Assignment 2 Solution

Madhi Nagarajan, nagarajm

February 15, 2020

This purpose of this assignment is based on creating a program for that handles chemical elements and has the ability to mathematically form molecules (MoleculeT), compounds (CompoundT), and balance chemical reactions (ReactionT). Testing and formatting of these classes were also done, utilizing pytest and flake8, to verify our personal code and our partner's code.

1 Testing of the Original Program

2 Results of Testing Partner's Code

3 Critique of Given Design Specification

4 Answers

a) With the natural language of A1, it was much easier to understand what the purpose was for each function or each file, thus it made it easier for me to implement. That being said, it does not give an exact specification on what parameters to take in or what the resulting output should exactly be. This means that the program depends on the developer's interpretation of the natural language specification. Thus, the end result may not maintain correctness it was expected to have.
This is where a formal specification has its advantages. It clearly describes the parameters and output of the program. However one such feature it's lacking is that it is not very descriptive. I found that this assignment was somewhat difficult to understand the purpose of certain functions/code or how it can utilized by other classes. Furthermore, the specification can be somewhat restrictive of what the developer can do since it specifies how exactly the function/class should be designed.

b)

c) I would change the current ChemTypes class to a store a list for each enum, containing both it's atomic number and atomic mass (eg. H = [1.01, 1]). We can then have a constructor (get_mass()) that returns an element's atomic mass. For MoleculeT, a mass constructor can be used again, but this time it multiplies the number of atoms with the atomic mass. For CompoundT, it would take the sum of all MoleculeT masses in its compound.

d)

e) Static typing is when the variable's type is kow at compile time, where as dynamic typing is when we don't have to define the type before compiling; the compiler "trusts the programmer". Static typing's main advantage is that TypeError bugs are caught early on by the compiler. It also provides a sense of readability/safety especially with large amounts of code and classes because the programmer can easily tell what type of parameters/returns are being handled by a method. However, static typing can also be a burden for programmers, as it becomes difficult to type fast, clean code since you have to define types all the time.

f)
```
pairs = [(x, y) for x in range(10) for y in range(10)
                x % 2 == 1 and y % 2 == 1 and x < y]
```

g)
```
def lengthlist(a):
    b = map(lambda x: 1, a)
    return sum(list(b))
```

h) The interface of a module enables the module's clients to utilize the service offered by the module.The implementation of an interface provides the services offered by the module. Often the inteface acts like a blueprint of the module, which one can then implement on top of for the module to work and provide functionality. However, the interface cannot do anything by itself; it needs to be implemented.

i)
  i) Abstraction Interfaces enable abstraction because interfaces produce an abstract model or blueprint on the methods that implementations can utilize. Interfaces should be created with abstraction in mind.
  ii) Anticipation of change Interfaces revolve around the idea of anticipation of change due to the fact that implementations may be changed during the development process or that newer implementations need to be made based on the same abstract model.

iii) Generality Generality is an approch to tackle more general problems first, then moving towards the complex ones. Interfaces should be created with generality in mind, so that it is flexible if several implementations are needed.

iv) Modularity Modularity is when we divide a complex system into modules. Interfaces can take advantage of this idea, as interfaces can be utilized for several modules/implementations, helping to keep modules simple and seperated.

v) Separation of concerns Seperation of concerns is similar to modularity, as its a principle that different concerns should be isolated. Seperation of concerns should be utilized with interfaces because while certain modules may have similar functionalities, it's important to keep them seperated. Interfaces help to do this, as modules can inherit from the same interfaces, but also extend on their own implementation.

