Assignment 2 Solution

Alice Ip,ipa1, 400078727

February 17, 2019

Introductory blurb.

1 Testing of the Original Program

When deciding on test cases, I used the pytest "python -m pytest –cov –cov-report html" feature as a guideline. This feature outputs the line coverage in html files for each python file and highlights the lines that were not covered in testing. This method allowed me to ensure that I covered all branches. However, some test cases made use of other class functions, and would show up as "covered" using this feature. As a result, I also made sure to make test cases to cover all possible results and exceptions in separate tests for each function in each module to ensure that individual functions worked. After running my test cases on my files, I discovered many errors in my program. An example of such an error would be in the "AALst.py" file, where if the num_alloc function was called on a department that had no elements inside, there would be a "KeyError". After this unsuccessful test case, I adjusted my file to have a branch to handle this particular situation. I was able to get a 100% coverage and pass on all 29 of my test cases

2 Results of Testing Partner's Code

When I ran my "all_Tests.py" file against my partner's SeqADT.py, DCapALst.py, SALst.py, all 29 test cases passed. Since the test cases only tested the outputs/results from calling the functions in their files, as long as my partner followed all assignment specifications that I also followed, this was an expected result. The assignment specification indicated the types of all data values, eliminating any ambiguity of interaction between the modules. Assumptions ,necessary exceptions, and the inputs and outputs of each function was explicitly stated, with only the data structure and the implementations left as a decision for my partner and I. After comparing our code for the modules, I discovered that my implementation was almost the exact same as my partners, with the exception that I used much more if statements and variables to hold boolean values, whereas my partner just returned an expression that resulted in a boolean value. The discovery regarding similarities was not a surprise, considering the strict assignment specifications.

3 Critique of Given Design Specification

Advantages: The expectations of inputs for SInfoT were made clear through the explicit description of values contained in the enum types GenT and DeptT. The names of parameters in arguments were consistent. For example, in the DCapALst, whenever the parameter passed in was a department name, it is represented using the character "d" each time. Each function in every module mostly had minimality, they had a clear purpose, and it was clear how it would be used. There was high cohesion inside each module, the functions inside were all related to the data structure it worked with. The modules had low coupling, did not excessively or unnecessarily rely on other modules to function properly. For example, the SeqADT would not access the DCapALst to check if the department that was in the SeqADT was already at full capacity, rather it is used inside the allocate function, which would be the main function that uses both modules. The next function of SeqADT, the sort function of SALst and the average function of SALst exercised a reasonable amount of generality. The next function only needs to access items in order, so it only accesses items in order. The sort and average functions of SALst allow for a user defined filter, and allows the functions to be used to extract information for various purposes depending on the user's need. The design specification is opaque: includes absolutely necessary information such as how the state variables will be stored, but allows the programmer to make their own implementation decisions on what will make the program as efficient as possible.

Disadvantages: The use of the python built in exceptions simplified the program much more, however, they could be customized to include meaningful statements. For example if the program tried to add a department into DCapALst that was already entered before, there could be a message that states "Department already in list". The program also does not check the inputs to see if they are valid. For example, the first and last names should have a character limit, and not contain any numbers of symbols. As well, when passing a majority of data into functions, it is not checked to see if it is the right type, and is just assumed to be the expected type.

4 Answers

a Contrast the natural language of A1 to the formal specification of A2. What are the advantages and disadvantages of each approach?

The natural language of A1 was very easy to understand just from reading, and explained exactly what the purpose and function of each function was. While I was working on A1, I knew exactly what the purpose of each function was, and what it did, and had an idea of how it might be used, or how it may use other functions in the program. This type of specification is much more beginner friendly, and allowed much more implementation freedom. At the same time, assumptions and exceptions were almost completely omitted. For example, the programmer would decide the outcome of a free choice student whose first choice was a department that was already at full capacity. As a result, the program may not work as expected by the person who gave the specification.

