Assignment 2 Solution

Rizwan Ahsan, ahsanm7

February 25, 2021

This report discusses the testing phase for .... It also discusses the results of running the same tests on the partner files. The assignment specifications are then critiqued and the requested discussion questions are answered.

1 Testing of the Original Program

The result of running test case on my program resulting in passing all the tests. A total of 21 tests were written which covered all possible methods. The test cases were written so that it would cover all the exceptions in the modules. Also all the methods were tested tested invidually. In this design specification all the modules followed a similar trend in methods and contained setters and getters and easy methods to test. So, overall the test cases that were written were fairly easy to write and covered all of the possible outcomes.

2 Results of Testing Partner's Code

By running the test cases that were written by me on my partner's code also returned in a succesful passing of all the tests. Most of the methods that were written were tested intesively to cover all the possile outcomes. It seems that my partner's code was also well written in accordance to the design specification. However, since my testing of sim function could be made better it seems that that also passed for my partner's code.

3 Critique of Given Design Specification

The design specification for the most part is very well. All of the methods naming were consistent throughout the specification. The getter methods started with get and the setter methods started with set which made it easier to identify what the method did.

The ordering of parameters were also consisted. All of the methods had necessary exception handling and the obvious information were assumed to be inputted correctly by the programmer. The design also had the quality of essential because it didn't contain any extra unnecessary functions. CircleT.py, TriangleT.py and BodyT.py were all implemented using Shape.py as an interface which allowed to precisely implement the only required functions. The design however is not minimal. On the Scene.py, the getters and setters for the unbalanced forces and initial velocities did two task at the same time. The rest of the modules maintained minimality. The entire design can be made minimal by changing on these above methods to make them perform a single task. The design has high cohesion because all of the modules were built together to perform a single task. And in order to perform that task they were very closely related to each other. The design had low coupling between other modules. All the modules that were implemented dependent on the implementation of Shape.py, however, this module acted as interface for all the other modules. The design could be made without Shape.py but it could increase the chances of making mistakes in the process. The design was opaque meaning it had very good information hiding. All of the specification only showed what each of the modules were intended to do however, how they need to be implemented were tasked on the programmer. Overall, the design specification that was provided was very good.

4 Answers

a) Yes, I think setters and getters need to be unit tested. One a small code base it doesn't make sense to get them tested since they return a certain value or change a certain value. However, in a very big code base, where different modules have high cohesion and depends on the getters or setters then the methods need to be checked that it performs properly as intended otherwise the program will give random errors and the root cause of that errors will be difficult to find. So, writting unit tests will help mitigate these problem.

b) In order to test the functions Fx and Fy, the functions need to be defined in the unit tests. Then, a Scene object needs to be created with proper parameters for which the output of Fx and Fy are known. Then we can test the setters and getters and see that they return the correct value that was calculated prior.

c) Matplotlib has a built in testing method can be run inside of pytest. The method is called compare_images(). This method takes in two images, 1 is the output of the graph from Plot.py and the other image is the expected graph, there is a third input which is the tolerance level. We can save the images of the plots using the savefig() function found in matplotlib and set a tolerance under which the images are equal. By doing this we can automate tests for Plot.py

d) close_enough : seq of $\mathbb{R} \times$ seq of $\mathbb{R} \rightarrow \mathbb{B}$
close_enough$(x_c, x_t) \equiv (i, j : \mathbb{R} | i \in [0..|x_c| - 1] \wedge j \in [0..|x_t| - 1] : \frac{||x_c - x_t||}{||x_t||} < \epsilon)$

e) No, there is no need to have exceptions for negative coordinates because this signifies the position of the center of mass of the shape. Since the shape can move in any direction, having negative coordinates can be useful in this case. It can be used to pinpoint the specific location of the shape and can determine it's behaviour of motion.

