethod
    def add(m, i):
        if isinstance(i, SInfoT):
            SALst.s.add(tuple((m, i)))
        else:
            raise KeyError

    ## @brief remove deletes the tuple of macid and student info from the set
    #  @param m MacId of the student (string)
    #  @exception throws KeyError if input m is not found in the set
    @staticmethod
    def remove(m):
        mac = None
        for i in SALst.s:
            if i[0] == m:
                mac = i
        if not mac:
            raise KeyError
        else:
            SALst.s.remove(mac)

    ## @brief elm checks if the input string is in the data structure
    #  @param m MacId of the student (string)
    #  @return true if MacId is found else False
    @staticmethod
    def elm(m):
        for i in SALst.s:
            if i[0] == m:
                return True
        return False

    ## @brief info gives information about input from the data structure
    #  @param m MacId of the student (string)
    #  @return information of the student (SInfoT)
    @staticmethod
    def info(m):
        for i in SALst.s:
            if m == i[0]:
                return i[1]
        raise KeyError

    ## @brief sort sorts the data structure
    #  @details it uses inbuilt sorted function to sort Data structure
    #           based on their gpa and the lambda function
    #  @param f lambda function expresssion
    #  @return the data structure sorted according to the parameter
    @staticmethod
    def sort(f):
        pre = []
        res = []
        for i in SALst.s:
```

```
        if f(i[1]):
            pre.append(i)
    sort = sorted(pre, key=lambda x: x[1].gpa)
    for i in sort:
        res.append(i[0])
    return list(reversed(res))

## @brief average gives the requested average
#  @param f lambda function expresssion
#  @details Gets the average of GPA only of those sets who matches
#           with the lambda expresssion
#  @return average of the gpa (Numeric)
@staticmethod
def average(f):
    try:
        count = 0
        add = 0
        if SALst.s is None:
            raise ValueError
        for i in SALst.s:
            if f(i[1]):
                count += 1
                add += i[1].gpa
        res = add / count
        return res
    except:
        raise ValueError

## @brief allocate allot students based on the provided conditions
#  @details Sorts the data structure (condtion->gpa more than or equal to the 4)
#           starts alloting students with free choice as priority then
#           students without freechoice. The limit of alloting is decided by
#            the department capacity
#  @exception Throws RuntimeError
@staticmethod
def allocate():
    AALst.init()
    f = SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0)
    for m in f:
        ch = SALst.info(m).choices
        AALst.add_stdnt(ch.next(), m)
    s = SALst.sort(lambda t: not (t.freechoice) and t.gpa >= 4.0)
    for m in s:
        ch = SALst.info(m).choices
        alloc = False
        while not alloc and not ch.end():
            d = ch.next()
            if (AALst.num_alloc(d) < DCapALst.capacity(d)):
                AALst.add_stdnt(d, m)
                alloc = True
        if not alloc:
            raise RuntimeError

@staticmethod
def get_gpa(m, s):
    for i in s:
        if m == i[0]:
            # print(i.gpa)
            return i.gpa
```

J Code for Read.py

```
## @file Read.py
#  @author Shivam Taneja
#  @brief functions for retrieving data from file
#  @date 01/02

from StdntAllocTypes import *
from DCapALst import *
from SALst import *
import re


## @brief load_stdnt_data reads in data from a file, storing it in SALst
#  @param s the name of the file to be read
def load_stdnt_data(s):
    with open(s, 'r') as infile:
        contents = infile.readlines()

    fname = []
    lname = []
    macids = []
    gpalist = []
    genderl = []
    freechoicel = []
    dept = []

    other = [x.strip().split(',') for x in contents]

    department = [re.findall("\[(.*?)\]", x) for x in contents]
    for i in range(len(other)):
        department[i] = [x.strip().split(', ') for x in department[i]]

    for i in range(len(department)):
        temp = department[i][0]
        dept.append(temp)

    for i in range(len(other)):
        length = len(other[i]) - 1
        count = 5
        for j in reversed(range(count, length)):
            other[i].pop(j)

    for i in range(len(other)):
        temp = other[i][1].replace(' ', '')
        fname.append(temp)

