Assignment 1 Solution

Hriday Jham, jhamh

January 28, 2021

This report discusses testing of the `ComplexT` and `TriangleT` classes written for Assignment 1. It also discusses testing of the partner's version of the two classes. The design restrictions for the assignment are critiqued and then various related discussion questions are answered.

1 Assumptions and Exceptions

ComplexT:

1. I assumed that the real and imaginary parts would be represented in float numbers.
2. For get_phi, since The phase of a complex number with both real and imaginary parts as 0 is undefined, I returned None if both real and imaginary parts of the complex are 0
3. Since the reciprocal of 0 is undefined, I assumed the reciprocal of 0 complex number to be None in recip
4. since any number when divided by 0 is undefined, I assumed the quotient of a division by 0 to be None in div

TraingleT:

1. I assumed that the lengths of all three sides of the triangle would be represented by integers
2. I assumed that a triangle is invalid if any of the sides is less than or equal to 0 in is_valid
3. I assumed that the triangle is invalid if the sum of any two sides is not greater than the third side in is_valid
4. Assumed that a triangle which is both scalene and right to be a right triangle.

2 Test Cases and Rationale

Test Cases for ComplexT functions:

1. real function: Tested to real function against complex numbers with a positive, negative and 0 real part
2. imag function: Tested to imag function against complex numbers with a positive, negative and 0 imaginary part
3. get_r function: Tested to get_r function for a positive, negative and 0 complex number.
4. get_phi function: Tested get_phi function for a complex number with real part ¿ 0 or y != 0 with complex number 5 - 3i, for a complex number with real part ¡ 0 and imaginary part = 0 with complex number -3 + 0i. I also tested that 0 + 0i complex number returns None.
5. equal function: Tested equal function with two equal complex numbers and two unequal complex numbers.
6. conj function: Tested conj function for three different complex numbers.
7. add function: Tested add function for three different pairs of complex numbers.
8. sub function: Tested sub function for three different pairs of complex numbers.
9. mult function: Tested mult function for three different pairs of complex numbers.
10. recip function: Tested recip function for three different complex numbers.
11. div function: Tested div function for three different pairs of complex numbers, one of which was a 0 complex number to test if it returns None.
12. sqrt function: Tested sqrt function against complex numbers with a positive, negative and 0 imaginary part.

Test Cases for TriangleT functions:

1. get_sides function: Tested get_sides function against two different triangles
2. equal function: Tested equal function against a pair of equal triangles and a pair of unequal triangles.

3. perim function: Tested perim function for two different pairs of triangles.

4. area function: Tested area function for two different pairs of triangles.

5. is_valid test: Tested is_valid function for two invalid triangles and a valid triangle.

6. tri_type function: Tested tri_type function for an equilateral, isosceles, scalene and right triangle.

3 Results of Testing Partner's Code

Partner's code failed three tests against my test_driver.

div and recip function failed for ComplexT because my test_driver was searching for None in undefined cases, but the partner's code raised exceptions in such cases.

tri_type function failed for partner's code as their code had spelled isosceles wrong. It was spelled isoceles instead of isosceles.

4 Critique of Given Design Specification

The only flaw I found with the Design specification was that it was not clearly defined whether we should raise exceptions in undefined cases or return None.

5 Answers to Questions

(a) ComplexT mutators:

- Constructor

ComplexT selectors:

- real
- imag
- get_r
- get_phi
- equal
- conj
- add

- sub
- mult
- recip
- div
- sqrt

TriangleT mutators:

- constructor

TriangleT selectors:

- get_sides
- equal
- perim
- area
- is_valid
- tri_type

(b) The two options for the state variables for these classes is to either use class variables or instance variables. In my files, I implemented instance variables for both ADTs

(c) Since complex numbers are vector quantities, implementing a greater than or less than function would only compare the magnitude of both complex numbers, and that does not tell us much.

(d) Yes, it is possible that three inputs given to TriangleT do not form a valid triangle in the following cases.

- One of the sides is negative or 0.
- The sum of any two sides is nto greater than the third.

