Assignment 2 Solution

Matthew Braden, bradenm, 400109876

February 26, 2019

Assignment 2 involved creating a variety of modules. Some of these modules include different tasks, such as reading from different text files containing different data types, allocating students to their program of choice, calculating averages for different genders, as well as many others.

1 Testing of the Original Program

When it came to choosing test cases for the assignment, the approach used involved testing every function at least once. When it came to certain functions that needed to be tested, a wider variety of test cases were needed to be used. For a lot of the functions, there was at least a normal test case, however for the majority of functions boundary and abnormal test cases were used in order to check the validity of the program.

Through analyzing results from testing, many bugs were discovered all over the code. However there was not enough time to fix every bug and as a result the code does not function as it should as some functions in the SALst module were not fully completed. Therefore failing a majority of the tests.

2 Results of Testing Partner's Code

When running the partners code, the python test cases went as expected as the abnormal test cases failed as they were meant to. When it came to the boundary and normal test cases the code passed as the output was successful. The partners code also was able to raise errors when expected. The partners code functioned similar to the original program that was tested, and is therefore a success.

3 Critique of Given Design Specification

The strong advantages of this design specification for assignment 2 was the ability to display all of the information that is necessary without making any assumptions on what the input files needed to look like. The information given about each module went into great depth of showing what functions that will be needed as well as the type in which the functions output when necessary. The one disadvantage of the specification is that the discrete math that was involved was a bit hard to read for a couple of the functions.

4 Answers

1. In A1 the way the assignment explained how to sort the students in order of gpa was much more clear and easy to understand than the way A2 did with the discrete math explanation. However with most of the other functions the discrete math was easy to understand, therefore making the process in A2 to be clear with less assumptions being made. A1 has a major disadvantage when reading the students information file as it was not specified what the file would actually look like, where as in A2 the students information was given and therefore should allow for the partners code to work when being integrated with another students code.

2. The specification should be modified to make a KeyError if the students gpa is out of the range of 0 to 12. You will not need to make a new ADT as you will just have to throw this exception in at the start of the functions in which it is necessary for the students gpa to be in between 0 and 12.

3. The documentation can take advantage of these similarities by copying what DCapALst does to SALst since the functions are almost exactly the same except for a few minor differences.

4. A2 is much more general than A1 as the layout of both the students and the departments data is the same for everyone. Unlike A1 where we would have free range to make assumptions of what the layout for the text files would like, A2 delivers with a specific format that cannot be altered.

5. The use of SeqADT is much more valuable over a regular list. This is due to the fact that the SeqADT is much more of a stable ADT to use when using a set of data. When data sets become large a simple list is harder to use as it becomes more difficult to keep knowledge of where each item is in the list compared to the use of a sequence in SeqADT.

6. The use of enums allows the items in the enumerated lists to be called in a much more simple manner. This is because each item in the enumerated list is assigned a value. So when the item is called in a different module a value will appear. These were not introduced in the specification for macids as the students macid cannot be assigned a certain value when using enums as it needs to be inputed from a text file.

E Code for StdntAllocTypes.py

```
## @file StdntAllocTypes.py
#  @brief Creates classes for different types
#  @author Matthew Braden
#  @date 2/11/2019

from SeqADT import *

## @brief Enumerated class of different genders
#  @details Class that contains genders related to a value
class GenT(enum):
    male = 0
    female = 1

## @brief Enumerated class of different departments
#  @details Class that contains departments related to a value
class DeptT(enum):
    civil = 0
    chemical = 1
    electrical = 2
    mechanical = 3
    software = 4
    materials = 5
    engphys = 6

## @brief NamedTuple class of different categories
#  @details Class that contains categories for student info with the data type
class SInfoT(NamedTuple):
    fname: str
    lname: str
    gender: GenT
    gpa: float
    choices: SeqADT(DeptT)
    freechoice: bool
```