    for i in range(len(other)):
        temp = other[i][2].replace(' ', '')
        lname.append(temp)

    for i in other:
        macids.append(i[0])

    for i in range(len(other)):
        temp = other[i][4].replace(' ', '')
        gpalist.append(float(temp))

    for i in range(len(other)):
        temp = other[i][3].replace(' ', '')
        if temp == 'male':
            genderl.append(GenT.male)
        else:
            genderl.append(GenT.female)

    for i in range(len(other)):
        temp = other[i][-1].replace(' ', '')
        if temp == "True":
            freechoicel.append(True)
        else:
            freechoicel.append(False)

    for i in range(len(other)):
        length = len(other[i])
        for j in range(length):
            (other[i][j]).replace('', '')

    for i in range(len(other)):
        indexofdept = department[i]
        other[i].insert(4, indexofdept)

    for i in range(len(other)):
        for index, item in enumerate(dept[i]):
            if item == "civil":
                dept[i][index] = DeptT.civil
            if item == "chemical":
                dept[i][index] = DeptT.chemical
            if item == "electrical":
                dept[i][index] = DeptT.electrical
            if item == "mechanical":
                dept[i][index] = DeptT.mechanical
            if item == "software":
                dept[i][index] = DeptT.software
            if item == "engphys":
                dept[i][index] = DeptT.engphys
            if item == "materials":
                dept[i][index] = DeptT.materials

    SALst.init()

    for i in range(len(other)):
        sinfo = SInfoT(fname[i], lname[i], genderl[i],
            gpalist[i], SeqADT(dept[i]), freechoicel[i])
        SALst.add(macids[i], sinfo)


## @brief load_dcap_data reads in data from a file, storing it in DCapALst
#  @param s the name of the file to be read
def load_dcap_data(s):
    with open(s, 'r') as infile:
        contents = infile.readlines()
    other = [x.strip().split(', ') for x in contents]

    for i in range(len(other)):
        if other[i][0] == "civil":
            other[i][0] = DeptT.civil
        if other[i][0] == "chemical":
            other[i][0] = DeptT.chemical
        if other[i][0] == "electrical":
            other[i][0] = DeptT.electrical
        if other[i][0] == "mechanical":
            other[i][0] = DeptT.mechanical
        if other[i][0] == "software":
            other[i][0] = DeptT.software
        if other[i][0] == "engphys":
            other[i][0] = DeptT.engphys
        if other[i][0] == "materials":
            other[i][0] = DeptT.materials

    for i in range(len(other)):
        other[i].append(float(other[i][1]))
        other[i].pop(1)

    DCapALst.init()
    for i in range(len(other)):
        DCapALst.add(other[i][0], other[i][1])
```

K Code for Partner's SeqADT.py

```
## @file SeqADT.py
#  @author Orlando Ortega
#  @brief A program takes in a sequence, and its able to iterate through it
#  @date  2/11/19

## @brief SeqADT Abstract Data type which is able to iterate through a sequence
class SeqADT:

  s = []
  i = 0

## @brief Constructs SeqADT
#  @details Assigns the array into variable s, which in this case is a sequence of T, furthermore it
     also initializes
#  i which acts as the index of the sequence
#  @param self The instance of the object itself
#  @param s The sequence to be iterated over
  def __init__(self, x):
    self.s = x
    self.i = 0

## @brief Restarts index
#  @details Assigns i the value of 0, in order to restart the iteration of the sequence back to the
     first element
#  @param self The instance of the object itself

  def start(self):
    self.i = 0

## @brief Iterates to the next element of the sequence
#  @details It returns the current element of the sequence while also increasing the index to get
     ready for the next element
#  @exception Throws a StopITeration error if index is outside the range of the sequence
#  @param self The instance of the object itself
#  @return The value at the current index of i in the Object's sequence
  def next(self):
    if (self.i >= len(self.s)):
      raise StopIteration
    currentIndex = self.i
    self.i += 1
    return self.s[currentIndex]

## @brief Checks for end of sequence
#  @details Checks the current index of the sequence to see if it is out of range, therefore
     concluding that the iteration has
#  ended
#  @param self The instance of the object itself
#  @return true if the current index is outside the length of the sequence, otherwise false.
  def end(self):
    return self.i >= len(self.s)
```

L Code for Partner's DCapALst.py

