Assignment 2 Solution

Michael Barreiros, barreirm

February 17, 2019

The purpose of the program created in Assignment 2 was to read and process data from first-year engineering students and allocate them into one of their choices.

1 Testing of the Original Program

The test_All.py module was to only test SeqADT, DCapALst, and SALst. AALst and StdntAllocTypes are tested as a biproduct of these test cases. The module that was not tested was the Read.py module as there would be too many test cases that would need to be created.

The tests in the test_All module tested for base cases, such as empty sets or initialization, tested all methods and tested for all proper calls of exceptions. Tests also tested for proper return values/expected values.

With this in mind, test_All.py had 32 test cases and my code passed all test cases. The coverage for my test cases not including Read.py or A2Examples.py was 98.33%.

1.1 Testing SeqADT

SeqADT had 11 tests in total:

- Init was tested 3 times, once with None as the input and checking if s was None another test to check that i was 0 in this case. The third test was testing if given a list of integers would s return that same list of integers.
- Start was tested 2 times, one test was checking if start was called right from the start that it would return zero. The other test was checking if next was called a few times then start was called that it would again return zero.
- Next was tested 4 times, 3 of the tests were making sure that the correct department was returned when calling next on a known list of departments. The fourth test was

checking that the StopIteration exception would raise if next was called too many times and would have gone out of bounds of the sequences.

- End was tested 2 times, one test checked to see that if end was called when we haven't reached the end of the sequence then it returned false. The other test was checking that it indeed returned true if we reached the end of the sequence.

1.2 Testing DCapALst

DCapALst had 8 tests in total:

- Init was tested 1 time. The test was checking that when DCapALst was initiated that s is an empty dictionary, .
- Add was tested 2 times, one tested for the correct output when calling DCapALst.s after adding two unique departments. The other test checked to see if the KeyError exception raised when trying to add a department that already exists in s.
- Remove was tested 2 times, one test checked to see after adding a department to an empty s, then removing the same department, if s was empty once again. The next test checked to see if the KeyError exception was raised when attempting to remove a department that was not in s.
- Elm was tested 1 time to see after adding a department to s then calling elm for that department if it returned true. Another test should have been added to see if elm was called for a department that was not in s if it returned false as expected.
- Capacity was tested 2 times, one test checked to see after a department was added to s if the correct capacity was returned when checking the capacity of that department. The other test checked to see if the KeyError exception was raised when attempting to check the capacity of a department that does not exist in s.

1.3 Testing SALst

SALst had 13 tests in total:

- Init was tested 1 time. The test was checking that when SALst is initiated that s is an empty dictionary, .
- Add was tested 2 times, one test checked to see after a student was added to an empty dictionary if s contained that student formatted in the appropriate way. The second test checked to see if the exception KeyError was raised if attempting to add a student that already exists in s.

- Remove was tested 2 times, one test checked to see after a student was added to an empty s then removed from s if s was empty once again. The second test checked to see if the exception KeyError was raised if attempting to remove a student that does not exist in s.
- Sort was tested 2 times, one test checked to see after adding 4 unique students to s then attempting to sort s by filtering out students that do not have free choice and that do not have a gpa above 4.0 and then sort by gpa if the output for that is as expected. The second test checked to see after sorting an s that is empty if an empty list will be returned.
- Average was tested 2 times, one test checked to see if after adding 4 unique students to s if the average returned is the expected average. The second test checked to see if the ValueError exception was raised after attempting to get the average when there is no members for the gender we ask for.
- Allocate was tested 4 times, one test checked to see after adding 2 departments and 4 students if students were properly allocated to one of the departments. This test checked also to see if free choice students took precedence over non free choice students. The second test checked to see after adding 2 departments and no students if after running allocate the department's allocation list was empty. The third test checked to see after adding 2 departments with 0 capacity and 1 student with no free choice and then trying to run allocate if the RuntimeError exception was raised. The fourth test checked to see after adding 2 departments and 3 students if everyone was allocated as expected.

