IT1312 – Data Structures & Algorithms
Programming Assignment

SUBMISSION INSTRUCTIONS:

1) Name the Python files containing your solutions according to the following requirement:

• Question 1a) – "**Sierpinski_Triangle.py**"	**[10 marks]**
• Question 1b) – "**Sierpinski_Circle.py**"	**[20 marks]**
• Question 2a) – "**Pythagoras_Trees.py**"	**[20 marks]**
• **TOTAL MARKS**	**[50 marks]**

2) At the beginning of every Python file to be submitted, include your "**Name, Student Admin no. and Tutorial Group**" as comments.

3) ZIP all Python files to be submitted into a zipped file and name it as "**ADMINNO_IT1312_PA.zip**" e.g., "**123456F_ IT1312_PA.zip**".

4) Submission Due Date: The deliverable is to be submitted to your module tutor via Brightspace **by 02 Feb 2025, Sunday, 11:59PM (Week 16)**. Submission link will be made available closer to the due date.

5) Late Submission Penalty:
   - Submissions made within 5 calendar days after the deadline will have a maximum score of 50% of the base mark.
   - Submissions made on or after the 6th day past the deadline will receive a 0 mark.

1. **Sierpinski Triangle and Sierpinski Circle**

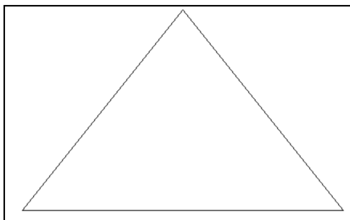
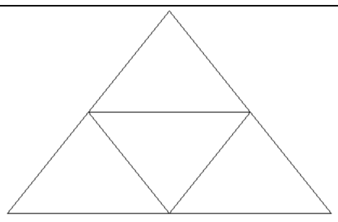
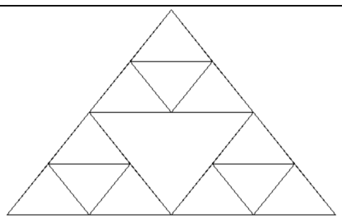
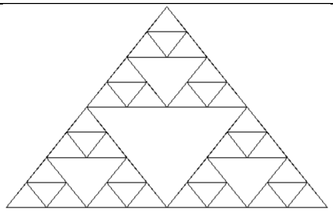
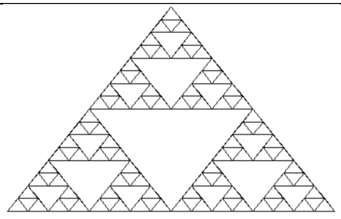
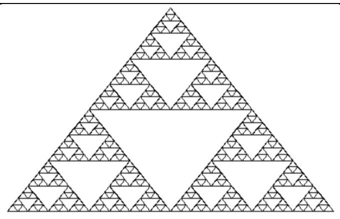
Background: *(https://en.wikipedia.org/wiki/Sierpi%C5%84ski_triangle)*

The Sierpinski triangle, also called the Sierpinski gasket or Sierpinski sieve, is a fractal and attractive fixed set with the overall shape of an equilateral triangle, subdivided recursively into smaller equilateral triangles. It is named after the Polish mathematician Wacław Sierpiński but appeared as a decorative pattern many centuries before the work of Sierpiński.

The programming technique of recursion can be used to generate the Sierpinski Triangle and other related fractal patterns. Fractals are infinitely complex patterns that are self-similar across different scales. To create a fractal, you can start with a simple pattern and repeat the pattern at smaller scales recursively.

a. Sierpinski Triangle [10 marks]

The Python code provided below is an implementation of an algorithm that generates the Sierpinski Triangle at different degree of complexities:

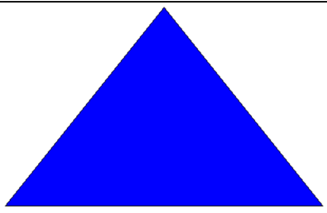
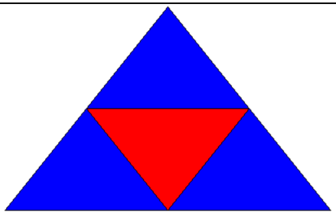
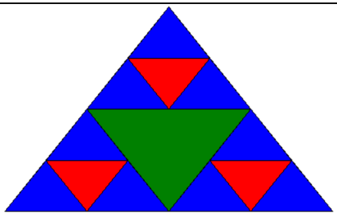
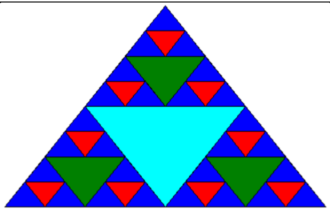
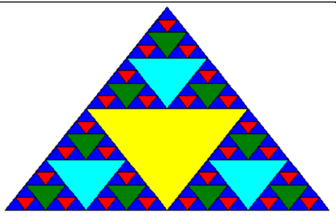
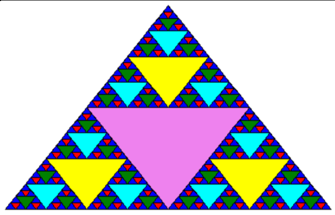
		
