Assignment 2 Solution

Rithvik Bhogadi and bhogadic

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

When testing the original program, the files *CircleT.py*, *TriangleT.py*, *BodyT.py*, *Scene.py* were tested. The result of the testing is that 19 of the test cases had passed which means that the original had no errors and all of the methods from the program have passed all the test cases. There were no failed test cases. The rational of the cases for the *CircleT.py* was to check if the method‘s output would match the output typed into the test cases. The testing consists of testing the exception handling ability of the python file. The rational of the test cases for the *TriangleT.py* follow the same rational as the *CircleT.py* class and checks its ability to handles exceptions as well. The rational for the *BodyT.py* file checks for what kind of shape that the object is made up of as well as throwing an expception if all objects are not of the same length. The rational for testing the *Scene.py* is checking if the shape, unbalanced forces and the initial velocity match the desired output.

2 Results of Testing Partner's Code

The results of testing partner file‘s combined with the original file‘s have been positive. The result of this testing exercise is that 19 of the test cases had passed which means that the original had no errors and all of the methods from the program have passed all the test cases. The partner file‘s *CircleT.py*, *TriangleT.py*, *BodyT.py*, *Scene.py* were tested. There are no problems with the original file‘s or the partner file‘s because both file‘s have no issue during compliation and are able to pass all of the test cases. This exercise teaches

an important concept which is regarding following the specifications listed by the MIS. This is because in doing so it allows for the program file's to be built in the best possible way. It is always helpful to have a guide and follow specified instructions as it only only allows for the quality of the program to be better. When comparing the results of this exercise between A1 and A2, there have been a couple of differences. In A1, in order to test the partner file's, instead of using *pytest*, the testing was done manually. This meant that each test case would have a desired output and each individual function would be tested. The problem with manual testing is that it takes longer than expected. In A2, in order to the partner file's we used *pytest*. The benefit of using *pytest* is that it less time consuming and as long as each test case method is pre-written and functionally working, the *pytest* will be of great help as it makes sure to test every test case method implemented at once.

3 Critique of Given Design Specification

The design specification of this assignment has been very clear and direct. This is because unlike the last assignment, this assignment had not allowed me to make multiple assumptions. Each step was clear and easy to understand. The specification for each module had stated the kind of methods that need to be implemented and even stated the type of exceptions that was required to be thrown in the case of an error. This assignment was informative as it had allowed me to learn about implementing and connecting different modules together while creating getters and setter in order to fund initialy velocites and unbalanced forces. By doing this assignment I have learnt on how to create graphs using *matplotlib* and as a result was able to accomplish one of the main tasks of this assignment. This assignment is beneficial for a student because it allows for the student to gian experience on how to implement functions that use mathematical notation to accomplish its functionality. A way in which the design specifications of this assignment may be improved is by trying to simplify the mathematical expressions in simpler expressions. Some of the mathematical expressions listed in the module specifications might not be easy to understand for students who may have never seen those expressiosn before. For example, the mathematical expressions within functions such as $sim()$, $mmom()$ and $cm()$ are examples of expressions that may be easy to comprehend. Another criticism of the design specifications is regarding *Plot.py*. This is because although there is no required test cases for *Plot.py*, in order to check on whether the correct graphs are created the specification does not specify on how to do so. The specification has to explain better about how to check and see if the correct graph is produced by stating that the $make_expt.py$ file may be used. Finally, the inclusion of *SciPy* module only leads to confusion for the students because some students may intepret as of they would be required to implement another

module even though it is not required.

4 Answers

a) No, it is not necessary to create a unit test for getters and setters. This is because the objective of unit testing to identify the real behaviour of the function. Getters and setters do not have an actual behaviour because they only set and get. Unless a getter or setter is does more than just set or get objects and actually consists of complex code, it is a waste of time to do unit testing for getters and setters.

b) In order to test the getters and setters of these state variables by testing the $sim()$ function within the *Scene.py*. This is because by testing the function, the function will use the state variable to achieve its functionality. In this scenario when testing, if the output from testing the function meets the desired output, then it means that the getters and setters have been able to get the value and set the value required.

c) In order to test *Plot.py* if required, export the graphs into images and then by comparing the images a conclusion can be made upon *Plot.py*. This can be done by importing *matplotlib* and then using $savefig()$ in order to save the graphs as images. This will alow for the tester to compare and spot any issues within the images.