```
## @file DCapALst.py
#  @author Orlando Ortega
#  @brief Controls Departments and their Capacities
#  @date  2/11/19

from StdntAllocTypes import *

## @brief An abstract data type that represents a Department and their Capacity, List
class DCapALst:

  s = {}
## @brief Constructs DCapALst object
#  @details takes in no arguments, and initiates an empty dictionary
  @staticmethod
  def init():
    DCapALst.s = {}

## @brief Adds departments and their capacities to the dictionary
#  @details Adds departments as keys, and their capacities as their values to the dictionary
#  @param d Department of type DeptT
#  @param n Current department capacity
#  @exception KeyError Throws error when department is already in the dictionary
  @staticmethod
  def add(d: DeptT, n: int):
    if(d in DCapALst.s):
      raise KeyError

    DCapALst.s[d] = n

## @brief Removes departments and their capacities from the dictionary
#  @details Removes departments and their capacities from the dictionary
#  @param d Department of type DeptT
#  @exception KeyError Throws error when department is not in the dictionary
  @staticmethod
  def remove(d: DeptT):
    if(d not in DCapALst.s):
      raise KeyError

    del DCapALst.s[d]

## @brief Checks to see if department is in the dictionary
#  @details Checks for department in dictionary and returns a boolean value
#  @param d Department of type DeptT
#  @return Returns boolean depending on whether the department is in the dictionary
  @staticmethod
  def elm(d: DeptT):
    return d in DCapALst.s

## @brief Displays Capacity
#  @details Receives department and returns it's capacity
#  @param d Department of type DeptT
#  @exception KeyError Throws error when department is not in the dictionary
  @staticmethod
  def capacity(d: DeptT):
    if(d not in DCapALst.s):
      raise KeyError

    return DCapALst.s[d]
```

M Code for Partner's SALst.py

```
## @file SALst.py
#  @author Orlando Ortega
#  @brief SALst
#  @date  2/11/19
from StdntAllocTypes import *
from AALst import *
from DCapALst import *
from typing import NamedTuple

## @brief An abstract data type for students
#  @details Stores a tuple with first index being the macid of a student, and the second being the
#  information of the student of type SInfoT
class StudentT(NamedTuple):
  macid: str
  info: SInfoT

## @brief Local Method for GPA
#  @details Returns the GPA of a given student
#  @param m Student's macid
#  @param s Student's info of type SInfoT
#  @return The student's gpa
def get_gpa(m,s: SInfoT):
  return s.gpa

## @brief An abstract data type for associating students
#  @details Associates student's macid with their student info, it is also able to sort, find the
     average and allocate these students
#  into their programs of choice
class SALst:

  s = []

## @brief Constructor for SALst
#  @details Initializes an empty list
  @staticmethod
  def init():
    SALst.s = []

## @brief Adds students to a list
#  @details Takes in the macid and information of a student and appends them to a list in the form of
     a tuple of type
#  StudentT
#  @param m Student's macid
#  @param i Student's info of type SInfoT
#  @exception Throws KeyError if the student is already in the list
  @staticmethod
  def add(m: str, i: SInfoT):
    for item in SALst.s:
      if m == item.macid:
        raise KeyError


    SALst.s.append(StudentT(m, i))

## @brief Removes students from the list
#  @details Takes in the macid of a student and removes them from the list
#  @param m Student's macid
#  @exception Throws KeyError if the student is not already in the list
  @staticmethod
  def remove(m: str):
    #keeps track of elements that have been iterated
    counter = len(SALst.s)

    #iterate through list
    for item in SALst.s:
      #if student was found remove them
      if m == item.macid:
        SALst.s.remove(item)
      else:
        counter -= 1

    #if the counter = 0, it means that all elements of the list have been viewed, therefore the
        student is not on the list
    if counter == 0:
      raise KeyError
```

