Assignment 2 Solution

Dhruv Bhavsar, bhavsd1

February 16, 2020

This report discusses the testing results for all the modules written such as ReactionT, CompoundT, etc. for Assignment 2. It also includes the test results of partner's code for these programs. I also discuss the quality of the given specifications and answer the given discussion questions.

1 Testing of the Original Program

For this assignment, the tests were written in pytest, an unit testing framework for python. What I noticed from my last assignment's test driver (without using pytest) and pytest, is that they both behaved and worked in the same fashion. Though, pytest is much easier better to use since it provides coverage analysis on your code which is really helpful since it tells you how much of the module you have tested thus far. For my test cases, I made my compounds and reactions have a matrix that is not n x n to ensure that the method works for all matrices. I also included a test case that does not have a solution, to ensure that my program throws an ValueError just like it was mentioned in the specifications. So my test file had 52 tests and I was able to pass all of them. As usual, testing the program helped me find errors in my code.

2 Results of Testing Partner's Code

When running my test driver against my partners code, I came across with 5 tests failed out of the 52 tests I have. They passed 47 tests. Upon further examination of the fails, I found out that my partners answers were right but they weren't in integers as they were in real numbers. For my ReactionT I was able to find a way to convert my real numbers into equivalent integer form. So my partner did pass those test cases, I should have included an equivalent calculator function in my code to able to compare two equivalent numbers. The other 3 errors were about checking if two elm sets are equal. At first I was really confused on why this error occurred. But after carefully going through the code for Set, I found that they used == in the member function, which is not correct. If they use ==, they should have included eq function in MoleculeT in order for == to work correctly. Right now it is comparing memory address with a Set. With the last assignment, my partner had one mistake with their code, adding an extra day in between the days. It was a minor error with his code. Same thing for this assignment, all they need to do is add the eq function in MoleculeT. When I added the eq function, all the tests passed (except the coeffs test but it was different equivalent form).

3 Critique of Given Design Specification

The specifications for the this assignment was given by mathematical notations, which in my opinion is much more concrete than specifications given in natural language. Though it can be hard to understand the meaning of the function with mathematical notation as it can get very complex quick. One of the strengths of this design is its high cohesion between modules. Components are very closely related as most of them use each other. This design is opaque as it uses information hiding to hide calculations in the **ReactionT** module especially.

4 Answers

a) As I mentioned before natural language in the specification for A1 was somewhat ambiguous in some aspects and the formal specification is more concrete in terms what it wants you to implement. The advantage of natural language specification is that it is much easier to understand what you need to do for the method. However it can be ambiguous in terms of implementation. Formal specification provides more concrete implementations but can be tricky to decipher.

b) The process of converting the strings to logical syntactic components is called parsing. You convert the string that is inputted into the respective Element by some sort of matching of strings. I think I would need to include a module named Parser that is responsible for parsing the string into an appropriate type/object. It will be stored in an array of strings, then iterate over the array and create the corresponding object of the molecule.

c) To calculate the atomic mass of the elements, compounds and reactions just add an extra field with the Element types that holds the mass of the element. Then make a function that goes through the list or set just like how num atoms works and sum up the mass based on the element and the number of atoms for respective element.

d) In usual chemistry, coefficients are not decimal numbers they are whole numbers. An algorithm to ensure the coefficients are whole numbers can be found in my **ReactionT**. The code for ReactionT was inspired from this `https://stackoverflow.com/questions/42637872/solve-system-of-linear-integer-equations-in-python`.

e) The difference between static typing and dynamic typing is that static you have to be explicit on what type you are using whereas with dynamic typing you don't need to declare the type, it will automatically interpret what type you need. The advantage of dynamic typing is that you don't need a generic type as generic type is the same thing as you declare it for a specific type. The disadvantage is that it might cause run time errors with the mismatch types. The advantage of static typing is that large amount of errors can be caught before runtime as there will be a type mismatch error. The disadvantage of static typing is that it can hold up rapid development as you have to be constantly worried about the correct types. Source: `quora.com/What-are-the-pros-and-cons-and-needs-for-static-dynamic-type-checker-or-both`

f) `[(i, i+2) for i in range(10) if not(i%2==0)]`

g) def len(n): return sum(list(map(lambda x: 1, n)))

h) Interface is defined to be a contract between the system and the environment. The interface informally describes what can pass between the system and environment. The implementation is defined as the implementation of the interface.

i)
  i) Abstraction is the process of focussing on the important details instead of worrying about the irrelevant details.
  ii) Anticipation of change is programming based on knowing that a change can arise and your program must be able to change as requirements and such change.
  iii) Generality is solving a general problem than a really specific problem. This is important in software engineering because you can now use that general solution for many more problems.
  iv) Modularity is when a complex system is broken down into smaller less complex systems and you work your way bottom up.
  v) Seperation of concerns is the principle of concerns being seperated and considered independently.