The formal specification of A2 was much more organized and explicit on what was expected out of each function. Assumptions, exceptions, inputs, outputs, uses are clearly stated. As long as the programmer followed such guidelines, the program will most likely work as expected. Since the guidelines are strict, it is very possible that a different person could work on each module, and still end with a fully functional program with limited coordination. However, it was much harder figuring out exactly what the purpose of each module was, in contrast with the natural language. The formal specification required more advanced math knowledge in order to comprehend the semantics of the transition statements.

b The specification makes the assumption that the gpa will be between 0 and 12. How would you modify the specification to change this assumption to an exception? Would you need to modify the specification to replace a record type with a new ADT?

To turn this into an exception, you could include in the add function of SALst.py file:

exception: $((i.gpa < 0 \vee i.gpa > 12) \Rightarrow \text{KeyError})$

Alternatively, this could be an exception in the Read.py load_stdnt_data function. Since gpa values are generally part of a set of discrete and finite numbers, a new enum similar to the GenT and DeptT enum could be made to restrict the values that gpa could be. The entire record-like type for SInfoT does not need to be replaced, just the type of gpa.

c If we ignore the functions sort, average and allocate, the two modules SALst and DCapALst are very similar. Ignoring the functions mentioned, how could the documentation be modified to take advantage of the similarities?

Instead of having two completely separate modules, you could have a parent class with the init, add, remove, and elm functions. SALst and DCapALst would be children that inherit from this parent class, and each have their own specializations (SALst would have sort, average, allocate functions, and DCapALst would have it's capacity function). This parent class could then be used for other problems where you may want to add, remove, and check if an item is inside.

d A1 had a question about generality of an interface. In what ways is A2 more general than A1

A2 has the SeqADT data type, which takes in a sequence of T. This T can be any data type, and it would still be able to return a value, and increase the iteration. It does not limit the use of this data type to our current problem. As well, we make use of passing functions to the sort and average functions in SALst.py. These functions are a filter for the list to be sorted and averaged, and can change depending on how the caller wants to use the information. On the other hand, depending on how you implemented the program in A1, most likely it would only have one parameter, the list, and any filtering done on this list would be hardcoded in the function.

e The list of choices for each students is represented by a custom object, SeqADT, instead of a Python list. For this specific usage, what are the advantages of using SeqADT over a regular list?

SeqADT has the next function, which will return the current value and increment by one. This limits the accessibility, because the order of which you can obtain values is fixed. In our problem, the SeqADT is used to hold the prefered department a student would like to be in, in order of most preferred to least preferred. Our program hopes to allocate students in the most satisfiable way, so there will only be a need to access and check the deparments in one order, there is no need for random access, and will prevent any possible errors relating to accessing the wrong preference of a student.

f Many of the strings in A1 have been replaced by enums in A2. For these cases, what advantages do enums provide? Why weren't enums also introduced in the specification for macids?

Enums allow for a standardized typing for values such as gender, and departments, which are from a finite and predetermined set of immutable values. In our problem, our enum values genT.male allows the programmer to easily identify that this is a representation for data that indicates that the student is male, in comparison to using the string "m" to represent male. We can implement consistency in our program by constantly using this type, and expect this particular type in a particular format when being used in user supplied functions such as sort, and average. Macids, on the other hand, should not be of the enum type, because there is an almost infinite number of possible macids, and are not predefined.

E Code for StdntAllocTypes.py

```
## @file StdntAllocTypes.py
#  @author Alice Ip ipa1
#  @title Student Allocation Types
#  @date 2019-02-11

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


# @brief An enum type for gender
class GenT(Enum):
    male = 0
    female = 1


# @brief An enum type for department names
class DeptT(Enum):
    civil = 0
    chemical = 1
    electrical = 2
    mechanical = 3
    software = 4
    materials = 5
    engphys = 6


# @brief An NamedTuple to store student information
class SInfoT(NamedTuple):
    fname: str
    lname: str
    gender: GenT
    gpa: float  # type is real
    choices: SeqADT
    freechoice: bool
```