degree 0	**degree 1**	**degree 2**
		
degree 3	**degree 4**	**degree 5**

Python Implementation – Sierpinski Triangle:

```
import turtle

def drawTriangle(points,myTurtle):
    myTurtle.up() # Pen up
    myTurtle.goto(points[0][0],points[0][1])
    myTurtle.down() # Pen down
    myTurtle.goto(points[1][0],points[1][1])
    myTurtle.goto(points[2][0],points[2][1])
    myTurtle.goto(points[0][0],points[0][1])


def getMid(p1,p2):
    return ( (p1[0]+p2[0]) / 2, (p1[1] + p2[1]) / 2)


def sierpinski(points,degree,myTurtle):

    # Draw a triangle based on the 3 points given
    drawTriangle(points,myTurtle)

    if degree > 0:
        sierpinski([points[0],
                        getMid(points[0], points[1]),
                        getMid(points[0], points[2])],
                    degree-1, myTurtle)
        sierpinski([points[1],
                        getMid(points[0], points[1]),
                        getMid(points[1], points[2])],
                    degree-1, myTurtle)
        sierpinski([points[2],
                        getMid(points[2], points[1]),
                        getMid(points[0], points[2])],
                    degree-1, myTurtle)


def main():
   myTurtle = turtle.Turtle()
   myTurtle.speed(10)    # adjust the drawing speed here
   myWin = turtle.Screen()

   # 3 points of the first triangle based on [x,y] coordinates
   myPoints = [[-200,-50],[0,200],[200,-50]]
   degree = 2            # Vary the degree of complexity here

   # first call of the recursive function
   sierpinski(myPoints,degree,myTurtle)

   myTurtle.hideturtle()# hide the turtle cursor after drawing is
                        # completed

   myWin.exitonclick()   # Exit program when user click on window


main()
```

Modify the given code to implement a Python program (**Sierpinski_Triangle.py**) that generates the Sierpinski Triangle such that triangles at different degrees are drawn with different colors.

The expected output of the modified code is as follows:

		
degree 0	**degree 1**	**degree 2**
		
degree 3	**degree 4**	**degree 5**

HINTS:

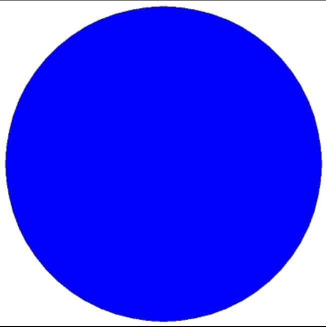
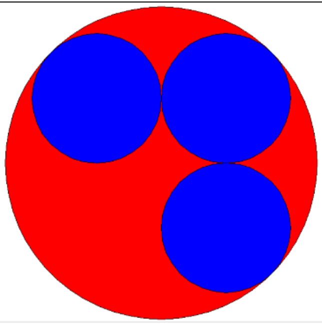
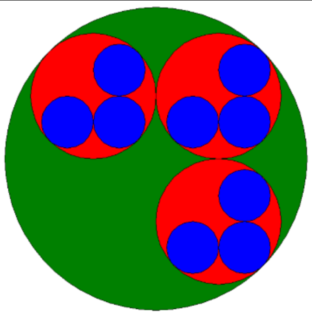
- Use Python's Turtle graphics to draw the required shapes. (*https://docs.python.org/3/library/turtle.html*)
- Use Turtle graphics' `fillcolor()`, `begin_fill()` and `end_fill()` methods to add colours to your Sierpinski triangles.
- Start by drawing a single large triangle (degree 0). Divide this large triangle into four new triangles by connecting the midpoint of each side (degree 1). Ignoring the middle triangle that you just created, apply the same procedure to each of the three corner triangles. Each time you create a new set of triangles, you recursively apply this procedure to the three smaller corner triangles.