d)
```
close_enough: seq of \mathbb{R} \rightarrow \beta
close_enough(x_{calc}, x_{true}) = ([+i : \mathbb{N} \mid 0..$nsteps$ - 1]
\epsilon
```

e) There does not have to be exception for negative coordinates because coordinates are relative to the origin. This means that negative coordinates are allowed. The sum of both numbers has to be positive and not equal to 0.

f) The state invariant is always satisified by the given specification because the specification is creating methods for a triangle. This means that a triangle must always have all of its sides have lengths greater than 0 and therefore, the state invariant will always be satisfied.

g)
```
s = [math.sqrt(x) for i in range(5,20)
if i % 2 == 0]
```

h)
```
   result = []
for ch in s:
    if not ch.isupper(): result.append(ch)
s = ''.join(result)
```

i) Abstraction is the process of removing object and hence creating a superset but instead, generality is the process of creating general concepts by abstracting properties. For example, if abstraction was piece of an object, then generality embodies the whole object with the inclusion of ther objects. This means that generality encapsulates abstraction as well. Overall, they are related because abstractions acts as a subset of generality.

j) The second scenario is better because we need low coupling in this case. This is because the second scenario allows for the user to create multiple sub modules before implementing a main module.

E Code for Shape.py

```
## @file Shape.py
#  @author Rithvik Bhogadi
#  @brief Defining different shapes for different physical objects
#  @date 02/16/2021

from abc import ABC, abstractmethod

## @brief Shapes provides an interface for circles, triangles and different types of shapes
## @details The method in the interface are abstract and need to be
#  overridden by the modules that inherit this interface


class Shape(ABC):

    @abstractmethod
    ## @brief a method which provides the x coordinate of the center mass
    def cm_x(self):
        pass

    @abstractmethod
    ## @brief a method which provides the y coordinate of the center mass
    def cm_y(self):
        pass

    @abstractmethod
    ## @brief a method which provides a mass
    def mass(self):
        pass

    @abstractmethod
    ## @brief a method that provides the inertia of the object
    def m_inert(self):
        pass
```

F Code for CircleT.py

```
## @file CircleT.py
#  @author Rithvik Bhogadi
#  @brief a module interface which inherits Shape.py and provides a CircleT shape object
#  @date 02/16/2021

from Shape import Shape

class CircleT(Shape):

    ## @brief constructor for class CircleT,
    # which consists of different parts of the object
    #  @param x represents the x coordinate of the shape
    #  @param y reprents the y coordinate of the shape
    #  @param r represents the radius of the shape
    #  @param m represents the mass of the shape
    def __init__(self, x, y, r, m):
        if not r > 0 and m > 0:
            raise ValueError
        self.x = x
        self.y = y
        self.r = r
        self.m = m


    ## @brief calculates the x coordinate of the shape
    #  @return return the x coordinate for CircleT
    def cm_x(self):
        return self.x

    ## @brief calculates the y coordinate of the shape
    #  @return return the y coordinate for CircleT
    def cm_y(self):
        return self.y

    ## @brief calculates the mass of the object
    #  @return returns the mass of the object
    def mass(self):
        return self.m

    ## @brief calculates the moment of inertia of the object
    #  @return returns the moment of inertia of the object
    def m_inert(self):
        return ((self.m*(self.r*self.r))/2)
```

G Code for TriangleT.py

```
## @file TriangleT.py
#  @author Rithvik Bhogadi
#  @brief a module interface which inherits Shape.py and provides a TriangleT shape object
#  @date 02/16/2021

from Shape import Shape

class TriangleT(Shape):

    ## @brief constructor for class TriangleT
    #  @param x represents the x coordinate
    #  @param y represents the y coordinate
    #  @param s represents the side length
    #  @param m represents the mass
    def __init__(self, x, y, s, m):
        if not s > 0 and m > 0:
            raise ValueError
        self.x = x
        self.y = y
        self.s = s
        self.m = m


    ## @brief calculates the x coordinate of the shape
    #  @return return the x coordinate for TriangleT
    def cm_x(self):
        return self.x

    ## @brief calculates the y coordinate of the shape
    #  @return return the y coordinate for TriangleT
    def cm_y(self):
        return self.y

    ## @brief calculates the mass of the object
    #  @return return the mass of the object
    def mass(self):
        return self.m

    ## @brief calculates the moment of inertia of the object
    #  @return returns the mmoment of inertia of the object
    def m_inert(self):
        return ((self.m*(self.s*self.s))/12)
```

