Assignment 2 Solution

Senni Tan tans28

February 15, 2020

The assignment implementation is specified in an MIS that consists of nine modules. The modules are implementations of a library of enumerated types that represents all chemical elements, two interfaces for functions, a generic abstract data type that represents a set and its two template modules which represents sets of elements and molecules respectively, and three abstract data types that represents molecules, compounds and chemical reactions. This report will be discussing the approach taken to test the implementation, the rationale for test case selections , comparison of the test results of partner files in A1 and A2, critiques of the specified design, and answers to the question related to the design in this assignment and the knowledge from lectures. Appendix E to Q are the code of my design and the partner's design for this assignment.

1 Testing of the Original Program

Testing was implemented by creating test cases in pytest and the test cases covered normal cases and boundary cases.

To test the functionality of Set.py, Set objects were created and used for testing all functions in Set.py.

- add fucntion: There were two Set objects and one Set has just one extra element than another; the remain elements are the same. The add function was tested by checking if the Set was equal to the another Set after it added that extra element.
- rm fucntion: There were two Set objects and one Set has just one extra element than another; the remain elements are the same. The rm function was tested by checking if the Set was equal to the another Set after it remove that extra element.
- member function: There was one Set objects and an integer which was an element of that Set. The member function was tested by checking if that integer was in the Set.

- size function: There was one Set and an integer that was same as the number of elements in the Set. The size function was tested by checking if the size of that Set was equal to that integer.

- equals function: There were two Set objects that contained exact same elements. The test was implemented by using the equals method to check if the two Sets are equal.

- to_seq function: There were a Set object and a list which contained the same elements as the Set. The test was implemented by checking if the result of Set applied to_seq method was equal to the list.

- result: Four tests failed. The test for size method passed.

- reason: The implementation of equals method that was used to compare Set objects was wrong. The equals method was implemented by checking the size and if every element contained in the Sets are the same; and to check if every element in this Set is also in that Set, the "in" expression was used then the elements were compared by pointers not the values, which caused the inaccuracy of the function.

Test for MoleculeT.py:

- To test the functions in MoleculeT.py, several MoleculeT objects were created and the functions were tested by comparing the equality of the functions output values and the expected output values.

- result: All tests passed.

Test for CompoundT.py:

- To test the functions in CompoundT.py, several CompoundT objects were created and the functions were tested by comparing the equality of the functions output values and the expected output values.

- result: All tests passed.

Test for ReactionT.py:

- To test the functions in ReactionT.py, several ReactionT objects were created and the functions were tested by comparing the equality of the functions output values and the expected output values.

- result: All tests passed.

2 Results of Testing Partner's Code

The same result as I tested my code. All test passed except the add, rm, member, equals, to_seq methods. The reason is that the partner implemented the equals method and member method using "in" which is in the same way I did. This caused the comparison of pointers in the tests but not values.

Compared with this section in A1 and in A2, I find that pytest is convinience for code testing in showing the number of tests passed and showing the number of tests failed because the result of pytest is just one line showing the combination of dots "."(passed test) and "F"(failed tested). Hard coding for tests in A1 takes more time in implementing the tests but the result will show a string on the screen indicated which tests passed and which tests failed; this is more convinent for find out and keep track of the failed tests.

3 Critique of Given Design Specification

Advantages:

- Consistency. The name conventions in the design are descriptive. The names of modules and functions represents the semantics well; for example, MoleculeT used for representing the chemical molecules and the add function adds and element in a Set.
- Essentiality. The specifications for modules and functions are neat and they don't have any unnecessary features and they don't have redundant features.
- Generality. The design specification is not general, it can predict how the module will be used; this benifits for actual implementation of the coding of the design.
- Minimality. Most of the modules are not minimal. Minimality actually cause heavy work in coding so not minimal modules are easier to implement.
- High Cohesion. The components in the modules are closely related so one module can use the component in other modules which is convienent for implementation because we don't need to code for the components again.

Disadvantages:

- Consistency. There are a few exception handling in the design specification, but not thorough enough. My opinions are that every functions that has parameters should have a type check since variables in python can be any types.

- Hign Coupling. The modules in the desig are strongly dependent on other modules so if one module goes wrong, other modules which depend on this module will go wrong as well, which is not good for debugging.

- Opacity. Because the design is hign coupling, the modules are highly related to each other. If there are changes in one module, the other modules which use this module need to be changed. Then there will be a large amount of changes, which is a drawback for maintenance.

The interface does not provide the programmer with checks that will allow them to avoid generation an exception because interface is used for showing the syntax of the functions not the semantic. Therefore, there won't be any checks for exception in the interface.

4 Answers

a) Advantages: natural language is easy to read and understand; formal specification is good for communication between people from different area because it uses a particular academic expression, anybody who has learned this expression can read and the formal specification has no ambiguous and the formal expression is neat.
Disvantages: natural language has long expression and causes more time to read, moreover, natural language sometimes has ambiguous statement which is bad for communication; formal specification is hard to read and understand.

b) ConvertElement: Read through every character in the string and find the element that the string represent and return the element.