2 Results of Testing Partner's Code

Testing my partner's code with my test module resulted in 12 failed and 20 passed tests. Discussion of failed tests:

- test_SeqADT_init(1 through 3), and test_SeqADT(1 and 2) failed because my partner named their state variables with the double underscore prefix while I did not use the double underscore prefix. When attempting to compare the state variable i or s to something the test would fail since I am not trying to access the state variables with the double underscore prefix.

- test_DCapAList_init1 failed because my partner chose to represent an empty set using set() while I chose to represent an empty set as , so when comparing set() to they are not equal and therefore fails my test.

- test_DCapAList_add1 failed because my partner used a NamedTuple to represent a department and capacity pair while I used the dictionary's method of adding entries. I feel as though my partner's method of creating a class for the NamedTuple is more effective as it leaves the data less ambiguous.

- test_DCapAList_remove1 failed for the same reason test_DCapAList_init1 failed. We represented out empty sets differently.

- test_SALst_allocate(1 through 4) failed because of the way we implemented a tuple of (dept: DeptT, cap: N). By making a seperate class to represent the tuple I was unable to compare it directly to my implementation of the tuple as a dictionary with the dept as the key and the cap as the value. This difference in implementation is what caused the failure.

I will not be attempting to "fix" these failures as the reason for these failures are due to the implementation differences for tuples and for naming of state variables as well as how we describe an empty set.

The test cases that passed are expecting values that were described in the specification, such as an int or a list of strings. When the expected output was not ambiguous, such as the structure of our sequences when outputted or even internally in the code, the test cases failed.

3 Critique of Given Design Specification

Advantages and disadvantages of the given design specification.

3.1 Advantages

The advantages of the given design specification is that there was very little ambiguity and allowed for a more robust program. I liked how we were given how to handle exceptions, what to include and what not to include by religously following the design specification. Making seperate classes to handle abstract data types was interesting and taught me a lot about coding in python and abstraction in code in general. Modularity was key here in this design specification with each module's methods doing only one task which made it easier to debug. The mathematical specification was appreciated since we could read the specification and understand what it was trying to describe with no questions asked. Overall I believe the design specification was very good and I appreciate the time the professor put in making such a detailed design specification.

3.2 Disadvantages

There are very little disadvantages. The design specification did not tell us how to implement a tuple exactly and after testing my partner's code it was clear that other people implemented tuples differently which caused some problems during testing. The only other disadvantage that I can think of is that the specification was so dependent on our understanding of discrete math and mathematical specifications that reading what exactly the specification was asking for and deciphering it took a lot of time.

4 Answers

a Natural language leaves room for creative freedom, if you understand how to code you can code however you'd like. However, when it comes to ambiguities and handling them this is where natural language specification breaks down. If you need to test your code or if somebody else has to test your code then they might not know how you implemented your code potentially resulting in failed test cases even though your program behaves the way that you expect it to. The formal specification is better in my opinion, it tells us how and when to handle exceptions, it tells us exactly how it expects the program to behave and it makes testing and coding easier if you can understand the specification. A disadvantage for formal specification is that it is formal, reading the specification takes time and knowledge on how to interpret all the symbols.

b The specification would handle this in the Read.py module. Either it will throw an expection while reading if it encounters a student whose gpa does not fall in the range or it does not include that student. The exact exception that gets thrown is something that I am not sure about, maybe the RuntimeError exception gets raised. I do not think that we would need to modify the specification to replace a record type with a new ADT because we would not be changing the student after it gets recorded, so there is no need for an ADT to describe students.

c The specification could include another module maybe named ALst that does the functions that are similar in both SALst and DCapALst so that we can reduce the lines of code that we have to write and also to increase modularity in our program.

d A big reason that A2 is more general than A1 is the presence of abstract data types and abstract objects. These are general things that we fill with data to create unique things but in the case of abstract data types we can create multiple instances of them simply with one class.

e The advantages of using a SeqADT instead of a python list is that we can create our own functions for traversing the list of choices and checking if the list has reached the end so that we do not go out of bounds. By using SeqADT we can handle exceptions easier because we have created functions that we know we can use to restart the iterator, or move to the next choice, or check if we reached the end of the sequence. We can also declare that when reading choices the elements of SeqADT have to be of type DeptT which is something we cannot do with a python list.