F Code for SeqADT.py

```
## @file SeqADT.py
#  @author Alice Ip ipa1
#  @Title Sequence Abstract Data Type
#  @date 2019-02-11

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

    ## @brief SeqADT constructor
    #  @details takes a sequence of T
    #  @param x sequence of t
    def __init__(self, x):

        self.__s = x
        self.__i = 0

    ## @brief start resets the i value to 0
    def start(self):

        self.__i = 0

    ## @brief next increases the value of i
    #  @return the value of the previous i
    def next(self):
        self.__i += 1

        if self.__i >= len(self.__s) + 1:
            raise StopIteration

        return self.__s[self.__i - 1]

    ## @brief end determines when to end iteration
    #  @return a boolean value if i >= s
    def end(self):
        if self.__i >= len(self.__s):
            return True
        else:
            return False
```

G Code for DCapALst.py

```
## @file DCapALst.py
#  @author Alice Ip ipa1
#  @Title Department Capacity List
#  @date 2019-02-11


# @brief List that holds departments and their capacity
class DCapALst:

    ## @brief init initializes data structure
    @staticmethod
    def init():

        DCapALst.s = []

    ## @brief add adds a department to the data structure
    #  @param d department name
    #  @param n current capacity of department
    @staticmethod
    def add(d, n):

        for department in DCapALst.s:
            if department[0] == d:
                raise KeyError

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

    ## @brief remove removes a department from data structure
    #  @param d department name
    @staticmethod
    def remove(d):

        found = False
        for x in range(0, len(DCapALst.s)):
            if DCapALst.s[x][0] == d:
                del DCapALst.s[x]
                found = True
        if (found is False):
            raise KeyError

    ## @brief elm checks to see if deparment is in list
    #  @param d department name
    #  @return true if department is in list, false otherwise
    @staticmethod
    def elm(d):

        for department in DCapALst.s:
            if department[0] == d:
                return True
        return False

    ## @brief capacity outputs the capacity of a department
    #  @param d department name
    #  @return department's capacity
    @staticmethod
    def capacity(d):

        found = False
        for department in DCapALst.s:
            if department[0] == d:
                found = True
                return department[1]
        if (found is False):
            raise KeyError
```

H Code for AALst.py

```
## @file AALst.py
#  @author Alice Ip ipa1
#  @title Allocation Association List Module
#  @date 2019-02-11

from StdntAllocTypes import *


## @brief List that holds the students allocated in a particular department
class AALst:

    # @brief init creates the allocation list data structure
    @staticmethod
    def init():

        AALst.s = {}

    ## @brief add adds a student to a department
    #  @param dep department name
    #  @param m name of student
    @staticmethod
    def add_stdnt(dep, m):
        if (dep not in AALst.s):
            AALst.s[dep] = []
            AALst.s[dep].append(m)
        else:
            AALst.s[dep].append(m)

    ## @brief lst_alloc outputs names of students in department
    #  @param d department name
    #  @return list of students
    @staticmethod
    def lst_alloc(d):

        return AALst.s[d]

    ## @brief num_alloc outputs number of students in a department
    #  @param d department name
    #  @return number of students in a department
    @staticmethod
    def num_alloc(d):

        if (d not in AALst.s):
            return 0

        return len(AALst.s[d])
```

