121COM: Introduction to Computing

Academic Year 2015/16

LabSheet 5

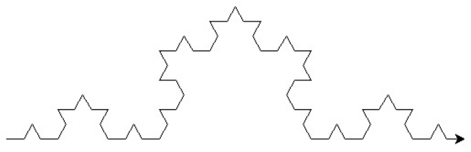
For use in labs the week beginning Mon 26th October 2015

Author: Dr Matthew England (Coventry University)

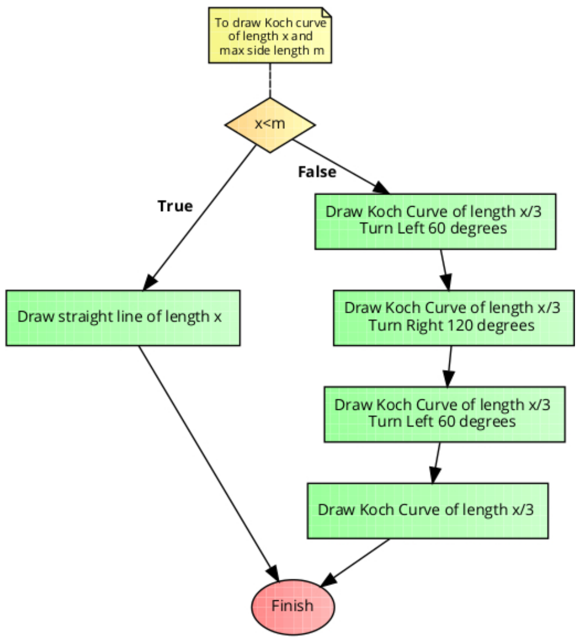
C Lab Exercises 1 - Fractals

Q1 In Q2 of Worksheet 5 the first image was a regular octagon. You should have drawn 8 sides of equal length, turning the turtle by $360/8 = 45$ degrees after each. Generalise these to define a function which can draw any regular polygon. It should have 2 parameters: side length and number of sides.

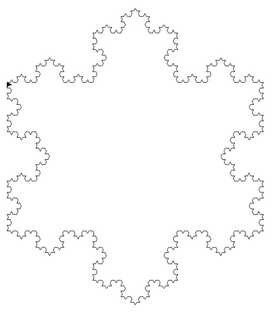
Q2 Implement the flow chart in a function `koch(x,m)` which draws **Koch Curves** of length x and max side length m.
The output from `kochCurve(500,50)` should look as below.



Experiment with different values of (x, m).


To draw Koch curve of length x and max side length m
x<m
True
False
Draw straight line of length x
Draw Koch Curve of length x/3 Turn Left 60 degrees
Draw Koch Curve of length x/3 Turn Right 120 degrees
Draw Koch Curve of length x/3 Turn Left 60 degrees
Draw Koch Curve of length x/3
Finish

Q3 A **Koch Snowflake** if formed by joining three Koch curves such that all ends meet. Write a function `kochSnowflake(x,m)` to create these. The output below is for `kochSnowflake(500,10)`.



A **Extension:** There are many generalisations to Koch curves. Try extending your function so the user can also choose the base degree of the turns to be something other than 60 degrees.
The wikipedia page gives some more ideas:
`www.wikipedia.org/wiki/Koch_snowflake`

I Lab Exercises 2 - More Recursion Exercises

Q4 A **palindrome** is a word (or any other sequence of symbols) which reads the same forward and backward, for example *racecar*. Write a recursive function `isPalindrome(S)` which returns a Boolean determining whether the string S is a palindrome.

Q5 In Python we can perform **floor division** using the operator `\\` to find how times one number can divide another. Write a recursive function to perform floor division which does not use the operator `\\`.

Q6 In Python we calculate powers using the symbol `**`. In this question we consider how the Python interpretor may have implemented this: do not use `**` in your solution.

The mathematical definition of taking a power is $n^p = n * n^{p-1}$ for positive integer p. Use this to create a recursive function `power(n,p)` which calculates n^p.

Next, note that an alternative definition is

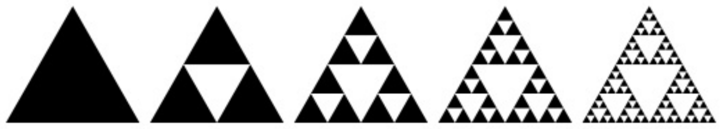
$$n^p = \begin{cases} n\left(n^2\right)^{\frac{p-1}{2}} & \text{if } p \text{ is odd} \\ \left(n^2\right)^{\frac{p}{2}} & \text{if } p \text{ is even.} \end{cases}$$

Implement this to create a new recursive function `fastPower(n,p)`. This should be faster than `power(n,p)`. Explain why this would be by counting how many calls are made to your functions for given values of n and p. If you want Python to do that counting for you, you'll probably need a **global variable**.

A Extended Task

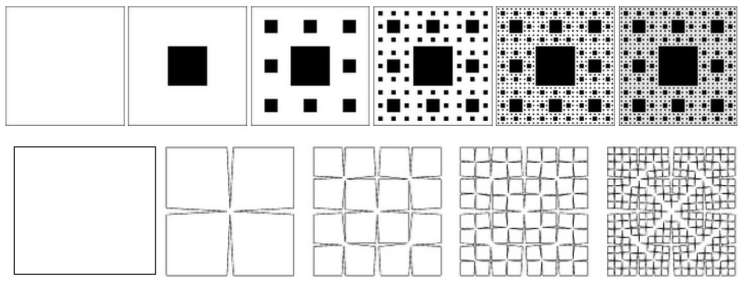
A **Sierpinski Triangle** of order 0 is just an equilateral triangle (all sides the same length). An order 1 Sierpinski Triangle is created by 3 smaller triangles coming together so their edges form the single larger one. An order 2 Sierpinski Triangle repeats this process within the 3 smaller triangles.

These examples, along with the order 3 and 4 Sierpinski triangles are shown below. In each case, the smallest triangles have been coloured in black to make clear the difference between triangles and gap.



Write Python code to draw the first 3 triangles. Generalise this to a function which will drawn the Sierpinski Triangle for a specified order.

Two similar fractals are the Sierpinski Carpet and the Cesaro Square. The first few orders of each are displayed below. Can you infer the patterns and write functions to draw these of any order?



If you have time, why not experiment with different colours for your fractals. Recursion can be beautiful!