Assignment 2 Solution

Daniel Guoussev-Donskoi

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

My tests were written in such a way that they tested every program with "logic" behind them extensively, but did minimal testing for getters and setters outside of checking for errors and exceptions. This is because the getter and setter functions didn't really have much "logic" behind them, while the more complex ones had potential boundary cases or exceptions to be thrown. My tests all passed up to the sim, which I was unable to develop a method of testing for. I greatly liked the progression of the modules, with us beginning with the most Abstract class, a general shape and making a CircleT object that was a form of shape, with the complexity slowly

2 Results of Testing Partner's Code

For the tests I made, all the exceptions were thrown as needed and the correct values were returned, with the main issues present being specifically rounding errors when I chose to check for a rounded value being returned and my partner did not.

3 Critique of Given Design Specification

While the formal notation was initially harder to interpret or access than one written in natural language, I found that once I began to understand MIS specification, it actually became far more effective at communicating details about the modules in question, specifically, the error cases for each, where exceptions would need to be raised as well as extreme clarity about the parameters that would be passed to each function and what was expected to be returned. I also found it extremely effective for the way certain modules inherited from other more abstract modules, and clearly illustrated how each should inherit from it's "parent" and behave.

4 Answers

a) As getters and setters don't really contain any "logic" they don't have to be unit tested because they're not executing a set of logical instructions, they're simply getting or setting what they're being told to set. Thus, a unit test will not really reveal any relevant information about a getter or a setter.

b) As said getters and setters are actually employed in the context of the Scene module, and set the unbalanced forces necessary for it, I would use Scene's built in method to get the unbalanced forces and compare it to an expected result for both Fx and Fy, thereby testing the getters and setters in question.

c) Matplotlib includes packages to compare two graphs or images produced by it, thus we could use compare images to examine a graph we had already produced against the one generated by Plot.py and return a boolean True or False depending on whether it was "close enough".

d) Math Specification: closeenough: (seq of $\mathbb{R}$ x seq of $\mathbb{R}$) $\rightarrow$Bool$closeenough = (+i$:R—$i\in$

[0..| x_{true} | - 1] : (x_{calc} - x_{true}) |) / (+i : $\mathbb{R}$ | i $\in$ [0..| x_{true} | - 1] : | x_{true} |) $< \epsilon$

e) Such an exception would be unnecessary as while the idea of "negative mass" or "negative length" are fundamentally nonsensical, we can have a negative position or speed, if we look at it in terms of velocity. After all, a body can move downwards when it's initial vertical velocity was positive, due to the effect of gravity. Thus, from a physics point of view and a programming point of view there is no need for such an exception.

f) The specification clearly outlined that an exception should be returned in the event that either m or s were not greater than zero. From this, we can logically determine that $m > 0$ and $s > 0$ as if either of them were not, value error would have been raised thus $m > 0$ and $s > 0$

g) sqrt(x) for x in range(5,20) if x mod 2 == 1

h)
```
def upperremover(string):
        purgedstring = " "
        for character in string:
            if (character.isupper() == False):
                purgedstring = purgedstring + character
        return purgedstring
```

i) Abstraction represents the idea of effectively ”ignoring unnecessary details” and focusing on a general template, that can represent multiple types of modules even if they differ on their specifics. Generality on the other hand, is the idea of representing multiple modules by focusing UPON the details and seeing what common details they have that makes the properties used in one similar to another.

j) It makes far more sense to have a single module inherited by many other modules than a module that inherits from many modules as while many inheriting one reduces complexity view abstraction, the other enhances it by adding many more moving parts to a single module which rather than simplifying anything, makes the module exponentially more complicated to understand and determine the issues with.

E Code for Shape.py

```
## @file Shape.py
#  @author Daniel Guoussev-Donskoi
#  @brief Contains a generic Shape type, to be inherited by others
#  @date 2021-02-16
from abc import ABC, abstractmethod

#@brief Shape is a class that implements a shape object containing an x and y coordinate pair
# as well as a mass and inertia variable which is then inherited by other objects

class Shape(ABC):
    #@brief Abstract method to initialises cm_x, inherited by other methods
    @abstractmethod
    def cm_x(self):
        pass
    #@brief Abstract method to intialise cm_y for y coordinate, inherited by other methods
    @abstractmethod
    def cm_y(self):
        pass
    #@brief Abstract method to initialise mass, inherited by other methods
    @abstractmethod
    def mass(self):
        pass
    #@brief Abstract method to initialise objects at inertia, inherited by other methods
    @abstractmethod
    def m_inert(self):
        pass
```

