Assignment 2 Solution

?

February 25, 2019

Introductory blurb.

1 Testing of the Original Program

Description of approach to testing. Rationale for test case selection. Summary of results. Any problems uncovered through testing.

2 Results of Testing Partner's Code

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

3 Critique of Given Design Specification

Advantages and disadvantages of the given design specification.

4 Answers

1.

E Code for StdntAllocTypes.py

```
## @file StdntAllocTypes.py
#  @title StdntAllocTypes
#  @author Joshua Guinness, guinnesj, 400134735
#  @date Febuary 11, 2019

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


## @brief Enumeration class for gender
class GenT(Enum):
    male = 1
    female = 2

## @brief Enumeration class for engineering departments
class DeptT(Enum):
    civil = 1
    chemical = 2
    electrical = 3
    mechanical = 4
    software = 5
    materials = 6
    engphys = 7

## @brief NamedTuple for info about each student
class SInfoT(NamedTuple):
    fname: str
    lname: str
    gender: GenT
    gpa: float
    choices: SeqADT
    freechoice: bool
```

F Code for SeqADT.py

```
## @file SeqADT.py
#  @title SeqADT
#  @author Joshua Guinness, guinnesj, 400134735
#  @date Febuary 11, 2019

## @brief A class which defines an abstract data type which is a sequence
class SeqADT:

    ## @brief Initializ the state variables
    #  @param p2 A sequence of T.
    def __init__ (self, x):
        self.s = x
        self.i = 0

    ## @brief Sets the integer variable to zero.
    def start(self):
        self.i = 0

    ## @brief Outputs the current element of the sequence then moving to the next element
    #  @return The current element of the sequence
    def next(self):
        if (self.i >= len(self.s)):
            raise StopIteration

        return self.s[self.i]
        self.i = self.i + 1

    ## @brief Checks to see if the end of the sequence is reached
    #  @return Boolean value about whether have reached end of sequence
    def end(self):
        if (self.i >= len(self.s)):
            return True
        else:
            return False
```

G Code for DCapALst.py

```
## @file DCapALst.py
#  @title DCapALst
#  @author Joshua Guinness, guinnesj, 400134735
#  @date Febuary 11, 2019

from StdntAllocTypes import *


## @brief Departments and their capacities and functions to preform operations on them
class DCapALst:

    ## @brief Makes the list empty
    @staticmethod
    def init():
        DCapALst.s = []

    ## @brief Adds a department and its capacity to the list
    #  @param p1 Department of type DeptT
    #  @param p2 Capacity of type integer
    @staticmethod
    def add(d, n):

        if (len(DCapALst.s) == 0):
            DCapALst.s.append((d, n))
        else:
            is_inside = False
            for i in DCapALst.s:
                if (i[0] == d):
                    is_inside = True

            if (is_inside == True):
                raise KeyError
            else:
                DCapALst.s.append((d, n))

    ## @brief Removes a department from the set
    #  @param p1 Department of type DeptT
    @staticmethod
    def remove(d):
        is_inside = False
        for i in DCapALst.s:
            if (i[0] == d):
                is_inside = True
                DCapALst.s.remove(i)

        if (is_inside == False):
            raise KeyError

    ## @brief Checks to see if a department already exists in the set
    #  @param p1 Department of type DeptT
    #  @return Boolean value about whether the department already exists
    @staticmethod
    def elm(d):
        is_inside = False
        for i in DCapALst.s:
            if (i[0] == d):
                is_inside = True

        if (is_inside == True):
            return True
        else:
            return False

    ## @brief Checks the current capacity of a department
    #  @param p1 Department of type DeptT
    #  @return Capacity of the passed department
    @staticmethod
    def capacity(d):
        is_inside = False
        for i in DCapALst.s:
            if (i[0] == d):
                is_inside = True
                return int(i[1])

        if (is_inside == False):
            raise KeyError
```