f) In the constructor for TriangleT, there is an exception that handles the state invariant. So, whenever the condition $(s > 0 \wedge m > 0)$ is not met, then the program throws a ValueError and stops the code. Since, the constructor is run whenever an object of TriangleT is created, this allows that the invariant is always satisfied by the given specification.

g)
```
[x**0.5 for x in range(5,20) if x%2 != 0]
```

h)
```
def remove_upper(word):
    output = ""
    for c in word:
        if c.upper() == c:
            continue
        output += c

    return output
```

i) Abstraction and generality are together related to the concept of information hiding. Whenever a generic module is created it can be used by many other modules to perform other tasks. For example, the built-in libraries of any programming language both follow the concept of information hiding. They are abstract in the sense that we don't to know the intricate details of the code, we just need to know what it does and general in the sense that it can be reused and modified to our needs as necessary.

j) The scenario in which a module is used by many other modules i.e Fan-in would be better in general. This is because in this case, all the other modules are dependent on the correct implementation of the parent module. So, if the dependent modules functions incorrectly then the root cause of the problem can be fixed by correcting only 1 module.

E Code for Shape.py

```
## @file Shape.py
#  @author Rizwan Ahsan, ahsanm7
#  @brief An interface for shape
#  @date 16/02/2021

from abc import ABC, abstractmethod


## @brief Shape provides an interface for all kinds of shape
class Shape(ABC):

    @abstractmethod
    ## @brief A generic method for finding the x-axis of center of mass
    #  @return a float indicating the x-axis of the shape
    def cm_x(self):
        pass

    @abstractmethod
    ## @brief A generic method for finding the y-axis of center of mass
    #  @return a float indicating the y-axis of the shape
    def cm_y(self):
        pass

    @abstractmethod
    ## @brief A generic method for finding the mass
    #  @return a float indicating the mass of the shape
    def mass(self):
        pass

    @abstractmethod
    ## @brief A generic method for finding the moment of inertia
    #  @return a float indicating the moment of inertia of the shape
    def m_inert(self):
        pass
```

F Code for CircleT.py

```
## @file CircleT.py
#  @author Rizwan Ahsan, ahsanm7
#  @brief Contains a ADT for circle
#  @date 16/02/2021

from Shape import Shape


## @brief CircleT is an abstract class for circle.
#  @details A circle is composed of x, y axis, radius and mass
class CircleT(Shape):

    ## @brief Constructor for the class CircleT. Assumes the arguments
    #         provided will be of correct type
    #  @param x x-axis of the circle
    #  @param y y-axis of the circle
    #  @param r radius of the circle
    #  @param m mass of the circle
    #  @throws ValueError throws if r and m value is not greater than 0
    def __init__(self, x, y, r, m):
        if r <= 0 or m <= 0:
            raise ValueError
        self.__x = x
        self.__y = y
        self.__r = r
        self.__m = m

    ## @brief Gets the x-axis of center of mass of circle
    #  @return Float representing the x-axis
    def cm_x(self):
        return self.__x

    ## @brief Gets the y-axis of center of mass of the circle
    #  @return Float representing the y-axis
    def cm_y(self):
        return self.__y

    ## @brief Gets the mass of the circle
    #  @return Float representing the mass
    def mass(self):
        return self.__m

    ## @brief Gets the moment of inertia of the circle
    #  @return Float representing the moment of inertia
    def m_inert(self):
        return (self.__m * self.__r ** 2) / 2
```

