Assignment 2 Solutions

Aditya Sharma

February 16, 2020

This is my report for Assignment 2. The task for this assignment was to develop a Python Program capable of balancing a chemical equation given a design specification.

1 Testing of the Original Program

The number of testing functions is roughly proportional to the number of methods in each class (excluding static methods). Each method has a dedicated test function to testing a particular criteria. Once again, we used PyTest to perform the Unit Testing. Unlike the previous assignment, we were given a formal specification, I leveraged this specification to make the testing process easier and more manageable.

Once again, I approached testing the program with a particular structure. Firstly, I would test the return type of each method to ensure that it matches the specification. Secondly, I would test common cases where the input was as simplistic as possible to ensure that the methods are able to perform basic tasks correctly. Finally, whenever it was applicable, I tested particular edge cases.

In some cases, I wasn't able to test the return type or edge cases. The reason for this is because sometimes the design specification would state that a method would return a "Sequence". In Python, lists, dictionaries and tuples classify as sequences. Thus, there is no fixed type that an output should be. For that reason, I didn't test the output types for methods that returned sequences. In addition, majority of the methods were straightforward with their purpose and potential implementation details. Thus, there often times were not many edge cases to consider.

I do need to stress the fact that a large majority of the methods were getters. Therefore, the standard approach to test the getters is to check that given the correct input, it is able to return the correct value. There are no edge cases when testing getters. There will usually be edge cases when testing a piece of code that requires a complicated component of implementation. For some test functions, I wrote one test method in a particular way that was able to test multiple components. For example: In my test functions, I frequently

compared lists and sets. This is because I am able to test 3 different cases with those types of data types. I am able to ensure that the number of elements within sets/lists are equal, that each element within one set/list exists in another, that the order of the elements between those two are correct.

1.1 Set Testing

1.1.1 Set.add

According to the design specification, the purpose of this method was to add an element to the existing set. When we consider the properties of a set (sequential, no duplicates), we can draw out two cases.

Firstly, if we add an element to a set that didn't exist before, then the element should exist in the set. If the element exists in the set, then the length of the set should increase by 1. Secondly, if we add an element that already exists in the set, then there should not be any addition. This means that the length of the set should remain the same.

- This test case handled the situation where an element that doesn't exist in the set was being added to the set. This ensures that the element is actually added.
- This test case handled the situation where an element that already existed in the set was being added to the set. This checks to ensure that the element doesn't get added because then there will be duplicates and sets can't have duplicates.

1.1.2 Set.rm

According to the design specification, the purpose of this method was to remove a particular element from the set.

- This test case handled the situation where we try removing an item from a set that doesn't exist. This checks to ensure that the correct exception is thrown.

1.1.3 Set.member

According to the design specification, the purpose of this method was to check if an element is a member of the set. In order to be a member, the element must exist within the set. It was a straightforward method and there were only two possibilities. Either an element is a member or it is not a member. Thus, the following two test cases were generated.

- This test case handled the situation where an element that didn't exist within a set had its membership checked. This checks to that false is returned.
- This test case handled the situation where an element was a member and had its membership checked. This test ensures that true is returned.

1.1.4 Set.size

According to the design specification, the purpose of this method was to return a numerical value that signified the size of the set. This had only one test case because the size of a set can only be an integer because of the python programming language.

- This test case handled the situation where the size of a set is checked and the correct actual numerical value is returned.

1.1.5 Set.equals

According to the design specification, the purpose of this method was to determine if two sets are equivalent; if they have the same number of elemens and each element in one set exists in another. However, once again, it was a straightforward request and there don't aren't edge cases.

- This test case handles the situation where two differently sized sets are being compared. This ensures that false is returned.
- This test case handles the situation where two sets composed of different elements are being compared. This ensures that false is returned.
- This test case handles the situation where two sets of the same size composed of the same elements are being compared. This ensures that true is returned.

1.1.6 Set.to_seq

According to the design specification, the purpose of this method was to return a sequence of all the elements from the set. Essentially, when we think about it, when we are returning this sequence, the sequence must have the same length and must be composed of the same elements. Assuming that a list is returned as the sequence.

- This test case handles the situation where a set is turned into a sequence. It checks to ensure all elements are returned, that no additional elements are returned, that the length is also the same as the length of the original set.

1.2 MoleculeT Testing

1.2.1 MoleculeT.get_num

According to the design specification, the purpose of this method is to return the number of elements within the molecule. Since each molecule is composed of an ElementT type, it needs to return the number of those particular elements. Once again, this is a straightforward request and the only way to test this is to ensure that the correct output is returned.

- This test case handles the case where the number of elements from a molecule must be returned. It checks to ensure that the correct number associated with that particular element was returned. By doing this, it also checks to ensure that it is of the correct type (Integer).