In such cases, the constructor should call the is_valid function implemented in the class, and if the function returns False, it should raise an error.

(e) It would be a good idea, because in such a case we could call the tri_type function implemented inside the constructor and the TriType variable would be a class instance.

(f) There is a direct relationship between performance and usability. If the performance of a program is inefficient, it would affect the usability and understandability of the code.

(g) Faking a design process is required by programmers when they are not sure on how to proceed with their program and come up with a proper design. This is usually done when writing big and complex projects and programmers usually understand the working of the design process along the way. This can be prevented when writing a code where the programmer completely understands the working of the project and has a clear design process that is to be followed.

(h) If we are trying to reuse a code which is not correct or even has minor bugs in it can drastically affect the whole project if we reuse it in many parts of the project. Therefore, it is necessary to perform rigorous unit testing on the parts of code that we are reusing. Even such, there could still be an edge case or a bug that we failed to test which could affect the reliability of the program.

(i) Some examples that programming languages are abstractions built on top of hardware are:

   - Operating Systems are a good example, as the actions we perform are to be converted to 0s and 1s and performing actions would be almost unusable without compilers and interpreters.
   - Washing Machines are another example as there are so many operations performed by a washing machine such as washing, drying, soaking etc. and they are programmed to be easy to use even by someone who has no idea about the hardware underneath.

F Code for complex_adt.py

```
## @file complex_adt.py
#  @author Hriday Jham
#  @brief Contains the ComplexT type to represent Complex
#  numbers with their real and imaginary parts
#  @date 02/21/2021
import math

## @brief ComplexT is a class that implements an ADT for the
#  mathematical concept of Complex Numbers
#  @details the ADT contains the real and imaginary parts of
#  complex number

class ComplexT:

    ## @brief constructor method for ComplexT
    #  @param x a float indicating the real part of the complex
    #  number
    #  @param y a float indicating the imaginary part of the
    #  complex number

    def __init__(self, x, y):
        self.x = x
        self.y = y

    ## @brief get the real part of the complex number
    #  @return a float indicating the real part

    def real(self):
        return self.x

    ## @brief get the imaginary part of the complex number
    #  @return a float indicating the imaginary part

    def imag(self):
        return self.y

    ## @brief get the absolute value of the complex number
    #  @return a float indicating the absolute value

    def get_r(self):
        return math.sqrt(math.pow(self.x,2) + math.pow(self.y,2))

    ## @brief get the phase of the complex number (in radians)
    #  @return a float indicating the phase

    def get_phi(self):
        if self.x > 0 or self.y != 0:
            return (2 * math.atan(self.y / (self.get_r() + self.x)))
        elif self.x < 0 and self.y == 0:
            return math.pi
        elif self.x == 0 and self.y == 0:
            return None

    ## @brief check if two complex numbers are equal
    #  @details two complex numbers are equal if their real and imaginary
    #  parts are equal
    #  @param complex_no a ComplexT object to compare with
    #  @return boolean indicating whether two complex numbers are equal

    def equal(self, complex_no):
        if self.x == complex_no.x and self.y == complex_no.y:
            return True
        else:
            return False

    ## @brief get the conjugate of the complex number
    #  @details a conjugate of a number is the reflection of the
    #  complex number about the real axis
    #  @return a float indicating the conjugate

    def conj(self):
        return ComplexT(self.x, -self.y)

    ## @brief get the sum of two complex numbers
    #  @details the sum of two complex numbers is the sum of both the
    #  real parts and imaginary parts.
```

```python
#  @param complex_no a ComplexT object to add
#  @return ComplexT object indicating the sum

def add(self, complex_no):
    return ComplexT(self.x + complex_no.x, self.y + complex_no.y)

## @brief get the difference of two complex numbers
#  @details the difference of two complex numbers is the difference of both the
#  real parts and imaginary parts.
#  @param complex_no a ComplexT object to subtract
#  @return ComplexT object indicating the difference

def sub(self, complex_no):
    return ComplexT(self.x - complex_no.x, self.y - complex_no.y)

## @brief get the product of two complex numbers
#  @param complex_no a ComplexT object to multiply
#  @return ComplexT object indicating the product

def mult(self, complex_no):
    return ComplexT((self.x * complex_no.x - self.y * complex_no.y), (self.x * complex_no.y +
        self.y * complex_no.x))

## @brief get the reciprocal of the complex number
#  @return ComplexT object indicating the reciprocal

def recip(self):
    if self.x == 0 and self.y == 0:
        return None
    else:
        real_part = self.x / (math.pow(self.x, 2) + math.pow(self.y, 2))
        imag_part = -1 * self.y / (math.pow(self.x, 2) + math.pow(self.y, 2))
        return ComplexT(real_part, imag_part)

## @brief get the quotient of two complex numbers
#  @param complex_no a ComplexT object to divide
#  @return ComplexT object indicating the quotient

def div(self, complex_no):
    if complex_no.equal(ComplexT(0.0, 0.0)):
        return None
    else:
        return self.mult(complex_no.recip())

## @brief get the positive square root of two complex numbers
#  @return a float indicating the root

def sqrt(self):
    sign = 0.0
    if self.y < 0:
        sign = -1.0
    elif self.y > 0:
        sign = 1.0
    else:
        sign = 1.0
    real_part = math.sqrt((self.x + self.get_r()) / 2)
    imag_part = sign * math.sqrt(( -self.x + self.get_r()) / 2)
    return ComplexT(real_part, imag_part)
```