F Code for CircleT.py

```
## @file CircleT.py
#  @author Daniel Guoussev-Donskoi
#  @brief CircleT is a class that initialises a Circle object to be moved through a scene in motion
#  @date 2021-02-16
from Shape import Shape

#@brief CircleT creates a Circle object, defined by it's coordinates, mass and movement from inertia
class CircleT(Shape):
    #@brief Constructor for CircleT, represents a Circle object as a coordinate pair, mass and
        movement from inertia
    #@param A pair of coordinates, it's radius and it's mass
    #@throws If radius or mass aren't positive, throws a value error
    def __init__(self, x, y, r, m):
        self.x = x
        self.y = y
        self.r = r
        self.m = m
        if (r <= 0) or (m <= 0):
            raise ValueError
    #@brief Sets the x coordinate of the Circle
    def cm_x(self):
        return self.x
    #@brief Sets the y coordinate of the Circle
    def cm_y(self):
        return self.y
    #@brief Sets the mass of the Circle
    def mass(self):
        return self.m
    #@brief Sets the movement from inertia of the Circle
    def m_inert(self):
        m = self.m
        r = self.r
        final = m * (r ** 2) / 2
        return final
```

G Code for TriangleT.py

```
## @file TriangleT.py
#  @author Daniel Guoussev-Donskoi
#  @brief A class that defines an equilateral triangle
#  @date 2021-02-16
from Shape import Shape

#@brief A class that inherits shape and defines an equilateral triangle using coordinates, side length
    and mass
class TriangleT(Shape):
    #@brief A constructor that takes x and y coords, side length and mass to initialise a triangle
    #@param x and y coordinates, side length, mass
    #@throws Throws an exception if side length or mass aren't positive, giving a Value Error
    def __init__(self, x, y, s, m):
        self.x = x
        self.y = y
        self.s = s
        self.m = m
        if (s <= 0 or m <= 0):
            raise ValueError
    #@brief Getter for x coordinate
    def cm_x(self):
        return self.x
    #@brief Getter for y coordinate
    def cm_y(self):
        return self.y
    #@brief Getter for mass
    def mass(self):
        return self.m
    #@brief Getter for movement from inertia
    def m_inert(self):
        m = self.m
        s = self.s
        final = m * (s ** 2) / 12
        return final
```

H Code for BodyT.py

```
## @file BodyT.py
#  @author Daniel Guoussev-Donskoi
#  @brief A class that defines a body of no particular shape
#  @date 2021-02-16
from Shape import Shape

#@brief A helper function that sums up a list of values
#@param A list m of numbers
def sum(m):
    total = 0
    for i in m:
        total = total + i
    return total


def cm(z, m):
    sum1 = sum(m)
    sum2 = 0
    length = len(z)
    for i in range(0, length):
        sum2 = sum2 + (z[i] * m[i])
    final = sum2 / sum1
    return final


def mmom(x, y, m):
    sum1 = 0
    for i in range(0, len(x)):
        sum1 = sum1 + (m[i] * ((x[i] ** 2) + (y[i] ** 2)))
    return sum1

#@brief A class that defines a Body object,
#@param Inherits from shape like triangle and circle
class BodyT(Shape):
    #@brief A Constructor that creates a Body object
    #@param Takes in coordinate sets of x and y  and the mass of the object
    def __init__(self, x, y, m):
        self.x = x
        self.y = y
        self.m = m
    #@brief A getter function that returns cm_x coordinates and masses
    def cm_x(self):
        x = self.x
        m = self.m
        return cm(x, m)
    #@brief a getter function that returns cm_y coordinates and masses
    def cm_y(self):
        y = self.x
        m = self.m
        return cm(y, m)
    #@brief a Getter function that returns the total mass
    def mass(self):
        m = self.m
        return sum(m)
    #@brief a getter function that returns the momentum from inertia
    def m_inert(self):
        m = self.m
        x = self.x
        y = self.y
        final = mmom(x, y, m) - (sum(m) * ((cm(x, m) ** 2) + (cm(y, m) ** 2)))
        return final
```