ConvertMolecule/ConvertCompound: Read through every character of the string and extract the elements and numbers; take the molecule H2O as example, "H" represents the element H and the number right after it represents the nuber of element H, and "O" represents the element O; convert those to elements and numbers repectively and use the original module to construct a molecule/compound by those elements and numbers and return.

ConvertReaction: Read the string and split the string by "=", the left string represents the left side of reaction and the right string represents the right side of reaction. Then split the left/right string by "+" and the strings will represents the molecules or compounds and use the ConvertMolecule/ConvertCompound to convert each of them. Then use the original module to construct a reaction and return.

c) In the ElementT we assign each element with a number which represents its mass. Then in the MoleculeT and CompoundT we add an instance variable "mass" and implement a function to calculate the mass and return.

d) The coefficients in chemistry use integers instead of real numbers. To turn these float coefficients into integers, we can implement a function. First we iterate throught the list of coefficients and for each float number we use 1 divided by the float number to get its denominator. Put the denominators in a list and find the least common multiple of those numbers. Then multiple each members in the coefficients list by the least common multiple. Here is the link for finding the least common multiple: https://www.geeksforgeeks.org/finding-lcm-two-array-numbers-without-using-gcd/

e) In a statically typed language, every variable is bounded to a type or to an object; once the variable is bounded to a type, it can be bounded only to the objects of that type. In dynamically typed language, every variable won't be bounded to a type, it can be any types.

   Advantages: static typing is better for reading and understanding the code, and it also benifits for reducing errors; dynamic typing is easier for writing the code and it's more tolerant to change. Disavantages: in static typing the program will occur more type errors and it is less tolerant to change; in dynamic typing it is easy to mess up the types between variables because there is no declaration of variables for you to read and figure out.

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

g) map(sum, [1 for x in list])

h) Interface is like syntax; it gives us the structure on how the module will be built. Implementation is like semantic; it gives us the information about what the module does.

i) i

   i) Abstraction guides the design of a module's interface by the process of focusing on what is important while ignoring what is irrelevant in the interface, and abstraction produces a model of an entity in which the irrelevant details of the entity are left out in the interface.

   ii) Anticipation of change guides the design of module's interface by information hiding in the interface.

iii) Generality guides the design of module's interface by solving a more general problem than the problem at hand whenever possible in the interface.

iv) Modularity guides the design of module's interface by applying that different parts of the system are considered separately through the interface and the parts of the system are considered separately from their composition by interface.

v) Separation of concerns guides the design of module's interface by isolating and considering different concerns separately in different modules through the interface.

E Code for ChemTypes.py

```
## @file ChemTypes.py
#  @author Senni Tan
#  @brief Definition of types used for representing chemistry elements
#  @date Feb 6, 2020

from enum import Enum, auto


## @brief Enumerated element type
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 Senni Tan
#  @brief Module that implement the interface of two functions
#  @date Feb 6,2020

from ChemTypes import *
from ElmSet import *
from abc import ABC, abstractmethod


## @brief An interface that has two functions: num_atoms and constit_elms
class ChemEntity(ABC):

    ## @brief Interface of function num_atoms
    @abstractmethod
    def num_atoms(self, element):
        pass

    ## @brief Interface of function constit_elms
    @abstractmethod
    def constit_elems(self):
        pass
```

G Code for Equality.py

```
## @file Equality.py
#  @author Senni Tan
#  @brief Module that implements an interface of a function equals
#  @date Feb 6, 2020

from abc import ABC, abstractmethod


## @brief An interface that contains the syntax of a function
class Equality(ABC):

    ## @ brief Interface of function equals
    @abstractmethod
    def equals(self, T):
        pass
```

H Code for Set.py

```
## @file Set.py
#  @author Senni Tan
#  @brief Module that creates a generic set
#  @date Feb 6, 2020

from Equality import *


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

    ## @brief Set constructor
    #  @details Put the elements of the input set or sequence into a
    #  new Set
    #  @param ss a sequence or a set of elements of some type
    def __init__(self, ss):
        self.s = ss.copy()

    ## @brief A function that adds an element into the Set
    #  @param e an input element
    def add(self, e):
        self.s = list(self.s)
        self.s.append(e)

    ## @brief A function that removes an element from the Set
    #  @details Raise ValueError if the element is not in the Set
    #  @param e an input element
    def rm(self, e):
        if e not in self.s:
            raise ValueError
        self.s.remove(e)

    ## @brief A function that checks if the given element is in the
    #  Set
    #  @param e an input element
    #  @return True if the given element is in the Set, false otherwise
    def member(self, e):
        return e in self.s

    ## @brief A function that determines how many elements in the Set
    #  @return The number of elements in the Set
    def size(self):
        num = 0
        for i in self.s:
            num += 1
        return num

    ## @brief A function that determines if the current Set object is
    #  equal to the given Set object
    #  @param R the given Set
    #  @return True if two Sets are in the same size and contains the
    #  same elements, false otherwise
    def equals(self, R):
        if not self.size() == Set(R).size():
            return False
        for i in list(self.s):
            if not Set(R).member(i):
                return False
        return True

    ## @brief A function that turns the current Set into a sequence
    #  return A set that is turned from Set
    def to_seq(self):
        return list(self.s)


'''test
s = Set({1,2,3,4})
print(s.s)
s.rm(4)
print(s.s)
s.add(5)
print(s.s)
print(s.member(1))
print(s.member(6))
n = s.size()
```

```
print(n)
s2 = Set([1,2,3,4])
s2.add(5)
s2.rm(4)
print(s2.s)
s3 = Set([1,2,3,5])
print(s.equals(s2.s))
print(s.equals(s3.s))
s = Set({1,2,3,4})
print(s.s)
s = s.to_seq()
print(s)
s3.add(6)
'''
```