G Code for triangle_adt.py

```
## @file triangle_adt.py
#  @author Hriday Jham
#  @brief Contains the TriangleT type to represent all three
#  sides of a triangle
#  @date 01/21/2021

import math
from enum import Enum, auto

## @brief TriangleT is a class that implements an ADT for the
#  mathematical concept of triangles.
#  @details the ADT contains the three sides of a triangle

class TriangleT:

    ## @brief constructor method for TriangleT
    #  @param x a float indicating first side of the Triangle
    #  @param y a float indicating second side of the Triangle
    #  @param z a float indicating third side of the Triangle

    def __init__(self, x, y, z):
        self.side1 = x
        self.side2 = y
        self.side3 = z

    ## @brief get the three sides of the triangle
    #  @return a tuple containing the three sides.

    def get_sides(self):
        return (self.side1, self.side2, self.side3)

    ## @brief check if two triangles are equal
    #  @details two triangles are equal if their sides are equal
    #  @param new_triangle a TriangleT object to compare with
    #  @return boolean indicating whether two triangles are equal

    def equal(self, new_triangle):
        tuple1 = self.get_sides()
        tuple2 = new_triangle.get_sides()
        if sorted(tuple1) == sorted(tuple2):
            return True
        else :
            return False

    ## @brief get the perimeter of the triangle
    #  @details the perimeter of as triangle is the sum of all three sides
    #  @return float indicating the perimeter

    def perim(self):
        return self.side1 + self.side2 + self.side3

    ## @brief get the area of the triangle
    #  @details the area of a triangle is the area it covers
    #  @return float indicating the area

    def area(self) :
        s = self.perim() / 2
        area = math.sqrt(s * (s - self.side1) * (s - self.side2) * (s - self.side3))
        return area

    ## @brief checks if the three sides form a valid triangle
    #  @details the sum of any two sides of a triangle should be greater than
    #  its third side. Also the length of a side cannot be negative or zero.
    #  @return boolean indicating whether the triangle is valid.

    def is_valid(self) :
        if self.side1 <= 0 or self.side2 <= 0 or self.side3 <= 0:
            return False
        elif (self.side1 + self.side2 > self.side3) and (self.side1 + self.side3 > self.side2) and
             (self.side2 + self.side3 > self.side1) :
            return True
        else :
            return False

    ## @brief checks the type of triangle.
    #  @details A triangle is equilateral if all three sides are equal, Isosceles
```

```python
    #  if two sides are equal, scalene if all three sides have different length, and
    #  right if one angle of the triangle is 90 degrees (pi/2 radian)
    #  @return TriType returning the type of triangle

    def tri_type(self):
        if self.side1 == self.side2 and self.side1 == self.side3:
            return TriType.equilat
        elif self.side1 == self.side2 or self.side1 == self.side3 or self.side2 == self.side3:
            return TriType.isosceles
        else:
            sides_list = sorted(self.get_sides())
            if (pow(sides_list[0], 2) + pow(sides_list[1], 2) == pow(sides_list[2], 2)):
                return TriType.right
            else :
                return TriType.scalene

## @brief TriType contains an enum for the types of triangles.

class TriType(Enum):
    equilat = auto()
    isosceles = auto()
    scalene = auto()
    right = auto()
```