I Code for SALst.py

```
## @file SALst.py
#  @author Alice Ip ipa1
#  @brief Student Allocation List
#  @date 2019-02-11

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

## @brief performs functions on a list of students
class SALst:

    s = []

    @staticmethod
    def init():
        SALst.s = []  # dictionary of StudentT

    ## @brief add will add macid and info into the s dictionary
    #  @param m macid of student
    #  @param i SInfoT of student

    @staticmethod
    def add(m, i):

        found = False
        for student in SALst.s:
            if student[0] == m:
                raise KeyError
        if found is False:
            SALst.s.append((m, i))

    ## @brief remove will remove m from s
    #  @param m macid of student

    @staticmethod
    def remove(m):

        found = False
        s = SALst.s
        for x in range(0, len(s)):
            student = s[x]
            if student[0] == m:
                del s[x]
                found = True
        if found is False:
            raise KeyError

    ## @brief elm checks if m is in s
    #  @param m string
    #  @return boolean indicating if m is in s

    @staticmethod
    def elm(m):

        found = False
        for student in SALst.s:
            if student[0] == m:
                return True
        if found is False:
            return False

    ## @brief info returns the student's information
    #  @param m macid of the student
    #  @return SInfoT of student

    @staticmethod
    def info(m):

        found = False
        for student in SALst.s:
            if student[0] == m:
                return student[1]
        if found is False:
            raise KeyError
```

```
## @brief sort sorts the s based on the filter given
#  @param f filter to sort by
#  @return list of macids
@staticmethod
def sort(f):
    filtered = []
    for student in SALst.s:
        if f(student[1]):
            filtered.append(student)

    for x in range(0, len(filtered) - 1):

        if (filtered[x][1].gpa < filtered[x + 1][1].gpa):
            temp = filtered[x]
            filtered[x] = filtered[x + 1]
            filtered[x + 1] = temp
    macid_list = []
    for y in filtered:
        macid_list.append(y[0])

    return(macid_list)

## @brief average calculates the average of a filtered list of students
#  @param f filter for students to calculate average of
#  @return a float representing average
@staticmethod
def average(f):

    filtered = []
    for student in SALst.s:
        if f(student[1]):
            filtered.append(student)

    if (len(filtered) == 0):
        raise ValueError

    average_sum = 0

    for x in range(0, len(filtered)):
        average_sum += filtered[x][1].gpa

    average = average_sum / (len(filtered))
    return(average)

## @brief allocate allocates students into deparments
#  @details allocates freechoice students, regular students, according to gpa
@staticmethod
def allocate():

    AALst.init()
    freechoice_list = SALst.sort(lambda t: t.freechoice and t.gpa >= 4)

    for m in freechoice_list:
        ch = SALst.info(m).choices
        AALst.add_stdnt(ch.next(), m)

    regular_list = SALst.sort(lambda t: not t.freechoice and t.gpa >= 4)

    for m in regular_list:
        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 Alice Ip ipa1
#  @title reads student and department files
#  @date 2019-02-11

from DCapALst import *
from SALst import *
from StdntAllocTypes import *
from SeqADT import *

convert_dept = {'civil': DeptT.civil, 'chemical': DeptT.chemical,
                'electrical': DeptT.electrical, 'mechanical': DeptT.mechanical,
                'software': DeptT.software, 'materials': DeptT.materials,
                'engphys': DeptT.engphys}
convert_gender = {'male': GenT.male, 'female': GenT.female}


## @brief load_stdnt_data reads a file of student data and puts into SALst
#  @param s is the file name

def load_stdnt_data(s):
    SALst.init()

    ifile = open(s, "r")
    for line in ifile:
        student_line = line.strip()
        student_line = student_line.split(" ")

        for x in range(0, len(student_line)):
            student_line[x] = student_line[x].replace(',', '')

        macid = student_line[0]
        first_name = student_line[1]
        last_name = student_line[2]
        gender = student_line[3].replace(',', '')
        gpa = student_line[4]
        courses = []

        for x in range(5, len(student_line) - 1):
            if (x == 5):
                student_line[5] = (student_line[5])[1::]
            if (x == (len(student_line) - 2)):
                student_line[x] = (student_line[x])[0:-1]
            courses.append(convert_dept[student_line[x]])

        freechoice = student_line[len(student_line) - 1]

        student_info = SInfoT(first_name, last_name,
                              gender, gpa, (courses), freechoice)

        SALst.add(macid, student_info)


## @brief load_dcap_data reads a file of deparment capacities and puts into SALst
#  @param s is the file name

def load_dcap_data(s):
    DCapALst.init()

    ifile = open(s, "r")
    for line in ifile:
        dept_line = line.strip()
        dept_line = dept_line.split(" ")
        capacity = int(dept_line[1])
        dept_name = dept_line[0].replace(',', '')
        dept_name = convert_dept[dept_name]
        DCapALst.add(dept_name, capacity)
```