H Code for AALst.py

```
## @file AALst.py
#  @title AALst
#  @author Joshua Guinness, guinnesj, 400134735
#  @date Febuary 11, 2019

from StdntAllocTypes import *


## @brief Departments and the students allocated to them
class AALst:

    ## @brief Makes the list empty
    @staticmethod
    def init():
        AALst.s = []

        for dept in (DeptT):
            AALst.s.append((dept, []))

    ## @brief Adds a student to a department
    #  @param p1 Department of type DeptT
    #  @param p2 MacId of student
    @staticmethod
    def add_stdnt(dep, m):
        for i in AALst.s:
            if (i[0] == dep):
                i[1].append(m)

    ## @brief Outputs a list of students allocated to the specified department
    #  @param p1 Department of type DeptT
    #  @return list of allocated macids to the specified department
    @staticmethod
    def lst_alloc(d):
        for i in AALst.s:
            if (i[0] == d):
                return i[1]

    ## @brief Checks the number of students allocated to a specified department
    #  @param p1 Department of type DeptT
    #  @return Number of students allocated to a specified department
    @staticmethod
    def num_alloc(d):
        for i in AALst.s:
            if (i[0] == d):
                return len(i[1])
```

I Code for SALst.py

```
## @file SALst.py
#  @title SALst
#  @author Joshua Guinness, guinnesj, 400134735
#  @date Febuary 11, 2019

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


## @brief Students and operations to preform on them
class SALst:

    ## @brief Makes the list empty
    @staticmethod
    def init():
        SALst.s = []

    ## @brief Adds a student to the list
    #  @param p1 macid of student
    #  @param p2 student info
    @staticmethod
    def add(m, i):
        is_inside = False
        for j in SALst.s:
            if (j[0] == m):
                is_inside = True

        if (is_inside == True):
            raise KeyError
        else:
            SALst.s.append((m, i))

    ## @brief Removes a student from the list
    #  @param p1 macid of a student
    @staticmethod
    def remove(m):
        is_inside = False
        for i in SALst.s:
            if (i[0] == m):
                is_inside = True
                SALst.s.remove(i)

        if (is_inside == False):
            raise KeyError

    ## @brief Checks to see if a student exists in the list
    #  @param p1 macid of a student
    #  @return Boolean about whether the student exists
    @staticmethod
    def elm(m):
        is_inside = False
        for i in SALst.s:
            if (i[0] == m):
                is_inside = True

        if (is_inside == True):
            return True
        else:
            return False

    ## @brief Gets the info about a particular student
    #  @param p1 macid of a student
    #  @return Info about the specified student
    @staticmethod
    def info(m):
        is_inside = False
        for i in SALst.s:
            if(i[0] == m):
                is_inside = True
                return i[1]

        if (is_inside == False):
            raise KeyError

    ## @brief Sorts the student in decreasing order of GPA
```

```
#  @param p1 lamda function
#  @return list of macids of sorted students
@staticmethod
def sort(f):
    l = []
    temp = SALst.s.copy()
    to_delete = []
    counter = 0
    for i in temp:
        if (f(i[1]) == False):
            to_delete.append(counter)
        counter = counter + 1

    to_delete.reverse()
    for i in to_delete:
        temp.remove(temp[i])

    while (len(temp) > 0):
        highest = -1
        element_number = 0
        for i in range(len(temp)):
            if (temp[i][1].gpa > highest):
                temp[i][1].gpa > highest
                element_number = i
                i = i+1

        l.append(temp[element_number][0])
        temp.remove(temp[element_number])

    return l

## @brief Checks to see if the end of the sequence is reached
#  @param p1 The instance of the class
#  @return Boolean value about whether have reached end of sequence
@staticmethod
def average(f):

    temp = SALst.s.copy()

    to_delete = []
    counter = 0
    for i in temp:
        if (f(i[1]) == False):
            to_delete.append(counter)
        counter = counter + 1

    to_delete.reverse()
    for i in to_delete:
        temp.remove(temp[i])

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

    total = 0
    number = len(temp)

    for i in temp:
        total = total + i[1].gpa

    return total/number

## @brief Checks to see if the end of the sequence is reached
#  @param p1 The instance of the class
#  @return Boolean value about whether have reached end of sequence
@staticmethod
def allocate():

    AALst.init()

    F = SALst.sort(lambda t: t.freechoice and t.gpa >= 4.0)
    for m in F:
        ch = SALst.info(m).choices
        AALst.add_stdnt(ch.next(), m)

    S = SALst.sort(lambda t: not t.freechoice and t.gpa >= 4.0)
    for m in S:
        ch = SALst.info(m).choices
        alloc = False
        while (not alloc and not ch.end()):
            d = ch.next()
            if (AALst.num_alloc(d) < DCapALst.capacity(d)):
                AALst.add_stdnt(d, m)
                alloc = True
        if (not alloc):
            raise RuntimeError
```