E Code for ChemTypes.py

```
## @file ChemTypes.py
#  @author Dhruv Bhavsar
#  @brief Definition of Element types
#  @date Feb 1, 2020

from enum import Enum, auto


## @brief A data type to hold all elements in the periodic table
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()
 Es = auto()
 Fm = auto()
 Md = auto()
 No = auto()
 Lr = auto()
 Rf = auto()
 Db = auto()
 Sg = 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
#  @author Dhruv Bhavsar
#  @brief Abstract interface for methods about chemical entities
#  @date Feb 1, 2020

from abc import ABC, abstractmethod


## @brief
class ChemEntity(ABC):

    ## @brief An abstract method for counting number of atoms
    #  @param element_t
    @abstractmethod
    def num_atoms(self, element_t):
        pass

    @abstractmethod
    def constit_elems(self):
        pass
```

G Code for Equality.py

```
## @file Equality.py
#  @author Dhruv Bhavsar
#  @brief Used for comparing two objects
#  @Date Feb 1, 2020

from abc import ABC, abstractmethod


## @brief An abstract class for comparing two objects
class Equality(ABC):

    ## @brief Abstract method for checking if two objects are equal
    #  @param other Object to compare against
    #  @return true is equal else false
    @abstractmethod
    def equals(self, other):
        pass
```

H Code for Set.py

```
## @file Set.py
#  @author Dhruv Bhavsar
#  @brief Class for Set building and applying methods to the Set
#  @date Feb 2, 2020

from Equality import *


## @brief Class that represents a Set
class Set(Equality):

    ## @brief Constructor that initializes the object with set
    #  @details Takes in a sequence and converts it into set and
    #           holds it in the state variable
    #  @param sequence - Takes in a sequence (list)
    def __init__(self, sequence):
        self.__set = set(sequence)

    ## @brief Add an element to the set
    #  @details Uses the add function of set object to add the element
    #  @param Element to add
    def add(self, element):
        self.__set.add(element)

    ## @brief Remove a specified from the set
    #  @details Uses the remove function from the Set Object
    #  @param Specified Element
    #  @throws ValueError if element not in set
    def rm(self, element):
        try:
            self.__set.remove(element)
        except KeyError:
            raise ValueError("Element is not in the set")

    ## @brief Checks if the element is in the set
    #  @param Element to check
    #  @return True if in the set else false
    def member(self, element):
        if element in self.__set:
            return True
        else:
            return False

    ## @brief Returns the size of the set
    #  @details Using len function
    #  @return the size of the set as an integer
    def size(self):
        return len(self.__set)

    ## @brief Checks if the two sets are the same
    #  @details First check if the two sets are the same then
    #           check if all the elements occur in the other set
    #  @param other_set - Set to check with
    #  @return True if the same else false
    def equals(self, other_set):
        if not (self.size() == other_set.size()):
            return False

        for elem in self.__set:
            if not other_set.member(elem):
                return False
        return True

    ## @brief Convert the set into a list for it to be iterable
    #  @return List of the sets
    def to_seq(self):
        return list(self.__set)
```

I Code for ElmSet.py

```
## @file ElemSet.py
#  @author Dhruv Bhavsar
#  @brief Class used to rename the Set, set of ElementT
#  @date Feb 3, 2020

from Set import *


## @brief Inherits from Set, just used for new name of Set
class ElmSet(Set):
    pass
```

J Code for MolecSet.py

```
## @file MolecSet.py
#  @author Dhruv Bhavsar
#  @brief Class used to rename the Set, set of MoleculeT
#  @date Feb 3, 2020

from Set import *


## @brief Inherits from Set, just used for new name of Set
class MolecSet(Set):
    pass
```