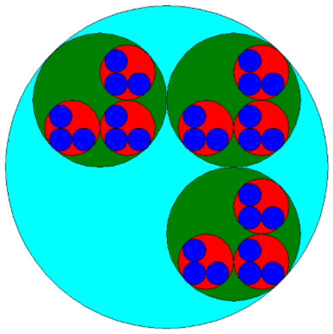
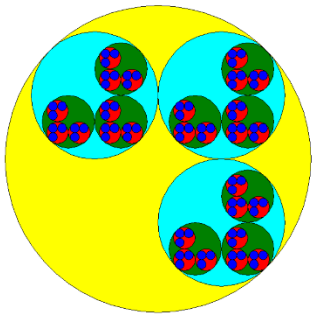
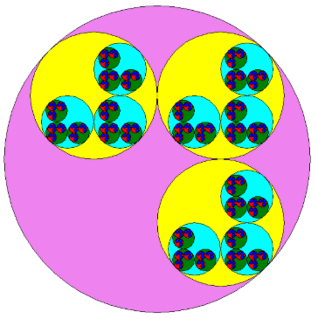
b. Sierpinski Circle [20 marks]

Like both the Sierpinski Triangle, Sierpinski Circle is a fractal pattern that can be generated recursively with circle (instead of triangle and square) as the building block.

Using the same programming technique of recursion, write a Python program (**Sierpinski_Circle.py**) to generate the Sierpinski Circle.

The expected output of the program is as follows:

		
degree 0	**degree 1**	**degree 2**

		
degree 3	**degree 4**	**degree 5**

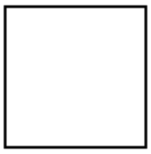
2. Pythagoras Tree

Background: *(https://en.wikipedia.org/wiki/Pythagoras_tree_(fractal))*

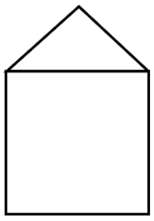
The Pythagoras tree is a fractal tree constructed from squares. It is named after Pythagoras because each triple of touching squares encloses a right triangle, in a configuration traditionally used to represent the Pythagorean theorem.

How to draw a Pythagoras Tree? (**https://www.wikihow.life/Draw-a-Pythagorean-Tree**)

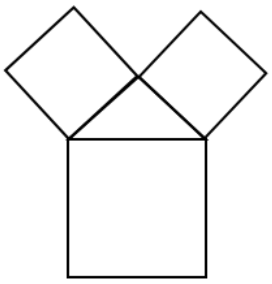
Step 1*: Begin by drawing a square near the bottom of your drawing surface.*



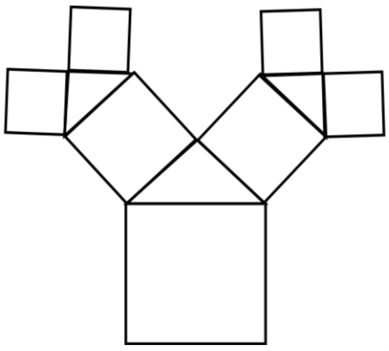
Step 2*: Draw a isosceles right triangle whose hypotenuse is the top edge of the square you just drew.*



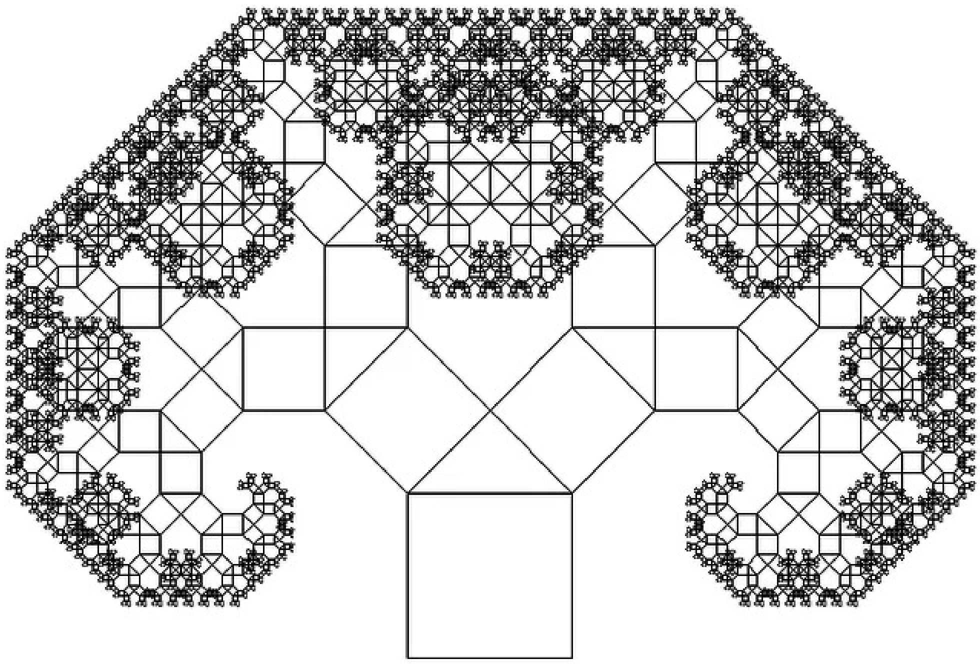
Step 3*: Draw squares along each of the other two sides of this isosceles triangle.*



