Assignment 2 Solution

Zayed Sheet

February 25, 2019

The following document is a thorough report on 2AA4 Assignment 2. The purpose of this software design exercise is to read from a formal design specification to produce a python program. This program uses several modules to take an input of students in first year engineering and place them in one of their desired programs of choice depending on whether they meet the requirements for that program.

1 Testing of the Original Program

When creating my test cases I didn't run into any errors, everything worked as expected and all test cases passed for my program. My main concern when creating the test cases was to check whether the code adhered to the project specifications. The first thing I did was try to cover all the basic cases that essentially assess the correctness of the code. These basic cases would have expected inputs and expected outputs to test the basic functionality of the code. They would also check for exceptions required in the project spec. Simply to further test the code I then added some cases that test the robustness of the program by essentially trying to break the code with unexpected inputs. For example, cases where the inputs would contain empty lists, empty tuples, etc. My test cases go as follows:

A. **TestSeqADT**

i) test_next
This case tests for the correctness of the next method of module SeqADT.

ii) test_empty_next
This case tests for robustness of the next method. It test what will happen if the next method is used on an empty list in SeqADT.

iii) test_start
This case tests for correctness of the start method of SeqADT.

iv) test_end & test_end_2
These two test cases are paired together to test the correctness of the end method of SeqADT.

B. **TestDCapALst**

i) test_add
This case tests for the correctness of the add method in module DCapALst.

ii) test_add_2
This case tests that an exception is raised when adding a department that already exists using the add method of DCapALst.

iii) test_remove
This case tests for the correctness of the remove method in DCapALst.

iv) test_remove_2
This case tests that an exception is raised when removing a department that already exists in the remove method of DCapALst.

v) test_elm & test_elm_2
These two test cases are paired together to test the correctness of the elm method of DCapALst.

vi) test_capacity
This case tests for the correctness of the capacity method in DCapALst.

vii) test_capacity_2
This case tests that an exception is raised when checking the capacity of a department that doesn't exist.

C. **TestSALst**

i) test_add
This case tests for the correctness of the add method in module SALst.

ii) test_add_2
This case tests that an exception is raised if an element that already exists is added to SALst.

iii) test_remove
This case tests for correctness of the remove method in module SALst.

iv) test_remove_2
This case tests that an exception is raised if an element that doesn't exist is removed from SALst.

v) test_elm $ test_elm_2
These test cases are paired together to test the correctness of the elm method of SALst.

vi) test_info
This test case tests for correctness of the info method in SALst.

vii) test_info_2
This case tests than an exception is raised if this method is used with an element that doesn't exist in SALst.

viii) test_sort & test_sort_2
These test for the full correctness of the sort method in SALst using different lambda functions as inputs.

ix) test_sort_3
This case tests the robustness of the sort method. SALst is empty in this test case and the output should also be empty.

x) test_average
This case tests that an exception is raised when no elements in the list contain the given GenT value.

xi) test_average_2 This case tests for the correctness of the average method.

xii) test_allocate
This case tests for the correctness of the allocate method. In this test cae SAlst contains several different students all with unique attributes.

xiii) test_allocate_2
This case tests for robustness of the allocate method. SALst is empty in this test case and the output should be empty departments for all departments in AAlst.

2 Results of Testing Partner's Code

After running my partner's code with $test_All.py$, $24 out of 28 test cases passed. The four that failed include Test$
$I learned a few things after testing and reviewing my partners code. First I learned to make some of my test case$

3 Critique of Given Design Specification

What I liked about the formal design specification is that you don't have to think about whether what you're implementing is correct or not, because as long as it follows the specifications than you don't need to worry. However what I dislike is that I found the

formal design specifications to be harder to read, and it didn't allow for much creativity or flexibility for the person coding it. For those reasons a project using formal specifications provided can be quite boring to code.
The only thing I would change about the design would be to use inheritance where applicable. Some of the classes contain nearly identical methods which felt really repetative to recode.