1.2.2 MoleculeT.get_elm

This method is similar to the previous method in the fact that its purpose is to return the element that a molecule is composed of. The only test case is to check that the correct molecule was returned. There are no edge cases.

- This test case handles the case where the element from a molecule must be returned. It checks to ensure that the correct element associated with that particular molecule was returned. By doing this, it also checks to ensure that it is of the correct type (ElementT) and not another type.

1.2.3 MoleculeT.num_atoms

According to the design specifications, the purpose of the method is to return the number of atoms for an element within a molecule. It needs to return the correct number. However, what if the element doesn't exist within the molecule. In that case it needs to return 0. In order to ensure that the method works perfectly, I will be testing this functionality.

- This test case handles the situation where the element exists within the molecule and it ensures that the correct number is retrieved.

- This test case handdles the situation where the element does not exist within the molecule and that 0 is retrieved.

1.2.4 MoleculeT.constit_elems

According to the design specifications, the purpose of the method is to return an ElmSet type that represents the element that makes up the molecule. This method is not complicated either. We just need to ensure that the return type is correct and the correct element is returned.

- This test case checks to ensure that the correct ElmSet type was returned.
- This test case checks to ensure that the correct element was returned given any molecule.

1.2.5 MoleculeT.equals_

According to the design specification, the purpose of the method is to ensure that one molecule is equal to another. In order to do this, we need to check that both molecules are composed of the same number and same type of elements.

- This test case checks to ensure that when given two different molecules, false is returned.
- This test case checks to ensure that when given two of the same molecules with different numbers, false is returned.
- This test case checks to ensure that when given two molecules with the same elements and same count, true is returned.

1.3 CompoundT Testing

1.3.1 CompoundT.get_molec_set

According to the design specification, since a CompoundT will be made up of various molecules, we need to return a the molecules. In order to do this and ensure that the correct output is proceeded, I need to check that all the elements from CompoundT are being returned, that the exact same number of values is also being returned.

- This test case checks to ensure that when the function is called, the correct elements and the same number of elemetns are returned. The implementation of this test case allows me to check two things at once.

1.3.2 CompoundT.num_atoms

According to the design specification, given a particular molecule within a compound, I need to return the number of atoms associated with that molecule's element. To ensure that the correct output is produced, I need to check to make sure that the correct number that corresponds to the molecule's element count is returned. In the event that a molecule that doesn't exist within a is specified, I need to check to ensure that the correct fall back is called (return 0).

- This case checks to ensure that the correct output that corresponds to the molecule's element number is produced.
- This case checks to ensure that the correct fallback output (0) is produced when it recieves a element input that doesn't exist within a Compound.

1.3.3 CompoundT.constit_elems

According to the design specification, this method needs to return a an ElmSet sequence composed of all the elements that make up the compound. Therefore, in order to test this method, I need to check to ensure that the correct elements are returned, the correct number of elements are returned and that their return types are the same.

- This case checks to ensure that the correct sequence of ElmSet elements are produced. It checks to ensure that only elements existing within the CompoundT are produced.
- This case checks to ensure that the same number of ElmSet elements are returned, it ensures that no extra elements are being returned.
- This case checks to ensure that the return type are both the same.

1.3.4 CompoundT.equals

According to the specification, the method needs to check if two CompoundT objects are equal. In order to determine equivalence, they need to have the same number molecules, elements and number of atoms for each element.

- This test case checks to ensure that when two compounds that have the same number of molecules, type of elements and number of atoms

- This test case checks to ensure that the implementation can recognize when the number of molecules, element composition and number of atoms for each element is different. I needed to implement this test case because I needed to check to ensure that the implementation handles those cases because those cases would result in a "false" answer.

1.4 ReactionT Testing

1.4.1 ReactionT.get_lhs and ReactionT.get_rhs

Once again, since these methods are both getters, in order to test them, I just need to make sure that the correct output was produced. Since there is no intricacy in the output, I only need to check the values to ensure that they match. There is no possible edge case because of the simplistic nature of the method.

1.4.2 ReactionT.get_lhs_coeff and ReactionT.get_rhs_coeff

According to the specifications, these methods need to return the balanced left and right side of the chemical equations that correspond to the coefficients. These methods do require a certain level of testing.

- This test case checks to ensure that the correct sequence of numbers is produced (coefficients). The reason I checked this was because depending it is entirely possible for an incorrect sequence of numbers to be produced.
- This test case also checks to ensure that the order of the sequence of numbers is correct. Since the order of the numbers matter because each number corresponds to a particular compound's coefficient, I had to ensure that they were correct and they matched.
- Finally, this test case checks to ensure that the correct number of coefficients were retrieved. This is because of the invariant state, the number of coefficients must match the number of compounds.