I Code for Scene.py

```
## @file Scene.py
#  @author Daniel Guoussev-Donskoi
#  @brief A Scene method that simulates the movement of an object
#  @date 2021-02-16
#  @details This class simulates a Scene where an object moves through space over time
from scipy.integrate import odeint

#@brief A Scene class
class Scene:
        #@brief A constructor for Scene, which initialises it
        #@param Takes in a Shape, and a set of unbalanced force functions and initial velocities
        def __init__(self,s,Fx,Fy,vx,vy):
                self.s = s
                self.Fx = Fx
                self.Fy = Fy
                self.vx = vx
                self.vy =vy
        #@brief A getter function to get the shape passed to the Scene
        def get_shape(self):
                s = self.s
                return s
        #@brief A getter function to get the force functions passed to the Scene
        def get_unbal_forces(self):
                Fx = self.Fx
                Fy = self.Fy
                return Fx,Fy
        #@brief A getter function to get the initial velocity passed to the Scene
        def get_init_velo(self):
                vx = self.vx
                vy = self.vy
                return vx,vy
        #@brief A setter function to set the shape used by a Scene object to a new shape
        #@param the new shape being set
        def set_shape(self,change):
                self.s = change
        #@brief A setter function to set the unbalanced forces to new ones
        #@param the new unbalanced forces being set
        def set_unbal_forces(self,changefx,changefy):
                self.Fx = changefx
                self.Fy = changefy
        #@brief A setter function to set the initial velocities to new ones
        #@param The new initial velocities being passed to set_init
        def set_init_velo(self,changevx,changevy):
                self.vx = changevx
                self.vy = changevy
        #@brief A simulation function, to simulate movement through the scene
        #@param the ending position of the object, as well as the number of moves to get there
        def sim(self,finalpos,moves):
                times = []
                for j in range(0,moves):
                        times.append((j * finalpos)/(moves - 1))
                final = odeint(self.__ode, [self.s.cm_x(),self.s.cm_y(),self.vx,self.vy],times)
                return times, final

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

J Code for Plot.py

```
## @file Plot.py
#  @author Daniel Guoussev-Donskoi
#  @brief A plot module that plots the Scene depicted in Scene.py
#  @date 2021-02-16
#  @details Uses matplotlib to illustrate a set of coordinates, based on the velocities of objects
     passed to it
import matplotlib.pyplot as plt

#@brief A plot function that takes in both a scene and a time to plot it over
#@param The scene passed to the function, and the time over which to plot it
def plot(w, t):
    x = []
    y = []
    for i in range(len(w)):
        x.append(w[i][0])
        y.append(w[i][1])
    fig, axs = plt.subplots(3)
    fig.suptitle("Motion")
    fig.tight_layout()
    axs[0].plot(t, x)
    axs[0].set_ylabel("x(m)")
    axs[0].set_xlabel("t(s)")
    axs[1].plot(t, y)
    axs[1].set_ylabel("x(m)")
    axs[1].set_xlabel("t(s)")
    axs[2].plot(x, y)
    axs[2].set_ylabel("x(m)")
    axs[2].set_xlabel("t(s)")
```

K Code for test_All.py

L Code for Partner's CircleT.py

```
## @file CircleT.py
#  @author Prakarsh Kamal
#  @brief CircleT module
#  @date 15 Feb 2021

from Shape import Shape

## @brief CircleT is a class that implements a Shape object


class CircleT(Shape):

    ## @brief constructor method for class CircleT,
    #  initializes a circle from 4 parameters
    #  @details The arguments provided to the access programs will be of the correct type
    #  @param x is the x-coordinate of circle
    #  @param y is the y-coordinate of circle
    #  @param r is the radius of circle
    #  @param m is the mass of circle
    #  @throws ValueError if r not > 0 and m not > 0
    def __init__(self, x, y, r, m):
        self.x = x
        self.y = y
        self.r = r
        self.m = m

        if not (self.r > 0 and self.m > 0):
            raise ValueError

    ## @brief getter method to get x coordinate of centre of mass
    #  @return returns a real
    def cm_x(self):
        return self.x

    ## @brief getter method to get y coordinate of centre of mass
    #  @return returns a real
    def cm_y(self):
        return self.y

    ## @brief getter method to get mass
    #  @return returns a real
    def mass(self):
        return self.m

    ## @brief method to calculate inertia
    #  @return returns a real
    def m_inert(self):
        return (self.m * self.r ** 2) / 2
```

M Code for Partner's TriangleT.py

```
## @file TriangleT.py
#  @author Prakarsh Kamal
#  @brief TriangleT module
#  @date 15 Feb 2021

from Shape import Shape

## @brief TriangleT is a class that implements a Shape object


class TriangleT(Shape):

    ## @brief constructor method for class TriangleT,
    #  initializes a triangle from 4 parameters
    #  @details The arguments provided to the access programs will be of the correct type
    #  @param x is the x-coordinate of triangle
    #  @param y is the y-coordinate of triangle
    #  @param s is the side length of triangle
    #  @param m is the mass of triangle
    #  @throws ValueError if s not > 0 and m not > 0
    def __init__(self, x, y, s, m):
        self.x = x
        self.y = y
        self.s = s
        self.m = m

        if not (self.s > 0 and self.m > 0):
            raise ValueError

    ## @brief getter method to get x coord of centre of mass
    #  @return returns a real
    def cm_x(self):
        return self.x