4 Answers

A) The advantages of using a formal specification is that there is no misinterpretation between the coder and the person assigning the project. What you see is what you must code. On the other hand natural language specifications can cause misinterpretation and may require the programmer to make several assumptions when coding the project.
One of the main disadvantages to formal specifications is that it doesn't allow for much creativity or flexibility for the programmer. It's also much more difficult to read as opposed to natural language, and if the programmer doesn't have a strong grasp of discrete mathematics than he or she may read the specifications incorrectly.

B) I would modify the read module specification to specify that the gpa must be between 0 and 12 otherwise an exception is raised. Based off our assumptions the gpa is always the fifth item in the student's attributes so it wouldn't be too hard to implement this. If we add the exception to the read module then no changes will be necessary to any of our record types because by the time any information is allocated the gpa would be guaranteed to be between 0 and 12.

C) Because DCapALst and SALst contain nearly identical methods than we could use inheritance to inherit all the methods from one class to another. This would mean less code is required overall which is easier for the programmer to code and also makes the program easier to read.

D) In A2 the average and sort functions are a lot more general because you can sort them based off the input parameters. The input to both of these methods is a lambda function and therefore we can generalize what we want to find the average for or what we want to sort. For example we could now use sort to remove everyone with free choice or everyone with below a 4.0 gpa, or both, or neither. We can also find the average of any group of students we'd like based off any attribute (doesn't necessarily have to be gender). The use of the SeqADT data type may also help to make things more geneeral because it can be used for any list.

E) A major advantage to using SeqADT as opposed to a regular list is that it makes it a lot easier to read, code and understand your program because the way its structured is closely related to how people think. Its simply a lot easier to read "while not choices.end" as opposed to "for i in range(list.length)" or easier to read "choices.next" as opposed to "i++".

F) Enums are useful for defining immutable, related sets of constant values. They're especially useful for strings which can be easily confused based on spelling, capitalization, casing etc. Making a string an enum also makes it more human readable. Lastly, enums support iteration which can also be beneficial (for example if you want to iterate over all your departments which I have done in my code). The reason we don't make MacIDs Enums is because MacIDs are very unique and therefore can't be represented by Enums. There's simply too many to represent them all and they're simply too unique.

E Code for StdntAllocTypes.py

```
## @file StdntAllocTypes.py
#  @author Zayed Sheet
#  @brief This module is to represent certain data as Enum or NamedTuple
#  @date 2019-02-11


from enum import Enum
from typing import NamedTuple
from SeqADT import *


## @brief class for GenT datatype
#  @details Enumerated datatype for gender
class GenT(Enum):
    male = 'male'
    female = 'female'


## @brief class for DeptT datatype
#  @details Enumerated datatype for departments
class DeptT(Enum):
    civil = 'civil'
    chemical = 'chemical'
    electrical = 'electrical'
    mechanical = 'mechanical'
    software = 'software'
    materials = 'materials'
    engphys = 'engphys'


## @brief Datatype for student information
#  @details NamedTuple datatype for students
class SInfoT(NamedTuple):
    fname: str
    lname: str
    gender: GenT
    gpa: float
    choices: SeqADT
    freechoice: bool
```

F Code for SeqADT.py

```
## @file SeqADT.py
#  @author Zayed Sheet
#  @brief Abstract data type that represents sequence for list of choices
#  @date 2019-02-11


## @brief Class for abstract data type that represents sequence for list of choices
#  @details Class for ADT that treats list of choices as a sequence
class SeqADT:

    ## @brief initializer for class that sets index to 0 and s to inputted list
    def __init__(self, x):
        self.__s = x  # s and i are state variables
        self.__i = 0  # __ is so you can access the state variables directly

    ## @brief sets the index of the data type to 0
    def start(self):  # pass in self if you're changing state variables
        self.__i = 0

    ## @brief method that allows user to access next item in list
    #  @return returns the next item in the list
    def next(self):
        if self.__i < len(self.__s):
            self.__i += 1
            return self.__s[self.__i - 1]
        else:
            raise StopIteration

    ## @brief method that lets user know if they're at the end of the list
    #  @return returns True if user is at end of list, otherwise false
    def end(self):
        if self.__i < len(self.__s):
            return False
        return True
```