K Code for CompoundT.py

```
## @file CompoundT.py
#  @author Dhruv Bhavsar
#  @brief Class for holding a MolecSet
#  @date Feb 3, 2020

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


## @brief Class that represents a Compound, inherits ChemEntity and Equality
class CompoundT(ChemEntity, Equality):

    ## @brief Constructor to initalize the object with a MolecSet
    #  @param molec_set - MolecSet to be stored in a Compound
    def __init__(self, molec_set):
        self.__C = molec_set

    ## @brief Return the Compound which is a MolecSet
    #  @return MolecSet
    def get_molec_set(self):
        return self.__C

    ## @brief Return the number of atoms in the MolecSet with the specified element
    #  @details Using functional programming functions, turn the
    #           MolecSet into a sequence to iterate over
    #           and find the number of atoms for each MolecSet
    #           with a specified element then sum up the list
    #  @param element - ElementT
    #  @return the total number of atoms of element in the Compound
    def num_atoms(self, element):
        return sum([m.num_atoms(element) for m in self.__C.to_seq()])

    ## @brief Returns the ElmSet of the all the different ElementT in the compound
    #  @details Using get_elm() function for MoleculeT to get the element
    #  @return ElmSet of all the ElementT
    def constit_elems(self):
        return ElmSet([m.get_elm() for m in self.__C.to_seq()])

    ## @brief Check if the two compounds are equal
    #  @param other_compound - other compound to compare with
    #  @return true if equal else false
    def equals(self, other_compound):
        return self.__C.equals(other_compound.get_molec_set())

    def __eq__(self, other):
        return self.equals(other)
```

L Code for ReactionT.py

```
## @file ReactionT.py
#  @author Dhruv Bhavsar
#  @brief Class that holds two compounds for the reactions
#  @date Feb 3, 2020

from ChemTypes import *
from CompoundT import *
from sympy import Matrix, lcm
import numpy as np


## @brief Class to simulate a reaction with balancing the equation
class ReactionT:

    ## @brief Constructor for ReactionT which balances before storing it
    #  @details Uses local functions to calculate the coeffs, then check
    #           if it is balanced else throw Value Error
    #           The reason I dont check for positive coeffs is
    #           because I make them positive in the calculation of coeffs
    #  @param left_compound - CompoundT right_compound - CompoundT
    def __init__(self, left_compound, right_compound):

        coeffs = self.__calc_coeffs__(left_compound, right_compound)

        left_coeffs = [coeffs[i] for i in range(len(left_compound))]
        right_coeffs = [coeffs[len(left_compound) + i]
                        for i in range(len(right_compound))]

        if not self.__is_balanced__(left_compound, right_compound, left_coeffs, right_coeffs):
            raise ValueError("Not Balanceable")

        self.__rhs = right_compound
        self.__lhs = left_compound
        self.__left_coeffs = left_coeffs
        self.__right_coeffs = right_coeffs

    ## @brief Return the left hand side of ReactionT
    #  @return the left side CompoundT
    def get_lhs(self):
        return self.__lhs

    ## @brief Return the right hand side of ReactionT
    #  @return the right side CompoundT
    def get_rhs(self):
        return self.__rhs

    ## @brief Return the coeffs for the left side
    #  @return list of coeffs
    def get_lhs_coeff(self):
        return self.__left_coeffs

    ## @brief Return the coeffs for the right side
    #  @return list of coeffs
    def get_rhs_coeff(self):
        return self.__right_coeffs

    ## @brief Return the ElementT in the compound
    #  @return ElmSet of elements
    def __elm_in_chem_eq__(self, c):
        new_set = []
        for i in c:
            new_set += i.constit_elems().to_seq()
        return ElmSet(new_set)

    # Function for calculating coefficients,
    # Build the matrix of the rows being the different elements
    # Each index is how many atoms of that element in there
    # The for loop creates each row meaning for a new element
    # The rest of the calculation to find the coefficients using a
    # linear system of equation solver Source:
    # https://stackoverflow.com/questions/42637872/solve-system-of-linear-integer-equations-in-python
    # Throws a Value Error if it can't be solved
    def __calc_coeffs__(self, left_compound, right_compound):
        try:
            left_compound_elms = self.__elm_in_chem_eq__(left_compound).to_seq()
            matrix = []
            for element in left_compound_elms:
                row = \
                    ([m.num_atoms(element)
                      for m in left_compound] + [m.num_atoms(element)
                                                 for m in right_compound])
                matrix.append(row)

            new_matrix = Matrix(matrix)
            v = new_matrix.nullspace()[0]
            m = lcm([val.q for val in v])
            x = m * v
            result = np.array([int(val) for val in x])
            coeffs = list(map(lambda c: abs(c), result))
            return coeffs
        except IndexError:
            raise ValueError("Cannot be balanced")

    # Function to find the number of atoms in a compound for a specified element
    # Also multiply the number of atoms by the coeff
    def __n_atoms__(self, comp, c, element):
        atoms = [c[i] * comp[i].num_atoms(element) for i in range(len(comp))]
        return sum(atoms)

    # Function to find if both the left side and right side have the same
    # number of atoms for a specific element
    def __is_bal_elm__(self, left_comp, right_comp, left_coef, right_coef, element):
        return self.__n_atoms__(left_comp, left_coef, element) \
            == self.__n_atoms__(right_comp, right_coef, element)

    # Function for checking if the reaction has the same elements on the right side
    # and left side. Then checks if the both sides are balanced
    def __is_balanced__(self, left_comp, right_comp, left_coef, right_coef):
        same_elms = \
            self.__elm_in_chem_eq__(left_comp).equals(self.__elm_in_chem_eq__(right_comp))

        balanced_elm = [self.__is_bal_elm__(left_comp, right_comp, left_coef, right_coef, e)
                        for e in self.__elm_in_chem_eq__(left_comp).to_seq()]

        return same_elms and balanced_elm
```