Step 4*: Repeat this construction on each of the two new squares.*

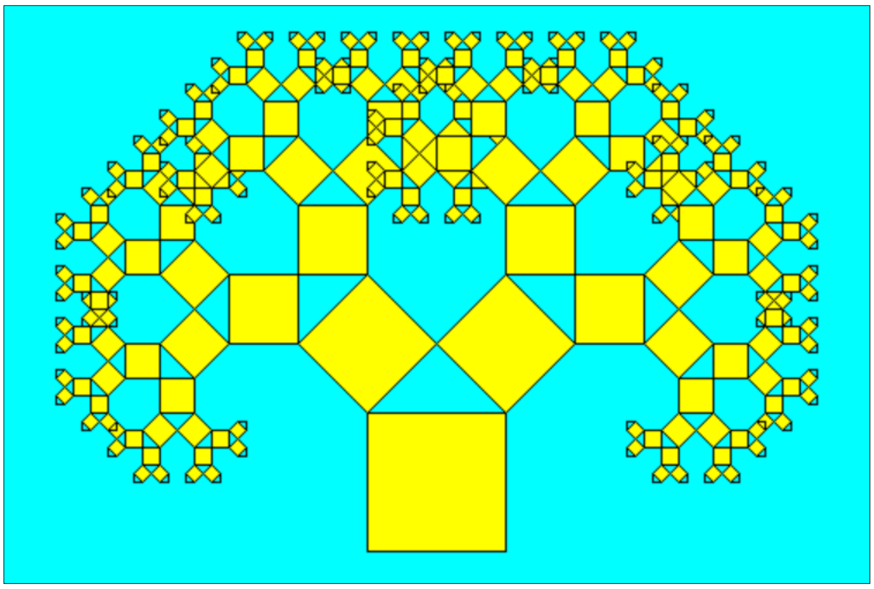


***Step 5**: Continue building triangles and squares in the same fashion (aka recursively) until you are happy with your tree.*



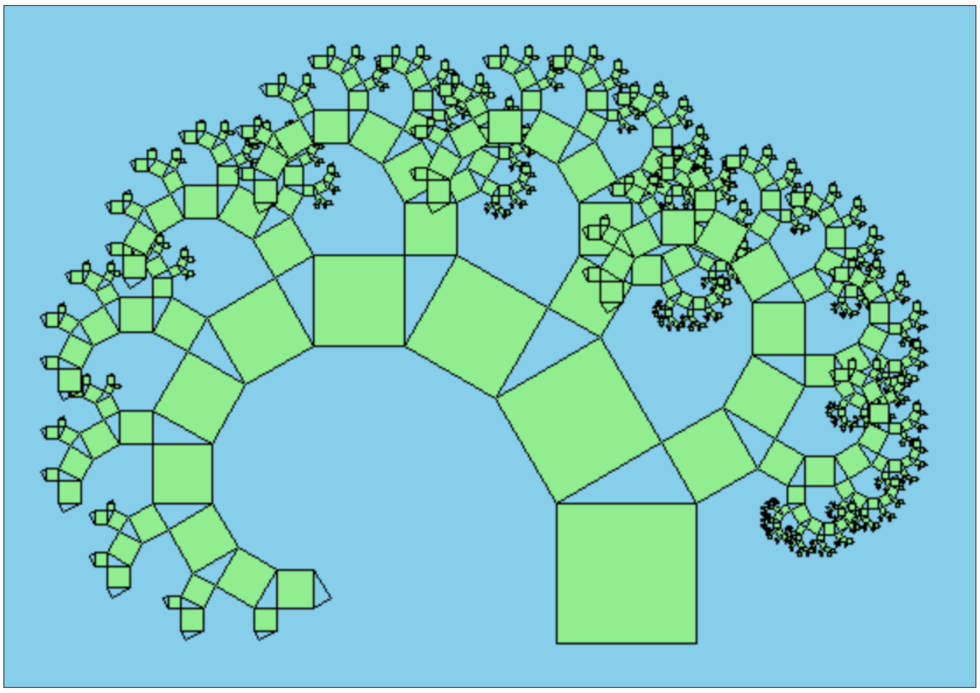
a. **<u>Three Types of Pythagoras Tree [20 marks]</u>**