G Code for DCapALst.py

```
## @file DCapALst.py
#  @author Zayed Sheet
#  @brief This module is for the DCapALst data type for eng streams
#  @date 2019-02-11


## @brief Class for datatype for department and department capacities
class DCapALst:

    ## @brief Intiializes a list for the class
    @staticmethod
    def init():  # initialize list
        DCapALst.s = []

    ## @brief Adds a department and its capacity into the list
    #  @details Adds a tuple containing the department as a DeptT type and the capacity for
    #  @        \n that department
    #  @param d department name as a DeptT type
    #  @param n Integer for capacity for department
    @staticmethod
    def add(d, n):
        for department in DCapALst.s:
            if d in department:  # if the program is not already in the list
                raise KeyError
        DCapALst.s.append((d, n))

    ## @brief Removes a department from the list
    #  @param d Department as a DeptT type
    @staticmethod
    def remove(d):
        for department in DCapALst.s:  # for every tuple in s
            if d in department:  # if the department is in this tuple
                DCapALst.s.remove(department)  # remove the tuple from the list
                return  # exit the function
        raise KeyError

    ## @brief Checks if a department exists in the list
    #  @param Department as a DeptT data type
    #  @return Returns true if department exists otherwise false
    @staticmethod
    def elm(d):
        for department in DCapALst.s:  # for every department tuple in s
            if d in department:  # if the department is in this tuple
                return True
        return False

    ## @brief Checks the capacity for a department
    #  @param d Department as a DeptT type
    @staticmethod
    def capacity(d):
        for department in DCapALst.s:  # for every tuple in s
            if d in department:  # if the department is in this tuple
                return department[1]
        raise KeyError
```

H Code for AALst.py

```
## @file AAlst.py
#  @author Zayed Sheet
#  @brief This module is for the AALst data type
#  @date 2019-02-11


from StdntAllocTypes import *


## @brief Class for AALst data type
#  @details This datatype has methods for intializing, adding an allocated student
#           \n and returning the list of allocated students or how many students are
#           \n allocated into a department
class AALst:

    ## @brief  Initializes the data type
    #  @details Creates a list of tuples containing department and an empty list
    @staticmethod
    def init():  # initialize list
        AALst.s = []
        for i in DeptT:
            AALst.s.append((i, []))

    ## @brief Adds a student to the datatype
    #  @details Adds a student into their desired department
    #  @param dep the department user is being allocated to
    #  @param m the user's macid
    @staticmethod
    def add_stdnt(dep, m):
        for i in AALst.s:
            if dep in i:
                i[1].append(m)

    ## @brief Returns the list of students allocated into a department
    #  @param d The desired department of type DeptT
    #  @return The list of students allocated into a department
    @staticmethod
    def lst_alloc(d):
        for i in AALst.s:
            if d in i:
                return i[1]

    ## @brief Returns how many students are allocated into a specific department
    #  @param d The department of type DeptT
    #  @return The number of students allocated into a department
    @staticmethod
    def num_alloc(d):
        for i in AALst.s:
            if d in i:
                return len(i[1])
```