M Code for test_All.py

```
## @file test_All.py
#  @author Dhruv Bhavsar
#  @brief Test Module using Pytest
#  @date Feb 8, 2020

import collections
from pytest import *
from ChemEntity import *
from ChemTypes import *
import ElmSet
import MolecSet
from Equality import *
from CompoundT import *
from MoleculeT import *
from ReactionT import *
from Set import *


# Set
class TestSet:

    def setup_method(self, method):
        x = [1, 12, 10, 15, 37]
        y = ["a", "b", "d", "r", "t"]
        self.intS = Set(x)
        self.stringS = Set(y)

    def teardown_method(self, method):
        self.intS = None
        self.stringS = None

    def test_int_add(self):
        self.intS.add(178)
        assert self.intS.member(178)

    def test_string_add(self):
        self.stringS.add("good")
        assert self.stringS.member("good")

    def test_int_rm(self):
        self.intS.rm(12)
        assert not self.intS.member(12)

    def test_string_rm(self):
        self.stringS.rm("a")
        assert not self.stringS.member("a")

    def test_int_rm_error(self):
        with raises(ValueError):
            self.intS.rm(2)

    def test_string_rm_error(self):
        with raises(ValueError):
            self.stringS.rm("z")

    def test_int_member(self):
        assert self.intS.member(1)

    def test_string_member(self):
        assert self.stringS.member("t")

    def test_int_size(self):
        assert self.intS.size() == 5

    def test_string_size(self):
        assert self.stringS.size() == 5

    def test_int_equals(self):
        new_set = Set([1, 12, 10, 15, 37])
        assert self.intS.equals(new_set)

    def test_string_equals(self):
        new_set = Set(["a", "b", "d", "r", "t"])
        assert self.stringS.equals(new_set)

    def test_int_not_equals(self):
        assert not self.intS.equals(Set([1, 2, 6, 54, 3]))

    def test_string_not_equals(self):
        assert not self.stringS.equals(Set(["a", "b"]))

    def test_int_to_seq(self):
        assert collections.Counter(
            self.intS.to_seq()) == collections.Counter([1, 12, 10, 15, 37])

    def test_string_to_seq(self):
        assert collections.Counter(
            self.stringS.to_seq()) == collections.Counter(["a", "b", "d", "r", "t"])

    def test_int_to_seq_invalid(self):
        assert not collections.Counter(
            self.intS.to_seq()) == collections.Counter([1, 12, 10, 15, 56, 37])

    def test_string_to_seq_invalid(self):
        assert not collections.Counter(
            self.stringS.to_seq()) == collections.Counter(["a", "b", "d", "r", "t", "tyu"])


# MoleculeT
class TestMoleculeT:

    def setup_method(self, method):
        self.f7 = MoleculeT(7, ElementT.F)
        self.h2 = MoleculeT(2, ElementT.H)
        self.o4 = MoleculeT(4, ElementT.O)

    def teardown_method(self, method):
        self.f7 = None
        self.h2 = None
        self.o4 = None

    def test_get_num(self):
        assert self.f7.get_num() == 7

    def test_get_elm(self):
        assert self.h2.get_elm() == ElementT.H

    def test_num_atoms_oxy(self):
        assert self.o4.num_atoms(self.o4.get_elm()) == 4

    def test_num_atoms_fluorine(self):
        assert self.f7.num_atoms(self.f7.get_elm()) == 7

    def test_constit_elems_hydrogen(self):
        assert self.h2.constit_elems().equals(ElmSet([ElementT.H]))

    def test_constit_elems_oxygen(self):
        assert self.o4.constit_elems().equals(ElmSet([ElementT.O]))

    def test_constit_elems_fluorine(self):
        assert self.f7.constit_elems().equals(ElmSet([ElementT.F]))

    def test_equals_hydrogen(self):
        assert self.h2.equals(self.h2)

    def test_equals_fluorine(self):
        assert self.f7.equals(MoleculeT(7, ElementT.F))

    def test_equals_oxygen(self):
        new_mole = MoleculeT(4, ElementT.O)
        assert self.o4.equals(new_mole)

    def test_not_equals(self):
        assert not self.f7.equals(self.o4)


# CompoundT
class TestCompoundT:

    def setup_method(self):
        self.h2so4 = CompoundT(MolecSet([MoleculeT(1, ElementT.S),
                                 MoleculeT(4, ElementT.O), MoleculeT(2, ElementT.H)]))
        self.xef5 = CompoundT(MolecSet([MoleculeT(5, ElementT.F), MoleculeT(1, ElementT.Xe)]))
        self.h2o2 = CompoundT(MolecSet([MoleculeT(2, ElementT.H), MoleculeT(2, ElementT.O)]))

    def teardown_method(self):
        self.h2o2 = None
        self.xef5 = None
        self.h2so4 = None

    def test_get_molec_set_h2so4(self):
        assert self.h2so4.get_molec_set().equals(
            MolecSet([MoleculeT(1, ElementT.S),
                      MoleculeT(4, ElementT.O), MoleculeT(2, ElementT.H)]))

    def test_get_molec_set_h2o2(self):
        assert self.h2o2.get_molec_set().equals(MolecSet([MoleculeT(2, ElementT.H),
                                                          MoleculeT(2, ElementT.O)]))

    def test_num_atoms_xef5(self):
        assert self.xef5.num_atoms(ElementT.F) == 5

    def test_num_atoms_xef5_2(self):
        assert self.xef5.num_atoms(ElementT.Xe) == 1

    def test_num_atoms_h2so4(self):
        assert self.h2so4.num_atoms(ElementT.F) == 0

    def test_num_atoms_h2so4_2(self):
        assert self.h2so4.num_atoms(ElementT.S) == 1

    def test_constit_elems_h2o2(self):
        assert self.h2o2.constit_elems().equals(ElmSet([ElementT.H, ElementT.O]))

    def test_constit_elems_h2so4(self):
        assert not self.h2so4.constit_elems().equals(ElmSet([ElementT.H, ElementT.O]))

    def test_constit_elems_xef5(self):
        assert self.xef5.constit_elems().equals(ElmSet([ElementT.Xe, ElementT.F]))

    def test_equals_1(self):
        assert self.xef5.equals(self.xef5)

    def test_equals_2(self):
        molec_set = CompoundT(MolecSet([MoleculeT(1, ElementT.S),
                                        MoleculeT(4, ElementT.O), MoleculeT(2, ElementT.H)]))
        assert self.h2so4.equals(molec_set)

    def test_equals_3(self):
        assert not self.h2o2 == self.h2so4


# ReactionT
class TestReactionT:

    def setup_method(self, method):
        self.xe = CompoundT(MolecSet([MoleculeT(1, ElementT.Xe)]))
        self.f2 = CompoundT(MolecSet([MoleculeT(2, ElementT.F)]))
        self.xef6 = CompoundT(MolecSet([MoleculeT(6, ElementT.F), MoleculeT(1, ElementT.Xe)]))

        self.xef6_reaction = ReactionT([self.xe, self.f2], [self.xef6])

        self.h2 = CompoundT(MolecSet([MoleculeT(2, ElementT.H)]))
        self.o2 = CompoundT(MolecSet([MoleculeT(2, ElementT.O)]))
        self.h2o = CompoundT(MolecSet([MoleculeT(2, ElementT.H), MoleculeT(1, ElementT.O)]))

        self.h2o_reaction = ReactionT([self.h2, self.o2], [self.h2o])

        self.c3h8 = CompoundT(MolecSet([MoleculeT(3, ElementT.C), MoleculeT(8, ElementT.H)]))
        self.co2 = CompoundT(MolecSet([MoleculeT(1, ElementT.C), MoleculeT(2, ElementT.O)]))

        self.co2h20_reaction = ReactionT([self.c3h8, self.o2], [self.co2, self.h2o])

    def teardown_method(self, method):
        self.xef5 = None
        self.co2h20 = None
        self.h20 = None

    def test_init_invalid(self):
        with raises(ValueError):
            na = CompoundT(MolecSet([MoleculeT(1, ElementT.Na)]))
            naoh = CompoundT(MolecSet([MoleculeT(1, ElementT.Na), MoleculeT(1, ElementT.H),
                                       MoleculeT(1, ElementT.O)]))
            ReactionT([na, self.h2o], [naoh])

    def test_get_lhs_1(self):
        assert self.h2o_reaction.get_lhs() == [self.h2, self.o2]

    def test_get_lhs_2(self):
        assert self.co2h20_reaction.get_lhs() == [self.c3h8, self.o2]

    def test_get_rhs_1(self):
        assert self.h2o_reaction.get_rhs() == [self.h2o]

    def test_get_rhs_2(self):
        assert self.xef6_reaction.get_rhs() == [self.xef6]

    def test_get_lhs_coeff_1(self):
        assert self.xef6_reaction.get_lhs_coeff() == [1, 3]

    def test_get_lhs_coeff_2(self):
        assert self.co2h20_reaction.get_lhs_coeff() == [1, 5]

    def test_get_lhs_coeff_3(self):
        assert self.h2o_reaction.get_lhs_coeff() == [2, 1]

    def test_get_rhs_coeff_1(self):
        assert self.xef6_reaction.get_rhs_coeff() == [1]

    def test_get_rhs_coeff_2(self):
        assert self.co2h20_reaction.get_rhs_coeff() == [3, 4]

    def test_get_rhs_coeff_3(self):
        assert self.h2o_reaction.get_rhs_coeff() == [2]
```