f Enums allow us to specify what we expect in our data types. In the case of DeptT we can declare that there are only 7 departments and enumerate them 1 through 7. With enums we do not have to account for potential string mismatching when testing for a certain department, as in the difference between "civil" and "Civil". String mismatching could be a problem that arises when trying to allocate a student to their choices, if a student's choice is a string we would have to try to force it into a known format perhaps using toLowerCase() in order to properly allocate that student. Another advantage is that enums provide safety, by limiting the values a variable can have we lessen the chance that when the program runs it won't have odd behaviour, instead the compiler would tell us that we have a type mismatch or something along those lines. Enums were not used for a student's macid because a macid is unique to every student. An enum declares what values something can have. If we were to implement macids as an enum then it would be a long enum consisting of every macid which makes little sense when we can just say that macid is a string and effectively have the same result with a lot less headache.

E Code for StdntAllocTypes.py

```
## @file StdntAllocTypes.py
#  @author Michael Barreiros
#  @brief StdntAllocTypes
#  @date 09/02/2019

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

## @brief GenT abstract data type for genders
#  @details can be male or female


class GenT(Enum):
    male = 1
    female = 2

## @brief DeptT abstract data type for departments
#  @details there are 7 unique departments


class DeptT(Enum):
    civil = 1
    chemical = 2
    electrical = 3
    mechanical = 4
    software = 5
    materials = 6
    engphys = 7

## @brief SInfoT abstract data type for Students
#  @details includes first name, last name, gender, gpa, 3 choices, and
#  whether or not they have free choice


class SInfoT(NamedTuple):
    fname: str
    lname: str
    gender: GenT
    gpa: float
    choices: type(SeqADT(DeptT))
    freechoice: bool
```

F Code for SeqADT.py

```
## @file SeqADT.py
#  @author Michael Barreiros
#  @brief SeqADT
#  @date 09/02/2019

## @brief An abstract data type for a sequence


class SeqADT:

    s = []
    i = 0
    ## @brief SeqADT constructor
    #  @details initalizes the sequence with a given sequence
    #  @param x is a sequence of type T that SeqADT will be initialized to
    #  @return returns itself, a SeqADT type
    def __init__(self, x):
        self.s = x
        self.i = 0

    ## @brief start method
    #  @details resets the iterator i to 0, which is the "start" of the
    #  sequence
    def start(self):
        self.i = 0

    ## @brief next method
    #  @details returns the sequence at i and adds one to the iterator, this
    #  effectively moves the iterator to the next element in the sequence
    #  @exception throws StopIteration if i is greater or equal to the
    #  size of s
    #  @return returns s[i] before i got one added to it
    def next(self):
        if self.i >= len(self.s):
            raise StopIteration
        temp = self.s[self.i]
        self.i = self.i + 1

        return temp

    ## @brief end method
    #  @details this function's purpose is to return whether or not i is
    #  at the end of s
    def end(self):
        return self.i >= len(self.s)
```

G Code for DCapALst.py

```
## @file DCapALst.py
#  @author Michael Barreiros
#  @brief DCapALst
#  @date 09/02/2019

# from StdntAllocTypes import GenT, DeptT, SInfoT

## @brief DCapALst is an abstract data dype


class DCapALst:
    s = {}

    ## @brief the constructor for DCapALst
    #  @details sets the sequence to be an empty sequence
    @staticmethod
    def init():
        DCapALst.s = {}

    ## @brief the elm function
    #  @details returns whether or not a department is an element
    #  of the sequence
    #  @param d the department name
    #  @return a boolean value of whether or not the department is
    #  in the sequence
    @staticmethod
    def elm(d):
        return d in DCapALst.s

    ## @brief the add function
    #  @details adds a department and its capacity to the sequence
    #  @param d the department name
    #  @param n the department capacity
    #  @exception KeyError if d is already in the sequence
    @staticmethod
    def add(d, n):
        if DCapALst.elm(d):
            raise KeyError
        DCapALst.s[d] = n

    ## @brief the remove function
    #  @details removes a department and its capacity value from the sequence
    #  @param d the department name
    #  @exception KeyEror if d is not in the sequence
    @staticmethod
    def remove(d):
        if not(DCapALst.elm(d)):
            raise KeyError
        del DCapALst.s[d]

    ## @brief the capacity function
    #  @details outputs the capacity value of a given department
    #  @param d the department name
    #  @exception KeyError if d is not in the sequence
    #  @return DCapALst.s[d] this is the capacity of the department
    #  that was given
    @staticmethod
    def capacity(d):
        if not(DCapALst.elm(d)):
            raise KeyError

        return DCapALst.s[d]
```