2 Results of Testing Partner's Code

Summary: My partner was able to pass all 28 of my test cases successfully.

My partner passed all 28 test cases. This came as a surprise because initially I was expecting to see errors and failed test cases similar to Assignment 1. However, when I didn't encounter the errors. I started to reflect and think about the potential reasons

why my partner didn't fail the test cases. I realized that from Assignment 1, only a few variables had changed: assignment difficulty, potential new partner and introduction of design specification. It dawned to me that the core reason for the result was because of the introduction of the design specification. In the previous assignment, we had our requirements stated in a natural language where it was open to introduction. However, in this assignment, everything was formalized and fixed. This reduced the number of assumptions that we had to make and gave us a better understanding of the entire program itself so we would be able to better implement particular methods.

3 Critique of Given Design Specification

In this assignment, I liked how we had a formal design specification provided. I felt like this provided a better understanding of particular tasks and goals that we had to achieve. This reduced the time spent trying to understand the task because the design specification clearly outlined the end goal of each method. The specification of the assignment imposed particular design decisions such as Inheritance. I felt like the introduction of Inheritance really made the overall program more structured, organized and hierarchical in a sense. This made it easier to understand the connection and relationship between modules.

However, I still believe that there can be room for improvement. Personally, I felt like there were too many classes and objects to keep track of which lead to massive confusion in implementing some methods in CompoundT and ReactionT. The fact that we created several objects that inherited from each other made it difficult to understand what exactly was being stored in each object and how we could access the information. For example: A CompoundT object is composed of a MolecSet object which is composed of a set of MoleculeTs which is composed of etc... This continuous application of "Inheritance" made it time consuming because I would have to keep referring to the respective module's template to understand which methods exist and which methods would be applicable. Perhaps in the future, an improvement that could be made would be to limit the "Inheritance" chain.

If I were to change the design of the of the project, I would only change the number of inheritances that it requires. Yes, I agree that it inheriting did enforce minimalism and essentiality, but it made debugging difficult because I found that if I made a mistake in Set.py, the error would translate into MolecSet and ElmSet. This made it difficult to debug at first because I was wondering what was going wrong with my implementation in MolecSet and ElmSet. Then, when I realized that both those objects inherited from Set.py, I was able to find the source of the error. In a sense, inheritances can be good, but for inexperienced developers like myself, it can be troublesome in the beginning.

- The design specification satisfied the quality criteria of being "Consistent". Consis-

tentency is defined as following the same naming conventions, being able to handle exceptions and being able to understand the rest of a program given partial knowledge of the program. The reason I say this is because if I were given the naming conventions, purpose and goal of each method, I'm fairly confident that I would be able to make the program. This is because the software engineering principles discussed in 7I were applied to the design of the project.

- The design specification satisfied the quality criteria of being "Essential". This is because it provided useful features and didn't include any useless features. I believe that this was satisfied because the concept of "Inheritance" was successfully incorporated. This is because we had a few different versions of "Set" such as "MolecSet" and "ElmSet" that leveraged the same methods from "Set". By implementing "Inheritance", we avoided developing the same routines for each of those objects, routines that would be doing the same task as those from "Set". This also something that I liked because we avoided duplicate work and it saved us time.

- The design specification satisfied the quality criteria of "Generality". This is because the specification was not super fine-tuned or specific. It didn't discuss the procedure of solving a very specific problem i.e: Balancing an equation with 3 compounds. Instead, it focused on a general problem of being able to solve all chemical equations. Furthermore, the specification provided routines that allowed the users to have a vast majority of freedom. It provided users with the ability to use particular routines in unique ways and still leave room for expansion. It was a specification that was designed for change. The way the the general tasks were conveyed were a little difficult to interpret because of my inexperience with discrete math; however, it did make the project better because we were able to avoid assumptions and were able to better understand the task at hand.

- The design specification did not satisfy "Minimalism" because it had access routines that offer two different services that are requested seperately by the user. According to Hoffman and Strooper, in order to be Minimal, developers must avoid developing access routines that offer two different services that are requested seperately. Many times within a module, there would be methods that return the length of a seqeuence while another determines if a member exists within a sequence. As you can see, these are completely different services that can be requested by the user. However, I didn't find this to be an issue during my time developing, instead, it just provided me with a greater degree of freedom when programming. I feel like Minimalism and Generality conflict with each other to a degree.

- The design specification satisfied the quality criteria for "High Cohesion" because all the components of the program (modules) were tightly related to each other. Often

times, the output from one module would become the input for another module after some processing. They were used together to finish the tasks at hand. This was a reason why the project was well designed because of how all the pieces fit together. It made it easier to develop and plan our development. However, sometimes, because of how they were so tightly dependent and related, an error from one module would also carry over into an error in the other module. This made it difficult to debug sometimes.

