Geometry with Python Turtle

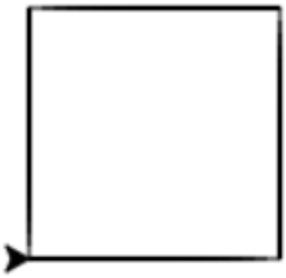
Introduction

In this lesson we are going to use the Turtle library to draw a variety of shapes. Each shape will combine Python for loops together with functions from the Turtle library

1 Drawing a square

The program below imports turtle and then uses a for loop to repeat moving forward and turning 90 degrees.

```
import turtle

for i in range(0,4):
  turtle.forward(100)
  turtle.left(90)
```

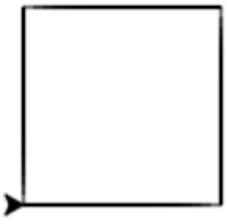


2 Using penup() and pendown() to draw separate squares

The program below does the following

1. Draws a square
2. Lifts a pen using penup()
3. Goes to a specific location using goto()
4. Punts the pen down using pendown() function
5. Draws another square

```
import turtle

for i in range(0,4):
  turtle.forward(100)
  turtle.left(90)

turtle.penup()
turtle.goto(-200,200)
turtle.pendown()

for i in range(0,4):
  turtle.forward(100)
  turtle.left(90)
```



3 Drawing a square filled with colour

The program below draws a square filled with colour, using the following process

1. Define the colour using the color() function
2. Before drawing a shape using the begin_fill() function
3. When the shape is complete use the end_fill() function

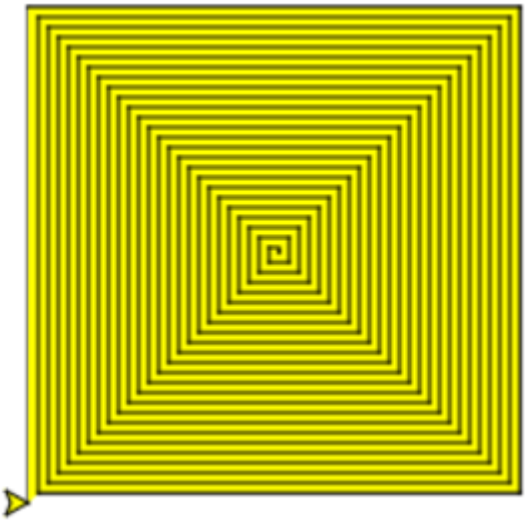
```
import turtle

#specify colour
turtle.color('green')

# Turtle begins fill here
turtle.begin_fill()
for i in range(0,4):
  turtle.forward(100)
  turtle.left(90)
#Turtle ends fill here
turtle.end_fill()
```

4 Drawing a filled in square spiral shape

The program below draws a square spiral filled with colour, using the following process

1. Define the colour using the color() function
2. Defin the fill colour using the fillcolor() function
3. Before drawing a shape using the begin_fill() function
4. Draw a square with a for loop repeating 100 times and the forward function having (i*2) as an argument
5. When the shape is complete use the end_fill() function

```
import turtle

for i in range(0,4):
  turtle.forward(100)
  turtle.left(90)

turtle.penup()
turtle.goto(-200,200)
turtle.pendown()

for i in range(0,4):
  turtle.forward(100)
  turtle.left(90)
```



Fractal Geometry with Python Turtle

Introduction

In this lesson we are going to learn how to write functions in Python that have a visual output. We will then get these functions to call themselves creating interesting visual patterns that are called fractals.

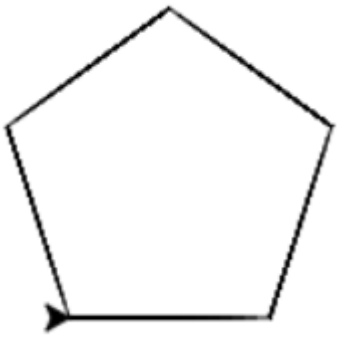
1.1 Drawing a polygon

The humble beginning of our journey starts with a polygon a geometric shape where each is equal with the same angle between each side. Some well known polygons are equilateral triangles, squares, pentagons, hexagons and many more.