E Code for ChemTypes.py

```
## @file ChemTypes.py
#  @title ChemTypes
#  @author Madhi Nagarajan
#  @brief This file acts as an enum class for all periodic elements.
#  @date February 12, 2020
from enum import Enum, auto


## @brief The class, ChemTypes, represents an enum class for all periodic elements.
#  @details The class, CompoundT, represents an enum class for all periodic elements
# for other classes to utilize.
class ElementT(Enum):
    H = auto()
    He = auto()
    Li = auto()
    Be = auto()
    B = auto()
    C = auto()
    N = auto()
    O = auto()
    F = auto()
    Ne = auto()
    Na = auto()
    Mg = auto()
    Al = auto()
    Si = auto()
    P = auto()
    S = auto()
    Cl = auto()
    Ar = auto()
    K = auto()
    Ca = auto()
    Sc = auto()
    Ti = auto()
    V = auto()
    Cr = auto()
    Mn = auto()
    Fe = auto()
    Co = auto()
    Ni = auto()
    Cu = auto()
    Zn = auto()
    Ga = auto()
    Ge = auto()
    As = auto()
    Se = auto()
    Br = auto()
    Kr = auto()
    Rb = auto()
    Sr = auto()
    Y = auto()
    Zr = auto()
    Nb = auto()
    Mo = auto()
    Tc = auto()
    Ru = auto()
    Rh = auto()
    Pd = auto()
    Ag = auto()
    Cd = auto()
    In = auto()
    Sn = auto()
    Sb = auto()
    Te = auto()
    I = auto()
    Xe = auto()
    Cs = auto()
    Ba = auto()
    La = auto()
    Ce = auto()
    Pr = auto()
    Nd = auto()
    Pm = auto()
    Sm = auto()
    Eu = auto()
    Gd = auto()
```

```
Tb = auto()
Dy = auto()
Ho = auto()
Er = auto()
Tm = auto()
Yb = auto()
Lu = auto()
Hf = auto()
Ta = auto()
W = auto()
Re = auto()
Os = auto()
Ir = auto()
Pt = auto()
Au = auto()
Hg = auto()
Tl = auto()
Pb = auto()
Bi = auto()
Po = auto()
At = auto()
Rn = auto()
Fr = auto()
Ra = auto()
Ac = auto()
Th = auto()
Pa = auto()
U = auto()
Np = auto()
Pu = auto()
Am = auto()
Cm = auto()
Bk = auto()
Cf = auto()
Fm = auto()
Md = auto()
No = auto()
Lr = auto()
Rf = auto()
Db = auto()
Bh = auto()
Hs = auto()
Mt = auto()
Ds = auto()
Rg = auto()
Cn = auto()
Nh = auto()
Fl = auto()
Mc = auto()
Lv = auto()
Ts = auto()
Og = auto()
```

F Code for ChemEntity.py

```
## @file ChemEntity.py
#  @title ChemEntity
#  @author Madhi Nagarajan
#  @brief This file is an abstract class which other classes utilize its methods
#  @date February 12, 2020
from abc import abstractmethod, ABC


## @brief The class, ChemEntity, represents an abstract class
#  @details The class, ChemEntity, represents an abstract class which other classes
#  utilize its methods
class ChemEntity(ABC):

    ## @brief The function is an abstract method
    #  @param R is a given value
    @abstractmethod
    def num_atoms(self, elem):
        pass

    ## @brief The function is an abstract method
    #  @param R is a given value
    @abstractmethod
    def constit_elems(self):
        pass
```

G Code for Equality.py

```
## @file Equality.py
#  @title Equality
#  @author Madhi Nagarajan
#  @brief This file is an abstract class which other classes utilize its methods
#  @date February 12, 2020
from abc import ABC, abstractmethod


## @brief The class, Equality, represents an abstract class
#  @details The class, Equality, represents an abstract class which other classes
#  utilize its methods
class Equality(ABC):

    ## @brief The function is an abstract method
    #  @param R is a given value
    @abstractmethod
    def equals(self, R):
        pass
```