K Code for test_All.py

```
# @file test_All.py
#  @author Alice Ip ipa1
#  @Title Testing File
#  @date 2019-02-11

import pytest
from StdntAllocTypes import *
from SeqADT import *
from DCapALst import *
from AALst import *
from SALst import *
from Read import *


class TestStdntAllocTypes:

    def test_sinfot(self):
        sinfo1 = SInfoT("first", "last", GenT.male, 12.0,
                        SeqADT([DeptT.civil, DeptT.chemical]), True)
        assert (sinfo1.fname == "first")


class TestRead:

    def test_dept_data(self):
        load_dcap_data("src/DeptCap.txt")
        assert(DCapALst.elm(DeptT.civil) == True)
        assert(DCapALst.capacity(DeptT.civil) == 100)
        assert(DCapALst.elm(DeptT.chemical) == True)
        assert(DCapALst.capacity(DeptT.chemical) == 100)
        assert(DCapALst.elm(DeptT.electrical) == True)
        assert(DCapALst.capacity(DeptT.electrical) == 100)
        assert(DCapALst.elm(DeptT.mechanical) == True)
        assert(DCapALst.capacity(DeptT.mechanical) == 100)
        assert(DCapALst.elm(DeptT.software) == True)
        assert(DCapALst.capacity(DeptT.software) == 100)
        assert(DCapALst.elm(DeptT.materials) == True)
        assert(DCapALst.capacity(DeptT.materials) == 100)
        assert(DCapALst.elm(DeptT.engphys) == True)
        assert(DCapALst.capacity(DeptT.engphys) == 100)

    def test_stdnt_data(self):
        load_stdnt_data("src/StdntData.txt")
        assert(SALst.elm("macid1") == True)
        assert(SALst.info("macid1") == SInfoT(fname='firstname',
                                              lname='lastname', gender='male', gpa='9.2',
                                              choices=[DeptT.software, DeptT.chemical,
                                                       DeptT.materials],
                                              freechoice='True'))


class TestSeqADT:

    def test_next(self):
        choices = SeqADT([DeptT.civil, DeptT.chemical])
        assert(choices.next() == DeptT.civil)
        assert(choices.next() == DeptT.chemical)
        with pytest.raises(StopIteration):
            choices.next()

    def test_start(self):
        choices = SeqADT([DeptT.civil, DeptT.chemical])
        choices.start()
        assert(choices.next() == DeptT.civil)

    def test_end_true(self):
        choices = SeqADT([])
        assert(choices.end() == True)

    def test_end_false(self):
        choices = SeqADT([DeptT.software, DeptT.chemical])
        assert(choices.end() == False)


class TestSALst:
```

```
def test_add_success(self):
    alice1 = SInfoT("first", "last", GenT.male, 5.0,
                    ([DeptT.civil, DeptT.chemical]), True)
    SALst.init()
    SALst.add("ipa1", alice1)
    assert(SALst.elm("ipa1") == True)

def test_add_failure(self):
    alice1 = SInfoT("first", "last", GenT.male, 5.0,
                    ([DeptT.civil, DeptT.chemical]), True)
    SALst.init()
    SALst.add("ipa1", alice1)
    with pytest.raises(KeyError):
        SALst.add("ipa1", alice1)

def test_remove_success(self):
    alice1 = SInfoT("first", "last", GenT.male, 5.0,
                    ([DeptT.civil, DeptT.chemical]), True)
    SALst.init()
    SALst.add("ipa1", alice1)
    SALst.remove("ipa1")
    assert(SALst.elm("ipa1") == False)

def test_remove_failure(self):

    SALst.init()
    with pytest.raises(KeyError):
        SALst.remove("ipa1")

def test_elm_true(self):
    alice1 = SInfoT("first", "last", GenT.male, 5.0,
                    ([DeptT.civil, DeptT.chemical]), True)
    SALst.init()
    SALst.add("ipa1", alice1)
    assert(SALst.elm("ipa1") == True)

def test_elm_false(self):
    SALst.init()
    assert(SALst.elm("ipa1") == False)

def test_sort(self):
    alice1 = SInfoT("first", "last", GenT.male, 5.0,
                    ([DeptT.civil, DeptT.chemical]), True)
    alice2 = SInfoT("first", "last", GenT.male, 13.0,
                    ([DeptT.civil, DeptT.chemical]), True)
    alice3 = SInfoT("first", "last", GenT.male, 7.0,
                    ([DeptT.civil, DeptT.chemical]), True)
    SALst.init()
    SALst.add("ipa1", alice1)
    SALst.add("ipa2", alice2)
    SALst.add("ipa3", alice3)
    id_list = SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0)
    assert(id_list == ['ipa2', 'ipa3', 'ipa1'])