J Code for Read.py

```
## @file Read.py
#  @title Read
#  @author Joshua Guinness, guinnesj, 400134735
#  @date Febuary 11, 2019

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


## @brief Loads in the student data and updates the state of the SALst module
#  @param A filename of student data
def load_stdnt_data(s):

    SALst.init()

    f = open(s, 'r')

    for line in f:
        temp = line.split(', ')
        student_info = []
        student_info.append(temp[1])
        student_info.append(temp[2])
        student_info.append(GenT[temp[3]])
        student_info.append(float(temp[4]))

        list_dept = []

        for i in range(5, len(temp) - 1):
            temp2 = temp[i].replace('[', "")
            temp3 = temp2.replace(']', "")
            temp4 = DeptT[temp3]

            list_dept.append(temp4)

        student_info.append(SeqADT(list_dept))
        string = temp[-1]
        string2 = string.replace('\n', "")
        if (string2 == "True"):
            student_info.append(True)
        else:
            student_info.append(False)

        final_info = SInfoT(student_info[0], student_info[1], student_info[2],
        student_info[3], student_info[4], student_info[5])
        SALst.add(temp[0], final_info)

    f.close()


## @brief Loads in the department data and updates the state of the DCapALst module
#  @param p1 A filename of department data
def load_dcap_data(s):

    DCapALst.init()

    f = open(s, 'r')
    for line in f:
        string = line.rstrip('\n')
        temp = string.split(', ')
        DCapALst.add(DeptT[temp[0]], temp[1])

    f.close()
```

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


class TestingClass:

    def setup_method(self, method):
        load_dcap_data("DeptCap.txt")
        load_stdnt_data("StdntData.txt")

    ## Testing the DCapALst modle

    # Checking to see whether departments correctly exist after reading in the data
    def test_CivilExists(self):
        assert DCapALst.elm(DeptT.civil)

    def test_ChemExists(self):
        assert DCapALst.elm(DeptT.chemical)

    def test_ElecExists(self):
        assert DCapALst.elm(DeptT.electrical)

    def test_MechExists(self):
        assert DCapALst.elm(DeptT.mechanical)

    def test_SoftExists(self):
        assert DCapALst.elm(DeptT.software)

    def test_MatExists(self):
        assert DCapALst.elm(DeptT.materials)

    def test_Phys(self):
        assert DCapALst.elm(DeptT.engphys)

    # Checking to see if getting the current capacity of the department works

    def test_GetCapacityCivil(self):
        assert DCapALst.capacity(DeptT.civil) == 100

    def test_GetCapacityMech(self):
        assert DCapALst.capacity(DeptT.mechanical) == 100

    def test_GetCapacityPhys(self):
        assert DCapALst.capacity(DeptT.engphys) == 100

    # Checking to see if removing a department works

    def test_RemoveDept(self):
        DCapALst.remove(DeptT.software)
        assert not DCapALst.elm(DeptT.software)

    # Checking to see if adding a department works

    def test_AddDept(self):
        DCapALst.remove(DeptT.software)
        DCapALst.add(DeptT.software, 100)
        assert DCapALst.elm(DeptT.software)
```

K Code for Partner's 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)
```

L Code for Partner's DCapALst.py

```python
## @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]
```

M Code for Partner's SALst.py

```python
## @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
```