H Code for Set.py

```
## @file Set.py
#  @title Set
#  @author Madhi Nagarajan
#  @brief This file acts as a data type for any Set. It inherits from the Equality class.
#  @date February 12, 2020

from Equality import Equality


## @brief The class, Set, represents a data type for all Sets
#  @details The class, Set, represents a data type for all Sets and does related operations
class Set(Equality):

    ## @brief Constructor for Set
    #  @details Constructor accepts one parameters that is a list.
    #  @param s is a parameter of type list.
    def __init__(self, s):
        self.T = s

    def __eq__(self, other):

        return self.equals(other)

    ## @brief The function adds a new value to the Set
    #  @param e is a new value that is being added to the Set
    #  @returns the Set after adding in the new value.
    def add(self, e):
        if not self.member(e):
            self.T.append(e)

    ## @brief The function removes a value from the Set
    #  @param e is the value being removed
    #  @returns the Set after removing the value.
    def rm(self, e):
        if self.member(e):
            self.T.remove(e)

    ## @brief The function checks if a value is in the Set
    #  @param e is the value being checked
    #  @returns a boolean value depending on whether the item is in the list
    def member(self, e):
        if e in self.T:
            return True
        else:
            return False

    ## @brief The function finds the size of the Set
    #  @returns an int value corresponding to the size of the Set
    def size(self):
        return len(self.T)

    ## @brief The function produces a list (of the Set) that is iterable
    #  @returns an iterable list value of the Set
    def to_seq(self):
        return self.T

    ## @brief The function checks whether current set is equal to a given set
    #  @param R is a given set
    #  @returns a boolean depending on if both sets equal
    def equals(self, R):
        if len(self.T) == len(R.T):
            if all(elem in R.to_seq() for elem in self.to_seq()):
                return True
        return False
```

I Code for ElmSet.py

```
## @file ElemSet.py
#  @title ElemSet
#  @author Madhi Nagarajan
#  @brief This file acts as a subclass of set
#  @date Feburary 12, 2020
from Set import *


## @brief The class, ElemSet, represents a subclass of set, for all Element Sets
#  @details ElemSet inherits all the common methods of Set.py, but acts
#  as a set for only elements.
class ElmSet(Set):

    ## @brief Constructor for ElmSet; inherits from Set
    #  @details Constructor accepts one parameter of a list to pass thru Set
    #  @param s is a list (of type Set) that gets passed thru Set
    def __init__(self, s):
        super().__init__(s)
```

J Code for MolecSet.py

```
## @file MolecSet.py
#  @title MolecSet
#  @author Madhi Nagarajan
#  @brief This file acts as a subclass of set
#  @date Feburary 12, 2020

from Set import *


## @brief The class, MolecSet, represents a subclass of set, for all Molecule Sets
#  @details ElemSet inherits all the common methods of Set.py, but acts as a set
#  for only Molecules
class MolecSet(Set):

    ## @brief Constructor for MolecSet; inherits from Set
    #  @details Constructor accepts one parameter of a list to pass thru Set
    #  @param s is a list (of type Set) that gets passed thru Set
    def __init__(self, s):
        super().__init__(s)
```

K Code for CompoundT.py

```
## @file CompoundT.py
#  @title CompoundT
#  @author Madhi Nagarajan
#  @brief This file acts as a data type for any Compound. It inherits from the
#  ChemEntity and Equality class.
#  @date February 12, 2020

from ChemEntity import ChemEntity
from ElmSet import ElmSet
from Equality import Equality


## @brief The class, CompoundT, represents a data type for Compounds
#  @details The class, CompoundT, represents a data type for Compounds
#  and does related operations
class CompoundT(ChemEntity, Equality):

    ## @brief Constructor for CompoundT
    #  @details Constructor accepts one parameter of a molecule set
    #  @param M is a MolecSet of the compound
    def __init__(self, M):
        self.__molec_set = M

    ## @brief The function returns the molec_set constructor the MoleculeT
    #  @returns a molecule set of all elements in a compound
    def get_molec_set(self):
        return self.__molec_set

    ## @brief The function calculates the number of atoms of a specific element in a Compound
    #  @param e is a given element
    #  @returns the number of atoms of that specific element in a Compound set.
    def num_atoms(self, e):
        sum = 0
        for molec in self.__molec_set.to_seq():
            if molec.get_elm() == e:
                sum += molec.get_num()
        return sum

    ## @brief The function produces an ElmSet of all the elements in a Compound
    #  @returns a list (ElmSet) of all the elements in a Compound
    def constit_elems(self):
        return ElmSet([m.get_elm() for m in self.__molec_set.to_seq()])

    ## @brief The function checks if this CompoundT equals another.
    #  @param D is a given CompoundT
    #  @returns A boolean depending on if the current CompoundT is equal to the
    #  given CompoundT
    def equals(self, D):
        if len(self.__molec_set.to_seq()) == len(D.__molec_set.to_seq()):
            if all(elem in D.molec_set for elem in self.__molec_set):
                return True
        return False
```