G Code for TriangleT.py

```
## @file TriangleT.py
#  @author Rizwan Ahsan, ahsanm7
#  @brief Contains a ADT for working with traingles
#  @date 16/02/2021

from Shape import Shape


## @brief An ADT class for representing triangles
class TriangleT(Shape):

    ## @brief Constructor for the class TriangleT. Assumes the arguments
    #         provided will be of correct type
    #  @param x x-axis of the triangle
    #  @param y y-axis of the triangle
    #  @param r radius of the triangle
    #  @param m mass of the triangle
    #  @throws ValueError throws if s and m value is not greater than 0
    def __init__(self, x, y, s, m):
        if s <= 0 or m <= 0:
            raise ValueError
        self.__x = x
        self.__y = y
        self.__s = s
        self.__m = m

    ## @brief Gets the x-axis of center of mass of the triangle
    #  @return Float representing the x-axis
    def cm_x(self):
        return self.__x

    ## @brief Gets the y-axis of center of mass of the triangle
    #  @return Float representing the y-axis
    def cm_y(self):
        return self.__y

    ## @brief Gets the mass of the triangle
    #  @return Float representing the mass
    def mass(self):
        return self.__m

    ## @brief Gets the moment of inertia of the triangle
    #  @return Float representing the moment of inertia
    def m_inert(self):
        return (self.__m * self.__s ** 2) / 12
```

H Code for BodyT.py

```
## @file BodyT.py
#  @author Rizwan Ahsan, ahsanm7
#  @brief Contains a ADT for working with bodies of any shape
#  @date 16/02/2021

from Shape import Shape


## @brief An ADT for representing bodies of various shape
#  @details Any object can be a body if it has center of mass, mass and
#           moment of inertia
class BodyT(Shape):

    ## @brief Contructor for the class BodyT. Assumes the provided arguments
    #         will be of correct type
    #  @param x List of all points in body along x-axis
    #  @param y List of all points in body along y-axis
    #  @param m List of all mass contained in the body at certain points
    #  @throws ValueError throws error if the length of x, y and m are not equal
    #          or any of the mass in m is less than 0
    def __init__(self, x, y, m):
        self.__cmx = self.__cm__(x, m)
        self.__cmy = self.__cm__(y, m)
        self.__m = sum(m)
        value = (self.__cm__(x, m) ** 2 + self.__cm__(y, m) ** 2)
        self.__moment = self.__mmom__(x, y, m) - sum(m) * value
        eq_len = len(x) == len(y) == len(m)
        mass_error = False
        for i in m:
            if i <= 0:
                mass_error = True

        if not eq_len or mass_error:
            raise ValueError

    @staticmethod
    def __cm__(z, m):
        return sum([z[i] * m[i] for i in range(len(m))]) / sum(m)

    @staticmethod
    def __mmom__(x, y, m):
        s = 0
        for i in range(len(m)):
            s += m[i] * (x[i] ** 2 + y[i] ** 2)
        return s

    ## @brief Gets the x-axis of the center of mass of body
    #  @return Float representing the x-axis
    def cm_x(self):
        return self.__cmx

    ## @brief Gets the y-axis of the center of mass of body
    #  @return Float representing the y-axis
    def cm_y(self):
        return self.__cmy

    ## @brief Gets the  mass of the  body
    #  @return Float representing the mass
    def mass(self):
        return self.__m

    ## @brief Gets the moment of inertia of body
    #  @return Float representing the moment of inertia
    def m_inert(self):
        return self.__moment
```