H Code for AALst.py

```
## @file AALst.py
#  @author Michael Barreiros
#  @brief AALst
#  @date 09/02/2019

from StdntAllocTypes import DeptT

## @brief AALst an abstract data type


class AALst:
    s = {}
    ## @brief the constructor for AALst
    #  @details goes through every department in DepT and sets its key
    #  to an empty list
    @staticmethod
    def init():
        for d in DeptT:
            AALst.s[d] = []

    ## @brief the add_stdnt function
    #  @details adds a string, m, to the department, dep.
    #  @param dep the department name
    #  @param m a string
    @staticmethod
    def add_stdnt(dep, m):
        if dep in AALst.s:
            AALst.s[dep].append(m)

    ## @brief the lst_alloc function
    #  @details returns the sequence of strings for a department
    #  @param d the department name
    #  @return AALst.s[d] the List associated with a department
    @staticmethod
    def lst_alloc(d):
        return AALst.s[d]

    ## @brief the num_alloc function
    #  @details outputs the size of the sequence that is the value to
    #  a department, d.
    #  @param d the department name
    #  @return the length of the sequence that is associated with a department
    @staticmethod
    def num_alloc(d):
        return len(AALst.s[d])
```

I Code for SALst.py

```
## @file SALst.py
#  @author Michael Barreiros
#  @brief SALst
#  @date 11/02/2019

# from StdntAllocTypes import GenT, DeptT, SInfoT
from AALst import AALst
from DCapALst import DCapALst

## @brief SALst an abstract data type for an allocated list of students


class SALst:

    s = {}

    ## @brief the constructor for SALst
    @staticmethod
    def init():
        SALst.s = {}

    ## @brief the elm function
    #  @details returns a boolean for whether or not m exists in the set
    #  @return a boolean value for whether or not m exists in the set
    @staticmethod
    def elm(m):
        return m in SALst.s

    ## @brief the add function
    #  @details adds a student by their macid m to the list
    #  @param m the student's macid
    #  @param i the student info of type SInfoT associated with the student
    #  @exception KeyError if the macid m already appears in the set
    @staticmethod
    def add(m, i):
        if SALst.elm(m):
            raise KeyError
        SALst.s[m] = i

    ## @brief the remove functtion
    #  @details removes a student by their macid m from the set
    #  @param m the student's macid
    #  @exception KeyError if the macid is not in the set
    @staticmethod
    def remove(m):
        if not(SALst.elm(m)):
            raise KeyError
        del SALst.s[m]

    ## @brief the info function
    #  @details this function returns the Student information for a given macid
    #  @param m the student's macid
    #  @exception KeyError if the given student doesn't exist in the set
    #  @return the student information of type SInfoT
    @staticmethod
    def info(m):
        if not(SALst.elm(m)):
            raise KeyError

        return SALst.s[m]

    ## @brief the sort function
    #  @details sorts all members of the set that are filtered by a function f
    #  @param f a function to be applied to the sequence. It takes aspects of
    #  SInfoT and returns a boolean
    #  @return L a sequence of strings that are sorted based on the function
    #  that was passed through
    @staticmethod
    def sort(f):
        usrtd = {}
        for macid in SALst.s:
            if f(SALst.info(macid)):
                usrtd[macid] = SALst.info(macid)
        ## newList was sorted using a line of code that was found
        #  on stackoverflow
        #  link is https://stackoverflow.com/questions/72899/
        # how-do-i-sort-a-list-of-dictionaries-by-a-value-of-the-dictionary
        srtd = sorted(usrtd, key=lambda k: SALst.info(k).gpa, reverse=True)

        return srtd

    ## @brief the average function
    #  @details computes the average following a criteria given through the
    #  function file
    #  @param f a function that filters the set
    #  @exception ValueError if fset is an empty set which would cause
    #  a division by zero
    #  @return a float value for the average
    @staticmethod
    def average(f):
        fset = {}
        accumulated_gpa = 0
        for macid in SALst.s:
            if f(SALst.info(macid)):
                fset[macid] = SALst.info(macid)
                accumulated_gpa = accumulated_gpa + SALst.info(macid).gpa

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

        return accumulated_gpa / len(fset)

    ## @brief the allocate function
    #  @details sorts freechoice students and other students then allocates
    #  freechoice students first and then allocates the other students
    #  @exception throws Runtimeerror if a student does not get allocated
    @staticmethod
    def allocate():
        AALst.init()
        freechoice_stdnts = SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0)
        other_stdnts = SALst.sort(lambda t: not(t.freechoice) and t.gpa >= 4.0)

        for macid in freechoice_stdnts:
            choices = SALst.info(macid).choices
            AALst.add_stdnt(choices.next(), macid)

        for macid in other_stdnts:
            choices = SALst.info(macid).choices
            allocated = False
            while(not(allocated) and not(choices.end())):
                dept = choices.next()
                if AALst.num_alloc(dept) < DCapALst.capacity(dept):
                    AALst.add_stdnt(dept, macid)
                    allocated = True
            if not(allocated):
                raise RuntimeError
```