F Code for SeqADT.py

```
## @file SeqADT.py
#  @brief Creates the Sequence data type
#  @author Matthew Braden
#  @date 2/11/2019

## @brief Class for the sequence of students
#  @details Class that contains start, next and end functions
class SeqADT:

    s = 0
    i = 0

    ## @brief Initializes data
    #  @details Initializes the sequence of students
    #  @param x Value of amount of students in sequence
    @staticmethod
    def __init__(x):
        SeqADT.s = x
        SeqADT.i = 0
        return SeqADT.s

    ## @brief Start of the sequence
    #  @details Starts the students sequence
    @staticmethod
    def start():
        SeqADT.i = 0

    ## @brief Next in sequence
    #  @details Returns a value of the next item in the sequence
    @staticmethod
    def next():
        if SeqADT.i  >= len(SeqADT.s):
            raise StopIteration
        SeqADT.i  += 1
        return SeqADT.s[SeqADT.i]

    ## @brief Ends the sequence
    #  @details Finishes the students sequence and returns a bool
    @staticmethod
    def end():
        return SeqADT.i >= len(SeqADT.s)
```

G Code for DCapALst.py

```
## @file DCapALst.py
#  @brief Creates class for departments
#  @author Matthew Braden
#  @date 2/11/2019

from StdntAllocTypes import *

## @brief Class for departments list
#  @details Class that contains all information about the departments
class DCapALst:

    s = []
    ## @brief Initializes data
    #  @details Initializes the list of departments
    @staticmethod
    def __init__():
        DCapALst.s = []

    ## @brief Adds data
    #  @details Adds a department to the DCapALst
    #  @param d The DeptT from StdntAllocTypes
    #  @param n The capacity of the department
    @staticmethod
    def add(d, n):
        if (d, n) in DCapALst.s:
            raise KeyError
        DCapALst.s.append((d, n))

    ## @brief Removes data
    #  @details Removes a department from the DCapALst
    #  @param d The DeptT from StdntAllocTypes
    @staticmethod
    def remove(d):
        if (d, n) not in DCapALst.s:
            raise KeyError
        if (d, n) in DCapALst.s:
            DCapALst.s.remove((d, n))

    ## @brief Returns a bool
    #  @details Returns weather or not the department is in DCapALst
    #  @param d The DeptT from StdntAllocTypes
    @staticmethod
    def elm(d):
        return ((d, n) in DCapALst.s)

    ## @brief Returns the capacity
    #  @details Checks if department is in the DCapALst and returns capacity
    #  @param d The DeptT from StdntAllocTypes
    @staticmethod
    def capacity(d):
        if (d, n) not in DCapALst.s:
            raise KeyError
        if (d, n) in DCapALst.s:
            return n
```

H Code for AALst.py

```
## @file AALst.py
#  @brief Creates class for allocation
#  @author Matthew Braden
#  @date 2/11/2019

from StdntAllocTypes import *

## @brief Class for allocation list
#  @details Class that contains allocation information
class AALst:

    s = []

    ## @brief Initializes data
    #  @details Initializes the list of departments
    @staticmethod
    def __init__():
        for d in DepT:
            AALst.s = (d, [])

    ## @brief Adds a student
    #  @details Adds a student to the AALst
    #  @param dep The DeptT from StdntAllocTypes
    #  @param m String of students
    @staticmethod
    def add_stdnt(d, m):
        for (d, L) in AALst.s:
            AALst.s.append((d, m))

    ## @brief Returns a string
    #  @details Returns a sequence of students as a string
    #  @param d The DeptT from StdntAllocTypes
    @staticmethod
    def lst_alloc(d):
        if (d, L) in AALst.s:
            return L

    ## @brief Returns a value
    #  @details Returns the number of students in a department
    #  @param d The DeptT from StdntAllocTypes
    @staticmethod
    def num_alloc(d):
        if (d, L) in AALst.s:
            return len(L)
```