N Code for Partner's Set.py

```
## @file Set.py
#  @author Zihao Du
#  @brief Module that creates the Set generic abstract data type
#  @date Feb 6, 2020

from Equality import *


## @brief A generic abstract data type that represents a set
class Set(Equality):

    ## @brief Set constructor
    #  @details Initializes a Set object whose state consists
    #  of a sequence of some type
    #  @param s sequence of some type
    def __init__(self, s):
        self._S = set(s)

    ## @brief Add an element into the Set object
    def add(self, e):
        self._S.add(e)

    ## @brief Remove a certain element in the set
    #  @throw If the element is not in the set, a ValueError will be raised
    # @param e The element the client wants to delete
    def rm(self, e):
        if (self.member(e) is False):
            raise ValueError
        self._S.remove(e)

    ## @brief Determine if a certain element is in the set
    # @return True if the element is in the set, False otherwise
    # @param e The element the client want to test
    def member(self, e):
        for x in self._S:
            if (x == e):
                return True
        return False

    ## @brief Obtain the number of elements in the Set
    # @return The number of elements in the Set
    def size(self):
        return len(self._S)

    ## @brief Inherit from Equality,
    #  determine if a certain Set object equals the current one
    # @return True if they are the same set, False otherwise
    # @param r The Set object to compare with the current Set
    def equals(self, r):
        if (r.size() != self.size()):
            return False
        for e in self._S:
            if (r.member(e) is False):
                return False
        return True

    ## @brief Obtain the list of elements in the set
    # @return List of elements in the set
    def to_seq(self):
        return list(self._S)

    ## @brief Magic function, redefine the meaning of equivalence
    # @return True if two sets are the same set
    # @param r The Set object to compare with the current Set
    def __eq__(self, r):
        if (r.size() != self.size()):
            return False
        for e in self._S:
            if (r.member(e) is False):
                return False
        return True
```