I Code for ElmSet.py

```
## @file ElemSet.py
#  @author Senni Tan
#  @brief A module that implements a set of type ElementT
#  @date Feb 8, 2020

from ChemTypes import *
from Set import *


## @brief An abstract data type that inherit everything from Set but
#  will be used particularly on type ElementT
class ElmSet(Set):

    pass


'''test
a = ElmSet([ElementT.H, ElementT.He, ElementT.Li, ElementT.Be, ElementT.B])
b = ElmSet([ElementT.C, ElementT.N, ElementT.O, ElementT.F, ElementT.Ne])
c = ElmSet([ElementT.H, ElementT.He, ElementT.Li, ElementT.Be, ElementT.B])
d = ElmSet([ElementT.C, ElementT.N, ElementT.O, ElementT.F, ElementT.Ne])
print(a.s)
print(b.s)
print(a.equals(c.s))
a.rm(ElementT.B)
print(a.equals(c.s))
print(a.s)
a.add(ElementT.B)
print(a.equals(c.s))
print(a.s)
print(b.member(ElementT.Ne))
print(b.member(ElementT.V))
print(a.size())
a.add(1)
print(a.size())
print(a.s)
print(a.to_seq())
'''
```

J Code for MolecSet.py

```
## @file MolecSet.py
#  @author Senni Tan
#  @brief Module that implements a set of type MoleculeT
#  @date Feb 8, 2020

from MoleculeT import *
from Set import *


## @brief An abstract data type that inherit everything from Set but
#  will be used particularly on type MoleculeT
class MolecSet(Set):

    pass


'''test
H2 = MoleculeT(2, ElementT.H)
O2 = MoleculeT(2, ElementT.O)
N2 = MoleculeT(2, ElementT.N)
s1 = MolecSet([H2])
print(s1.s)
print(s1.s[0].get_elm())
print(s1.s[0].get_num())
print(s1.s[0].num_atoms(ElementT.H))
print(s1.s[0].num_atoms(ElementT.O))
s1.add(O2)
print(s1.s)
print(s1.s[1].get_elm())
print(s1.s[1].get_num())
print(s1.s[1].num_atoms(ElementT.H))
print(s1.s[1].num_atoms(ElementT.O))
print(s1.member(O2))
print(s1.member(MoleculeT(2, ElementT.O))) #false
s1.rm(O2)
print(s1.member(O2))
print(s1.size())
s1 = s1.to_seq()
print(s1)
'''
```

K Code for CompoundT.py

```
## @file CompoundT.py
#  @author Senni Tan
#  @brief Module that implements a type of CompoundT
#  @date Feb 8, 2020

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


## @brief An abstract data type that represents a compound
class CompoundT(ChemEntity, Equality):

    ## @brief CompoundT constructor
    #  @details Construct a compound with a given set of molecules
    #  @param M the given set of molecules
    def __init__(self, M):
        self.molec_set = M

    ## @brief The getter method of the class CompoundT
    #  @return The set of molecules contained in the compound
    def get_molec_set(self):
        return self.molec_set

    ## @brief Determine the number of the given element contained in
    #  the compound
    ## @param e the element that needs to be checked
    ## @return The number of the given element contained in the compound
    def num_atoms(self, e):
        num = 0
        for molecule in self.molec_set.to_seq():
            num += molecule.num_atoms(e)
        return num

    ## @brief Determine the elements contained in the compound
    #  @return The set of elements contained in the compound
    def constit_elems(self):
        s = ElmSet([])
        for m in self.get_molec_set().to_seq():
            s.add(m.get_elm())
        return s

    ## @brief Determine if the given compound is equal to this one
    #  @param D the given compound
    #  @return True if two compound contain the same set of molecules,
    #  false otherwise
    def equals(self, D):
        this = self.molec_set
        that = D.get_molec_set()
        if not this.size() == that.size():
            return False
        for i in list(this.s):
            if not that.member(i):
                return False
        return True


'''test
O = MoleculeT(1, ElementT.O)
H2 = MoleculeT(2, ElementT.H)
a = MoleculeT(4, ElementT.H)
H2O = CompoundT(MolecSet([H2, O]))
print(H2O.get_molec_set())
print(H2O.num_atoms(ElementT.H))
print(H2O.num_atoms(ElementT.O))
print(H2O.num_atoms(ElementT.N))
e = H2O.constit_elems()
e = e.to_seq()
print(e)
o = MoleculeT(1, ElementT.O)
h2 = MoleculeT(2, ElementT.H)
h2o = CompoundT(MolecSet([h2, o]))
print(h2o.get_molec_set())
print(h2o.equals(h2o))
```

```
print(h2o.equals(H2O)) #false
'''
```