I Code for SALst.py

```
## @file SALst.py
#  @author Zayed Sheet
#  @brief This module is for the SALst data and is mainly used to allocate students
#  @date 2019-02-11


from AALst import *
from DCapALst import *


## @brief Class for allocating students into programs
#  @details This class allocated students into their programs based off several aspects
#           \n like if its full or not. The class has several different methods to do this
#           \n including sort, remove, allocate.
class SALst:

    ## @brief Initializes a list for list of students
    @staticmethod
    def init():
        SALst.s = []

    ## @brief Adds a student into the list of students
    #  @param m Students macid
    #  @param i Students information of type SinfoT
    @staticmethod
    def add(m, i):
        for student in SALst.s:  # for every element in SALst
            if m in student:  # if the macid is present in that element
                raise KeyError
        SALst.s.append((m, i))  # otherwise append student to the list

    ## @brief Removes a student from the list
    #  @param m Students macid
    @staticmethod
    def remove(m):
        for student in SALst.s:  # for every student in SAlst
            if m in student:  # if the given macid is in this tuple
                SALst.s.remove(student)  # remove that student from the list
                return  # exit function
        raise KeyError  # otherwise if macid isn't present anywhere, raise keyerror

    ## @brief Checks if a student is in the list
    #  @param m Students Macid
    #  @return Returns true or false based off whether student is in student list
    @staticmethod
    def elm(m):
        for student in SALst.s:  # for every student in s list
            if m in student:  # if student has the given macid
                return True  # return true
        return False  # otherwise if no student has this macid return false

    ## @brief Returns a students information
    #  @param m Students Macid
    #  @return Students information as a data type SinfoT
    @staticmethod
    def info(m):
        for student in SALst.s:  # for every student in s list
            if m in student:  # if student has the given macid
                return student[1]  # return student SinfoT
        raise KeyError  # otherwise if no student has this macid return exception

    ## @brief Filters and sorts the list of students based off gpa
    #  @details Sorts the students in student list based off their GPA first. Then filters the
    #  @students based off the function passed in as a parameter.
    #  @param f Function passed in to filter students based off function requirements
    #  @return Returns the list of sorted and filtered students
    @staticmethod
    def sort(f):
        SALst.sorted_list = []  # initialize sorted list variable

        a_list = sorted(SALst.s, key=lambda x: x[1].gpa, reverse=True)  # sort the list

        for student in a_list:  # for every student in filtered list
            if f(student[1]):  # if the student meets the function requirements
                SALst.sorted_list.append(student[0])  # add that student's macid to the list
```

```
        return SALst.sorted_list

    ## @brief Calculated the average GPA of students based off the passed in function's
    #         \n requirements
    #  @param f Function passed in to filter students based off function requirements
    #  @return Returns the average GPA
    @staticmethod
    def average(f):
        total = 0
        num_student = 0
        for student in SALst.s:
            if f(student[1]):
                total += student[1].gpa
                num_student += 1
        if num_student == 0:
            raise ValueError
        return total / num_student

    ## @brief Allocates students into their program based off several factor
    #  @details Allocates students into their program. It first prioritizes students with
    #           \n free choice, then allocates students in order of highest to lowest GPA.
    #           \n students who failed are not allocated and filtered out. The AALst data
    #           \n type is used for all allocation.
    @staticmethod
    def allocate():
        AALst.init()
        f = SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0)
        for student in f:
            ch = SALst.info(student).choices
            AALst.add_stdnt(ch.next(), student)

        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
```

J Code for Read.py

```
## @file Read.py
#  @author Zayed Sheet
#  @brief This file contains functions that extracts student/department data from text files.
#  @date 2019-02-11


from SALst import *
import re


## @brief Uses the data from a text file to add all students
#  @details Reads from a text file and uses the data to create
#  @param s This parameter is a text file that contains a student on each line.
def load_stdnt_data(s):
    f = open(s, "r")
    lines = f.read().splitlines()
    f.close()
    SALst.init()
    dept = []
    free = True

    for i in lines:
        s = re.split(', \[|\], |, ', i)
        for i in range(5, len(s)-1):
            dept.append(DeptT(s[i]))

        if s[len(s)-1] == "True":
            free = True
        else:
            free = False

        info = SInfoT(s[1], s[2], GenT(s[3]), float(s[4]), SeqADT(dept), free)
        SALst.add(s[0], info)


## @brief  Obtains information on the engineering streams and their respective capacities.
#  @param s This parameter is a text file that contains the engineering streams
def load_dcap_data(s):
    DCapALst.init()
    with open(s, "r") as f:
        lines = [line.strip() for line in f if line.strip()]
        for i in range(len(lines)):  # for every line
            dept = lines[i].split(', ')  # creates a list with department and capacity
            temp = DeptT(dept[0])
            DCapALst.add(temp, int(dept[1]))
    f.close()
```