O   Code for Partner's MoleculeT.py

```
## @file ChemTypes.py
#  @author Zihao Du
#  @brief Module that creates the MoleculeT ADT
#  @date Feb 6, 2020

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


## @brief An abstract data type that represents a Molecule, a child class of
#  ChemEntity and Equality
class MoleculeT(ChemEntity, Equality):

    ## @brief MoleculeT constructor
    #  @details Initializes a MoleculeT object whose state consists
    #  of a natural number and a ElementT object
    #  @param n a natural number(the subscript)
    #  @param e the ElementT object
    def __init__(self, n, e):
        self._num = n
        self._elm = e

    ## @brief Get number of atoms in the molecule
    # @return The number of atoms in the molecule
    def get_num(self):
        return self._num

    ## @brief Get element of the molecule
    # @return The element of the molecule
    def get_elm(self):
        return self._elm

    ## @brief Obtain the number of a certain element in the molecule
    # @return The number of atoms of the element in the molecule
    # @param e ElementT object the client is interested in
    def num_atoms(self, e):
        if (e != self._elm):
            return 0
        else:
            return self._num

    ## @brief Get the element in the molecule in a ElmSet
    # @return ElmSet that contains the element in the molecule
    def constit_elems(self):
        s1 = ElmSet([self._elm])
        return s1

    ## @brief Determine if a molecule is equal to the current molecule
    # @return True if they are the same, otherwise False
    # @param m A moleculeT object to compare with the current one
    def equals(self, m):
        if((m._elm == self._elm) & (m._num == self._num)):
            return True
        else:
            return False
```