    ## @brief getter method to get y coord of centre of mass
    #  @return returns a real
    def cm_y(self):
        return self.y

    ## @brief getter method to get mass
    #  @return returns a real
    def mass(self):
        return self.m

    ## @brief method to calculate inertia
    #  @return returns a real
    def m_inert(self):
        return (self.m * self.s ** 2) / 12
```

N Code for Partner's BodyT.py

```
## @file BodyT.py
#  @author Prakarsh Kamal
#  @brief BodyT module
#  @date 15 Feb 2021

from Shape import Shape

## @brief BodyT is a class that implements a Shape object


class BodyT(Shape):

    ## @brief constructor method for class BodyT,
    #  initializes a body from 3 parameters
    #  @details The arguments provided to the access programs will be of the correct type
    #  @param x is a sequence of reals (x coords)
    #  @param y is a sequence of reals (y coords)
    #  @param m is a sequence of reals (mass)
    #  @throws ValueError if len of x, y, m is not equal
    #  @throws ValueError if elements of m not > 0
    def __init__(self, x, y, m):
        if not(len(x) == len(y) == len(m)):
            raise ValueError
        for i in m:
            if i <= 0:
                raise ValueError
        else:
            self.cmx = cm(x, m)
            self.cmy = cm(y, m)
            self.m = sum(m)
            self.moment = mmom(x, y, m) - sum(m) * (cm(x, m)**2 + cm(y, m)**2)

    ## @brief getter method to get x coord of centre of mass
    #  @return returns a real
    def cm_x(self):
        return self.cmx

    ## @brief getter method to get y coord of centre of mass
    #  @return returns a real
    def cm_y(self):
        return self.cmy

    ## @brief getter method to get mass
    #  @return returns a real
    def mass(self):
        return self.m

    ## @brief method to calculate inertia
    #  @return returns a real
    def m_inert(self):
        return self.moment


## @brief local function
#  @param z is a sequence of reals
#  @param m is a sequence of reals
#  @return returns a real
def cm(z, m):
    count = 0
    for i in range(len(m)):
        ans = z[i] * m[i]
        count += ans
    return count / sum(m)


## @brief local function
#  @param x is a sequence of reals
#  @param y is a sequence of reals
#  @param m is a sequence of reals
#  @return returns a real
def mmom(x, y, m):
    count = 0
    for i in range(len(m)):
        ans = m[i] * (x[i]**2 + y[i]**2)
        count += ans
    return count
```

O Code for Partner's Scene.py

```python
## @file Scene.py
#  @author Prakarsh Kamal
#  @brief Scene module
#  @date 15 Feb 2021

from scipy.integrate import odeint

## @brief Scene is a class that implements Shape and odeint
#  to simulate scene motion


class Scene:

    ## @brief constructor method for class Scene
    #  initializes Scene from 5 parameters
    #  @param s is a Shape instance
    #  @param fx is unbalanced force in x direction
    #  @param fy is unbalanced force in y direction
    #  @param vx is initial velocity in x direction
    #  @param vy is 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 getter method to get shape instance
    #  @return returns instance of type Shape
    def get_shape(self):
        return self.s

    ## @brief getter method to get unbalanced forces in x and y directions
    #  @return returns two functions (real to real)
    def get_unbal_forces(self):
        return self.fx, self.fy

    ## @brief getter method to get initial velocity in x and y directions
    #  @return returns 2 reals
    def get_init_velo(self):
        return self.vx, self.vy

    ## @brief setter method (mutator) for shape in current scene
    #  @param s is Shape instance
    def set_shape(self, s):
        self.s = s

    ## @brief setter method (mutator) for forces in current scene
    #  @param fx is a function (real to real)
    #  @param fy is a function (real to real)
    def set_unbal_forces(self, fx, fy):
        self.fx = fx
        self.fy = fy

    ## @brief setter method (mutator) for velocities in current scene
    #  @param vx is velocity in x direction in current scene
    #  @param vy is velocity in y direction in current scene
    def set_init_velo(self, vx, vy):
        self.vx = vx
        self.vy = vy

    ## @brief method to implement simulation of scene
    #  @param t_final is a real (final time)
    #  @param nsteps is a natural (no. of steps)
    #  @return returns a sequence of reals (t) and
    #  returns a sequence of sequence length 4 of reals
    def sim(self, t_final, nsteps):
        t = []
        for i in range(0, nsteps - 1):
            ans = (i * t_final) / (nsteps - 1)
            t += [ans]
        return t, odeint(self.__ode, [self.s.cm_x(), self.s.cm_y(), self.vx, self.vy], t)

    def __ode(self, w, t):
        return [w[2], w[3], self.fx(t) / self.s.mass(), self.fy(t) / self.s.mass()]
```