I Code for Scene.py

```
## @file Scene.py
#  @author Rizwan Ahsan, ahsanm7
#  @brief Contains an ADT for working with different types of scenes
#  @date 16/02/2021
#  @details Demonstrates the movement of a shape with time

from scipy.integrate import odeint


## @brief An ADT for representing a scene
class Scene():

    ## @brief Constructor for the Scene class
    #  @param s A Shape object
    #  @param fx unbalanced force function in x direction
    #  @param fy unbalanced force function in y direction
    #  @param vx initial velocity in x direction
    #  @param vy initial velocity in y direction
    def __init__(self, s, fx, fy, vx, vy):
        self.__s = s
        self.__fx = fx
        self.__fy = fy
        self.__vx = vx
        self.__vy = vy

    ## @brief Gets the shape object that was passed in the constructor
    #  @return A shape object
    def get_shape(self):
        return self.__s

    ## @brief Gets the unbalanced forces in x, y directions
    #  @return A tuple of unbalanced forces
    def get_unbal_forces(self):
        return self.__fx, self.__fy

    ## @brief Gets the initial velocities in x, y directions
    #  @return A tuple of velocities
    def get_init_velo(self):
        return self.__vx, self.__vy

    ## @brief Changes the set variable which contains shape object
    #         to the new object
    def set_shape(self, s):
        self.__s = s

    ## @brief Changes the unbalanced forces in both x, y directions
    def set_unbal_forces(self, fx, fy):
        self.__fx = fx
        self.__fy = fy

    ## @brief Changes the initial velocities in both x, y directions
    def set_init_velo(self, vx, vy):
        self.__vx = vx
        self.__vy = vy

    ## @brief Simulates the movement of the shape with time
    #  @return tuple of list
    def sim(self, tf, nsteps):
        t = sorted([(i * tf) / (nsteps - 1) for i in range(nsteps)])

        def ode(w, tau):
            third = self.__fx(tau) / self.__s.mass()
            fourth = self.__fy(tau) / self.__s.mass()
            return [w[2], w[3], third, fourth]

        odeint_val = odeint(ode, [self.__s.cm_x(), self.__s.cm_y(), self.__vx, self.__vy], t)
        return t, odeint_val
```

J Code for Plot.py

```
## @file Plot.py
#  @author Rizwan Ahsan, ahsanm7
#  @brief A library used for plotting graphs
#  @date 16/02/2021
#  @details Displays 3 seperate graphs denoting x versus t, y versus t and
#           y versus x respectively

from matplotlib import pyplot as plt


## @brief A function to plot graphs
#  @throws ValueError throws if the length of w and t are not equal
def plot(w, t):
    if len(w) != len(t):
        raise ValueError

    x = []
    y = []

    for i in range(len(w)):
        x.append(w[i][0])
        y.append(w[i][1])

    fig, axes = plt.subplots(3)
    fig.suptitle('Motion simulation')
    axes[0].plot(t, x)
    axes[1].plot(t, y)
    axes[2].plot(x, y)

    axes[0].set(ylabel='x (m)')
    axes[1].set(ylabel='y (m)')
    axes[2].set(xlabel='x (m)', ylabel='y (m)')

    plt.show()
```