P Code for Partner's CompoundT.py

```
## @file CompoundT.py
#  @author Zihao Du
#  @brief Module that creates the CompoundT ADT
#  @date Feb 6, 2020

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


## @brief An abstract data type that represents a compound(set of molecules), a
#  child class of ChemEntity and Equality
class CompoundT(ChemEntity, Equality):

    ## @brief CompoundT constructor
    #  @details Initializes a CompoundT object whose state consists
    #  of a MolecSet
    #  @param m A MolecSet
    def __init__(self, m):
        self.__C = m

    ## @brief Obtain the MolecSet of the compound
    # @return The MolecSet of the compound
    def get_molec_set(self):
        return self.__C

    ## @brief Inherit from ChemEntity,
    #  obtain the number of atoms of a certain element in the compound
    # @return The number of atoms of a certain element in the compound
    # @param e ElementT object the client is interested in
    def num_atoms(self, e):
        sum = 0
        for m in (self.__C).to_seq():
            sum += m.num_atoms(e)
        return sum

    ## @brief Inherit from ChemEntity,
    #  obtain the set of Elements in the compound
    # @return An ElmSet containing all elements that appears in the compound
    def constit_elems(self):
        set1 = ElmSet([])
        for m in (self.__C).to_seq():
            set1.add(m.get_elm())
        return set1

    ## @brief Inherit from Equality,
    #  determine if two compounds are the same
    # @return True if two compounds have the same molecules inside
    # @param d CompoundT object to compare with the current one
    def equals(self, d):
        return (self.__C.equals(d.get_molec_set()))

    ## @brief Magic method, determine if two compounds are the same
    # @return True if two compounds have the same molecules inside
    # @param d CompoundT object to compare with the current one
    def __eq__(self, d):
        return (self.__C.equals(d.get_molec_set()))
```