L Code for ReactionT.py

```
## @file ReactionT.py
#  @author Senni Tan
#  @brief Module that implements a type of ReactionT
#  @date Feb 8, 2020

from ChemTypes import *
from CompoundT import *
from sympy import *
from sympy.solvers.solveset import linsolve


## @brief A local function that determines if every element in a
#  set is positive
#  @param s the given set that needs to be checked
#  @return True if all elements are positive, false otherwise
def pos(s):
    for i in s:
        if i <= 0:
            return False
    return True


## @brief A local function that determines the number of atom 'e'
#  in a given chemical reaction
#  @param C the given sequence of compounds which represents
#  a chemical reaction
#  @param c the given set of coefficients
#  @param e the given element that needs to be checked
#  @return The number of atoms of element 'e' in the reaction
def n_atoms(C, c, e):
    length = len(C)
    num = 0
    for i in range(length):
        num += c[i] * C[i].num_atoms(e)
    return num


## @brief A local function that determines the elements in the
#  chemical equation
#  @param C     the given sequence of compounds which represents
#  a chemical reactiona
#  @return The set of elements in the chemical equation
def elm_in_chem_eq(C):
    s = []
    for c in C:
        a = c.constit_elems().s
        for i in a:
            s.append(i)
    return s


## @brief A local function that determines if the given element
#  is balanced in the given chemical equation with      the given
#  coefficients
#  @param L sequence of CompoundT that represents the compounds of
#  left side of the chemical equation
#  @param R sequence of Compound that represents the compounds of
#  right side of the chemical equation
#  @param cL sequence of number that represents the coefficients
#  of the left side of the chemical equation
#  @param cR sequence of number that represents the coefficients
#  of the left side of the chemical equation
#  @param e the given element
#  @return True if the given elements is balanced in the given
#  chemical equaiton, false otherwise
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 A local function that determines if the given chemical
#  equation is balanced with the given compounds and coefficients
#  @param L sequence of CompoundT that represents the compounds of
#  left side of the chemical equation
#  @param R sequence of Compound that represents the compounds of
```

```
#  right side of the chemical equation
#  @param cL sequence of number that represents the coefficients
#  of the left side of the chemical equation
#  @param cR sequence of number that represents the coefficients
#  of the left side of the chemical equation
#  @return True if the given chemical equation is balanced, false
#  otherwise
def is_balanced(L, R, cL, cR):
    if not elm_in_chem_eq(L) == elm_in_chem_eq(R):
        return False
    for e in elm_in_chem_eq(L):
        if not is_bal_elm(L, R, cL, cR, e):
            return False
    return True


## @brief A local function that turns the result into a list
#  @details The linsolve function of the sympy library is used
#  in this program to solve a matrix. The default result will be a
#  tuple and the result needs to be turned in a list
#  @param r the tuple of the result
#  @return The list of the result
def result_into_list(r):
    r = list(r)
    r = list(r[0])
    return r


## @brief A local function that turns the result into a list of
#  number
#  @details The linsolve function of the sympy library is used
#  in this program to solve a matrix. The default result will be a
#  relation between symbols and the result needs to be turned into
#  numbers
#  @param result the list of the result
#  @return The list of the result turned into numbers
def result_into_coeff(result, sym):
    r = []
    for i in result:
        i = i.subs(sym, 1)
        r.append(i)
    return r


## @brief A local function that determine the greatest common factor
#  of two numbers
#  @param a one of the two given numbers
#  @param b one of the two given numbers
#  @return the greatest common factor of two given numbers
def gcd(a, b):
    while b > 0:
        a, b = b, a % b
    return a


## @brief A local function that determine the greatest common factor
#  of a list of numbers
#  @param a the given list of numbers
#  @return the greatest common factor of all numbers in the list
def gcd_of_list(a):
    result = a[0]
    for i in a[1:]:
        result = gcd(result, i)
    return result


## @brief A local function that determine the factorial of 100
#  @The number of factorial of 100
def factorial_100():
    n = 100
    r = 1
    while n > 0:
        r *= n
        n -= 1
    return r


## @brief A local function that turns a list of numbers that
#  represents the coefficients by fractions into representing
#  by integers
```

```
#  @param r the given list of numbers
#  @return The list of numbers that represents the coefficients by
#  integer
def frac_into_int(r):
    fac100 = factorial_100()
    result = []
    for i in r:
        num = i * fac100
        result.append(num)
    divisor = gcd_of_list(result)
    new = []
    for i in result:
        num = i // divisor
        new.append(num)
    return new


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

    ## @brief The constructor of the ReactionT
    #  @param l the sequence of compoundT that represents the compounds
    #  on the left side of the chemical equation
    #  @param r the sequence of compoundT that represents the compounds
    #  on the righr side of the chemical equation
    def __init__(self, l, r):
        self.coeffL = []
        self.coeffR = []
        if not elm_in_chem_eq(l) == elm_in_chem_eq(r):
            raise ValueError
        elements = elm_in_chem_eq(l)
        # construct symbols
        tau0 = symbols('tau0')

        # construct matrix of left side elements
        left_matrix = []
        for e in elements:
            elm_matrix_row = []
            for compound in l:
                a = compound.num_atoms(e)
                elm_matrix_row.append(a)
            left_matrix.append(elm_matrix_row)
        left_len = len(left_matrix[0])
        # construct matrix of right side elements
        right_matrix = []
        for e in elements:
            elm_matrix_row = []
            for compound in r:
                a = compound.num_atoms(e)
                elm_matrix_row.append(a)
            right_matrix.append(elm_matrix_row)
        # right matrix times -1
        for row in right_matrix:
            for i in range(len(row)):
                row[i] *= -1
        # join the right matrix and left matrix
        matrix = []
        for i in range(len(elements)):
            combined_row = left_matrix[i] + right_matrix[i]
            matrix.append(combined_row)
        # put an zero at the end of each row
        for i in range(len(elements)):
            matrix[i].append(0)
        # solve for matrix
        result = linsolve(Matrix(matrix))
        # turn result in to a list
        result = result_into_list(result)
        result = result_into_coeff(result, tau0)
        # get rid of fraction
        result = frac_into_int(result)
        # put result in to left coefficient and right coefficient
        for i in range(left_len):
            self.coeffL.append(result[i])
        slice_r = result[left_len:]
        for i in slice_r:
            self.coeffR.append(i)
        # check value error for coefficients
        if not pos(self.coeffL):
            raise ValueError
        if not pos(self.coeffR):
```

```
            raise ValueError
        if not is_balanced(l, r, self.coeffL, self.coeffR):
            raise ValueError

        self.lhs = l.copy()
        self.rhs = r.copy()

    ## @brief The getter method of class ReactionT
    #  @return The list of compounds on the left side of the equation
    def get_lhs(self):
        return self.lhs

    ## @brief The getter method of class ReactionT
    #  @return The list of compounds on the rignt side of the equation
    def get_rhs(self):
        return self.rhs

    ## @brief The getter method of class ReactionT
    #  @return The list of coefficients on the left side of the equation
    def get_lhs_coeff(self):
        return self.coeffL

    ## @brief The getter method of class ReactionT
    #  @return The list of coefficients on the right side of the
    #  equation
    def get_rhs_coeff(self):
        return self.coeffR
```