The Pythagoras Tree drawn in the above illustrations is based on squares and "**Isosceles Right Triangle**". *(NOTE: Isosceles right triangle is a two-dimensional three-sided figure in which one angle measures 90°, and the other two angles measure 45° each). Using this approach will generate a "**Symmetrical / Balanced Pythagoras Tree**" as shown below:*

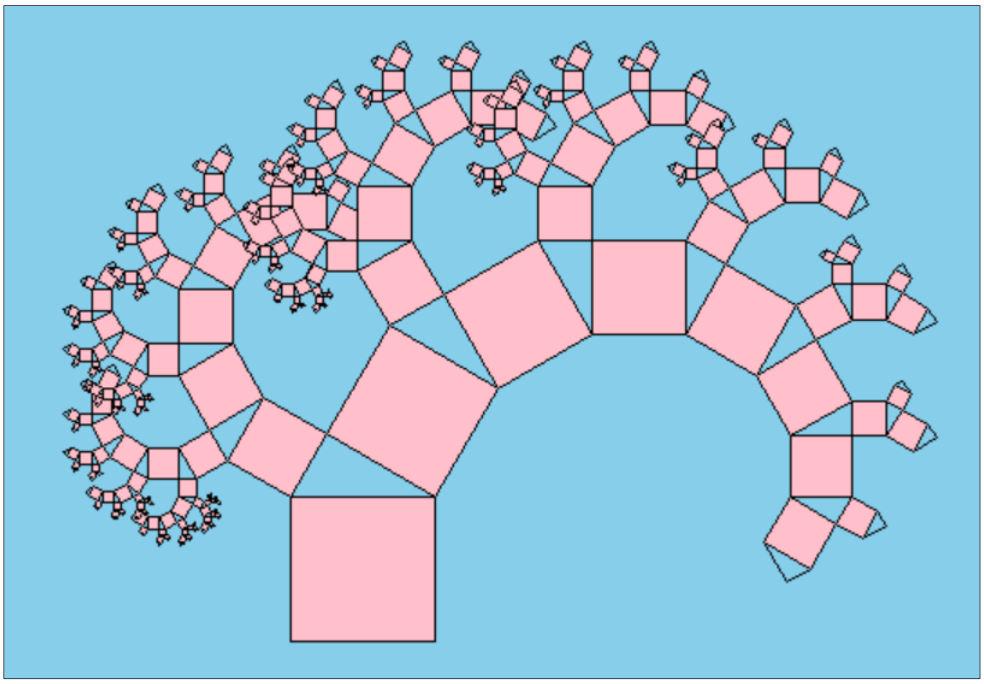


Symmetrical / Balanced Pythagoras Tree

If an isosceles right triangle was not used, but instead replaced by a "**Right Triangle**" *(NOTE: Right triangle is a two-dimensional three-sided figure in which one angle measures 90°, and the other two angles are acute angles)*, this will generate either a "**Left-Leaning Pythagoras Tree**" or "**Right-Leaning Pythagoras Tree**" as shown below:



Left-Leaning Pythagoras Tree



Right-Leaning Pythagoras Tree

Using the programming technique of recursion, write a Python program (**Pythagoras_Trees.py)** to generate the 3 types of Pythagoras Trees:

- Symmetrical / Balanced Pythagoras Tree
- Left-Leaning Pythagoras Tree
- Right-Leaning Pythagoras Tree

Besides generating the Pythagoras Trees according to the expected output as shown in the diagrams above, you are encouraged to go beyond the basic requirements and infuse your creativity into the program. Add your personal touch by incorporating creative elements, introducing variations, or implementing features that elevate the visual appeal of the fractals.

-- End of Programming Assignment --