J Code for Read.py

```
## @file Read.py
#  @author Michael Barreiros
#  @brief Read
#  @date 11/02/2019

from StdntAllocTypes import GenT, DeptT, SInfoT
from DCapALst import DCapALst
from SALst import SALst
from SeqADT import SeqADT

## @brief the read module


class Read:
    ## @brief the load stdnt data function
    #  @details reads from a file and saves the data in a SALst data type.
    #  This will go through each line and take it as though it were a student.
    #  Then it assigns each entry to its correct position and adds the student
    #  to the list of type SALst
    @staticmethod
    def load_stdnt_data(s):
        SALst.init()
        stdnt_data = open(s, "r")
        stdnt_data = stdnt_data.read()
        stdnt_data = stdnt_data.replace(",", "")
        stdnt_data = stdnt_data.replace("[", "")
        stdnt_data = stdnt_data.replace("]", "")
        stdnt_data = stdnt_data.splitlines()

        for student in stdnt_data:
            stdnt_info = student.split()
            macid = stdnt_info[0]
            f_name = stdnt_info[2]
            l_name = stdnt_info[3]
            if (stdnt_info[3] == "male"):
                gender = GenT.male
            else:
                gender = GenT.female
            gpa = float(stdnt_info[4])
            choices = []
            for i in range(5, len(stdnt_info) - 1):
                if stdnt_info[i] == "civil":
                    choices.append(DeptT.civil)
                if stdnt_info[i] == "chemical":
                    choices.append(DeptT.chemical)
                if stdnt_info[i] == "electrical":
                    choices.append(DeptT.electrical)
                if stdnt_info[i] == "mechanical":
                    choices.append(DeptT.mechanical)
                if stdnt_info[i] == "software":
                    choices.append(DeptT.software)
                if stdnt_info[i] == "materials":
                    choices.append(DeptT.materials)
                if stdnt_info[i] == "engphys":
                    choices.append(DeptT.engphys)
            if (stdnt_info[-1] == "True"):
                free_choice = True
            else:
                free_choice = False

            info = SInfoT(f_name, l_name, gender, gpa,
                          SeqADT(choices), free_choice)
            SALst.add(macid, info)

    ## @brief the load dcap data function
    #  @details this function reads from a file that is known to be the file
    #  for the department capacities. This function puts the appropriate
    #  capacity with its department and stores it as a DCapALst data type
    @staticmethod
    def load_dcap_data(s):
        DCapALst.init()
        dept_data = open(s, "r")
        dept_data = dept_data.read()
        dept_data = dept_data.replace(",", "")
        dept_data = dept_data.splitline()
```

```
for dept_and_cap in dept_data:
    dept_info = dept_and_cap.split()
    if dept_info[0] == "civil":
        dept = DeptT.civil
    if dept_info[0] == "chemical":
        dept = DeptT.chemical
    if dept_info[0] == "electrical":
        dept = DeptT.electrical
    if dept_info[0] == "mechanical":
        dept = DeptT.mechanical
    if dept_info[0] == "software":
        dept = DeptT.software
    if dept_info[0] == "materials":
        dept = DeptT.materials
    if dept_info[0] == "engphys":
        dept = DeptT.engphys

    cap = int(dept_info[1])
    DCapALst.add(dept, cap)
```