K Code for test_driver.py

```
## @file test_All.py
#  @author Rizwan Ahsan, ahsanm7
#  @brief Contains methods for testing all modules in the file
#  @date 16/02/2021
#  @details Tests Shape, CircleT, TriangleT, BodyT, Scene and Plot

import pytest
from CircleT import CircleT
from TriangleT import TriangleT
from BodyT import BodyT
from Scene import Scene
from Plot import plot


class TestCircleT():

    def setup_method(self, method):
        self.c = CircleT(1.0, 10.0, 0.5, 1.0)

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

    def test_cmx(self):
        assert self.c.cm_x() == 1.0

    def test_cmy(self):
        assert self.c.cm_y() == 10.0

    def test_mass(self):
        assert self.c.mass() == 1.0

    def test_m_inert(self):
        assert self.c.m_inert() == 1.0 * 0.5 ** 2 / 2

    def test_error(self):
        with pytest.raises(ValueError):
            CircleT(1.0, 10.0, -0.5, -2.0)


class TestTriangleT():

    def setup_method(self, method):
        self.t = TriangleT(1.0, -10.0, 5, 17.5)

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

    def test_cmx(self):
        assert self.t.cm_x() == 1.0

    def test_cmy(self):
        assert self.t.cm_y() == -10.0

    def test_mass(self):
        assert self.t.mass() == 17.5

    def test_m_inert(self):
        assert self.t.m_inert() == 17.5 * 5 ** 2 / 12

    def test_error(self):
        with pytest.raises(ValueError):
            TriangleT(2.0, 0.0, -1.0, 5.0)


class TestBodyT():

    def setup_method(self, method):
        self.b = BodyT([11, 9, 9, 11], [11, 11, 9, 9], [10, 10, 10, 10])

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

    def test_cmx(self):
        assert self.b.cm_x() == 10

    def test_cmy(self):
        assert self.b.cm_y() == 10

    def test_mass(self):
        assert self.b.mass() == 40

    def test_m_inert(self):
        assert self.b.m_inert() == 80

    def test_error1(self):
        with pytest.raises(ValueError):
            BodyT([11, 9, 9, 11], [11, 11, 9, 9], [10, 10, 10])

    def test_error2(self):
        with pytest.raises(ValueError):
            BodyT([11, 9, 9, 11], [11, 11, 9, 9], [10, 10, 10, -20])


class TestScene():

    def setup_method(self, method):
        self.t = TriangleT(1.0, -10.0, 5, 17.5)
        self.g = 9.81
        self.m = 1
        self.s = Scene(self.t, self.Fx, self.Fy, 10, 20)

    def teardown_method(self, method):
        self.t1 = None

    def Fx(self, t):
        return 5 if t < 5 else 0

    def Fy(self, t):
        return -self.g * self.m if t < 3 else self.g * self.m

    def test_get_shape(self):
        assert self.s.get_shape() == self.t

    def test_get_init_val(self):
        assert self.s.get_init_velo() == (10, 20)

    def test_set_init_val(self):
        self.s.set_init_velo(0, 0)
        assert self.s.get_init_velo() == (0, 0)

    def test_set_shape(self):
        a = CircleT(1.0, 10.0, 0.5, 1.0)
        self.s.set_shape(a)
        assert self.s.get_shape() == a

    def sim_values(self):
        t_calc, wsol = self.s.sim(10, 50)
        w_calc = []
        for w1 in wsol:
            w_calc.append([w1[0], w1[1], w1[2], w1[3]])

        t_true = [0.0, 0.20408163265306123, 0.40816326530612246, 0.6122448979591837,
                  0.8163265306122449, 1.0204081632653061, 1.2244897959183674, 1.4285714285714286,
                  1.6326530612244898, 1.836734693877551, 2.0408163265306123, 2.2448979591836733,
                  2.4489795918367347, 2.6530612244897958, 2.857142857142857, 3.061224489795918,
                  3.2653061224489797, 3.4693877551020407, 3.673469387755102, 3.877551020408163,
                  4.081632653061225, 4.285714285714286, 4.489795918367347, 4.6938775510204085,
                  4.8979591836734695, 5.1020408163265305, 5.3061224489795915, 5.510204081632653,
                  5.714285714285714, 5.918367346938775, 6.122448979591836, 6.326530612244898,
                  6.530612244897959, 6.73469387755102, 6.938775510204081, 7.142857142857143,
                  7.346938775510204, 7.551020408163265, 7.755102040816326, 7.959183673469388,
                  8.16326530612245, 8.36734693877551, 8.571428571428571, 8.775510204081632,
                  8.979591836734693, 9.183673469387756, 9.387755102040817, 9.591836734693878,
                  9.795918367346939, 10.0]

        w_true = [[1.0, 10.0, 0.0, 0.0],
                  [1.10412328258546, 9.795710119567326, 1.0204081632653061, -2.0020408163265304],
                  [1.416493128483002, 9.18284048191635, 2.040816326530612, -4.004081632653062],
                  [1.9371095383122396, 8.161391085831385, 3.0612244897959187, -6.006122448979593],
                  [2.665972512073173, 6.731361931312434, 4.081632653061225, -8.008163265306123],
                  [3.603082049765802, 4.892753018359498, 5.1020408163265305, -10.010204081632654],
                  [4.748438151390126, 2.645564346972576, 6.122448979591836, -12.012244897959185],
                  [6.102040816946145, -0.010204082848332074, 7.142857142857142, -14.014285714285714],
                  [7.663890046433865, -3.074552271103233, 8.163265306122451, -16.016326530612247],
                  [9.433985839853273, -6.547480217792112, 9.183673469387756, -18.018367346938778],
                  [11.41232819720438, -10.428987922914978, 10.204081632653063, -20.02040816326531],
                  [13.598917118487176, −14.719075386471825, 11.224489795918368, −22.022448979591836],
                  [15.993752603701672, −19.417742608462667, 12.244897959183675, −24.02448979591837],
                  [18.596834652847857, −24.524989588887475, 13.265306122448981, −26.026530612244898],
                  [21.408163265925747, −30.0408163277463, 14.285714285714288, −28.028571428571436],
                  [24.427738916534295, −35.928451567707256, 15.306122448979593, −28.82938761996386],
                  [27.655560657475576, −41.60771018236045, 16.326530612244902, −26.827346803637326],
                  [31.091628962348537, −46.878389038579634, 17.346938775510203, −24.8253059873108],
                  [34.73594383115321, −51.74048813636485, 18.367346938775512, −22.823265170984268],
                  [38.588505263889566, −56.19400747571607, 19.387755102040817, −20.821224354657737],
                  [42.64931326055763, −60.23894705663334, 20.408163265306133, −18.8191835383312],
                  [46.918367821157375, −63.875306879116586, 21.428571428571434, −16.81714272200467],
                  [51.39566894568879, −67.10308694316583, 22.448979591836732, −14.815101905678157],
                  [56.081216634151936, −69.92228724878112, 23.469387755102037, −12.813061089351622],
                  [60.975010886546784, −72.33290779596243, 24.489795918367353, −10.811020273025086],
                  [66.05102279679654, −74.33494507893066, 25.000001250241873, −8.808979456698568],
                  [71.15306386827447, −75.928406109244, 25.000001250241873, −6.806938640372042],
                  [76.25510493975241, −77.11328738112334, 25.000001250241873, −4.804897824045504],
                  [81.35714601123036, −77.88958889456869, 25.000001250241873, −2.802857007718976],
                  [86.45918708270828, −78.25731064958006, 25.000001250241873, −0.8008161913924476],
                  [91.5612281541862, −78.21645264615745, 25.000001250241873, 1.2012246249340786],
                  [96.66326922566415, −77.76701488430085, 25.000001250241873, 3.2032654412606156],
                  [101.76531029714208, −76.90899736401026, 25.000001250241873, 5.205306257587145],
                  [106.86735136861999, −75.64240008528569, 25.000001250241873, 7.207347073913669],
                  [111.96939244009792, −73.96722304812712, 25.000001250241873, 9.209387890240198],
                  [117.07143351157588, −71.88346625253457, 25.000001250241873, 11.211428706566736],
                  [122.1734745830538, −69.39112969850804, 25.000001250241873, 13.213469522893265],
                  [127.2755156545317, −66.49021338604754, 25.000001250241873, 15.215510339219785],
                  [132.37755672600963, −63.18071731515305, 25.000001250241873, 17.217551155546314],
                  [137.47959779748757, −59.46264148582455, 25.000001250241873, 19.219591971872852],
                  [142.5816388689655, −55.335985898062084, 25.000001250241873, 21.22163278819938],
                  [147.68367994044343, −50.80075055186563, 25.000001250241873, 23.22367360452591],
                  [152.78572101192137, −45.856935447235195, 25.000001250241873, 25.225714420852437],
                  [157.88776208339928, −40.50454058417077, 25.000001250241873, 27.227755237178965],
                  [162.98980315487725, −34.743565962672335, 25.000001250241873, 29.2297960535055],
                  [168.09184422635522, −28.574011582739885, 25.000001250241873, 31.231836869832044],
                  [173.19388529783316, −21.995877444373505, 25.000001250241873, 33.23387768615858],
                  [178.29592636931108, −15.009163547573152, 25.000001250241873, 35.235918502485106],
                  [183.397967440789, −7.613869892338798, 25.000001250241873, 37.23795931881163],
                  [188.50000851226693, 0.1900035213295581, 25.000001250241873, 39.24000013513816]]

        return t_calc, t_true, w_calc, w_true

    def test_sim1(self):
        tc, tt = self.sim_values()[0], self.sim_values()[1]
        num = []
        for i in range(len(tc)):
            num.append(tc[i] − tt[i])

        assert (max(num) / max(tt)) < 0.005
```