L Code for ReactionT.py

```
## @file ReactionT.py
#  @title ReactionT
#  @author Madhi Nagarajan
#  @brief This file acts as a data type for any Reaction.
#  @date Feburary 12, 2020


from numpy import linalg
from CompoundT import *
from MoleculeT import *
from Set import *
from ElmSet import *
from MolecSet import *


## @brief The class, ReactionT, represents a data type for Reactions
#  @details The class, ReactionT, represents a data type for Reactions and
#  does chemical reaction related operations
class ReactionT:

    ## @brief Constructor for ReactionT; fills out left-hand and right
    #  @details Constructor accepts two parameters, lhs and rhs. It fills out
    #  left-hand and right-hand
    # sides of the Reaction, as well as filling out the correct
    # LHS and RHS coefficients.
    #  @param L is a list/Set of Compounds of all LHS compounds in the reaction
    #  @param R is a list/Set of Compounds of all RHS compounds in the reaction
    def __init__(self, L, R):
        self.__lhs = L
        self.__rhs = R

        coeffs = self.chem_balance(L, R)
        self.__coeff_L = coeffs[0]
        self.__coeff_R = coeffs[1]

        if not (self.is_balanced(L, R, coeffs[0], coeffs[1]) and self.pos(
                coeffs[0]) and self.pos(coeffs[1])):
            raise ValueError('Unbalanced/invalid coefficients')

    ## @brief The function returns the LHS constructor of ReactionT
    #  @returns a list of LHS compounds
    def get_lhs(self):
        return self.__lhs

    ## @brief The function returns the RHS constructor of ReactionT
    #  @returns a list of RHS compounds
    def get_rhs(self):
        return self.__rhs

    ## @brief The function returns the LHS coefficient constructor of ReactionT
    #  @returns a list of LHS coefficients for all LHS compounds
    def get_lhs_coeff(self):
        return self.__coeff_L

    ## @brief The function returns the RHS coefficient constructor of ReactionT
    #  @returns a list of RHS coefficients for all RHS compounds
    def get_rhs_coeff(self):
        return self.__coeff_R

    ## @brief The function checks if all elements of a Set are positive
    #  @param s is the Set
    #  @returns a boolean value
    @staticmethod
    def pos(s):
        for i in s:
            if i <= 0:
                return False
        return True

    ## @brief The function finds the number of atoms in a compound
    #  @param C os a set of CompoundT
    #  @param c is a set of natural numbers
    #  @param e is a given element of type ElementT
    #  @returns the total number of atoms
    @staticmethod
    def n_atoms(C, c, e):
```

```
    atoms = 0
    for i in range(len(C)):
        atoms += c[i] * C[i].num_atoms(e)
    return atoms

## @brief The function finds the number of ...
#  @param C os a set of CompoundT
#  @param e is a given element of type ElementT
#  @returns the total number of atoms
@staticmethod
def elem_num(C, e):
    atoms = []
    for i in range(len(C)):
        atoms.append(C[i].num_atoms(e))
    return atoms

## @brief The function finds all the ElementTs in a CompoundT set
#  @param C os a set of CompoundT
#  @returns a set of these elements
@staticmethod
def elm_in_chem_eq(C):
    ret = []
    for comp in C:
        ret.append(comp.constit_elems().to_seq())
    return Set(ret)

## @brief The function checks if the number of atoms of
# an ElementT are the same on both LHS and RHS
#  @param L is a LHS set of CompoundT
#  @param R is a RHS set of CompoundT
#  @param L is a LHS set of balancing coefficients
#  @param R is a RHS set of balancing coefficients
#  @param e is a given element of type ElementT
#  @returns a set of these elements
def is_bal_elm(self, L, R, cL, cR, e):
    return self.n_atoms(L, cL, e) == self.n_atoms(R, cR, e)

## @brief The function checks if the LHS and RHS are balanced
#  @param L is a LHS set of CompoundT
#  @param R is a RHS set of CompoundT
#  @param L is a LHS set of balancing coefficients
#  @param R is a RHS set of balancing coefficients
#  @returns a set of these elements
def is_balanced(self, L, R, cL, cR):
    eq_atoms = all([self.is_bal_elm(L, R, cL, cR, elm)
                    for elm in self.elm_in_chem_eq(R).to_seq()])
    return eq_atoms

## @brief The function forms a matrix of a given side
#  @param C is a set of CompoundTs, either LHS or RHS
#  @returns a list/2D matrix
def matrix(self, C):
    comp_C = self.elm_in_chem_eq(C).to_seq()
    mat = []
    for comp_elms in comp_C:
        for elm in comp_elms:
            nums = self.elem_num(C, elm)
            mat.append(nums)
    return mat

## @brief The function balances the chemical reaction utilizing numpy
#  @param L is a LHS set of CompoundT
#  @param R is a RHS set of CompoundT
#  @returns a list containing both LHS and RHS coefficients
def chem_balance(self, L, R):
    mat1 = self.matrix(L)
    mat2 = self.matrix(R)

    mat = []
    for i in range(len(mat1)):
        neg = [-x for x in mat2[i]]
        mat.append(mat1[i] + neg)
    mat.append([0 for i in range(len(L) + len(R) - 1)])
    mat[-1].append(1)

    mat_B = [[0] for i in range(len(mat) - 1)]
    mat_B.append([1])
    calc = linalg.lstsq(mat, mat_B)[0].tolist()
    co_L = []
    co_R = []
```

```
for i in range(len(calc)):
    if i < len(L):
        co_L.append(round(calc[i][0], 5))
    else:
        co_R.append(round(calc[i][0], 5))

return [co_L, co_R]
```