H Code for BodyT.py

```
## @file BodyT.py
#  @author Rithvik Bhogadi
#  @brief Provides a sequence of mass in space
#  @date 02/16/2021

from Shape import Shape

class BodyT(Shape):

    ## @brief constructor for class BodyT
    #  @param cmx1 A x coordinate of the BodyT
    #  @param cmy1 A y coordinate of the BodyT
    #  @param m1 Provides one of the masses of BodyT
    def __init__(self, cmx1, cmy1, m1):

        def mmom(x,y,m):
            addo = [(m[i] * (x[i] ** 2 + y[i] ** 2)) for i in range(len(m))]
            return sum(addo)

        def cm(z,m):
            diver = [z[i] * m[i] for i in range(len(m))]
            return sum(diver) / sum(m)

        if not len(cmx1) == len(cmy1) == len(m1):
            raise ValueError
        self.cmx = cm(cmx1,m1)
        self.cmy = cm(cmy1,m1)
        self.m = sum(m1)
        self.moment = mmom(cmx1, cmy1, m1) - (self.m * (self.cmx ** 2 + self.cmy ** 2))


    ## @brief calculates the x coordinate of the shape
    #  @return return the x coordinate for BodyT
    def cm_x(self):
        return self.cmx

    ## @brief calculates the y coordinate of the shape
    #  @return return the x coordinate for BodyT
    def cm_y(self):
        return self.cmy

    ## @brief calculates the mass of the object
    #  @return return the mass of the object
    def mass(self):
        return self.m

    ## @brief calculates the moment of inertia of the object
    #  @return return the moment of inertia of the object
    def m_inert(self):
        return self.moment
```

I Code for Scene.py

```
## @file Scene.py
#  @author Rithvik Bhogadi
#  @brief A module interface which gets/sets shapes,unbalanced forces and initial velocities
#  @date  02/16/2021

from Shape import Shape
from scipy.integrate import odeint


class Scene():

    ## @brief constructor for class Scene
    #  @param s represents Shape
    #  @param Fx represnts the unbalanced force function in x dir
    #  @param Fy represents the unbalanced force function y dir
    #  @param vx represents the initial velocity in x dir
    #  @param vy represents the 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 gets the shape of the object
    #  @return returns the shape of the object
    def get_shape(self):
        return self.s

    ## @brief gets the unbalanced forces of the shape
    #  @return returns the unbalanced forces in the x and y dir
    def get_unbal_forces(self):
        return self.Fx, self.Fy

    ## @brief gets the initial velocity of the shape
    #  @return returns the initial velocities in the x and y dir
    def get_init_velo(self):
        return self.vx, self.vy

    ## @brief sets the shape
    #  @param s the shape of the object
    #  @return returns the newly set shape of the object
    def set_shape(self, s):
        self.s = s

    ## @brief sets the unbalanced forces in x dir and y dir
    #  @param Fx the force of the object in the x dir
    #  @param Fy the force of the object in the y dir
    #  @return returns the newly set forces of the object in the x dir and y dir
    def set_unbal_forces(self, Fx, Fy):
        self.Fx = Fx
        self.Fy = Fy

    ## @brief sets the initial velocities in x dir and y dir
    #  @param vx the initial velocity of the object in the x dir
    #  @param vy the initial velocity of the object in the y dir
    #  @return returns the newly set velocities of the object in the x dir and y dir
    def set_init_velo(self, vx, vy):
        self.vx = vx
        self.vy = vy

    def ode(self,w,t):
        return [w[2],w[3],self.Fx(t)/self.s.mass(),self.Fy(t)/self.s.mass()]

    ## @brief Simulating the different shapes
    #  @param t_final represents the final sequence of numbers
    #  @param n_steps represents the number of steps
    #  @return returns a sequence of numbers
    def sim(self,t_final,n_steps):
        t = []
        for i in range(n_steps):
            t.append((i*t_final)/(n_steps-1))
        return t, odeint(self.ode, [self.s.cm_x(),self.s.cm_y(),self.vx,self.vy], t)
```

J Code for Plot.py

```
## @file
#  @author
#  @brief
#  @date
#  @details

import matplotlib.pyplot as plt

def plot(w,t):
    if not (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, (xvt, yvt, yvx) = plt.subplots(3)
    fig.suptitle("Motion Simulation")
    yvx.plot(x, y)
    yvx.set(xlabel="x(m)", ylabel="y(m)")
    xvt.plot(t,x)
    xvt.set(xlabel="t(m)", ylabel="x(m)")
    yvt.plot(t,y)
    yvt.set(xlabel="t(m)", ylabel="y(m)")
    plt.show()
```