H Code for test_driver.py

```
## @file test_driver.py
#  @author Hriday Jham
#  @brief File containing tests for all the functions implemented
#  in ComplexT and TriangleT
#  @date 01/21/2020

import math
from complex_adt import ComplexT
from triangle_adt import TriangleT, TriType

real_test = True
imag_test = True
get_r_test = True
get_phi_test = True
complex_equal_test = True
conj_test = True
add_test = True
sub_test = True
mult_test = True
recip_test = True
div_test = True
sqrt_test = True
get_sides_test = True
tri_equal_test = True
perim_test = True
area_test = True
is_valid_test = True
tri_type_test = True


print("ComplexT function tests:")

a = ComplexT(1.0, 2.0)
c = a.real()
if c != 1:
    real_test = False

a = ComplexT(-1.0, 2.0)
c = a.real()
if c != -1:
    real_test = False

a = ComplexT(0.0, 0.0)
c = a.real()
if c != 0:
    real_test = False

if real_test == True:
    print("real test passes")
else :
    print("real test FAILS")
#

a = ComplexT(1.0, 2.0)
c = a.imag()
if c != 2:
    imag_test = False

a = ComplexT(-1.0, -2.0)
c = a.imag()
if c != -2:
    imag_test = False

a = ComplexT(0.0, 0.0)
c = a.imag()
if c != 0:
    imag_test = False

if imag_test == True:
    print("imag test passes")
else:
    print("real test FAILS")

a = ComplexT(1.0, 2.0)
c = a.get_r()
if c != math.sqrt(5):
```

```
    get_r_test = False

a = ComplexT(-1.0, -2.0)
c = a.get_r()
if c != math.sqrt(5):
    get_r_test = False

a = ComplexT(0.0, 0.0)
c = a.get_r()
if c != 0:
    get_r_test = False

if get_r_test == True:
    print("get_r test passes")
else:
    print("get_r test FAILS")

a = ComplexT(5.0, -3.0)
c = a.get_phi()
if c != -0.5404195002705842:
    get_phi_test = False

a = ComplexT(-3.0, 0.0)
c = a.get_phi()
if c != math.pi:
    get_phi_test = False

#a = ComplexT(0.0, 0.0)
#c = a.get_phi()
if c != None:
    get_phi_test = False

#if get_phi_test == True:
#     print("get_phi test passes")
#else:
#     print("get_phi test FAILS")

a = ComplexT(-3.0, 0.0)
b = ComplexT(-3.0, 0.0)
c = a.equal(b)
if c == False:
    complex_equal_test = False

a = ComplexT(0.0, 0.0)
b = ComplexT(1.0, 3.0)
c = a.equal(b)
if c != False:
    complex_equal_test = False

if complex_equal_test == True:
    print("equal test passes")
else:
    print("equal test FAILS")

a = ComplexT(5.0, -3.0)
c = a.conj()
if c.equal(ComplexT(5.0, 3.0)) == False:
    conj_test = False

a = ComplexT(-3.0, 0.0)
c = a.conj()
if c.equal(ComplexT(-3.0, 0.0)) == False:
    conj_test = False

a = ComplexT(1.0, 1.0)
c = a.conj()
if c.equal(ComplexT(1.0, -1.0)) == False:
    conj_test = False

if conj_test == True:
    print("conj test passes")
else:
    print("conj test FAILS")

a = ComplexT(5.0, -3.0)
b = ComplexT(-3.5, 4.5)
c = a.add(b)
if c.equal(ComplexT(1.5, 1.5)) == False:
    add_test = False
```

```python
a = ComplexT(-3.0, 0.0)
b = ComplexT(-3.0, 45.0)
c = a.add(b)
if c.equal(ComplexT(-6.0, 45.0)) == False:
    add_test = False

a = ComplexT(1.0, 1.0)
b = ComplexT(-1.0, -1.0)
c = a.add(b)
if c.equal(ComplexT(0.0, 0.0)) == False:
    add_test = False

if add_test == True:
    print("add test passes")
else:
    print("add test FAILS")

a = ComplexT(5.0, -3.0)
b = ComplexT(-3.5, 4.5)
c = a.sub(b)
if c.equal(ComplexT(8.5, -7.5)) == False:
    sub_test = False

a = ComplexT(-3.0, 0.0)
b = ComplexT(-3.0, 45.0)
c = a.sub(b)
if c.equal(ComplexT(0.0, -45.0)) == False:
    sub_test = False

a = ComplexT(1.0, 1.0)
b = ComplexT(-1.0, -1.0)
c = a.sub(b)
if c.equal(ComplexT(2.0, 2.0)) == False:
    sub_test = False

if sub_test == True:
    print("sub test passes")
else:
    print("sub test FAILS")

a = ComplexT(5.0, -3.0)
b = ComplexT(-3.5, 4.5)
c = a.mult(b)
if c.equal(ComplexT(-4.0, 33.0)) == False:
    mult_test = False

a = ComplexT(-3.0, 0.0)
b = ComplexT(-3.0, 45.0)
c = a.mult(b)
if c.equal(ComplexT(9.0, -135.0)) == False:
    mult_test = False

a = ComplexT(1.0, 1.0)
b = ComplexT(-1.0, -1.0)
c = a.mult(b)
if c.equal(ComplexT(0.0, -2.0)) == False:
    mult_test = False

if mult_test == True:
    print("mult test passes")
else:
    print("mult test FAILS")

a = ComplexT(5.0, -3.0)
c = a.recip()
if c.equal(ComplexT(5/34, 3/34)) == False:
    recip_test = False

a = ComplexT(-3.0, 0.0)
c = a.recip()
if c.equal(ComplexT(-1/3, 0.0)) == False:
    recip_test = False

a = ComplexT(1.0, 1.0)
c = a.recip()
if c.equal(ComplexT(1/2, -1/2)) == False:
    recip_test = False

if recip_test == True:
    print("recip test passes")
```

```
else:
    print("recip test FAILS")

a = ComplexT(5.0, -3.0)
b = ComplexT(-3.5, 4.5)
c = a.div(b)
if c.equal(ComplexT(-0.953846153846154, -0.3692307692307692)) == False:
    div_test = False

a = ComplexT(0.0, 0.0)
b = ComplexT(-3.0, 45.0)
c = a.div(b)
if c.equal(ComplexT(0.0, 0.0)) == False:
    div_test = False

a = ComplexT(1.0, 1.0)
b = ComplexT(0.0, 0.0)
c = a.div(b)
if c != None:
    div_test = False

if div_test == True:
    print("div test passes")
else:
    print("div test FAILS")

a = ComplexT(5.0, -3.0)
c = a.sqrt()
if c.equal(ComplexT(2.3271175190399496, -0.644574237324647)) == False:
    sqrt_test = False
    print("1")

a = ComplexT(-3.0, 0.0)
c = a.sqrt()
if c.equal(ComplexT(0.0, 1.7320508075688772)) == False:
    sqrt_test = False
    print("2")

a = ComplexT(1.0, 1.0)
c = a.sqrt()
if c.equal(ComplexT(1.09868411346781, 0.4550898605622274)) == False:
    sqrt_test = False
    print("3")

if sqrt_test == True:
    print("sqrt test passes")
else:
    print("sqrt test FAILS")

print("")
print("TriangleT tests:")

a = TriangleT(4, 5, 6)
c = a.get_sides()
if c != (4, 5, 6):
    get_sides_test = False

a = TriangleT(3,5, 6)
c = a.get_sides()
if c != (3, 5, 6):
    get_sides_test = False

if get_sides_test == True:
    print("get_sides test passes")
else:
    print("get_sides test FAILS")

a = TriangleT(4, 5, 6)
b = TriangleT(6, 4, 5)
c = a.equal(b)
if c != True:
    tri_equal_test = False

a = TriangleT(3,5, 6)
b = TriangleT(3,5,7)
c = a.equal(b)
if c != False:
    tri_equal_test = False

if tri_equal_test == True:
```

```
    print("equal test passes")
else:
    print("equal test FAILS")

a = TriangleT(4, 5, 6)
c = a.perim()
if c != 15:
    perim_test = False

a = TriangleT(3,5, 6)
c = a.perim()
if c != 14:
    perim_test = False

if perim_test == True:
    print("perim test passes")
else:
    print("perim test FAILS")

a = TriangleT(4, 5, 6)
c = a.area()
if c != 9.921567416492215:
    area_test = False

a = TriangleT(3,5, 6)
c = a.area()
if c != 7.483314773547883:
    area_test = False

if area_test == True:
    print("area test passes")
else:
    print("area test FAILS")

a = TriangleT(0, 3, 4)
c = a.is_valid()
if c == True:
    is_valid_test = False

a = TriangleT(1, 2, 3)
c = a.is_valid()
if c == True:
    is_valid_test = False

a = TriangleT(4, 5, 6)
c = a.is_valid()
if c == False:
    is_valid_test = False

if is_valid_test == True:
    print("is_valid test passes")
else:
    print("is_valid test FAILS")

a = TriangleT(1, 1, 1)
c = a.tri_type()
if c != TriType.equilat:
    tri_type_test = False

a = TriangleT(3, 2, 3)
c = a.tri_type()
if c != TriType.isosceles:
    tri_type_test = False

a = TriangleT(3, 4, 5)
c = a.tri_type()
if c != TriType.right:
    tri_type_test = False

a = TriangleT(3, 5, 7)
c = a.tri_type()
if c != TriType.scalene:
    tri_type_test = False

if tri_type_test == True:
    print("tri_type test passes")
else:
    print("tri_type test FAILS")
```