M Code for test_All.py

```
## @file test_All.py
#  @author Senni Tan
#  @brief Tests implementation of python files Set.py, MoleculeT.py,
#  CompoundT.py and ReactionT.py
#  @date Feb 8, 2020

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


## @brief Tests methods from Set.py
class TestSet:

    ## @brief Methods to test the add method of the Set class
    def test_add(self):
        nums = Set([1, 2, 3, 4])
        nums.add(5)
        assert nums.equals(Set([1, 2, 3, 4, 5]).s)

    ## @brief Methods to test the rm method of the Set class
    def test_rm(self):
        s1 = Set([1, 2, 3, 4])
        expect = Set([1, 2, 3])
        s1.rm(4)
        assert s1.equals(expect.s)

    ## @brief Methods to test the member method of the Set class
    def test_member(self):
        s = Set([1, 2, 3, 4])
        assert s.member(1)

    ## @brief Methods to test the size method of the Set class
    def test_size(self):
        s = Set([1, 2, 3, 4])
        size1 = s.size()
        assert size1 == 4

    ## @brief Methods to test the equals method of the Set class
    def test_equals(self):
        s1 = Set([1, 2, 3, 4])
        s2 = Set([1, 2, 3, 4])
        assert s1.equals(s2.s)

    ## @brief Methods to test the to_seq method of the Set class
    def test_to_seq(self):
        s1 = Set([1, 2, 3, 4])
        s1 = s1.to_seq()
        assert s1 == [1, 2, 3, 4]


## @brief Tests methods from MoleculeT.py
class TestMoleculeT:

    ## @brief Methods to test the get_num method of the MoleculeT class
    def test_get_num(self):
        h2 = MoleculeT(2, ElementT)
        num = h2.get_num()
        assert num == 2

    ## @brief Methods to test the get_elm method of the MoleculeT class
    def test_get_elm(self):
        h2 = MoleculeT(2, ElementT.H)
        elm = h2.get_elm()
        assert elm == ElementT.H

    ## @brief Methods to test the num_atoms method of the MoleculeT class
    def test_num_atoms(self):
        h2 = MoleculeT(2, ElementT.H)
```

```
        n1 = h2.num_atoms(ElementT.H)
        n2 = h2.num_atoms(ElementT.O)
        assert n1 == 2
        assert n2 == 0

    ## @brief Methods to test the constit_elems method of the MoleculeT class
    def test_constit_elems(self):
        h2 = MoleculeT(2, ElementT.H)
        s = h2.constit_elems()
        assert s.s[0] == ElementT.H

    ## @brief Methods to test the equals method of the MoleculeT class
    def test_equals(self):
        h2 = MoleculeT(2, ElementT.H)
        h2_2 = MoleculeT(2, ElementT.H)
        assert h2.equals(h2)
        assert h2.equals(h2_2)


## @brief Tests methods from CompoundT.py
class TestCompoundT:

    ## @brief Methods to test the get_num method of the CompoundT class
    def test_get_molec_set(self):
        oxy = MoleculeT(1, ElementT.O)
        h2 = MoleculeT(2, ElementT.H)
        h2o = CompoundT(MolecSet([h2, oxy]))
        s = h2o.get_molec_set().to_seq()
        assert s[0].get_elm() == ElementT.H
        assert s[0].get_num() == 2
        assert s[1].get_elm() == ElementT.O
        assert s[1].get_num() == 1

    ## @brief Methods to test the num_atoms method of the CompoundT class
    def test_num_atoms(self):
        oxy = MoleculeT(1, ElementT.O)
        h2 = MoleculeT(2, ElementT.H)
        h2o = CompoundT(MolecSet([h2, oxy]))
        h_num = h2o.num_atoms(ElementT.H)
        o_num = h2o.num_atoms(ElementT.O)
        assert h_num == 2
        assert o_num == 1

    ## @brief Methods to test the constit_elms method of the CompoundT class
    def test_constit_elems(self):
        O = MoleculeT(1, ElementT.O)
        H2 = MoleculeT(2, ElementT.H)
        H2O = CompoundT(MolecSet([H2, O]))
        s = H2O.constit_elems()
        assert s.s == [ElementT.H, ElementT.O]

    ## @brief Methods to test the equals method of the CompoundT class
    def test_equals(self):
        O = MoleculeT(1, ElementT.O)
        H2 = MoleculeT(2, ElementT.H)
        H2O = CompoundT(MolecSet([H2, O]))
        o = MoleculeT(1, ElementT.O)
        h2 = MoleculeT(2, ElementT.H)
        h2o = CompoundT(MolecSet([h2, o]))
        assert H2O.equals(H2O)
        assert not H2O.equals(h2o)


## @brief Tests methods from ReactionT.py
class TestReactionT:

    ## @brief Methods to test the get_lhs method of the ReactionT class
    def test_get_lhs(self):
        H2 = MoleculeT(2, ElementT.H)
        O2 = MoleculeT(2, ElementT.O)
        O = MoleculeT(1, ElementT.O)
        H2O = CompoundT(MolecSet([H2, O]))
        H2 = CompoundT(MolecSet([H2]))
        O2 = CompoundT(MolecSet([O2]))
        L = [H2, O2]
        R = [H2O]
        reaction = ReactionT(L, R)
        l = reaction.get_lhs()
        l = l[0].get_molec_set().s
        l = l[0]
```

```
        assert l.get_elm() == ElementT.H
        assert l.get_num() == 2

    ## @brief Methods to test the get_rhs mehtod of the ReactionT class
    def test_get_rhs(self):
        H2 = MoleculeT(2, ElementT.H)
        O2 = MoleculeT(2, ElementT.O)
        O = MoleculeT(1, ElementT.O)
        H2O = CompoundT(MolecSet([H2, O]))
        H2 = CompoundT(MolecSet([H2]))
        O2 = CompoundT(MolecSet([O2]))
        L = [H2, O2]
        R = [H2O]
        reaction = ReactionT(L, R)
        r = reaction.get_rhs()
        r = r[0].get_molec_set().s
        r1 = r[0]
        r2 = r[1]
        assert r1.get_elm() == ElementT.H
        assert r1.get_num() == 2
        assert r2.get_elm() == ElementT.O
        assert r2.get_num() == 1

    ## @brief Methods to test the get_lhs_coeff method of the
    #  ReactionT class
    def test_lhs_coeff(self):
        H2 = MoleculeT(2, ElementT.H)
        O2 = MoleculeT(2, ElementT.O)
        O = MoleculeT(1, ElementT.O)
        H2O = CompoundT(MolecSet([H2, O]))
        H2 = CompoundT(MolecSet([H2]))
        O2 = CompoundT(MolecSet([O2]))
        L = [H2, O2]
        R = [H2O]
        reaction = ReactionT(L, R)
        cL = reaction.get_lhs_coeff()
        assert cL[0] == 2
        assert cL[1] == 1

    ## @brief Methods to test the get_rhs_coeff method of the ReactionT class
    def test_rhs_coeff(self):
        H2 = MoleculeT(2, ElementT.H)
        O2 = MoleculeT(2, ElementT.O)
        O = MoleculeT(1, ElementT.O)
        H2O = CompoundT(MolecSet([H2, O]))
        H2 = CompoundT(MolecSet([H2]))
        O2 = CompoundT(MolecSet([O2]))
        L = [H2, O2]
        R = [H2O]
        reaction = ReactionT(L, R)
        cR = reaction.get_rhs_coeff()
        assert cR[0] == 2
```