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

def test_average_value(self):
    alice1 = SInfoT("first", "last", GenT.male, 5.0,
                    ([DeptT.civil, DeptT.chemical]), True)
    alice2 = SInfoT("first", "last", GenT.male, 13.0,
                    ([DeptT.civil, DeptT.chemical]), True)
    alice3 = SInfoT("first", "last", GenT.male, 7.0,
                    ([DeptT.civil, DeptT.chemical]), True)
    SALst.init()
    SALst.add("ipa1", alice1)
    SALst.add("ipa2", alice2)
    SALst.add("ipa3", alice3)

    average_value = SALst.average(lambda x: x.gender == GenT.male)
    true_value = average_value - 8.333333333333334
    close = False
    if (true_value < 0.0005):
        close = True
    assert(close is True)

def test_allocate_free(self):
    DCapALst.init()
```

```
        DCapALst.add(DeptT.civil, 2)

        sinfo1 = SInfoT("first", "last", GenT.male, 12.0,
                        SeqADT([DeptT.civil, DeptT.chemical]), True)

        SALst.init()
        SALst.add("stdnt1", sinfo1)

        AALst.init()
        SALst.allocate()
        assert(AALst.lst_alloc(DeptT.civil) == ['stdnt1'])

    def test_allocate_regular(self):
        DCapALst.init()
        DCapALst.add(DeptT.civil, 2)

        sinfo1 = SInfoT("first", "last", GenT.male, 12.0,
                        SeqADT([DeptT.civil, DeptT.chemical]), False)

        SALst.init()
        SALst.add("stdnt1", sinfo1)

        AALst.init()
        SALst.allocate()
        assert(AALst.lst_alloc(DeptT.civil) == ['stdnt1'])

    def test_not_allocated(self):
        DCapALst.init()
        DCapALst.add(DeptT.civil, 2)

        sinfo1 = SInfoT("first", "last", GenT.male, 12.0,
                        SeqADT([]), False)

        SALst.init()
        SALst.add("stdnt1", sinfo1)

        AALst.init()

        with pytest.raises(RuntimeError):
            SALst.allocate()


    def test_info_none(self):
        SALst.s = []
        with pytest.raises(KeyError):
            SALst.info('ipa1')

    def test_info_present(self):
        alice1 = SInfoT("first", "last", GenT.male, 5.0,
                        ([DeptT.civil, DeptT.chemical]), True)
        alice2 = SInfoT("first", "last", GenT.male, 13.0,
                        ([DeptT.civil, DeptT.chemical]), True)
        alice3 = SInfoT("first", "last", GenT.male, 7.0,
                        ([DeptT.civil, DeptT.chemical]), True)
        SALst.init()
        SALst.add("ipa1", alice1)
        SALst.add("ipa2", alice2)
        SALst.add("ipa3", alice3)
        info_present = SALst.info('ipa1')
        assert(info_present == SInfoT("first", "last", GenT.male,
                                      5.0, ([DeptT.civil, DeptT.chemical]), True))


class TestAALst:

    def test_add_to_existing(self):
        AALst.init()
        AALst.add_stdnt(DeptT.software, "ipa1")
        AALst.add_stdnt(DeptT.software, "ipa2")
        assert(AALst.lst_alloc(DeptT.software) == ["ipa1", "ipa2"])
        assert(AALst.num_alloc(DeptT.software) == 2)


class TestDCapALst:

    def test_add_existing(self):
        DCapALst.init()
        DCapALst.add(DeptT.software, 100)
        with pytest.raises(KeyError):
            DCapALst.add(DeptT.software, 100)
```

```
def test_remove_True(self):
    DCapALst.init()
    DCapALst.add(DeptT.software, 100)
    DCapALst.remove(DeptT.software)
    assert(DCapALst.elm(DeptT.software) == False)

def test_remove_False(self):
    DCapALst.init()
    with pytest.raises(KeyError):
        DCapALst.remove(DeptT.engphys)

def test_elm_True(self):
    DCapALst.init()
    DCapALst.add(DeptT.materials, 100)
    assert(DCapALst.elm(DeptT.materials) == True)

def test_elm_False(self):
    DCapALst.init()
    assert(DCapALst.elm(DeptT.civil) == False)
    assert(DCapALst.elm(DeptT.chemical) == False)
    assert(DCapALst.elm(DeptT.electrical) == False)
    assert(DCapALst.elm(DeptT.mechanical) == False)
    assert(DCapALst.elm(DeptT.software) == False)
    assert(DCapALst.elm(DeptT.materials) == False)
    assert(DCapALst.elm(DeptT.engphys) == False)

def test_capacity_True(self):
    DCapALst.init()
    DCapALst.add(DeptT.materials, 50)
    assert(DCapALst.capacity(DeptT.materials) == 50)

def test_capacity_False(self):
    DCapALst.init()
    with pytest.raises(KeyError):
        DCapALst.capacity(DeptT.materials)
```