```
## @brief Checks if the student is in the list
#  @details Takes in the macid of a student and checks if they are in the student list
#  @param m Student's macid
#  @return Boolean value of whether the student is in the list
  @staticmethod
  def elm(m: str):
    for item in SALst.s:
      if m == item.macid:
        return True

    return False

## @brief Returns a student's information
#  @details Takes in the macid of a student and returns their information
#  @param m Student's macid
#  @exception Throws KeyError if student is not in the list
#  @return Student information
  @staticmethod
  def info(m: str):
  #keeps track of elements that have been iterated
   counter = len(SALst.s)

  #iterate through list
   for item in SALst.s:
     #if student was found remove them
     if m == item.macid:
       return item.info
     else:
       counter -= 1

   #if the counter = 0, it means that all elements of the list have been viewed, therefore the student
        is not on the list
   if counter == 0:
     raise KeyError

## @brief Returns a sorted list of students
#  @details Takes in a function and finds the students that satisfy it, once those students are found
     it returns a list
#  of those students in descending gpa order. Furthermore, the algorithm used to sort is insertion
     sort, of which geeksforgeeks
#  and the textbook Algorithms were consulted for advice
#  @param f Function that takes in as a parameter SInfoT and returns a boolean
#  @return Sorted student list
  @staticmethod
  def sort(f):
    sorted_studentT_list = []
    finished_list = []

    #appends students that satisfy the function condition into a temporary list
    for item in SALst.s:
      if (f(item.info)):
        sorted_studentT_list.append(item)

    # Iterate through every element on the list starting from the second position
    for i in range(1, len(sorted_studentT_list)):

      key = sorted_studentT_list[i]

      # Move those students with a higher gpa closer to the start of the array to achieve a descending
          GPA order
      j = i-1
      while j >= 0 and get_gpa(key.macid,key.info) >
           get_gpa(sorted_studentT_list[j].macid,sorted_studentT_list[j].info) :
        #exchanges both elements if the gpa if the gpa of the second element is higher than that of
             the first element
        sorted_studentT_list[j+1] = sorted_studentT_list[j]
        j -= 1

      sorted_studentT_list[j+1] = key

    #appends the macid's of the sorted students, into a temporary list(this list is therefore sorted)
    for item in sorted_studentT_list:
      finished_list.append(item.macid)

    return finished_list

## @brief Returns the average gpa of students
#  @details Takes in a function and finds the students that satisfy it, once those students are found
     it returns the
#  average GPA of those students
#  @exception Throws ValueError if there are no students that satisfy the function requirement
#  @param f Function that takes in as a parameter SInfoT and returns a boolean
#  @return Students' average gpa
  @staticmethod
  def average(f):
    counter = 0
    total = 0
    average = 0

    #counts and adds each of the student's gpa that satisfy the function
    for item in SALst.s:
      if(f(item.info)):
        total += item.info.gpa
        counter += 1

    #if there were students that satisfied the requirement return their average gpa
    if(counter == 0):
      raise ValueError
    else:
      average = total/counter
      return average

## @brief Allocates students into their respective choices
#  @details First sorts students that are in the freechoice list and proceeds to allocate those first
#  (in order of GPA, highest to lowest), it then proceeds to those students that aren't and conducts
#  the exact same procedure
#  @exception Throws RuntimeError if a student could not be placed into any program of their choice
  @staticmethod
  def allocate():

    AALst.init()

    #allocates students that are in the freechoice list and have a gpa higher than 4 into their
        program choices
    F = SALst.sort((lambda t: t.freechoice and t.gpa >= 4.0))
    for m in F:
      ch = SALst.info(m).choices
      AALst.add_stdnt(ch.next(), m)

    #allocates students that are not in the freechoice list and have a gpa higher than 4 into their
        program choices
    S = SALst.sort(lambda t: not(t.freechoice) and t.gpa >= 4.0)
    for m in S:
      ch = SALst.info(m).choices

      #checks for the next choice if their top choice is out of capacity in order to priority
      alloc = False
      while(not(alloc) and not(ch.end())):
        d = ch.next()
        if(AALst.num_alloc(d) < DCapALst.capacity(d)):
          AALst.add_stdnt(d, m)
          alloc = True

      #if a student could not be allocated an error is raised
      if(not(alloc)):
        raise RuntimeError
```