- The design specification did not satisfy the quality criteria of "Low Coupling". Low Coupling is defined as a program's modules not being strongly dependent on other modules. In this case, because "Inheritance" was so frequently used, several of the modules inherited from other modules. As a result, the modules were strongly dependent on other modules. Therefore, the quality criteria of "Low Coupling" was violated to a degree. Sometimes this made it difficult to work with the project because an error from one module would carry over into an error in the other module. This made it a pain to troubleshoot and debug the program. Perhaps in the future, limiting the number of inheritances that occur would be beneficial.

- The design specification did satisfy the quality criteria of being "Opaque". This is because it successfully enforced information hiding. It was able to enforce information hiding because if a change in the implementation occured, the interface itself would not change. This is assuming that the change in the implementation still follows the output requirements of the design specification.

The interface also allows for checks that prevent exception generation. This is because of the State Invariants that are mentioned. We have to develop our programs in such a way that exception checks are avoided. This is something that played into our thinking process when designing the thinking of our implementation in the modules.

4 Answers

a) The natural language specification made it easier to understand the task on the surface level. It gave us a high-level overview of our task and made it easier to understand the end-goal. As a result, the high-level understanding allowed us to quickly begin planning our approach to developing the program. However, the fact that it only gave us a high-level overview meant that several low-level parts were left open to interpretations and there were several assumptions that were made. When I compare this to the formal specification of assignment 2, I realize the difference is enormous. Unlike assignment 1, we were able to still quickly understand the end goal of the

program and we were able to better understand the requirements of each method. This is because the requirements were conveyed in a more specific format and defined format compared to the loose style of natural language. Furthermore, because we had strict requirements to follow, minimal assumptions were made and this resulted in a reduction of errors.

b) The process of converting strings to logical syntactic components is called "Parsing". We would need to add a module that would be responsible for string manipulation and storage. The module's sole purpose would be to perform particular manipulations on strings. Personally, if I were to create the module, I would name it "Parser". Within the module, I would first need to split the string into reactants and products which would be stored in different variables. Afterwards, I would need to split both the reactants and products into their respective compounds. Each compound would be stored in a list. For each compound, I would break each compound into a list of their molecules which would contain a tuple of the element and its count. Afterwards, I would use this information to perform the conversion into ElementT, MoleculeT, MolecSet and CompoundT types.

c) If we wanted to calculate the mass of the elements, molecules and compounds, the first step would be to gather information pertaining to the mass of elements. The next step would be to assign each element their respective masses. This means, I would have to assign each ElementT a particular mass. This could be accomplished by creating a module (ElementMass) that would take an ElementT as an input and would output an ElementT object that contains its mass. With regards to the implementation, I would personally implement a hashtable where I can set ElementT types as keys and their respective masses as values. Upon recieving the input, I would create a new ElementT object that contains its respective mass. The final step would be to implement a module (CalculateMass) that would be responsible for calculating the masses of elements, molecules and compounds. In order to do this, the CalculateMass module would have a minimum of 3 instance methods that would rely on a series of static methods to calculate the masses. Each instance method would take a different input type. This way, we would have 1 method that would calculate the mass of elements, 1 method to calculate the mass of molecules and 1 method to calculate the mass of compounds. Therefore, to quickly summarize, if I wanted to calculate the masses, I would end up creating addditional modules and integrating those modules into my calculations.

d) This differs from the usual conventions in chemistry because in actual chemistry, floats are not valid coefficients. Chemistry only accepts positive integer values as coeffients. I can modify my current algorithm to change the floats to real numbers. I can do this

by finding the LCM of all the numbers and then multiply all the numbers by the LCM. By doing this, I can get the smallest whole number for all the coefficients.

e) Dynamic typing is more concise; we can eliminate writing verbose declarations and typecasing logic. Dynamic typing also allows us to program much faster because we can avoid boilerplate code. However, dynamic typing does not catch type mismatches when they should be caught, this increases the time a developer would spend debugging. Static typing allows us to catch type mismatches much sooner, usually immediately when they occur. This saves the developer lots of time and effort. In addition, static typing improves the organization and maintainaibility of a program.