M Code for test_All.py

```
## @file test_All.py
#  @title test_Al
#  @author Madhi Nagarajan
#  @brief This file is the test class
#  @date Feburary 12, 2020

from CompoundT import *
from MoleculeT import *
from Set import *
from ElmSet import *
from MolecSet import *
from ChemTypes import *
from ReactionT import *


class Test_All:

    def setup_method(self, method):
        self.s1 = Set([3, -6, 4, 0, 12, 9])
        self.e1 = ElmSet([ElementT.H, ElementT.O])

        self.m1 = MoleculeT(2, ElementT.H)
        self.m2 = MoleculeT(7, ElementT.O)
        self.m3 = MoleculeT(2, ElementT.H)
        self.m4 = MoleculeT(2, ElementT.O)
        self.m5 = MoleculeT(1, ElementT.O)

        self.c1 = CompoundT(MolecSet([self.m1, self.m2]))
        self.c2 = CompoundT(MolecSet([self.m3]))
        self.c3 = CompoundT(MolecSet([self.m4]))
        self.c4 = CompoundT(MolecSet([self.m3, self.m5]))

        # 2 H2 + O2 --> 2 H2O
        self.r1 = ReactionT([self.c2, self.c3], [self.c4])

        # 2 N --> N2
        self.m8 = MoleculeT(1, ElementT.N)
        self.m9 = MoleculeT(2, ElementT.N)
        self.c8 = CompoundT(MolecSet([self.m8]))
        self.c9 = CompoundT(MolecSet([self.m9]))
        self.r2 = ReactionT([self.c8], [self.c9])

    def teardown_method(self, method):
        pass

    def test_Set_add(self):
        self.s1.add(6)
        self.s1.equals(Set([3, -6, 4, 0, 12, 9, 6]))
        self.s1.add(9)
        self.s1.equals(Set([3, -6, 4, 0, 12, 9, 6]))

    def test_Set_rm(self):
        self.s1.rm(3)
        self.s1.rm(5)
        assert self.s1 == Set([-6, 4, 0, 12, 9])
        assert self.s1 == Set([-6, 4, 0, 12, 9])

    def test_Set_member(self):
        assert self.s1.member(3)
        assert not(self.s1.member(5))

    def test_Set_to_seq(self):
        assert self.s1.to_seq() == [3, -6, 4, 0, 12, 9]

    def test_Set_equals(self):
        assert self.s1.equals(Set([3, -6, 4, 0, 12, 9]))
        assert not(self.s1.equals(Set([5])))

    def test_ElmSet_add(self):
        self.e1.add(ElementT.C)
        assert self.e1 == ElmSet([ElementT.H, ElementT.O, ElementT.C])

    def test_MoleculeT_num_atoms(self):
        assert self.m1.num_atoms(ElementT.H) == 2
        assert self.m2.num_atoms(ElementT.C) == 0
```

```
def test_MoleculeT_constit_elems(self):
    assert self.m1.constit_elems().to_seq() == [ElementT.H]

def test_MoleculeT_equals(self):
    assert self.m1.equals(MoleculeT(2, ElementT.H))
    assert self.m1 == MoleculeT(2, ElementT.H)

def test_CompoundT_get_molec_set(self):
    assert self.c1.get_molec_set() == MolecSet([self.m1, self.m2])

def test_CompoundT_num_atoms(self):
    assert self.c1.num_atoms(ElementT.C) == 0
    assert self.c1.num_atoms(ElementT.H) == 2

def test_CompoundT_constit_elems(self):
    assert self.c1.constit_elems() == ElmSet([ElementT.H, ElementT.O])

def test_ReactionT_get_lhs(self):
    assert self.r1.get_lhs() == [self.c2, self.c3]

def test_ReactionT_get_rhs(self):
    assert self.r1.get_rhs() == [self.c4]

def test_ReactionT_get_lhs_coeff(self):
    assert self.r1.get_lhs_coeff() == [1, 0.5]
    assert self.r2.get_lhs_coeff() == [2]

def test_ReactionT_get_rhs_coeff(self):
    assert self.r1.get_rhs_coeff() == [1]
    assert self.r2.get_rhs_coeff() == [1]
```