I Code for SALst.py

```
## @file SALst.py
#  @brief This edits a student allocated list
#  @author Matthew Braden
#  @date 2/11/2019

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

## @brief Class for student allocation list
#  @details Class that contains student allocation functions
class SALst:

    s = []
    ## @brief Initializes data
    #  @details Initializes an empty list
    @staticmethod
    def __init__():
        SALst.s = []

    ## @brief Adds student data
    #  @details Adds a new student to the list
    #  @param m Student string
    #  @param i SInfoT
    @staticmethod
    def add(m, i):
        if (m, i) in SALst.s:
            raise KeyError
        SALst.s.append((m, i))

    ## @brief Removes student data
    #  @details Removes a student from SALst
    #  @param m String of students
    @staticmethod
    def remove(m):
        if (m, i) not in SALst.s:
            raise KeyError
        if (m, i) in SALst.s:
            SALst.s.remove((m, i))

    ## @brief Returns a bool
    #  @details Returns if a student is in SALst
    #  @param m String of students
    @staticmethod
    def elm(m):
        return ((m, i) in SALst.s)

    ## @brief Info student data
    #  @details Returns the information of students data
    #  @param m Students string
    @staticmethod
    def info(m):
        if (m, i) not in SALst.s:
            raise ValueError
        if (m, i) in SALst.s:
            return i

    ## @brief Sorts student data
    #  @details Adds a new student to the list
    #  @param f SInfoT
    @staticmethod
    def sort(f):
        sortLst = sorted(f, key = lambda s: SALst.s.get_gpa(m, s) , reverse = True)
        return sortLst
    #sorted line 63 was found from https://tinyurl.com/y9xdx4ep

    ## @brief Average the gpa
    #  @details Averages the gpa of genders
    #  @param f SInfoT
    @staticmethod
    def average(f):
        for ((m, i) in s) and f(i):
            if not i:
            raise ValueError
        sum = 0
        for ((m, i) in s) and f(i):
```

```
            fset = i
            for j in fset:
                sum += j.gpa
        ave = sum / len(fset)
        return ave

    ## @brief Allocates students
    #  @details Puts the students in one of their three choices
    @staticmethod
    def allocate():
        free = SALst.sort(f)
        if t.freechoice and (t.gpa >= 4.0):
            for m in free:
                ch = SALst.info(m).choices
                AALst.add_stdnt(ch.next(), m)

        student = SALst.sort(f)
        if (not t.freechoice) and (t.gpa >= 4.0):
            for m in student:
                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

    ## @brief Gets gpa
    #  @details Returns the students gpa
    #  @param m String of students
    #  @param s List of data
    @staticmethod
    def get_gpa(m, s):
        if (m, i) in s:
            return i.gpa
```

J Code for Read.py

```
## @file Read.py
#  @brief This edits a student allocated list
#  @author Matthew Braden
#  @date 2/11/2019

from DCapALst import *
from SALst import *

## @brief Loads students data
#  @details Reads students data from a file and puts it into a list
#  @param s Filename of file being used
def load_stdnt_data(s):
    file = open(s, "r")
    stdnt = file.read()
    stdnt = stdnt.replace(",", "")
    stdnt = stdnt.replace("[", "")
    stdnt = stdnt.replace("]", "")
    stdnt = stdnt.splitlines()
    file.close()
    SALst.__init__()
    stdnt_data = SALst.s
    for i in stdnt:
        stdnti = i.split()
        stdnt_lst = []
        for j in stdnti:
            stdnt_lst.append(j)
        stdnt_data.append(stdnt_lst)

## @brief Loads departments data
#  @details Reads departments data from a file and puts it into a list
#  @param s Filename of file being used
def load_dept_data(s):
    file = open(s, "r")
    dept = file.read()
    dept = dept.replace(",", "")
    dept = dept.splitlines()
    file.close()
    DCapALst.__init__()
    dept_data = DCapALst.s
    for i in dept:
        depti = i.split()
        dept_lst = []
        for j in depti:
            dept_lst.append(j)
        dept_data.append(dept_lst)
```

K Code for Partner's SeqADT.py

```
## @file SeqADT.py
#  @author Meijing Li
#  @brief Provides an abstract data type to represent students' department choices
#  @date 11/02/2019


## @brief An abstract data type that represents a sequence of DeptT
## @details This ADT defines a sequence of student's choices
class SeqADT:

    ## @brief SeqADT constructor
    #  @details takes a list of DeptT which represents the student's choices.
    #  @param x list of DeptT
    def __init__(self, x):
        self.__s = x  # x is a list of T
        self.__i = 0

    ## @brief start moves back to the student's first choice
    #  @details reinitializes the index indicator to the beginning of the list
    def start(self):
        self.__i = 0

    ## @brief next returns the current choice and then moves to the next choice
    #  @return the current choice before moving on to the next
    #  @throws StopIteration Throw out exception if there's no more choice
    def next(self):
        if (self.__i >= len(self.__s)):
            raise StopIteration
        self.__i += 1
        return (self.__s)[self.__i - 1]

    def end(self):
        return self.__i >= len(self.__s)
```