L Code for Partner's CircleT.py

```
## @file CircleT.py
#  @author Emily Sanderson 400088490 sanderse
#  @brief A template module that inherits Shape to create a circle
#  @date12/01/2021

from Shape import Shape


## @brief An abstract data type that represents a Circle
class CircleT(Shape):

    ## @brief CircleT constructor
    #  @details given the position, radius and mass, creates a Circle
    #  @param xs x-direction position of circle center of mass
    #  @param ys y-direction position of circle center of mass
    #  @param rs radius of circle
    #  @param ms mass of circle
    #  @throws ValueError if radius or mass are not greater than 0
    def __init__(self, xs, ys, rs, ms):
        if not (rs > 0 and ms > 0):
            raise ValueError()
        self.__x = xs
        self.__y = ys
        self.__r = rs
        self.__m = ms

    ## @brief cm_x returns the x position of the center of mass
    #  @return value representing the x-pos of the center of mass
    def cm_x(self):
        return self.__x

    ## @brief cm_y returns the y position of the center of mass
    #  @return value representing the y-pos of the center of mass
    def cm_y(self):
        return self.__y

    ## @brief mass returns the mass of CircleT
    #  @return value representing the mass of CircleT
    def mass(self):
        return self.__m

    ## @brief cm_x returns the moment of inertia of CircleT
    #  @return value of moment of inertia
    def m_inert(self):
        return (self.__m * (self.__r ** 2)) / 2
```