N Code for Partner's Set.py

```
## @file Set.py
#  @author Hafez Issa
#  @brief Module which holds a Set of an abstract type
#  @details Inherits Equality
#  @date February 08, 2020

from Equality import *


## @brief An abstract data type for storing and
#  operating on sequences of type T
class Set(Equality):

    ## @brief Set constructor
    #  @details Initializes a Set object whose states
    #  consist of a sequence of abstract type
    #  @param s Sequence of abstract type
    def __init__(self, s):
        self.S = set(s)

    ## @brief Union sequence S with element e of abstract type
    #  @param e Element to be added into sequence S
    def add(self, e):
        self.S.add(e)

    ## @brief Remove element e of abstract type from sequence S
    #  @details exception, if element e is not contained in
    #  sequence S, raise ValueError
    #  @param e Element to be removed from sequence S
    def rm(self, e):
        if (e in self.to_seq()):
            self.S.remove(e)
        else:
            raise ValueError

    ## @brief Check if element e of abstract type is in sequence S
    #  @param e Element to compare
    #  @return Boolean representing whether the sequence S contains element e
    def member(self, e):
        return True if e in self.to_seq() else False

    ## @brief Measure length of sequence of abstract type
    #  @return Natural representing the length of the sequence
    def size(self):
        return len(self.to_seq())

    ## @brief Return a sequence of all the elements in the set
    #  @return Sequence representing the set of all elements
    def to_seq(self):
        return list(self.S)

    ## @brief Compare two sequences of abstract type
    #  @details Equality is reflexive, order should not matter
    #  check if size is equal and if all elements of one sequence is
    #  in the other sequence
    #  @param R Sequence of abstract type
    #  @return Boolean representing whether Sequence S is equal
    #  to Sequence R
    def equals(self, R):
        if not (self.size() == R.size()):
            return False

        for elem_s in self.to_seq():
            if (R.member(elem_s)):
                continue
            else:
                return False
        return True
```