I Code for Partner's complex_adt.py

```
## @file complex_adt.py
#  @author Steven K.
#  @brief Python module for Complex Numbers
#  @date 2021-01-16

import math


## @brief Class for Complex Numbers
class ComplexT:

        ## @brief Constructor for Complex Numbers
        #  @param x The real part of the complex number
        #  @param y The imaginary part of the complex number
    def __init__(self,x,y):
        self.x = x
        self.y = y

    ## @brief Returns the real part of a complex number
    #  @return Real part of the complex number
    def real(self):
        return self.x

    ## @brief Returns the imaginary part of a complex number
    #  @return Imaginary part of the complex number
    def imag(self):
        return self.y

    ## @brief Calculates the absolute value of a complex number
    #  @return Absolute value of a complex number
    def get_r(self):
        return math.sqrt(self.x**2 + self.y**2)

    ## @brief Calculates the phase of a complex number
    #  @throws Exception thrown if the phase of the complex number is not defined
    #  @return Phase of a complex number
    def get_phi(self):
        if (self.x < 0) and (self.y == 0):
            return math.pi
        elif (self.real() == 0) and (self.imag() == 0):
            raise Exception("Phase of this number is not defined")
        else:
            return 2 * math.atan(self.y / (self.get_r()+self.x))

    ## @brief Compares two complex numbers and checks if they are equal
    #  @param c Second complex number
    #  @return True if both complex numbers are equal, or False if they are not
    def equal(self,c):
        if (self.real() == c.real()) and (self.imag() == c.imag()):
            return True
        else:
            return False


    ## @brief Calculates the conjugate of a complex number
    #  @return Conjugate of the complex number
    def conj(self):
        return ComplexT(self.x, self.y*(-1))

    ## @brief Adds two complex numbers
    #  @param c Second complex number
    #  @return Result of the complex number addition
    def add(self,c):
        return ComplexT((self.x+c.real()),(self.y+c.imag()))

    ## @brief Subtracts two complex numbers
    #  @param c Second complex number
    #  @return Result of the complex number subtraction
    def sub(self,c):
        return ComplexT((self.x-c.real()),(self.y-c.imag()))

    ## @brief Multiplies two complex numbers
    #  @param c Second complex number
    #  @return Result of the complex number multiplication
    def mult(self,c):
        return ComplexT((self.x*c.real()-self.y*c.imag()),(self.x*c.imag()+self.y*c.real()))
```

```
    ## @brief Calculates the reciprocal of a complex number
    #  @throws Exception thrown if the complex number is zero
    #  @return Reciprocal of the complex number
    def recip(self):
        if (self.x == 0) and (self.y == 0):
            raise Exception("Division by zero is not possible")
        else:
            return ComplexT((self.x/(self.x**2+self.y**2)),(self.y/(self.x**2+self.y**2))*(-1))

    ## @brief Divides a complex number by another complex number
    #  @param c Denominator
    #  @throws Exception thrown if the denominator is zero
    #  @return Result of the complex number division
    def div(self,c):
        if (c.real() == 0) and (c.imag() == 0):
            raise Exception("Division by zero is not possible")
        else:
            return self.mult(c.recip())

    ## @brief Calculates the square root of a complex number
    #  @return Square root of the complex number
    def sqrt(self):
        if (self.y < 0):
            return
                ComplexT(math.sqrt(((self.x+self.get_r())/2)),(-1)*math.sqrt(((self.x*(-1)+self.get_r())/2)))
        else:
            return
                ComplexT(math.sqrt(((self.x+self.get_r())/2)),math.sqrt(((self.x*(-1)+self.get_r())/2)))
```