K Code for test_driver.py

```
## @file test_driver.py
#  @author Rithvik Bhogadi
#  @brief tests the methods from various modules
#  @date 02/16/2021

from CircleT import CircleT
from TriangleT import TriangleT
from BodyT import BodyT
from Scene import Scene

from pytest import *

class TestCircleT:

    # initialize an instance of Set for each test
    def setup_method(self, method):
        self.object1 = CircleT(1,2,3,45)

    # reset state variables
    def teardown_method(self, method):
        self.object1 = None

    # test_to_cmx method
    def test_to_cmx(self):
        assert self.object1.cm_x()==1

    # test_to_cmy method
    def test_to_cmy(self):
        assert self.object1.cm_y()==2

    # test_to_mass method
    def test_to_mass(self):
        assert self.object1.mass()==45

    # test_to_m_inert method
    def test_to_m_inert(self):
        assert self.object1.m_inert()==202.5

    def test_toexceptionr(self):
        with raises(ValueError):
            CircleT(1,2,-4,6)


class TestTriangleT:

    # initialize an instance of Set for each test
    def setup_method(self, method):
        self.object2 = TriangleT(2,4,5,6)

    # reset state variables
    def teardown_method(self, method):
        self.object2 = None

    # test_to_cmx method
    def test_to_cmx(self):
        assert self.object2.cm_x()==2


    def test_to_cmy(self):
        assert self.object2.cm_y()==4


    def test_to_mass(self):
        assert self.object2.mass()==6


    def test_to_m_inert(self):
        assert self.object2.m_inert()==12.5

    def test_toexceptions(self):
        with raises(ValueError):
            TriangleT(2,4,-5,7)

class TestBodyT:

    # initialize an instance of Set for each test
```

```
    def setup_method(self, method):
        self.cmxs = [3.0, 5.0, 2.0, 4.0]
        self.cmys = [6.0, 9.0, 5.0, 8.0]
        self.ms = [2.0, 3.0, 1.0, 4.0]
        self.object3 = BodyT(self.cmxs, self.cmys, self.ms)

    # reset state variables
    def teardown_method(self, method):
        self.object3 = None

    # test_to_cmx method
    def test_to_cmx(self):
        assert self.object3.cm_x()==(39.0/10.0)


    def test_to_cmy(self):
        assert self.object3.cm_y()==(76.0/10.0)


    def test_to_mass(self):
        assert self.object3.mass()==10.0

    def test_to_m_inert(self):
        assert self.object3.m_inert()==(757-729.7)

    def test_toexceptionlists(self):
        with raises(ValueError):
            self.cmxs = [3.0, 5.0, 2.0, 4.0]
            self.cmys = [6.0, 9.0, 5.0, 8.0]
            self.ms = [2.0, 3.0, 1.0]
            BodyT(self.cmxs, self.cmys, self.ms)

class TestScene:

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


    def Fy(self, t):
        g = 9.81  # accel due to gravity (m/s^2)
        m = 1  # mass (kg)
        return -g * m if t < 3 else g * m

    # initialize an instance of Set for each test
    def setup_method(self, method):
        self.object4 = Scene(CircleT(1,2,3,4), self.Fx, self.Fy, 4, 5)
    # reset state variables
    def teardown_method(self, method):
        self.object4 = None

    def test_getshape(self):
        assert self.object4.get_shape().cm_x() == CircleT(1,2,3,4).cm_x()

    def test_getinitvelo(self):
        assert self.object4.get_init_velo() == (4,5)

    def test_setshape(self):
        self.object4.set_shape(CircleT(2,3,4,5))
        assert self.object4.get_shape().cm_x() == CircleT(2,3,4,5).cm_x()

    def test_setinitvelo(self):
        self.object4.set_init_velo(79,300)
        assert self.object4.get_init_velo() == (79,300)
```

L Code for Partner's CircleT.py

```
## @file CircleT.py
#  @author Shiraz Masliah
#  @date February 16, 2021

from Shape import Shape


#  @brief CircleT is a subclass of Shape
class CircleT(Shape):

    ## @brief Circle constructor
    def __init__(self, x, y, r, m):

        #  @details Raises value error if the radius is less than zero
        if not(r > 0):
            raise ValueError("radius value must be greater 0")
        #  @details Raises value error if the mass is less than zero
        if not(m > 0):
            raise ValueError("mass value must be greater 0")

    #  @details Initialized circle objects
    #  @params x, y, radius, and mass
        self.x = x
        self.y = y
        self.r = r
        self.m = m
```