K Code for test_All.py

```
import pytest
from SeqADT import *
from SALst import *
from DCapALst import *


class TestSeqADT:
    def test_next(self):
        s = SeqADT([1, 2, 3])
        s.next()
        s.next()
        assert s.next() == 3

    def test_empty_next(self):
        s = SeqADT([])
        with pytest.raises(StopIteration):
            s.next()

    def test_start(self):
        s = SeqADT([1, 2, 3])
        s.next()
        s.next()
        s.start()
        assert s.next() == 1

    def test_end(self):
        s = SeqADT([1, 2])
        s.next()
        s.next()
        assert s.end() == True

    def test_end_2(self):
        s = SeqADT([1, 2])
        assert s.end() == False


class TestDCapALst:

    DCapALst.init()

    def test_add(self):
        DCapALst.add(DeptT.civil, 20)
        assert DCapALst.s == [(DeptT.civil, 20)]

    def test_add_2(self):
        with pytest.raises(KeyError):
            DCapALst.add(DeptT.civil, 20)

    def test_remove(self):
        DCapALst.remove(DeptT.civil)
        assert DCapALst.s == []

    def test_remove_2(self):
        with pytest.raises(KeyError):
            DCapALst.remove(DeptT.civil)

    def test_elm(self):
        assert DCapALst.elm(DeptT.civil) == False

    def test_elm_2(self):
        DCapALst.add(DeptT.civil, 20)
        assert DCapALst.elm(DeptT.civil) == True

    def test_capacity(self):
        assert DCapALst.capacity(DeptT.civil) == 20

    def test_capacity_2(self):
        with pytest.raises(KeyError):
            DCapALst.capacity(DeptT.electrical)


class TestSALst:
    SALst.init()

    def test_add(self):
        sinfo1 = SInfoT("first", "last", GenT.male, 12.0,
```

```
                    SeqADT([DeptT.civil, DeptT.chemical]), False)
    SALst.add("lastf", sinfo1)
    assert SALst.s == [("lastf", sinfo1)]

def test_add_2(self):
    sinfo1 = SInfoT("first", "last", GenT.male, 12.0,
                    SeqADT([DeptT.civil, DeptT.chemical]), False)
    with pytest.raises(KeyError):
        SALst.add("lastf", sinfo1)

def test_remove(self):
    sinfo1 = SInfoT("first", "last", GenT.male, 12.0,
                    SeqADT([DeptT.civil, DeptT.chemical]), False)
    SALst.remove("lastf")
    assert SALst.s == []

def test_remove_2(self):
    sinfo1 = SInfoT("first", "last", GenT.male, 12.0,
                    SeqADT([DeptT.civil, DeptT.chemical]), False)
    with pytest.raises(KeyError):
        SALst.remove("lastf")

def test_elm(self):
    assert SALst.elm("firstf") == False

def test_elm_2(self):
    sinfo1 = SInfoT("first", "last", GenT.male, 12.0,
                    SeqADT([DeptT.civil, DeptT.chemical]), False)
    SALst.add("lastf", sinfo1)
    assert SALst.elm("lastf") == True

def test_info(self):
    SALst.init()
    sinfo1 = SInfoT("first", "last", GenT.male, 12.0,
                    SeqADT([DeptT.civil, DeptT.chemical]), False)
    SALst.add("lastf", sinfo1)
    assert SALst.info("lastf") == sinfo1

def test_info_2(self):
    SALst.init()
    sinfo1 = SInfoT("first", "last", GenT.male, 12.0,
                    SeqADT([DeptT.civil, DeptT.chemical]), False)
    SALst.add("lastf", sinfo1)
    with pytest.raises(KeyError):
        assert SALst.info("test")

def test_sort(self):
    SALst.init()
    sinfo1 = SInfoT("first", "last", GenT.male, 12.0,
                            SeqADT([DeptT.civil, DeptT.chemical]), False)

    sinfo2 = SInfoT("Zayed", "Sheet", GenT.male, 10.5,
                            SeqADT([DeptT.software, DeptT.chemical]), True)

    sinfo3 = SInfoT("Dom", "Buswoki", GenT.male, 11.0,
                            SeqADT([DeptT.software, DeptT.chemical]), False)

    sinfo4 = SInfoT("Farzad", "Valki", GenT.male, 1.0,