J Code for Partner's triangle_adt.py

```
## @file triangle_adt.py
#  @author Steven K.
#  @brief Python module for Triangles
#  @date 2020-01-20

import math
from enum import Enum

## @brief Enumeration class for all triangles types
class TriType(Enum):
    equilat = "Equilateral triangle"
    isoceles = "Isoceles triangle"
    scalene = "Scalene triangle"
    right = "Right-angle triangle"

## @brief Class for Triangles
class TriangleT:

    ## @brief Constructor for Complex Numbers
    #  @param a First side of the triangle
    #  @param b Second side of the triangle
    #  @param c Third side of the triangle
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

    ## @brief Getter method for all the sides of a triangle
    #  @return Tuple containing 3 sides of the triangle
    def get_sides(self):
        return (self.a,self.b,self.c)

    ## @brief Constructor for Complex Numbers
    #  @param t Second triangle
    #  @return True if both Triangles are equal and False if not
    def equal(self,t):
        setA = set(self.get_sides())
        setB = set(t.get_sides())
        if (setA == setB):
            return True
        else:
            return False
```

```
## @brief Calculates the perimeter of a triangle
#  @return Perimeter of the triangle
def perim(self):
    return (self.a+self.b+self.c)

## @brief Checks if the triangle is valid (mathematically possible)
#  @return True if it's a valid triangle and False if not
def is_valid(self):
    a = self.a
    b = self.b
    c = self.c
    if (a + b <= c) or (a + c <= b) or (b + c <= a):
        return False
    else:
        return True

## @brief Calculates the area of a triangle
#  @throws Exception thrown if the triangle is not valid
#  @return Area of the triangle
def area(self):
    if self.is_valid():
        semiP = self.perim()/2
        return math.sqrt(semiP*(semiP-self.a)*(semiP-self.b)*(semiP-self.c))
    else:
        raise Exception("Please enter a valid triangle")

## @brief Returns the type of the triangle
#  @throws Exception thrown if the triangle is not valid
#  @return TriType object corresponding to the type of triangle
def tri_type(self):
    a = self.a
    b = self.b
    c = self.c
    if self.is_valid():
        if (a**2 + b**2) == c**2:
            return TriType.right
        elif a == b == c:
            return TriType.equilat
        elif (a == b) or (b == c) or (a == c):
            return TriType.isoceles
        else:
            return TriType.scalene
    else:
        raise Exception("Please enter a valid triangle")
```