N Code for Partner's Set.py

```
## @file Set.py
#  @author Benjamin Kostiuk
#  @brief Module that defines the Set ADT
#  @details Assumes that the Set constructor is called for each object instance
#           before any other access methods are called.
#  @date 02/01/2020

from Equality import Equality


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

    ## @brief Set constructor
    #  @details Initializes a Set object whose state consists of a set of elements
    #  @param s Sequence of elements with which to initialize the Set
    def __init__(self, s):
        self.S = set(s)

    ## @brief Add an element to the set
    #  @param Element to be added to the set
    def add(self, e):
        self.S = self.S.union({e})

    ## @brief Remove an element from the set
    #  @param e Element to be removed from the set
    #  @throws ValueError if element to be removed cannot be found in the set
    def rm(self, e):
        if not self.member(e):
            raise ValueError("Cannot remove element not found in set.")
        self.S = self.S.difference({e})

    ## @brief Determine whether an element is in the set
    #  @param e Element to check whether in the set
    #  @return True if the element is in the set, otherwise false
    def member(self, e):
        return e in self.S

    ## @brief Get the size of the set
    #  @return The size of the set
    def size(self):
        return len(self.S)

    ## @brief Determine if the Set is equal to another set
    #  @details A Set is considered equal if all elements in one set are in another
    #  @param r Set to compare with
    #  @return True if the two Sets are equal, otherwise false
    def equals(self, r):
        if self.size() != r.size():
            return False

        for element in self.S:
            if not r.member(element):
                return False
        return True

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

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

O Code for Partner's MoleculeT.py