                            SeqADT([DeptT.software, DeptT.chemical]), False)


    SALst.add("sheetz", sinfo2)
    SALst.add("buswoki", sinfo3)
    SALst.add("lastf", sinfo1)
    SALst.add("valkif", sinfo4)

    assert SALst.sort(lambda t: not t.freechoice and t.gpa >= 4.0) == \
           ["lastf","buswoki"]

def test_sort_2(self):
    SALst.init()
    sinfo1 = SInfoT("first", "last", GenT.male, 12.0,
                            SeqADT([DeptT.civil, DeptT.chemical]), False)

    sinfo2 = SInfoT("Zayed", "Sheet", GenT.male, 10.5,
                            SeqADT([DeptT.software, DeptT.chemical]), True)

    sinfo3 = SInfoT("Dom", "Buswoki", GenT.male, 11.0,
                            SeqADT([DeptT.software, DeptT.chemical]), False)
```

```
    sinfo4 = SInfoT("Farzad", "Valki", GenT.male, 1.0,
                            SeqADT([DeptT.software, DeptT.chemical]), False)


    SALst.add("sheetz", sinfo2)
    SALst.add("buswoki", sinfo3)
    SALst.add("lastf", sinfo1)
    SALst.add("valkif", sinfo4)

    assert SALst.sort(lambda t: t.gpa >= 0) == \
           ["lastf", "buswoki", "sheetz", "valkif"]

def test_sort_3(self):
    SALst.init()
    assert SALst.sort(lambda t: t.gpa >= 0) == []

def test_average(self):
    with pytest.raises(ValueError):
        assert SALst.average(lambda x: x.gender == GenT.male)

def test_average_2(self):
    sinfo5 = SInfoT("Cat", "Gonzal", GenT.female, 8,
                            SeqADT([DeptT.software, DeptT.chemical]), True)
    sinfo6 = SInfoT("Pedram", "Yazdinia", GenT.male, 8,
                            SeqADT([DeptT.software, DeptT.chemical]), False)
    sinfo3 = SInfoT("Dom", "Buswoki", GenT.male, 10,
                            SeqADT([DeptT.software, DeptT.chemical]), False)

    SALst.add("gonzalc", sinfo5)
    SALst.add("yazdiniap", sinfo6)
    SALst.add("buswoki", sinfo3)

    assert SALst.average(lambda x: x.gender == GenT.male) == 9

def test_allocate(self):
    SALst.init()
    DCapALst.init()
    DCapALst.add(DeptT.civil,5)
    DCapALst.add(DeptT.software, 4)
    DCapALst.add(DeptT.electrical, 3)
    sinfo1 = SInfoT("first", "last", GenT.male, 12.0,
                            SeqADT([DeptT.civil, DeptT.chemical]), False)

    sinfo2 = SInfoT("Zayed", "Sheet", GenT.male, 10.5,
                            SeqADT([DeptT.software, DeptT.chemical]), True)

    sinfo3 = SInfoT("Dom", "Buswoki", GenT.male, 11.0,
                            SeqADT([DeptT.software, DeptT.chemical]), False)

    sinfo4 = SInfoT("Farzad", "Valki", GenT.male, 1.0,
                            SeqADT([DeptT.software, DeptT.chemical]), False)

    sinfo5 = SInfoT("Cat", "Gonzal", GenT.female, 8.4,
                            SeqADT([DeptT.software, DeptT.civil]), True)

    sinfo6 = SInfoT("Pedram", "Yazdinia", GenT.male, 8.0,
                            SeqADT([DeptT.electrical, DeptT.chemical]), False)



    SALst.add("gonzalc", sinfo5)
    SALst.add("yazdiniap", sinfo6)
    SALst.add("buswoki", sinfo3)
    SALst.add("lastf", sinfo1)
    SALst.add("valkarif", sinfo4)
    SALst.add("sheetz", sinfo2)
    SALst.allocate()

    assert AALst.s == [(DeptT.civil, ["lastf"]),(DeptT.chemical, []), (DeptT.electrical,
        ["yazdiniap"]),
                       (DeptT.mechanical, []), (DeptT.software, ["sheetz", "gonzalc", "buswoki"]),
                       (DeptT.materials, []), (DeptT.engphys, [])]

def test_allocate_2(self):
    SALst.init()
    SALst.allocate()

    assert AALst.s == [(DeptT.civil, []),(DeptT.chemical, []), (DeptT.electrical, []),
                       (DeptT.mechanical, []), (DeptT.software, []), (DeptT.materials, []),
                       (DeptT.engphys, [])]
```