The program below will do the following:

1. Specify the number of sides
2. Specify the length of each side
3. Use a for loop to draw a polygon with the specified number of sides and side length

```
from turtle import *

sides = 5
side_length = 80

for i in range(0,sides):
  forward(side_length)
  left(360/sides)
```



1.2 Polygon function

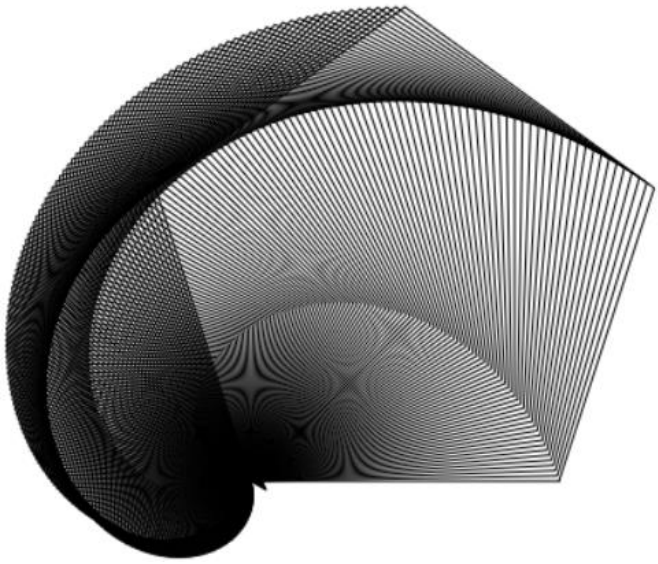
Now that we can draw any polygon simply by specifying sides and side_length, we can write a python function that uses sides and side_length to draw any polygon:

```
from turtle import *

def polygon(sides, side_length):
  for i in range(0,sides):
    forward(side_length)
    left(360/sides)

polygon(6,80)
```

1.3 Recursive polygon

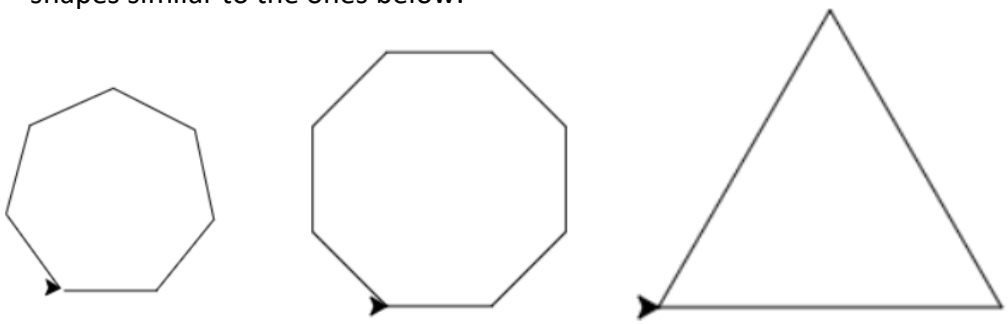
The polygon function is changes to call itself so that it generates smaller polygons turned to the left until the side_length = 1

```
from turtle import *
speed(100)

def polygon(sides,side_length):
  for i in range(0,sides):
    forward(side_length)
    left(360/sides)
  if side_length > 1:
    left(1)
    polygon(sides,side_length-1)

polygon(5,180)
```

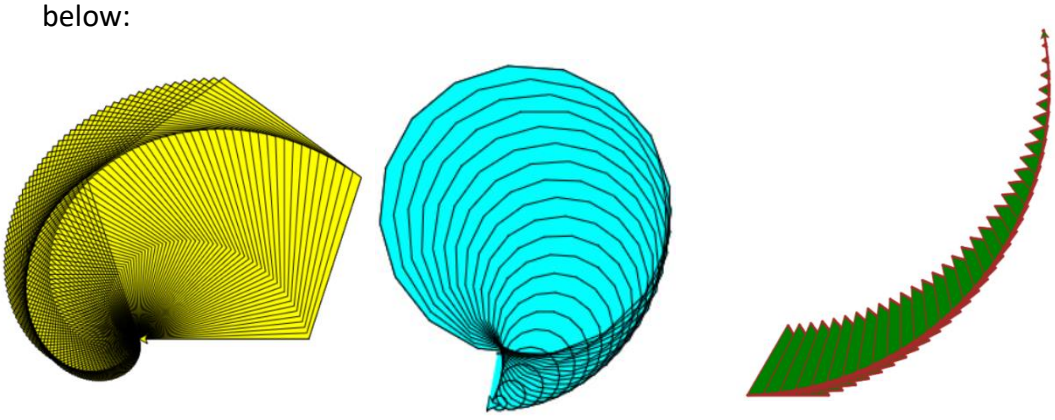


Challenges:

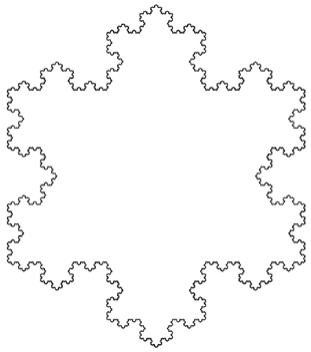
1. Can you change the size, angle and the number of sides of each polygon to create shapes similar to the ones below:



2. Can you use coloring of the fill and of the pen to draw shapes similar to the ones below:

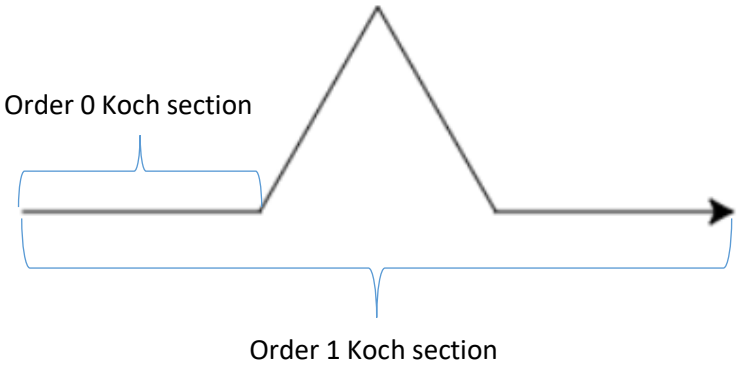


2.1 Drawing an order 1 Koch section



The fractal you see on the right is known as a Koch snowflake or the Koch Island. It is the result of repeating the simple pattern that you see pictured below. This patter is called order 1 Koch section and it is made up of 4 order 0 sections which are just straight lines.

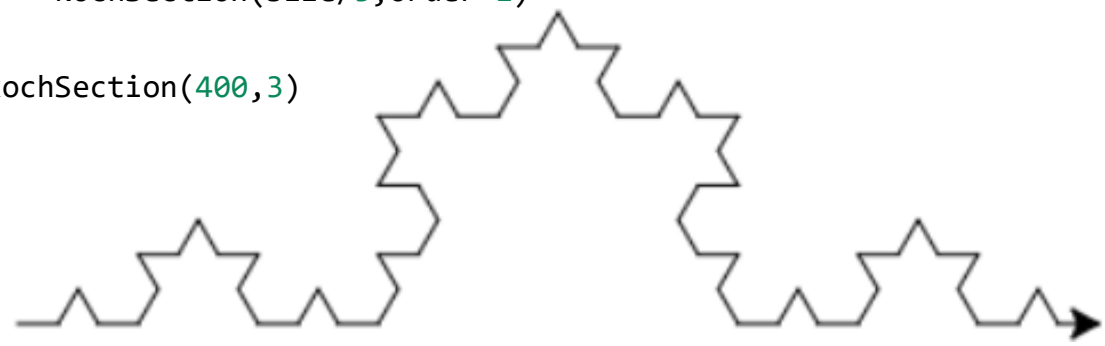
```
size = 100

forward(size)
left(60)
forward(size)
right(120)
forward(size)
left(60)
forward(size)
```



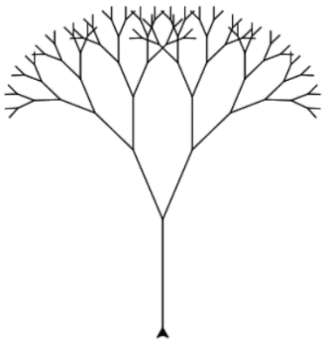
2.2 Creating a Koch fractal

Now that we can draw a simple order 1 Koch section we can formulate our code in a recursive way so that for example we could draw an order 3 section which is made up of 4 order 2 sections each of which is made up of 4 order 1 sections.

```
def kochSection(size,order):
  if order == 0:
    forward(size)
  else:
    kochSection(size/3,order-1)
    left(60)
    kochSection(size/3,order-1)
    right(120)
    kochSection(size/3,order-1)
    left(60)
    kochSection(size/3,order-1)

kochSection(400,3)
```



3.1 Drawing a simple fractal tree



Fractal structures are not only found in imaginary mathematical shapes. They are actually really common in nature, trees, sea shells, ferns and cacti all have these patterns

Our first step to drawing a tree will be to draw the shape below which is basically a branch from which further branches will grow:

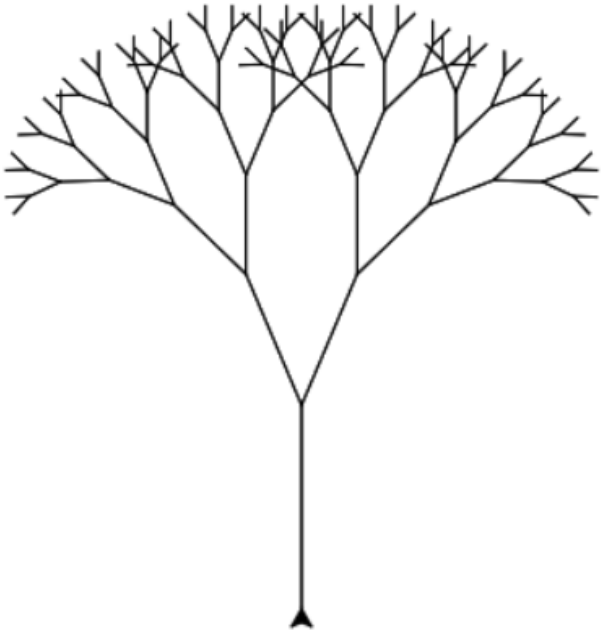
Here is the code to do that:

```
from turtle import *

left(90)
angle = 30
branch_len = 100

forward(branch_len)
left(angle)
forward(branch_len*0.7)
backward(branch_len*0.7)
right(angle*2)
forward(branch_len*0.7)
backward(branch_len*0.7)
left(angle)
backward(branch_len)
```

3.2 Drawing branching fractal tree

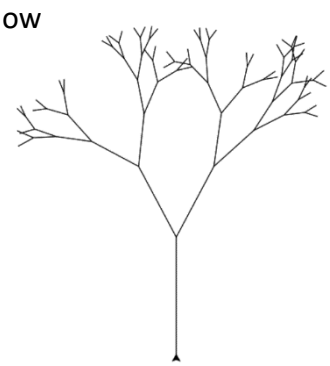
Now that we have created a basic tree pattern we can again re format the code into a function that calls itself. This time however we are going to have the tree sprout smaller branches until the brach reaches a size limit

```
from turtle import*

speed(100)
left(90)

def branch(branch_len,angle):
  if branch_len < 10:
    return
  else:
    forward(branch_len)
    right(angle)
    branch(branch_len*0.7,angle)
    left(angle*2)
    branch(branch_len*0.7,angle)
    right(angle)
    backward(branch_len)

branch(100,23)
```

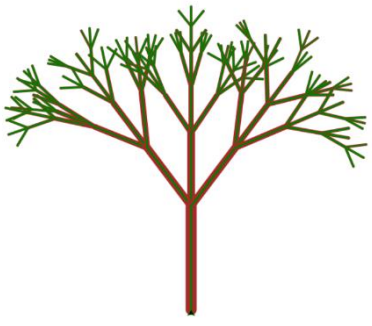


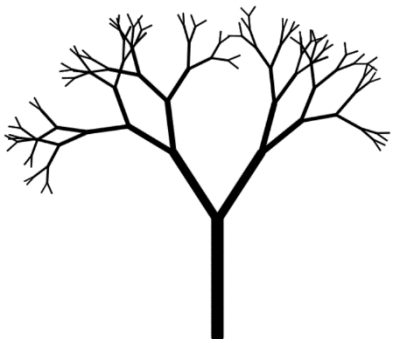
Challenges:

1. Can you use a random angle together with a random branch length to create trees like the ones below



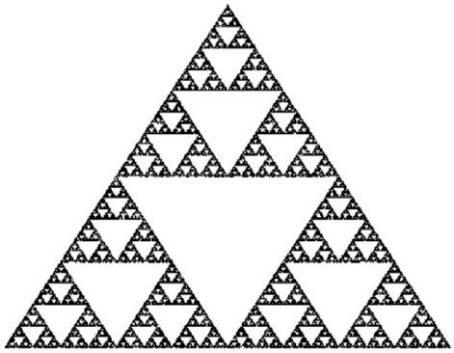
2. Can you use a thicker pen for the longer branches so that your trees look like this:



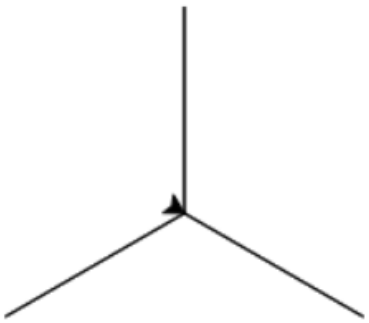


4.1 Drawing a basic Sierpinski section

Another famous fractal shape is the Sierpinski triangle. You can see it pictured on the right. We are going to draw a similar shape using a very similar pattern where you have 3 lines separated by 120 degrees:



```
for i in range(0,3):
  forward(100)
  backward(100)
  left(120)
```



4.2 Drawing branching Sierpinski triangle

Now we can create a structure where each branch splits into 3 and each of those splits into 3. As this happens the overall structure begins to resemble the Sierpinski triangle:

```
def sierpinski(length,depth):
  if depth == 0:
    return
  else:
    for i in range(0,3):
      forward(length)
      sierpinski(length/2, depth -1)
      backward(length)
      left(120)


sierpinski(80,6)
```