M Code for Partner's TriangleT.py

```
## @file TriangleT.py
#  @author Shiraz Masliah
#  @date February 16, 2021

from Shape import Shape


#  @brief TriangleT is a subclass of Shape
class TriangleT(Shape):

    ## @brief Triangle constructor
    def __init__(self, x, y, s, m):

        #  @details Raises value error if s is less than zero
        if not(s > 0):
            raise ValueError("s value must be greater 0")
        #  @details Raises value error if the mass is less than zero
        if not(m > 0):
            raise ValueError("mass value must be greater 0")

        #  @details Initialized triangle objects
        #  @params x, y, s, and mass
        self.x = x
        self.y = y
        self.s = s
        self.m = m

    ## @details Overrides Shape class method
    #  @returns 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 Shiraz Masliah
#  @date February 16, 2021

from Shape import Shape


#  @brief BodyT is a subclass of Shape
class BodyT(Shape):

    def __init__(self, x, y, m):

        #  @details Raises value error if the lengths
        #  of each list are not equal
        if not(len(x) == len(y) and len(y) == len(m)):
            raise ValueError("Lists not of equal lengths")
        #  @details Raises value error if any value in list m
        #  are less than zero
        for i in m:
            if not (i > 0):
                raise ValueError("m > 0")

        #  @details Initialized circle objects
        #  @params cmx, cmy, sum, and moment
        self.cmx = self.cm(x, m)
        self.cmy = self.cm(y, m)
        self.m = self.sum(m)
        squared = ((self.cm(x, m))**2 + (self.cm(y, m))**2)
        self.moment = self.mmom(x, y, m) - (self.sum(m)) * squared

    ## @brief Method for getting a number
    #  @returns cmx
    def cm_x(self):
        return self.cmx

    ## @brief Method for getting a number
    #  @returns cmy
    def cm_y(self):
        return self.cmy

    ## @brief Method for getting a number
    #  @returns mass
    def mass(self):
        return self.m

    ## @brief Method for getting a number
    #  @returns Moment of inertia
    def m_inert(self):
        return (self.moment)

    # @details local functions

    ## @brief Method adding numbers in list m
    #  @returns Summation of numbers in list m
    def sum(self, m):
        summation = 0
        for r in m:
            summation += r
        return summation

    ## @brief Method for finding cm
    #  @returns The multiple of each number in two lists
    #  and divides the product by the sum of all numbers in
    #  list m
    def cm(self, z, m):
        result = 0
        for i in range(len(m)):
            result += z[i] * m[i]
        return result / sum(m)

    ## @brief Method for finding the moment
    #  @returns the multiple of m by x and y squared
    #  for each number in each list
    def mmom(self, x, y, m):
        result = 0
        for i in range(len(m)):
```

```
    result += m[i] * (x[i]**2 + y[i]**2)
return result
```

O Code for Partner's Scene.py

```
## @file Scene.py
#  @author Shiraz Masliah
#  @date February 16, 2021

#  @brief CircleT is a subclass of Shape
#  @details uses library SciPy
from Shape import Shape
import scipy.integrate as spy


## @brief Scene constructor
class Scene(Shape):

    #  @details Initialized scene objects
    #  @params s, fx, fy, vx, and vy
    def __init__(self, s, fx, fy, vx, vy):
        self.s = s
        self.fx = fx
        self.fy = fy
        self.vx = vx
        self.vy = vy

    ## @brief Method for getting a number
    #  @returns s
    def get_shape(self):
        return self.s

    ## @brief Method for getting fx and fy
    #  @returns fx and fy
    def get_unbal_forces(self):
        return (self.fx, self.fy)

    ## @brief Method for getting vx and vy
    #  @returns vx and vy
    def get_init_velo(self):
        return (self.vx, self.vy)

    ## @brief Method defining s
    #  @returns s
    def set_shape(self, s):
        self.s = s

    ## @brief Method defining fx and fy
    def set_unbal_forces(self, fx, fy):
        self.fx = fx
        self.fy = fy

    ## @brief Method defining vx and vy
    def set_init_velo(self, vx, vy):
        self.vx = vx
        self.vy = vy

    ## @brief Method calculates the sim
    #  @returns t and ode list
    def sim(self, tfinal, nsteps):
        t = []
        for i in range(nsteps):
            t.append(i * tfinal / (nsteps - 1))
        return (t, spy.odeint(self.ode, [self.s.cm_x(), self.s.cm_y(), self.vx, self.vy], t))

    ## @brief Local function
    ## @returns a ode list
    def ode(self, w, t):
        result = [w[2], w[3], (self.fx(t) / self.s.mass()), (self.fy(t) / self.s.mass())]
        return result
```