L Code for Partner's SeqADT.py

```
## @file SeqADT.py
#  @author Justin Rosner, rosnej1
#  @brief Defining a class to store student choices
#  @date 02/06/2019


## @brief An abstract data type that iterates through a sequence
class SeqADT:

    ## @brief SeqADT constructor
    #  @details Takes a sqeuence of type T and returns an ADT that allows the
    #  user to iterate through a sequence
    #  @param x is a sequence of type T
    def __init__(self, x):

        self.__s = x
        self.__i = 0

    ## @brief start sets the counter back to 0 (ie. the start)
    def start(self):
        self.__i = 0

    ## @brief Iterates to the next element in the sequence
    #  @throw Throws StopIteration exception when i goes out of range
    #  @return Returns the next element in the sequence
    def next(self):
        if self.__i >= len(self.__s):
            raise StopIteration
        else:
            temp = self.__i
            self.__i += 1
            return (self.__s[temp])

    ## @brief Checks to see if the sequence is at the end
    #  @return A boolen value of True or False
    def end(self):
        if self.__i < len(self.__s):
            return False
        else:
            return True
```

M Code for Partner's DCapALst.py

```
## @file DCapALst.py
#  @author Justin Rosner, rosnej1
#  @brief Setting the department capacities for each engineering faculty
#  @date 02/05/2019

from StdntAllocTypes import *
from typing import NamedTuple


## @brief This class defines the tuple (dept: DeptT, cap: N)
class _DepartmentCapT(NamedTuple):
    dept: DeptT
    cap: int


## @brief An abstract object for storing the department capacities
class DCapALst:

    # A set of tuples defined as (dept: DeptT, cap: N)
    s = set()

    ## @brief Initialize empty data structure
    @staticmethod
    def init():
        DCapALst.s = set()

    ## @brief Adds a department and its capacity to the list
    #  @param d represents a department of type DeptT
    #  @param n is an integer representing the capacity of the department
    #  @throw Throws KeyError when the user tries to add a duplicate key
    @staticmethod
    def add(d, n):
        for department in DCapALst.s:
            if department.dept == d:
                raise KeyError
        DCapALst.s.add(_DepartmentCapT(d, n))

    ## @brief Deletes a department and its corresponding capacity from the list
    #  @param d represents a department of type DeptT
    #  @throw Throws KeyError when the department the user wants to delete is
    #  not in the current department list
    @staticmethod
    def remove(d):
        for department in DCapALst.s.copy():
            if department.dept == d:
                DCapALst.s.remove(department)
                return
        raise KeyError

    ## @brief Checks to see if an element already exists in the list
    #  @param d represents a department of type DeptT
    #  @return True if the element exists, and False if it does not
    @staticmethod
    def elm(d):
        for department in DCapALst.s:
            if department.dept == d:
                return True
        return False

    ## @brief Gives the user back the department capacity of thier choosing
    #  @param d represents a department of type DeptT
    #  @throw Throws KeyError if the department of choosing is not in the list
    #  @return Returns an integer value representing the capacity of the department
    @staticmethod
    def capacity(d):
        for department in DCapALst.s:
            if department.dept == d:
                return department.cap
        raise KeyError
```