K Code for test_All.py

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


class TestAll:

    def setup_method(self, method):
        self.sinfo1 = SInfoT("first", "last", GenT.male, 12.0, SeqADT(
                             [DeptT.civil, DeptT.chemical]), True)
        self.sinfo2 = SInfoT("first1", "last", GenT.female, 7.0, SeqADT(
                             [DeptT.civil, DeptT.chemical]), True)
        self.sinfo3 = SInfoT("first2", "last", GenT.male, 12.0, SeqADT(
                             [DeptT.civil, DeptT.chemical]), False)
        self.sinfo4 = SInfoT("first3", "last", GenT.male, 12.0, SeqADT(
                             [DeptT.civil, DeptT.chemical]), False)

        self.choices = SeqADT([DeptT.civil, DeptT.mechanical, DeptT.engphys])
        self.SeqADT_None = SeqADT(None)
        self.SeqADT_list = SeqADT([1, 2, 3, 4])
        self.DCapALst = DCapALst()
        self.SALst = SALst()

    def teardown_method(self, method):
        self.sinfo1 = None
        self.sinfo2 = None
        self.sinfo3 = None
        self.sinfo4 = None
        self.choices = None
        self.SeqADT_None = None
        self.SeqADT_list = None
        self.DCapALst = self.DCapALst.s.clear()
        self.SALst = self.SALst.s.clear()

    def test_SeqADT_init1(self):
        assert not self.SeqADT_None.s

    def test_SeqADT_init2(self):
        assert self.SeqADT_None.i == 0

    def test_SeqADT_init3(self):
        assert self.SeqADT_list.s == [1, 2, 3, 4]

    def test_SeqADT_start1(self):
        self.choices.start()
        assert self.choices.i == 0

    def test_SeqADT_start2(self):
        self.choices.next()
        self.choices.next()
        self.choices.start()
        assert self.choices.i == 0

    def test_SeqADT_next1(self):
        assert self.choices.next() == DeptT.civil

    def test_SeqADT_next2(self):
        self.choices.next()
        assert self.choices.next() == DeptT.mechanical

    def test_SeqADT_next3(self):
        self.choices.next()
        self.choices.next()
        assert self.choices.next() == DeptT.engphys

    def test_SeqADT_next4(self):
        with pytest.raises(StopIteration):
            self.choices.next()
            self.choices.next()
            self.choices.next()
            self.choices.next()

    def test_SeqADT_end1(self):
```

```
    assert not(self.choices.end())

def test_SeqADT_end2(self):
    self.choices.next()
    self.choices.next()
    self.choices.next()
    assert self.choices.end()

def test_DCapALst_init1(self):
    assert self.DCapALst.s == {}

def test_DCapALst_add1(self):
    self.DCapALst.add(DeptT.civil, 10)
    self.DCapALst.add(DeptT.chemical, 20)
    assert self.DCapALst.s == {DeptT.civil: 10, DeptT.chemical: 20}