M Code for Partner's TriangleT.py

```
## @file TriangleT.py
#  @author Emily Sanderson 400088490 sanderse
#  @brief A template module that inherits Shape to create a triangle
#  @date12/01/2021

from Shape import Shape


## @brief An abstract data type that represents a Triangle
class TriangleT(Shape):

    ## @brief TriangleT constructor
    #  @details given the position, side length and mass, creates a Triangle
    #  @param xs x-direction position of triangle center of mass
    #  @param ys y-direction position of triangle center of mass
    #  @param ss side length (equilateral triangle)
    #  @param ms mass of Triangle
    #  @throws ValueError if side length or mass are not greater than 0
    def __init__(self, xs, ys, ss, ms):
        if not (ss > 0 and ms > 0):
            raise ValueError()
        self.__x = xs
        self.__y = ys
        self.__s = ss
        self.__m = ms

    ## @brief cm_x returns the x position of the center of mass
    #  @return value representing the x-pos of the center of mass
    def cm_x(self):
        return self.__x

    ## @brief cm_y returns the y position of the center of mass
    #  @return value representing the y-pos of the center of mass
    def cm_y(self):
        return self.__y

    ## @brief mass returns the mass of TriangleT
    #  @return value representing the mass of TriangleT
    def mass(self):
        return self.__m

    ## @brief cm_x returns the moment of inertia of TriangleT
    #  @return value of moment of inertia
    def m_inert(self):
        return (self.__m * (self.__s ** 2)) / 12
```