L Code for Partner's DCapALst.py

```
## @file DCapALst.py
#  @author Meijing Li
#  @brief Provides an abstract object to store department capacities
#  @date 11/02/2019

# hide this exported module to pass flake8
# from StdntAllocTypes import *


# each field is represented as {DeptT: N}
## @brief An abstract object that represents a dictionary of department capacities
#  @details the keys of dictionary are DeptT and the values are corresponding int capacities
class DCapALst:
    s = {}  # s: a dictionary

    ## @brief DCapALst constructor
    #  @details inilializes the state variable as an empty dictionary
    @staticmethod
    def init():
        DCapALst.s = {}

    ## @brief add inserts new department with its capacity into the dictionary
    #  @param d department to be inserted
    #  @param n capacity number
    #  @throws KeyError department to be insearted is already in dictionary
    @staticmethod
    def add(d, n):
        if d in DCapALst.s:
            raise KeyError
        else:
            DCapALst.s[d] = n

    ## @brief remove deletes the department from the dictionary
    #  @param d department to be deleted
    #  @throws KeyError department is not in dictionary
    @staticmethod
    def remove(d):
        if d not in DCapALst.s:
            raise KeyError
        else:
            del DCapALst.s[d]

    ## @brief elm checks whether the department is in dictionary or not
    #  @param d department to be checked
    @staticmethod
    def elm(d):
        return (d in DCapALst.s)

    ## @brief capacity returns the capacity of the depertment
    #  @param d department
    #  @return number of capacity of the input department
    #  @throws KeyError department is not in dictionary
    @staticmethod
    def capacity(d):
        if d not in DCapALst.s:
            raise KeyError
        else:
            return DCapALst.s[d]
```

M Code for Partner's SALst.py

```
## @file SALst.py
#  @author Meijing Li
#  @brief Provides an abstract object to store student records
#  @date 11/02/2019

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


## @brief An abstract object that represents a list of student records
#  @details each student record is a tupe of macid and information
class SALst:
    s = []  # a list of StudenT, where StudenT = (macid: string, info: SInfoT)

    ## @brief SALst constructor
    #  @details inilializes the state variable as an empty list
    @staticmethod
    def init():
        SALst.s = []

    ## @brief add inserts new student tuple to the list
    #  @param m macid of student to be inserted
    #  @param i student's information of type SInfoT
    #  @throws KeyError student to be inserted is already in list
    @staticmethod
    def add(m, i):
        for t in SALst.s:
            if m == t[0]:
                raise KeyError
        student_t = (m, i)
        SALst.s.append(student_t)

    ## @brief remove deletes the student tuple from the list
    #  @param m macid of student to be deleted
    #  @throws KeyError student is not in list
    @staticmethod
    def remove(m):
        detector = False  # detect if <m, i> in s
        for t in SALst.s:
            if m == t[0]:
                detector = True
                SALst.s.remove(t)
        if (detector is False):
            raise KeyError

    ## @brief elm checks whether the student tuple is in list or not
    #  @param m macid of student to be checked
    @staticmethod
    def elm(m):
        for t in SALst.s:
            if m == t[0]:
                return True
        return False

    ## @brief info returns the student information
    #  @param m macid of student
    #  @return student information of type SInfoT
    #  @throws KeyError student is not in list
    @staticmethod
    def info(m):
        for t in SALst.s:
            if m == t[0]:
                return t[1]
        raise KeyError

    ## @brief sort returns a list of student macids in descending order
    #         based on their GPAs
    #  @param f a selector function. Only students satisfying f would be included
    #  @return a list of macids of specified students where
    #          these students are sorted in descending order
    @staticmethod
    def sort(f):
        # sort the whole list of tuples in descending order based on the gpa
        sorted_s = sorted(SALst.s, key=lambda tup: tup[1].gpa, reverse=True)
        lst = []
```

```
        for t in sorted_s:
            if (f(t[1])):  # only put students who satisfy f into the output list
                lst.append(t[0])
        return lst

    ## @brief average calculates the student average GPA
    #  @param f a selector function. Only students satisfying f would be included
    #  @return average GPA of the spesified students
    #  @throws ValueError no students in the list, or no students satifying f
    @staticmethod
    def average(f):
        total = 0.0
        num = 0
        for t in SALst.s:
            if (f(t[1])):
                total += t[1].gpa
                num += 1
        if (num == 0):  # no such student exists
            raise ValueError
        else:
            return total / num

    ## @brief allocate distributes students to the departments based on their choices
    #  @details freechoice students have priority to be allocated first
    #  @throws RuntimeError there exists student remianing unallocated
    @staticmethod
    def allocate():
        AALst.init()
        # allocate freechoice students first:
        free_lst = SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0)
        for m in free_lst:
            ch = SALst.info(m).choices  # ch is SeqADT(DeptT)
            AALst.add_stdnt(ch.next(), m)

        # allocate other students:
        nonfree_lst = SALst.sort(lambda t: not t.freechoice and t.gpa >= 4.0)
        for m in nonfree_lst:
            ch = SALst.info(m).choices
            alloc = False
            while (not alloc and not ch.end()):
                d = ch.next()
                # the department is not full
                if(AALst.num_alloc(d) < DCapALst.capacity(d)):
                    AALst.add_stdnt(d, m)
                    alloc = True
            if (not alloc):
                raise RuntimeError
```