(i,j) if odd(i) and odd(j) for i in range(1,10) for j in range(i+1,10) –¿ "odd(i)" is a function that returns "True" if "i" is odd. We can check if they are odd by doing i modulus 2 != 0.

f) def summation(listInput): return sum(list(map(add, listInput))) –¿ "add" is a function that results in 1 for all elements in the list

g) An interface is a pathway of sorts between different components of a program that exchange information. It can be thought of as an end point that communicates with another entity to either recieve, give or exchange information. Implementation is the actual execution process/details that results in the completion of a particular task. Implementation is the details of how a particular task was completed or executed, it is the inner logic of a function/ module.

h) In Software Engineering, Separation of Concerns is a principle that promotes making decisions separately. It promotes treating each category as its own subset with its own set of available information and then using that information to make a decision regarding the design of a software product. Modularity is a principle that discusses breaking a complex system into smaller pieces called modules. A program composed of such pieces is said to be modular. This principle guides the design by breaking up larger methods into smaller methods capable of doing simpler tasks. It works hand in hand with Separation of Concerns because it allows you to deal with modules in isolation with information pertaining to only that singular module. Anticipation of Change is a principle that discusses how software products must be designed with the idea for change to occur. This allows for the evolution of software products and also contributes to its maintenance. It guides the design of a module's interface by not containing it to work with super niche data types or by constraining it to perform super specific and unchangeable operations. It guides the design by allowing for a degree of freedom so that change can occur. Abstraction is a principle in which we ignore the particular details and only focus on the important aspects of a particular situation.

Essentially, it is when we look at a situation with an abstract picture. Usually, it is when we look at a particular program considering only its functions and how we can interact/use it. We would ignore its inner workings. It guides a module's interface by letting the purpose define the inner workings and by ignoring details. Generality is a principle that discusses how a general solution is provided to a particular task instead of the specialized solution. It allows us to successfully execute the desired task and sometimes also allows for the user to use it in a creative way. For example: Microsoft word was developed to write notes, but now it can be used to write resumes. It guides the development of the modules interface by making a developer create general solutions and implementations of particular modules.

E Code for ChemTypes.py

```
## @file ChemTypes.py
#  @author Aditya Sharma
#  @brief This file creates a module related to the elements on the periodic table


from enum import Enum, auto

## @brief This class implements a ____. It is a _____.
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 Aditya Sharma
#  @brief The file contains an interface.

from abc import ABC, abstractmethod


##  @brief An interface that will be used later on to implement particular behaviours.
class ChemEntity(ABC):
    ## @brief This is an abstract method. It returns the number of atoms in an element.
    #  @param elm This parameter an an ElementT Type.
    @abstractmethod
    def num_atoms(self, elm):
        pass

    ## @brief This is an abstract method.
    @abstractmethod
    def constit_elems(self):
        pass
```

G Code for Equality.py

```
## @file Equality.py
#  @author Aditya
#  @brief This file contains the Equality class.
from abc import ABC, abstractmethod


## @brief An interface that will be used later on to implement particular behaviours.
class Equality(ABC):
    ## @brief This is an abstract method.
    #  @param t This parameter is a sequence.
    @abstractmethod
    def equals(self, t):
        pass
```

H Code for Set.py

```
## @file Set.py
#  @author Aditya Sharma
#  @brief This file contains the Set class.

from Equality import *
## @brief Inherits from the "Equality" class.


class Set(Equality):
    ## @brief This is a constructor. It creates a Set object.
    #  @param s This parameter is a sequence containing elements of a particular type.
    def __init__(self, s):
        self.S = set(s)

    ## @brief This mutator adds elements of a particular type to the Set object.
    #  @param e It is of the same data type as the other elements in the set.
    def add(self, e):
        self.S.add(e)

    ## @brief This mutator removes elements of a particular type from the Set object.
    #  @param e It is of the same data type as the other elements in the set.
    #  @throw ValueEror It is thrown if an element is being removed that doesn't exist.
    def rm(self, e):
        try:
            self.S.remove(e)
        except:
            raise ValueError

    ## @brief This method checks if the given input is a member of the set.
    #  @param e It is of the same data type as the other elements in the set.
    #  @return A boolean type. True signifies the input is a member and False doesn't.
    def member(self, e):
        new_set = set([e])
        return new_set.issubset(self.S)

    ## @brief This is a getter.
    #  @return A natural number (integer) type. It represents the length of the set.
    def size(self):
        return len(self.S)

    ## @brief This method determies if the input set is equal to the given set.
    #  @param r It is a sequence of the same type as the elements in the Set object.
    #  @return A boolean type. True signifies that set equivalence and False doesn't.
    def equals(self, r):
        return set(r.to_seq()) == self.S

    ## @brief This method creates a sequencial representhation of the set.
    #  @return A list type. It represents a sequence of data types.
    def to_seq(self):
        return list(self.S)
```

I Code for ElmSet.py

```
## @file ElemSet.py
#  @author Aditya
#  @brief This file contains the ElmSet class.
from Set import *


## @brief Inherits properties from the "Set" class. It turns a Set object to an ElmSet object.
class ElmSet(Set):
    pass
```

J Code for MolecSet.py

```
## @file MolecSet.py
#  @author Aditya Sharma
#  @brief This file contains the MolecSet class.

from Set import *


## @brief This class inherits from the "Set" class
class MolecSet(Set):
    pass
```