```
## @file MoleculeT.py
#  @author Benjamin Kostiuk
#  @brief Module defines the MoleculeT ADT for molecule representation
#  @date 02/01/2020

from Equality import Equality
from ChemEntity import *


## @brief An abstract data type that represents a molecule
class MoleculeT(ChemEntity, Equality):

    ## @brief MoleculeT constructor
    #  @details Initializes a MoleculeT object whose state consists of
    #           an ElementT and the number of that element in the molecule
    #  @param m Number of the ElementT in molecule
    #  @param e ElementT in the molecule
    def __init__(self, n, e):
        self.num = n
        self.elm = e

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

    ## @brief Get the ElementT in the molecule
    #  @return The ElementT in the molecule
    def get_elm(self):
        return self.elm

    ## @brief Get the number of atoms of a given ElementT in the molecule
    #  @param e ElementT to check for in molecule
    #  @return The number of atoms of the specified ElementT in the molecule
    def num_atoms(self, e):
        if self.elm == e:
            return self.num
        return 0

    ## @brief Return an ElmSet of the ElementT in the molecule
    #  @return An ElmSet of the ElementT in the molecule
    def constit_elems(self):
        return ElmSet([self.elm])

    ## @brief Determine if the molecule is equal to another molecule
    #  @details Two molecules are considered equal if they are composed of the same
    #           ElementT and have the same number of atoms.
    #  @param m MoleculeT to compare with
    #  @return True if the molecules are equal, otherwise false
    def equals(self, m):
        return self.elm == m.get_elm() and self.num == m.get_num()

    def __eq__(self, value):
        return self.equals(value)

    def __hash__(self):
        return hash(str(self.num) + str(self.elm))
```

P Code for Partner's CompoundT.py

```
## @file CompoundT.py
#  @author Benjamin Kostiuk
#  @brief Module defines the CompoundT ADT for chemical compound representation
#  @date 02/01/2020

from MoleculeT import *
from MolecSet import *


## @brief An abstract data type that represents a chemical compound
class CompoundT(ChemEntity, Equality):

    ## @brief CompoundT constructor
    #  @details Initializes a CompoundT object whose state consists of a MolecSet
    #  @param m MolecSet of molecules in the chemical compound
    def __init__(self, m):
        self.C = m

    ## @brief Get the MolecSet of molecules in the chemical compound
    #  @return The MolecSet of molecules in the chemical compound
    def get_molec_set(self):
        return self.C

    ## @brief Get the number of atoms of a given ElementT in the chemical compound
    #  @param e ElementT to check for in chemical compound
    #  @return The number of atoms of the specified ElementT in the chemical compound
    def num_atoms(self, e):
        count = 0
        for m in self.C.to_seq():
            count += m.num_atoms(e)
        return count

    ## @brief Return an ElmSet of the ElementTs in the chemical compound
    #  @return An ElmSet of the ElementTs in the chemical compound
    def constit_elems(self):
        return ElmSet([m.get_elm() for m in self.C.to_seq()])

    ## @brief Determine if the chemical compound is equal to another chemical compound
    #  @details Two chemical compounds are considered equal if they have
    #           all the same molecules in them
    #  @param d CompoundT to compare with
    #  @return True if the chemical compounds are equal, otherwise false
    def equals(self, d):
        return self.C.equals(d.get_molec_set())