N Code for testAll.py

```
## @file test_All.py
#  @brief This tests the programs functions
#  @author Matthew Braden
#  @date 2/11/2019
import pytest

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

## @brief Class testing function
#  @details Test for the module SeqADT
class TestSeqADT:

    def test_next_sequence_function(self):
        seq = SeqADT([5, 6, 7, 8])
        assert seq.next() == 6

    def test_end_sequence(self):
        seq = SeqADT([10, 5, 6, 31])
        assert seq.end() == True

    def test_fail_for_start(self):
        seq = SeqADT([])
        assert seq.start() == None

    def test_next_sequence_fail(self):
        seq = SeqADT([])
        assert seq.next() == None

    def test_boundary_end(self):
        seq = SeqADT([])
        assert seq.end() == True

## @brief Class testing function
#  @details Test for the module DCapALst
class TestDCapALst:

    def test_raises_error_for_add(self):
        with pytest.raises(KeyError):
            DCapALst([[software, 100]]).add(DeptT.software, 100)

    def test_raises_error_for_remove(self):
        with pytest.raises(KeyError):
            DCapALst().remove(software, 100)

    def test_elm_function_boundary(self):
        dept = DCapALst()
        assert dept.elm() == True

    def test_elm_function(self):
        dept = DCapALst().add(DeptT.software, 100)
        assert dept.elm(software) == True

    def test_elm_to_fail(self):
        dept = DCapALst()
        assert dept.elm(software) == False

    def test_raises_error_for_capacity(self):
        while pytest.raises(KeyError):
            DCapALst().capacity(Software)

    def test_capacity_function(self):
        dept = DCapALst()
        assert dept.capacity(Software) == 100


## @brief Class testing function
#  @details Test for the module SALst
class TestSALst:

    def test_raises_error_for_add(self):
        with pytest.raises(KeyError):
```

```
        SALst([["student1", (brownc, Charlie, Brown, male, 3.9, [engphys, software, chemical,
            materials], True)]]).add("student1", (brownc, Charlie, Brown, male, 3.9, [engphys,
            software, chemical, materials], True))

def test_raises_error_for_remove(self):
    with pytest.raises(KeyError):
        SALst().remove("student1")

def test_for_elm(self):
    SAL = SALst().add("student1", (brownc, Charlie, Brown, male, 3.9, [engphys, software,
        chemical, materials], True))
    assert SAL.elm("student1") == True

def test_for_elm_failures(self):
    SAL = SALst()
    assert SAL.elm("student1") == False

def test_elm_boundary(self):
    SAL = SALst()
    assert dept.elm() == True

def test_for_info_error(self):
    with pytest.raises(ValueError):
        SALst().info("student1")

def test_for_get_gpa(self):
    gpa = SALst()
    assert gpa.get_gpa() == 0

def test_for_sort(self):
    sort = SALst().sort((brownc, Charlie, Brown, male, 3.9, [engphys, software, chemical,
        materials], True), (smithj2, John, Smith, male, 7.0, [mechanical, electrical, materials,
        mechanical, electrical], False))
    assert sort == (smithj2, John, Smith, male, 7.0, [mechanical, electrical, materials,
        mechanical, electrical], False), (brownc, Charlie, Brown, male, 3.9, [engphys, software,
        chemical, materials], True)

def test_for_allocate_boundary(self):
    alloc = SALst().allocate((brownc, Charlie, Brown, male, 3.9, [engphys, software, chemical,
        materials], True))
    assert not (alloc == [engphys: brownc])

def test_for_average(self):
    ave = SALst()
    assert ave.average((brownc, Charlie, Brown, male, 6.0, [engphys, software, chemical,
        materials], True), (smithj2, John, Smith, male, 7.0, [mechanical, electrical, materials,
        mechanical, electrical], False)) == 6.5
```