K Code for CompoundT.py

```
## @file CompoundT.py
#  @author Aditya Sharma
#  @brief This file contains the CompoundT class

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


## @brief This is a class for compounds
class CompoundT(ChemEntity, Equality):
    ## @brief This is a constructor.
    #  @param M It is a MolecSet type. It represents a sequence of MoleculeT types.
    def __init__(self, M):
        self.__C = M

    ## @brief This is a getter. Its purpose is to retrieve the MolecSet.
    #  @return A MolecSet type. It represents a sequence of MoleculeT types.
    def get_molec_set(self):
        return self.__C

    ## @brief This method finds the number of certain atoms in a molecular sequence.
    #  @param e This parameter is an ElementT type.
    #  @return An integer type. It represents the count of certain atoms in a sequence.
    def num_atoms(self, e):
        count = 0
        molecule_set = self.get_molec_set()
        sequence = molecule_set.to_seq()
        for i in range(len(sequence)):
            count += sequence[i].num_atoms(e)
        return count

    ## @brief This is a method that finds elements
    #  @return A ElmSet type. It represents elements from a molecule sequence.
    def constit_elems(self):
        array = []
        molecule_set = self.get_molec_set()
        sequence = molecule_set.to_seq()
        for i in range(len(sequence)):
            array.append(sequence[i].get_elm())
        return ElmSet(array)

    ## @brief This method determines if two sets of molecules are equal
    #  @param D This parameter is a moleculeT type.
    #  @return A boolean response. True means both molecule sets are equal. False does not.
    def equals(self, D):
        if self.get_molec_set().size() != D.get_molec_set().size():
            return False
        hashtable = {}
        for molecule in self.get_molec_set().to_seq():
            element = molecule.get_elm()
            element_count = molecule.get_num()
            if hashtable.get(element) is None:
                hashtable[element] = element_count
        for molecule in D.get_molec_set().to_seq():
            element = molecule.get_elm()
            element_count = molecule.get_num()
            if hashtable.get(element) is None:
                return False
            elif hashtable.get(element) != element_count:
                return False

        return True
```