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

Q Code for Partner's ReactionT.py

```
## @file ReactionT.py
#  @author Benjamin Kostiuk
#  @brief Module defines the ReactionT ADT for representing checmical reactions
#  @date 02/05/2020

from CompoundT import *

import numpy as np
from sympy import Matrix, lcm


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

    ## @brief ReactionT constructor
    #  @details Initializes a ReactionT object whose state consists of a sequence of
    #           reactants a sequence of its coefficients, a sequence of products
    #           and a sequence of its coefficients. The sequences of coefficients
    #           are computed as to balance the chemical reaction with an equal
    #           number of elements on both sides.
    #  @param reactants Sequence of CompoundT in the left-hand side of the chemical
    #                   reaction, known as reactants
    #  @param products Sequence of CompoundT in the right-hand side of the chemical
    #                  reaction, known as products
    #  @throws ValueError if the elements in the reactants do not match the elements
    #                     in the products, the two sides of the reaction cannot be
    #                     balanced, any of coefficients are non-positive or if the
    #                     the sequences of coefficients do not match their
    #                     respective side of the chemical reaction
    def __init__(self, reactants, products):
        # Get ElmSet of ElementTs in L and R
        lhs_elems = self.__elements_in_equation__(reactants)
        rhs_elems = self.__elements_in_equation__(products)

        # Check that elements in the reactants and the products are the same
        if not lhs_elems.equals(rhs_elems):
            raise ValueError("Elements in reactants must match elements in products.")

        # Get coefficient matrix to solve linear equation
        lhs_coeffs, rhs_coeffs = self.__solve_matrix__(reactants, products, lhs_elems)

        # Check if length of lists match
        if len(lhs_coeffs) != len(reactants) or len(rhs_coeffs) != len(products):
            raise ValueError("Cannot match coefficients to reactants and products.")

        # Check if coefficients are balanced
        for element in lhs_elems.to_seq():
            if not self.__is_balanced__(reactants, products, lhs_coeffs, rhs_coeffs, element):
                raise ValueError("Invalid ReactionT. Reaction cannot be balanced.")

        self.lhs = reactants
        self.rhs = products
        self.coeff_L = lhs_coeffs
        self.coeff_R = rhs_coeffs

    ## @brief Get the sequence of reactants of the chemical reaction
    #  @return The sequence of CompoundT in the left-hand side of the chemical reaction
    def get_lhs(self):
        return self.lhs

    ## @brief Get the sequence of products of the chemical reaction
    #  @return The sequence of CompoundT in the right-hand side of the chemical reaction
    def get_rhs(self):
        return self.rhs

    ## @brief Get the sequence of coefficients in the left-hand side of the chemical reaction
    #  @return The sequence of coefficients in the left-hand side of the chemical reaction
    def get_lhs_coeff(self):
        return self.coeff_L

    ## @brief Get the sequence of coefficients in the right-hand side of the chemical reaction
    #  @return The sequence of coefficients in the right-hand side of the chemical reaction
    def get_rhs_coeff(self):
        return self.coeff_R

    # Returns an ElmSet of ElementT in a list of CompoundTs
```

```
def __elements_in_equation__(self, equation):
    elems = []
    for compound in equation:
        elems += compound.constit_elems().to_seq()
    return ElmSet(elems)

# Check if a ReactionTs coefficients for reactants and products are balanced
def __is_balanced__(self, reactants, products, left_coeffs, right_coeffs, element):
    lhs_count, rhs_count = 0, 0
    # Count element for left hand side
    for i in range(len(reactants)):
        if left_coeffs[i] <= 0:
            raise ValueError("Invalid ReactionT. Coefficients must be positive.")
        lhs_count += left_coeffs[i] * reactants[i].num_atoms(element)

    # Count element for right hand side
    for i in range(len(products)):
        if right_coeffs[i] <= 0:
            raise ValueError("Invalid reaction. Coefficients must be positive.")
        rhs_count += right_coeffs[i] * products[i].num_atoms(element)

    return lhs_count == rhs_count

# Return right and left coefficients solved from a list of reactants and products
def __solve_matrix__(self, reactants, products, elems):
    # Create a coefficient matrix to solve
    coeff_matrix = []
    for e in elems.to_seq():
        row = [compnd.num_atoms(e) for compnd in reactants]
        row += [-compnd.num_atoms(e) for compnd in products]
        coeff_matrix.append(row)

    # Check if reaction is null
    if(reactants == [] and products == []):
        return [], []
    else:
        # Solve for lhs and rhs coefficients
        # Uses algorithm proposed here:
        #
            https://stackoverflow.com/questions/42637872/solve-system-of-linear-integer-equations-in-python
        matrix = Matrix(coeff_matrix)
        null_vectors = matrix.nullspace()

        if null_vectors == []:
            raise ValueError("Invalid ReactionT. Reaction cannot be balanced.")

        null_vectors = null_vectors[0]
        multiple = lcm([val.q for val in null_vectors])
        x = multiple * null_vectors
        solution = np.array([int(val) for val in x]).tolist()

        lhs_coeffs = solution[:len(reactants)]
        rhs_coeffs = solution[len(reactants):]

    return lhs_coeffs, rhs_coeffs
```