def test_DCapALst_add2(self):
    with pytest.raises(KeyError):
        self.DCapALst.add(DeptT.civil, 10)
        self.DCapALst.add(DeptT.civil, 10)

def test_DCapALst_remove1(self):
    self.DCapALst.add(DeptT.civil, 10)
    self.DCapALst.remove(DeptT.civil)
    assert self.DCapALst.s == {}

def test_DCapALst_remove2(self):
    with pytest.raises(KeyError):
        self.DCapALst.remove(DeptT.mechanical)

def test_DCapALst_elm(self):
    self.DCapALst.add(DeptT.civil, 10)
    assert self.DCapALst.elm(DeptT.civil)

def test_DCapALst_capacity1(self):
    self.DCapALst.add(DeptT.civil, 10)
    assert self.DCapALst.capacity(DeptT.civil) == 10

def test_DCapALst_capacity2(self):
    with pytest.raises(KeyError):
        self.DCapALst.capacity(DeptT.civil)

def test_SALst_init1(self):
    assert self.SALst.s == {}

def test_SALst_add1(self):
    self.SALst.add("student", ["test"])
    assert self.SALst.s == {"student": ["test"]}

def test_SALst_add2(self):
    with pytest.raises(KeyError):
        self.SALst.add("student", ["test"])
        self.SALst.add("student", ["test"])

def test_SALst_remove1(self):
    self.SALst.add("student", ["test"])
    self.SALst.remove("student")
    assert self.SALst.s == {}

def test_SALst_remove2(self):
    with pytest.raises(KeyError):
        self.SALst.remove("student")

def test_SALst_sort1(self):
    self.SALst.add("s1", self.sinfo1)
    self.SALst.add("s2", self.sinfo2)
    self.SALst.add("s3", self.sinfo3)
    self.SALst.add("s4", self.sinfo4)

    g = self.SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0)
    assert g == ["s1", "s2"]

def test_SALst_sort2(self):
    g = self.SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0)
    assert g == []

def test_SALst_average1(self):
    self.SALst.add("s1", self.sinfo1)
    self.SALst.add("s2", self.sinfo2)
    self.SALst.add("s3", self.sinfo3)
```

```
        self.SALst.add("s4", self.sinfo4)

        assert self.SALst.average(lambda x: x.gender == GenT.male) == 12.0

    def test_SALst_average2(self):
        with pytest.raises(ValueError):
            self.SALst.add("s1", self.sinfo1)
            self.SALst.add("s3", self.sinfo3)
            self.SALst.add("s4", self.sinfo4)
            self.SALst.average(lambda x: x.gender == GenT.female)

    def test_SALst_allocate1(self):
        self.DCapALst.add(DeptT.chemical, 2)
        self.DCapALst.add(DeptT.civil, 2)
        self.SALst.add("s1", self.sinfo1)
        self.SALst.add("s2", self.sinfo2)
        self.SALst.add("s3", self.sinfo3)
        self.SALst.add("s4", self.sinfo4)

        self.SALst.allocate()
        assert AALst.lst_alloc(DeptT.civil) == ["s1", "s2"]

    def test_SALst_allocate2(self):
        self.DCapALst.add(DeptT.chemical, 2)
        self.DCapALst.add(DeptT.civil, 2)

        self.SALst.allocate()
        assert AALst.lst_alloc(DeptT.civil) == []

    def test_SAL_allocate3(self):
        with pytest.raises(RuntimeError):
            self.DCapALst.add(DeptT.civil, 0)
            self.DCapALst.add(DeptT.chemical, 0)
            self.SALst.add("s3", self.sinfo3)
            self.SALst.allocate()

    def test_SALst_allocate4(self):
        self.DCapALst.add(DeptT.chemical, 1)
        self.DCapALst.add(DeptT.civil, 2)
        self.SALst.add("s1", self.sinfo1)
        self.SALst.add("s2", self.sinfo2)
        self.SALst.add("s3", self.sinfo3)

        self.SALst.allocate()
        assert AALst.lst_alloc(DeptT.chemical) == ["s3"]
```

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

```python
## @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
```