L Code for ReactionT.py

```
## @file reactionT.py
#  @author Aditya
#  @brief This file contains the ReactionT
#  Online citation:
     https://stackoverflow.com/questions/45220032/how-to-balance-a-chemical-equation-in-python-2-7-using-matrices


from MoleculeT import *
from ChemTypes import *
from ChemEntity import *
from Equality import *
from ElmSet import *
from MolecSet import *
from Set import *
import numpy as np


## @brief This class is responsible for balancing a chemical equation
class ReactionT:
    ## @brief This is a constructor
    #  @param L This parameter is a sequence of CompoundT types.
    #  @param R This parameter is a sequence of CompoundT types.
    def __init__(self, L, R):
        self.__lhs = L
        self.__rhs = R

        try:
            left_coeffs, right_coeffs = self.__balancer()
            if is_balanced(L, R, left_coeffs, right_coeffs) and pos(left_coeffs) and pos(right_coeffs):
                self.__coeffL, self.__coeffR = left_coeffs, right_coeffs
        except:
            raise ValueError

    ## @brief This getter retrives the compound sequence on the left side of the equation
    #  @return A sequence of CompoundT types.
    def get_lhs(self):
        return self.__lhs

    ## @brief This getter retrives the compound sequence on the right side of the equation
    #  @return A sequence of CompoundT types.
    def get_rhs(self):
        return self.__rhs

    ## @brief This getter retrieves a real number sequence from the equation's left side.
    #  @return A real number sequence. It represents the coefficient values of each compound
    def get_lhs_coeff(self):
        return self.__coeffL

    ## @brief This getter retrieves a real number sequence from the equation's right side.
    #  @return A real number sequence. It represents the coefficient values of each compound
    def get_rhs_coeff(self):
        return self.__coeffR

    ## @brief This is a local function used to balance equations
    #  @return 2 sequences of real numbers representing the coefficients of both sides
    #  Citation: Jash, Zackary (Discussion & Debug)
    def __balancer(self):
        all_elements = elm_in_chem_eq(self.get_lhs()).to_seq()
        all_compounds = []

        # Create array of compounds
        for i in range(len(self.get_lhs())):
            all_compounds.append(self.get_lhs()[i])
        for i in range(len(self.get_rhs())):
            all_compounds.append(self.get_rhs()[i])

        # AX = B and we are solving for X
        b = []
        a = []

        for i in range(len(all_elements)):
            singular_element = all_elements[i]
            count_of_atoms = self.get_lhs()[0].num_atoms(singular_element)
            b.append(count_of_atoms)

        for i in range(len(all_elements)):
```

```
            temp = []
            for j in range(1, len(all_compounds)):
                singular_element = all_elements[i]
                element_counter = all_compounds[j].num_atoms(singular_element)
                temp.append(element_counter)
            a.append(temp)

        # Numpy Linear Equation Solver
        x = np.linalg.lstsq(a, b, rcond=-1)
        left_coeffs = [1]
        right_coeffs = []

        # Get coefficients for left and right sides by traversing X
        for i in range(len(self.get_lhs()) - 1):
            left_coeffs.append(round(x[0][i], 3))
        for i in range(len(self.get_lhs()) - 1, len(self.get_rhs()) + len(self.get_lhs()) - 1):
            right_coeffs.append(round(x[0][i], 3))

        # Eliminate negatives
        for i in range(len(left_coeffs)):
            if left_coeffs[i] < 0:
                left_coeffs[i] = abs(left_coeffs[i])

        return left_coeffs, right_coeffs


## @brief This static method ensures every element within a sequence is positive.
#  @param s This parameter is a sequence of real numbers.
#  @return A boolean type. True signifies that every element is positive. False doesn't.
def pos(s):
    for i in range(len(s)):
        if s[i] < 0:
            return False
    return True


## @brief This static method finds the count of an atoms in a sequence of compounds.
#  @param C This parameter is a sequence of CompoundT types.
#  @param c This parameter is a sequence of real numbers. It represents the coefficients
#  @param e This parameter is a ElementT type. It represents the element.
#  @return A natural number type. This represents the count of the atoms.
def n_atoms(C, c, e):
    count = 0
    for i in range(len(C)):
        count += c[i] * C[i].num_atoms(e)
    return count


## @brief This method finds the set of elements from each compound in a given sequence.
#  @param C This parameter is a sequence of CompoundT types.
#  @return This method returns an sequence of ElementT type.
def elm_in_chem_eq(C):
    element_seq = C[0].constit_elems()
    for compound in C:
        compound_element_seq = compound.constit_elems()
        for elements in compound_element_seq.to_seq():
            element_seq.add(elements)
    return element_seq


## @brief This local function checks if the count of atoms of an element is equal.
#  @param L This parameter is a sequence of CompoundT types.
#  @param R This parameter is a sequence of CompoundT types.
#  @param cL This parameter is a sequence of real number types.
#  @param cR This parameter is a sequence of real number types.
#  @return A Boolean type. True signifies that equivalent counts of atoms. False doesn't.
def is_bal_elm(L, R, cL, cR, e):
    return n_atoms(L, cL, e) == n_atoms(R, cR, e)


## @brief This local function checks if both sides are balanced
#  @param L This parameter is a sequence of CompoundT types.
#  @param R This parameter is a sequence of CompoundT types.
#  @param cL This parameter is a sequence of real number types.
#  @param cR This parameter is a sequence of real number types.
#  @return A boolean type. True signifies balance. False doesn't.
def is_balanced(L, R, cL, cR):
    left = elm_in_chem_eq(L)
    right = elm_in_chem_eq(R)
    for element in left.to_seq():
```

```
    if not is_bal_elm(L, R, cL, cR, element):
        return False
return (left.equals(right))
```