O Code for Partner's MoleculeT.py

```
## @file MoleculeT.py
#  @author Hafez Issa
#  @brief Template module used to store and access objects of MoleculeT
#  @Date February 08, 2020

from ChemTypes import *
from Equality import *
from ChemEntity import *
from ElmSet import *


## @brief ADT of a MoleculeT object with its attributes
class MoleculeT(ChemEntity, Equality):

    ## @brief Constructor method for MoleculeT
    #  @param n Natural representing number of elements in MoleculeT
    #  @param e ElementT representing the element in MoleculeT
    def __init__(self, n, e):
        self.num = n
        self.elm = e

    ## @brief Getter method, returns ElementT type
    #  @return ElementT representing element of type ElementT
    def get_elem(self):
        return self.elm

    ## @brief Getter method, return Natural
    #  @return Natural representing the number of elements in MoleculeT
    def get_num(self):
        return self.num

    ## @brief Getter for number of atoms of specific element in MoleculeT
    #  @param e ElementT object to be described
    #  @return Natural number representing the number of atoms of the ElementT
    def num_atoms(self, e):
        if e == self.get_elem():
            return self.get_num()
        else:
            return 0

    ## @brief Convert elements of MoleculeT into ElmSet
    #  @return ElmSet representing the sequence of elements in MoleculeT
    def constit_elems(self):
        return ElmSet([self.get_elem()])

    ## @brief Check if two MoleculeT object are equal
    #  @param m MoleculeT object used to compare
    #  @return Boolean representing equality between two MoleculeT objects
    def equals(self, m):
        if (m.get_elem() == self.get_elem() and m.get_num() == self.get_num()):
            return True
        return False
```

P Code for Partner's CompoundT.py

```python
## @file CompoundT.py
#  @author Hafez Issa
#  @brief Template module used to store and access objects of CompoundT
#  @date February 08, 2020

from MoleculeT import *
from MolecSet import *
from Equality import *
from ElmSet import *
from MolecSet import *


## @brief ADT of a CompoundT object with its attributes
class CompoundT(ChemEntity, Equality):

    ## @brief Constructor of CompoundT object
    #  @param M MolecSet of MoleculeTs in the compound
    def __init__(self, M):
        self.C = M

    ## @brief Getter method
    #  @return MolecSet of all molecules in the CompoundT object
    def get_molec_set(self):
        return self.C

    ## @brief Get number of atoms of specific element e in the CompoundT object
    #  @param e ElementT to be described
    #  @return Natural number representing the number of e atoms in CompoundT object
    def num_atoms(self, e):
        num = 0
        for m in self.get_molec_set().to_seq():
            num += m.num_atoms(e)
        return num

    ## @brief Get list of elements in the CompoundT object
    #  @return ElmSet of all elements in the CompoundT object
    def constit_elems(self):
        atom = ElmSet([])
        for m in self.get_molec_set().to_seq():
            atom.add(m.get_elem())
        return atom

    ## @brief Check if two CompoundT object are equal
    #  @param D CompoundT object used to compare
    #  @return Boolean representing equality between two CompoundT objects
    def equals(self, D):
        return True if self.get_molec_set().equals(D.get_molec_set()) else False
```