L Code for Partner's SeqADT.py

```
## @file SeqADT.py
#  @title Sequence ADT
#  @author Dominik Buszowiecki
#  @date February 9, 2019


## @brief An abstract data type that represents a sequence of values
class SeqADT:

    ## @brief SeqADT constructor
    #  @details Initializes the state variables of SeqADT. The state variables are a list that
    #           is given as a parameter and a variable used to index the list
    #           (initialized to 0).
    #  @param x A list of values
    def __init__(self, x: list):
        self.__s = x
        self.__i = 0

    ## @brief start will reset the index state variable to 0
    def start(self):
        self.__i = 0

    ## @brief next will return the next value in the sequence
    #  @exception throws StopIteration if there is no more items in the sequence
    #  @return value of next item in the sequence
    def next(self):
        if self.__i >= len(self.__s):
            raise StopIteration
        self.__i += 1
        return self.__s[self.__i - 1]

    ## @breif end will check if there are more items in the sequence
    #  @return True if there are no more items in the sequence, otherwise False
    def end(self) -> bool:
        return self.__i >= len(self.__s)
```

M Code for Partner's DCapALst.py

```
## @file DCapALst.py
#  @title Department Capacity Association List
#  @author Dominik Buszowiecki
#  @date February 9, 2019

from StdntAllocTypes import *


## @brief An abstract data type containing the capacities of engineering departments as a list
class DCapALst:

    ## @brief Initializes the Department Capacity List to be empty
    @staticmethod
    def init():
        DCapALst.s = []

    ## @brief Adds a department and its capacity to the list
    #  @exception throws KeyError if the given department has been added before
    #  @param d A department of type StdntAllocTypes.DeptT
    #  @param n An integer representing the capacity of the department (d parameter)
    @staticmethod
    def add(d: DeptT, n: int):
        for i in DCapALst.s:
            if d == i[0]:
                raise KeyError
        DCapALst.s.append((d, n))

    ## @breif Removes a department and its capacity from the list
    #  @exception throws KeyError if the given department is not in DCapALst
    #  @param d A department of type StdntAllocTypes.DeptT to be removed
    @staticmethod
    def remove(d: DeptT):
        for i in range(0, len(DCapALst.s)):
            if d == DCapALst.s[i][0]:
                del DCapALst.s[i]
                return
        raise KeyError

    ## @brief elm checks if a department has been added
    #  @param d A department of type StdntAllocTypes.DeptT
    #  @return True if the department has been added, otherwise False
    @staticmethod
    def elm(d: DeptT) -> bool:
        for i in DCapALst.s:
            if d == i[0]:
                return True
        return False

    ## @brief capacity returns the capacity of a department
    #  @exception throws KeyError if the department given is not in DCapALst
    #  @param d A department of type StdntAllocTypes.DeptT
    #  @return An integer representing the capacity of the department given as a parameter.
    @staticmethod
    def capacity(d: DeptT) -> bool:
        for i in DCapALst.s:
            if d == i[0]:
                return i[1]
        raise KeyError
```