M Code for test_All.py

```
## @file test_All.py
#  @author Aditya Sharma
#  @brief This file is the test file.

from Set import *
from MoleculeT import *
from CompoundT import *
from ReactionT import *
import pytest


class TestSet:

        def setup_method(self, method):
                self.setA = Set([1, 2, 3, 4, 5])
                self.setB = Set([5, 4, 3, 2, 0])
                self.setC = Set([3, 2, 1, 4, 5])

        def teardown_method(self, method):
                self.setA = None

        def test_add_a(self):
                self.setA.add(5)
                assert self.setA.size() == 5

        def test_add_b(self):
                self.setA.add(6)
                assert self.setA.size() == 6
                assert self.setA.member(6) == True

        def test_rm(self):
                with pytest.raises(ValueError):
                        self.setA.rm(6)

        def test_member_a(self):
                assert self.setA.member(5) == True

        def test_member_b(self):
                assert self.setA.member(6) == False

        def test_size(self):
                assert self.setA.size() == 5

        def test_equals_a(self):
                assert self.setA.equals(self.setB) == False

        def test_equals_b(self):
                assert self.setA.equals(self.setC) == True

        def test_to_seq(self):
                assert self.setA.to_seq() == [1,2,3,4,5]


class TestMoleculeT:
        def setup_method(self, method):
                self.moleculeA = MoleculeT(2, ElementT.H)
                self.moleculeB = MoleculeT(1, ElementT.H)
                self.moleculeC = MoleculeT(2, ElementT.H)
                self.moleculeD = MoleculeT(2, ElementT.O)

        def teardown_method(self, method):
                self.moleculeA = None
                self.moleculeB = None
                self.moleculeC = None
                self.moleculeD = None

        def test_get_num(self):
                assert self.moleculeA.get_num() == 2

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

        def test_num_atoms_a(self):
                assert self.moleculeA.num_atoms(ElementT.H) == 2

        def test_num_atoms_b(self):
```

```
                assert self.moleculeA.num_atoms(ElementT.S) == 0

        def test_constit_elems_a(self):
                assert type(self.moleculeA.constit_elems()) == type(ElmSet([ElementT.H]))

        def test_constit_elems_b(self):
                assert self.moleculeA.constit_elems().to_seq() == [ElementT.H]

        def test_equals_a(self):
                assert self.moleculeA.equals(self.moleculeB) == False

        def test_equals_b(self):
                assert self.moleculeA.equals(self.moleculeC) == True

        def test_equals_c(self):
                assert self.moleculeA.equals(self.moleculeD) == False


class TestCompoundT:
        def setup_method(self, method):
                self.moleculeA = MoleculeT(2, ElementT.H)
                self.moleculeB = MoleculeT(1, ElementT.S)
                self.moleculeC = MoleculeT(4, ElementT.O)
                self.moleculeD = MoleculeT(2, ElementT.O)

                self.molecSetA = MolecSet([self.moleculeA, self.moleculeB, self.moleculeC])
                self.molecSetB = MolecSet([self.moleculeA, self.moleculeD])

                self.compoundA = CompoundT(self.molecSetA)
                self.compoundB = CompoundT(self.molecSetB)

        def teardown_method(self, method):
                self.moleculeA = None
                self.moleculeB = None
                self.moleculeC = None
                self.moleculeD = None

                self.molecSetA = None
                self.molecSetB = None

                self.compoundA = None
                self.compoundB = None

        def test_get_molec_set(self):
                assert self.compoundA.get_molec_set().to_seq() == MolecSet([self.moleculeA,
                    self.moleculeB, self.moleculeC]).to_seq()

        def test_num_atoms_a(self):
                assert self.compoundA.num_atoms(ElementT.H) == 2

        def test_num_atoms_b(self):
                assert self.compoundA.num_atoms(ElementT.Be) == 0

        def test_constit_elems(self):
                assert set(self.compoundA.constit_elems().to_seq()) == set(ElmSet([ElementT.H,
                    ElementT.S, ElementT.O]).to_seq())

        def test_equals_a(self):
                assert self.compoundA.equals(self.compoundA) == True

        def test_equals_b(self):
                assert self.compoundA.equals(self.compoundB) == False


class TestReactionT:
        def setup_method(self, method):

                self.moleculeA = MoleculeT(4, ElementT.C)
                self.moleculeB = MoleculeT(10, ElementT.H)

                self.moleculeC = MoleculeT(2, ElementT.O)

                self.moleculeD = MoleculeT(1, ElementT.C)
                self.moleculeE = MoleculeT(2, ElementT.O)

                self.moleculeF = MoleculeT(2, ElementT.H)
                self.moleculeG = MoleculeT(1, ElementT.O)

                self.molecSetA = MolecSet([self.moleculeA, self.moleculeB])
                self.molecSetB = MolecSet([self.moleculeC])
```

```
        self.molecSetC = MolecSet([self.moleculeD, self.moleculeE])
        self.molecSetD = MolecSet([self.moleculeF, self.moleculeG])

        self.compoundA = CompoundT(self.molecSetA)
        self.compoundB = CompoundT(self.molecSetB)
        self.compoundC = CompoundT(self.molecSetC)
        self.compoundD = CompoundT(self.molecSetD)

        self.reactionA = ReactionT([self.compoundA, self.compoundB], [self.compoundC,
            self.compoundD])

def teardown_method(self, method):
        self.moleculeA = None
        self.moleculeB = None

        self.moleculeC = None

        self.moleculeD = None
        self.moleculeE = None

        self.moleculeF = None
        self.moleculeG = None

        self.molecSetA = None
        self.molecSetB = None
        self.molecSetC = None
        self.molecSetD = None

        self.compoundA = None
        self.compoundB = None
        self.compoundC = None
        self.compoundD = None

def test_get_lhs(self):
        assert self.reactionA.get_lhs() == [self.compoundA, self.compoundB]

def test_get_rhs(self):
        assert self.reactionA.get_rhs() == [self.compoundC, self.compoundD]

def test_get_lhs_coeff(self):
        assert self.reactionA.get_lhs_coeff() == [1,6.5]

def test_get_rhs_coeff(self):
        assert self.reactionA.get_rhs_coeff() == [4,5]
```

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
```