Q Code for Partner's ReactionT.py

```
## @file ReactionT.py
#  @author Hafez Issa
#  @brief Module to balance the chemical equation of sequences CompoundT's
#  @date February 08, 2020

from CompoundT import *
from ChemTypes import *
import numpy as np


## @brief ADT which represents a ReactionT object, a chemical equation
class ReactionT():

    ## @brief Constructor for a chemical equation, ReactionT object
    #  @details exception, ValueError
    #  @param L Sequence of CompoundT's
    #  @param R Sequence of CompoundT's
    def __init__(self, L, R):
        self.lhs = L
        self.rhs = R
        self.coeffL = []
        self.coeffR = []

        temp_coeffL = []
        temp_coeffR = []

        compounds_total = []
        elems_in_total = __elm_in_chem_eq(self.lhs.to_seq())

        for i in self.lhs:
            compounds_total.append(i)

        for i in self.rhs:
            compounds_total.append(i)

        for i in compounds_total:
            a.append(i.num_atoms())

        for l in L:
            temp = l.get_molec_set().to_seq()
            for i in temp:
                temp_coeffL.append(i.get_num())
        self.coeffL = temp_coeffL

        for r in R:
            temp = r.get_molec_set().to_seq()
            for i in temp:
                temp_coeffR.append(i.get_num())
        self.coeffR = temp_coeffR

    ## @brief Getter method
    #  @return Sequence of type CompoundT
    def get_lhs(self):
        return self.lhs

    ## @brief Getter method
    #  @return Sequence of type CompoundT
    def get_rhs(self):
        return self.rhs

    ## @brief Getter method
    #  @return Sequence of type Real
    def get_lhs_coeff(self):
        return self.coeffL

    ## @brief Getter method
    #  @return Sequence of type Real
    def get_rhs_coeff(self):
        return self.coeffR

    ## @brief Check if sequence is positive
    #  @param s Sequence of type Real
    #  @return Boolean representing whether all elements in the sequence is positive
    def __pos(s):
        for i in s:
            if (i <= 0):
```

```
            return False
    return True

## @brief Count number of atoms
#  @param C Sequence of type CompoundT
#  @param c Sequence of type Real
#  @param e ElementT
#  return Natural Number which represents the number of atoms
def __n_atoms(C, c, e):
    count = 0
    for i in range(C.get_molec_set().size()):
        count += c[i] * C.get_molec_set().to_seq()[i].num_atoms(e)
    return count

## @brief Find elements in the sequence
#  @param C Sequence of type CompoundT
#  @return ElmSet containing the elements in the CompoundT
def __elm_in_chem_eq(C):
    elem = ElmSet([])
    for c in C.get_molec_set().to_seq():
        elem.add(c.constit_elems())
    return elem

## @brief
#  @param L Sequence of type CompoundT
#  @param R Sequence of type CompoundT
#  @param cL Sequence of type Real
#  @param cR Sequence of type Real
#  @param e ElementT
#  @return Boolean
def __is_bal_elm(L, R, cL, cR, e):
    if (n_atoms(L, cL, e) == n_atoms(R, cR, e)):
        return True
    return False

## @brief
#  @param L Sequence of type CompoundT
#  @param R Sequence of type CompoundT
#  @param cL Sequence of type Real
#  @param cR Sequence of type Real
#  @return Boolean representing whether the equations are balanced
def __is_balanced(L, R, cL, cR):
    if not (elm_in_chem_eq(L) == elm_in_chem_eq(R)):
        return False

    for e in elem_in_chem_eq(L):
        if not (is_bal_elm(L, R, cL, cR, e)):
            return False
    return True
```