N Code for Partner's BodyT.py

```
## @file BodyT.py
#  @author Emily Sanderson 400088490 sanderse
#  @brief A template module that inherits Shape to create a body
#  @date12/01/2021

from Shape import Shape
import numpy as np


## @brief An abstract data type that represents a Body
class BodyT(Shape):

    ## @brief BodyT constructor
    #  @details given a sequence of x, y and mass values, creates a BodyT
    #  @param xs seq of x-direction positions of Body center of the center of mass
    #  @param ys seq of y-direction positions of Body center of the center of mass
    #  @param ms seq of masses of Body
    #  @throws ValueError if moment or mass are not greater than 0
    def __init__(self, xs, ys, ms):
        if not (len(xs) == len(ys) and len(ys) == len(ms)):
            raise ValueError()
        for j in ms:
            if j <= 0:
                raise ValueError()
        self.__cmx = np.dot(xs, ms) / sum(ms)
        self.__cmy = np.dot(ys, ms) / sum(ms)
        self.__m = sum(ms)
        squared = self.__m * (self.__cmx ** 2 + self.__cmy ** 2)
        self.__moment = np.dot(ms, np.add(np.square(xs), np.square(ys))) - squared

    ## @brief cm_x returns the center of mass in the x direction
    #  @return value representing the x-pos of the center of mass
    def cm_x(self):
        return self.__cmx

    ## @brief cm_y rthe center of mass in the x direction
    #  @return value representing the y-pos of the center of mass
    def cm_y(self):
        return self.__cmy

    ## @brief mass returns the mass of BodyT
    #  @return value representing the mass of BodyT
    def mass(self):
        return self.__m

    ## @brief cm_x returns the moment of inertia of BodyT
    #  @return value of moment of inertia
    def m_inert(self):
        return self.__moment
```

O Code for Partner's Scene.py

```
## @file Scene.py
#  @author Emily Sanderson
#  @brief Creates the scene
#  @date 12/02/2021
#  @details Returns the scene

from scipy.integrate import odeint


## @brief An abstract data type that represents a scene
class Scene():

    ## @brief Scene constructor
    #  @details Creates a Scene Module given unbalanced force and velocity
    #  @param s xShape input
    #  @param Fx unbalanced force function in x dir
    #  @param Fy unbalanced force function in y dir
    #  @param vx initial velocity in x dir
    #  @param vy initial velocity in y dir
    def __init__(self, s, Fx, Fy, vx, vy):
        self.__s = s
        self.__Fx = Fx
        self.__Fy = Fy
        self.__vx = vx
        self.__vy = vy

    ## @brief Getter that returns the shape object
    #  @return Returns shape
    def get_shape(self):
        return self.__s

    ## @brief Getter that returns the unbalanced forces formulas
    #  @return Returns the unbalanced forces formulas
    def get_unbal_forces(self):
        return self.__Fx, self.__Fy

    ## @brief Getter that returns the initial velocities
    #  @return Returns the initial velocities
    def get_init_velo(self):
        return self.__vx, self.__vy

    ## @brief Setter that sets new shape to current shape
    #  @param sprime Another Shape
    def set_shape(self, sprime):
        self.__s = sprime

    ## @brief Setter that sets new unbalanced forces equations
    #  @param Fxprime New Fx Formula
    #  @param Fyprime New Fy Formula
    def set_unbal_forces(self, Fxprime, Fyprime):
        self.__Fx = Fxprime
        self.__Fy = Fyprime

    ## @brief Setter that sets new initial velocities
    #  @param Fxprime New x velocity
    #  @param Fyprime New y velocity
    def set_init_velo(self, vxprime, vyprime):
        self.__vx = vxprime
        self.__vy = vyprime

    ## @brief Calculates the values of t and runs odeint formula
    #  @param t_final length of simulation
    #  @param nsteps Number of steps taken in
    def sim(self, t_final, nsteps):
        t = []
        for i in range(nsteps):
            t.append((i * t_final) / (nsteps - 1))

        def ode(w, tau):
            third = self.__Fx(tau) / self.get_shape().mass()
            fourth = self.__Fy(tau) / self.get_shape().mass()
            return [w[2], w[3], third, fourth]
        init_cond = [self.__s.cm_x(), self.__s.cm_y(), self.__vx, self.__vy]
        return t, odeint(ode, init_cond, t)
```