Q Code for Partner's ReactionT.py

```
## @file ReactionT.py
#  @author Zihao Du
#  @brief Module that creates the ReactionT ADT
#  @date Feb 6, 2020

from ChemTypes import *
from CompoundT import *
from numpy import *


## @brief An abstract data type that represents a chemical reaction
class ReactionT:

    ## @brief ReactionT constructor
    #  @details Initializes a ReactionT object whose state consists
    #  of two sequences of CompoundT, two sequences of real numbers.
    #  The constructor will calculate the two sequences of coefficients using
    #  linear system of equations solver.
    #  @throw If any coefficient is zero or negative,
    #  or the number of compounds is not number of elements plus one,
    #  or the reaction cannot be balanced, a ValueError will be raised.
    #  @param l sequence of CompoundT representing the left hand side of the reaction
    #  @param r sequence of CompoundT representing the right hand side of the reaction
    def __init__(self, l, r):
        if (__elm_in_chem_eq__(l) != __elm_in_chem_eq__(r)):
            raise ValueError
        elements = __elm_in_chem_eq__(l)
        a = []
        b = []
        for elm in elements.to_seq():
            b.append([-(l[0].num_atoms(elm))])
            row = []
            for m in (l + r)[1:]:
                row.append(m.num_atoms(elm))
            a.append(row)
        if (len(a) != len(a[0])):
            raise ValueError
        coeff = linalg.solve(a, b)
        lhsc = [1]
        rhsc = []
        for i in range(len(l) - 1):
            lhsc.append(float(coeff[i][0]))
        for i in range(len(r)):
            rhsc.append(float(coeff[len(l) - 1 + i][0]))
        rhsc = [-x for x in rhsc]
        if(__is_balanced__(l, r, lhsc, rhsc) is False):
            raise ValueError
        if (__pos__(rhsc) is False or __pos__(lhsc) is False):
            raise ValueError
        self._lhs = l
        self._rhs = r
        self._coeffl = lhsc
        self._coeffr = rhsc

    ## @brief Obtain the sequence of CompoundT representing the left hand side
    # @return The sequence of CompoundT representing the left hand side
    def get_lhs(self):
        return self._lhs

    ## @brief Obtain the sequence of CompoundT representing the right hand side
    # @return The sequence of CompoundT representing the right hand side
    def get_rhs(self):
        return self._rhs

    ## @brief Obtain the list of real numbers representing the left hand side coefficients
    # @return The sequence of real numbers representing the left hand side coefficients
    def get_lhs_coeff(self):
        return self._coeffl

    ## @brief Obtain the list of real numbers representing the right hand side coefficients
    # @return The sequence of real numbers representing the right hand side coefficients
    def get_rhs_coeff(self):
        return self._coeffr


## @brief Determine if a sequence of real numbers have a positive or zero element in it
```

```
#  @param s List of real numbers
#  @return True if all elements are positive, otherwise False
def __pos__(s):
    for m in s:
        if (m <= 0):
            return False
    return True


## @brief Count the number of a certain atom on one side of the reaction
#  @param cs List of CompoundT
#  @param c List of coefficients
#  @param e A certain Element the client wants to count
#  @return The number of atoms of a certain element in one side of the reaction
def __n_atoms__(cs, c, e):
    sum = 0
    for i in range(len(cs)):
        sum += (c[i] * cs[i].num_atoms(e))
    return sum


## @brief List all the elements that appear in a sequence of CompoundT
#  @param C List of CompoundT
#  @return A ElmSet of elements that appears in the sequence of compounds
def __elm_in_chem_eq__(c):
    s = ElmSet([])
    for com in c:
        for e in (com.constit_elems().to_seq()):
            s.add(e)
    return s


## @brief Determine if two sides of a reaction achieve a balance in a certain element
#  @param l List of CompoundT on one side
#  @param r List of CompoundT on the other side
#  @param cl List of coefficients on one side
#  @param cl List of coefficients on the other side
#  @param e A certain Element the client wants to count
#  @return True if the number of atoms of the element is equal, otherwise False
def __is_bal_elm__(l, r, cl, cr, e):
    return (__n_atoms__(l, cl, e) == __n_atoms__(r, cr, e))


## @brief Determine if two sides of a reaction is balanced
#  @param l List of CompoundT on one side
#  @param r List of CompoundT on the other side
#  @param cl List of coefficients on one side
#  @param cl List of coefficients on the other side
#  @return True if they achieve a balance, otherwise False
def __is_balanced__(l, r, cl, cr):
    for e in (__elm_in_chem_eq__(l)).to_seq():
        if (__is_bal_elm__(l, r, cl, cr, e) is False):
            return False
    if (__elm_in_chem_eq__(l) != __elm_in_chem_eq__(r)):
        return False
    return True
```