N Code for Partner's SALst.py

```
## @file SALst.py
#  @author Justin Rosner, rosnej1
#  @brief Student Allocation Module
#  @date 02/09/2019

from StdntAllocTypes import *
from AALst import *
from DCapALst import *
from operator import itemgetter


## @brief This function gets the GPA of a desired student
#  @param m is a string representing a students macid
#  @param s is a set of type StudentT
#  @return a float representing the gpa of the student
def _get_gpa(m, s):
    if m in s:
        return (s[m].gpa)


## @brief An abstract object for allocating students into their second year streams
class SALst:

    # A dictionary of StudentT which is a tuple of (macid: string, info: SInfoT)
    s = {}
    ## @brief Initializes the empty data structure
    @staticmethod
    def init():
        AALst.init()
        SALst.s = {}

    ## @brief This function adds a student to the list s
    #  @param m is a string representing the students macid
    #  @param i is information about the student of type SInfoT
    #  @throw Throws KeyError if macid is already in the list
    @staticmethod
    def add(m, i):
        if m in SALst.s:
            raise KeyError
        SALst.s[m] = i

    ## @brief This function removes a student from the list s
    #  @param m is a string representing the students macid
    #  @throw Throws KeyError if the student is not in the list
    @staticmethod
    def remove(m):
        if m in SALst.s:
            del SALst.s[m]
            return
        raise KeyError

    ## @brief This function checks to see if a student is already in the list
    #  @param m is a string representing the macid of a student
    #  @return A boolean value True if the macid is in the list and False if not
    @staticmethod
    def elm(m):
        if m in SALst.s:
            return True
        return False

    ## @brief This function gets the information about the student
    #  @param m is a string representing a students macid
    #  @throw Throws an exception ValueError when the student is not in the list
    #  @return Returns the information (SInfoT) of the student
    @staticmethod
    def info(m):
        if m in SALst.s:
            return (SALst.s[m])
        raise KeyError

    ## @brief This function returns a list of students sorted by gpa
    #  @param f is a lambda function that checks if a student has free choice and a gpa >= 4.0
    #  @return A sequence of students in order of decreasing gpa
    @staticmethod
    def sort(f):
        unsorted_list = []
```

```
        sorted_list = []

        # Iterate through the list of type StudentT
        for student in SALst.s:
            if f(SALst.s[student]):
                # Make a temp dict that contains macid and gpa and append this to a list
                temp_dict = {'macid': student,
                             'gpa': _get_gpa(student, SALst.s)}
                unsorted_list.append(temp_dict)

        temp_list = sorted(unsorted_list, key=itemgetter('gpa'), reverse=True)

        # Go through the now sorted list and append just the macid to the final list
        for element in temp_list:
            sorted_list.append(element['macid'])

        return (sorted_list)

    ## @brief This function returns the gpa of a gender specific set of students
    #  @param f is a lambda function that checks the gender of the student
    #  @throw Throws ValueError when the gender specific subset of students is NULL
    #  @return A float value representing the average gpa of a specified subset
    #  of students
    @staticmethod
    def average(f):
        total_gpa = 0.0
        count = 0

        for student in SALst.s:
            if f(SALst.s[student]):
                total_gpa += _get_gpa(student, SALst.s)
                count += 1

        if count == 0:
            raise ValueError

        return (total_gpa / count)

    ## @brief This function allocates the students into their upper year programs
    #  @throw Throws RuntimeError if student runs out of choices
    @staticmethod
    def allocate():
        # Adding students with freechoice to their department
        freechoice_list = SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0)
        for macid in freechoice_list:
            ch = SALst.info(macid).choices
            AALst.add_stdnt(ch.next(), macid)

        # Adding the students with no free choice to their department
        normal_student_list = SALst.sort(lambda t: not(t.freechoice) and t.gpa >= 4.0)
        for macid in normal_student_list:
            ch = SALst.info(macid).choices
            alloc = False

            while (not(alloc) and not(ch.end())):
                dept = ch.next()
                if (AALst.num_alloc(dept) < DCapALst.capacity(dept)):
                    AALst.add_stdnt(dept, macid)
                    alloc = True

            if (not(alloc)):
                raise RuntimeError
```