N Code for Partner's SALst.py

```
## @file SALst.py
#  @title Student Association List
#  @author Dominik Buszowiecki
#  @date February 9, 2019

from StdntAllocTypes import *
from AALst import *
from DCapALst import *
from typing import Callable


## @brief An abstract data type of all first year engineerng students
class SALst:

    ## @brief init initializes the list of students to be empty
    @staticmethod
    def init():
        SALst.s = []

    ## @brief Adds a student into the SALst
    #  @exception throws KeyError if the student given has been added before
    #  @param m A string of a student's macid
    #  @param i Information of a student given with the data type StdntAllocTypes.SInfoT
    @staticmethod
    def add(m: str, i: SInfoT):
        for student in SALst.s:
            if student[0] == m:
                raise KeyError
        SALst.s.append((m, i))

    ## @brief Removes a student from the SALst
    #  @exception throws KeyError if a student to be removed is not found
    #  @param m A string of a student's macid
    @staticmethod
    def remove(m: str):
        for i in range(0, len(SALst.s)):
            if SALst.s[i][0] == m:
                del SALst.s[i]
                return
        raise KeyError

    ## @brief elm checks if a student is already in the SALst
    #  @param m A string of a student's macid
    #  @return True if a student is in SALst, otherwise False
    @staticmethod
    def elm(m: str):
        for student in SALst.s:
            if student[0] == m:
                return True
        return False

    ## @brief returns the information assoaciated with a student
    #  @exception throws KeyError if the student is not found
    #  @param m A string of a student's macid
    #  @return A students information with the type StdntAllocTypes.SInfoT
    @staticmethod
    def info(m: str) -> SInfoT:
        for student in SALst.s:
            if student[0] == m:
                return student[1]
        raise KeyError

    ## @brief Sorts a subset of students based on GPA
    #  @details The method is given a function that is able to filter a student. The filter
    #           function takes in a student (SInfoT) and returns True if they pass the filter.
    #           The method will return a list of macids that passed the filter, sorted by
    #           their GPA in descending order.
    #  @param f A filtering function that returns a boolean
    #  @return A list of strings (each string is a macid) sorted by their GPA in
    #          descending order
    @staticmethod
    def sort(f: Callable[[], bool]) -> list:
        temp_l = []
        for student in SALst.s:
            if f(student[1]):
                temp_l.append(student)
        temp_l = sorted(temp_l, key=lambda gpa_student: gpa_student[1].gpa, reverse=True)
        sorted_list = []
        for i in temp_l:
            sorted_list.append(i[0])
        return sorted_list

    ## @brief Computes the average of a particular subset of students
    #  @details The method is given a function that is able to filter a student. The function
    #           takes in a student(SInfoT) and returns True if they pass the filter. The
    #           method will then compute the average GPA amongst students who passed the
    #           filter.
    #  @exception throws ValueError if there are no students that pass the filter function.
    #  @param f A filtering function that returns a boolean
    #  @return A float representing the average GPA amongst a subset of students
    @staticmethod
    def average(f: Callable[[], bool]) -> float:
        i = 0
        size = 0
        for student in SALst.s:
            if f(student[1]):
                i += student[1].gpa
                size += 1
        if size == 0:
            raise ValueError
        else:
            return i / size

    ## @breif Allocates students in SALst into their program
    #  @details Students are allocated into a department in AALst.
    #  Students with free choice are allocated first. The remaining students are allocated in
    #  a order based on their GPA, a student is allocated into their highest preferred choice
    #  that is not full in capacity.
    #  @exception throws RuntimeError if all of a student's choices are full.
    @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
```