COMP1021
Introduction to Computer Science

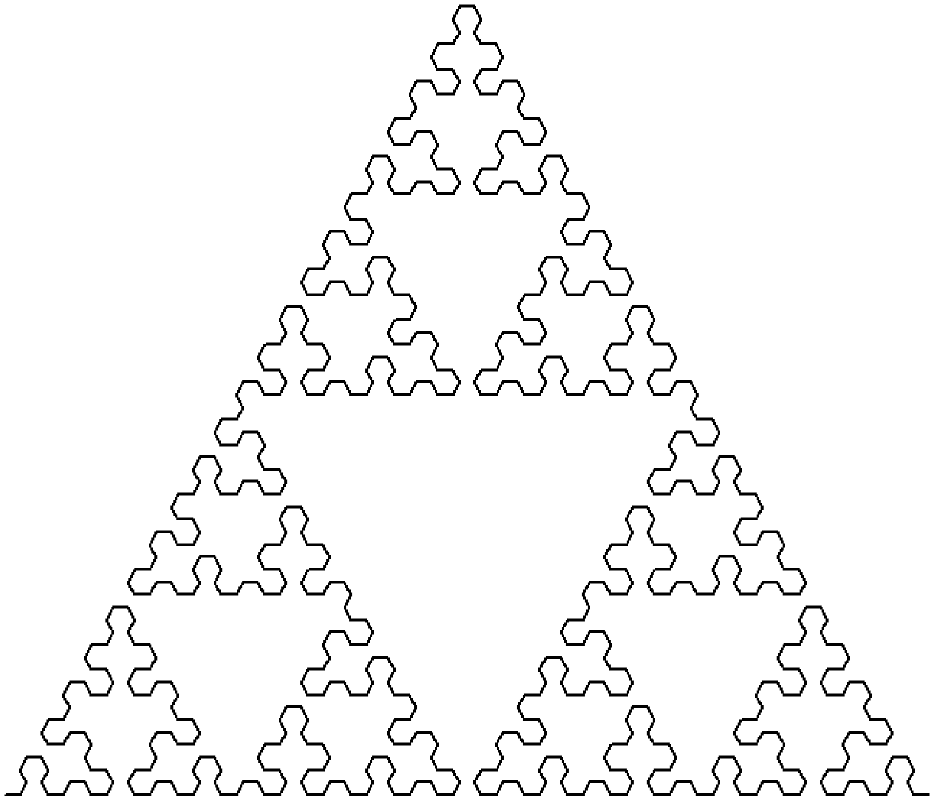
L-System
Computer Graphics

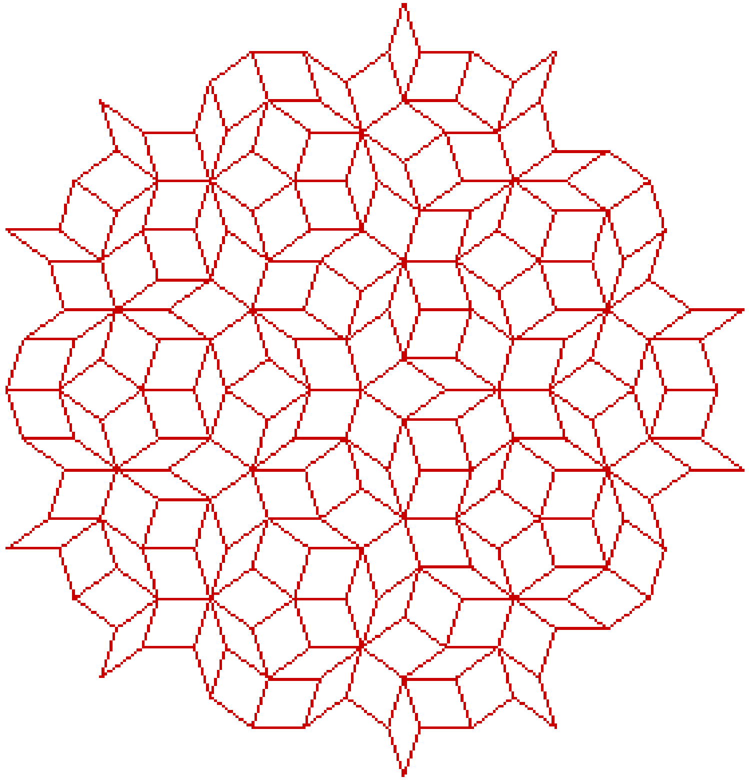
David Rossiter and Gibson Lam

Graphics Programming

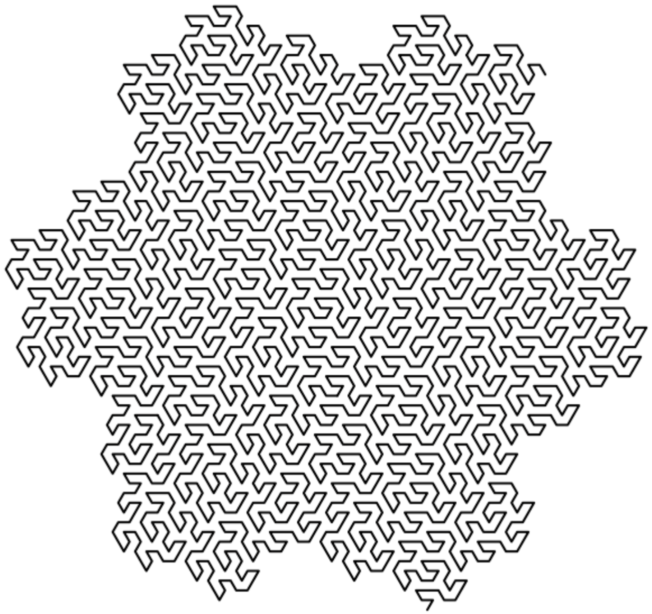
- We know about using turtle graphics to generate ‘simple’ computer graphics
- In this presentation we will look at a more advanced approach called ‘L-system’
- L-system is short for ‘Lindenmayer system’
- There are many special images that can be created using this system

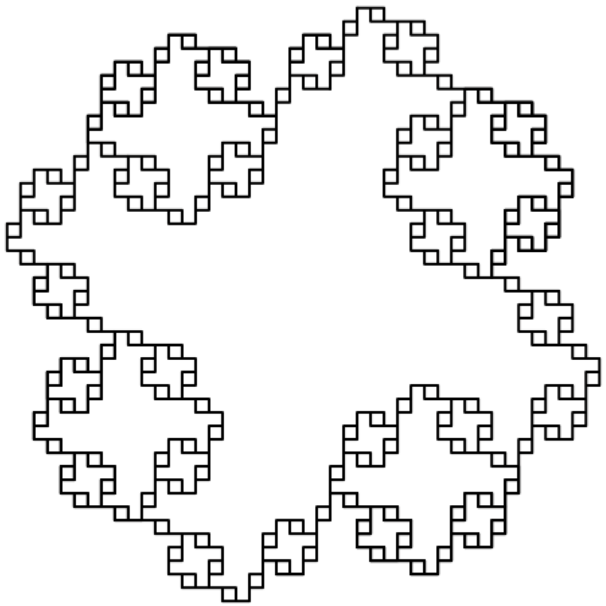
Mr Aristid Lindenmayer

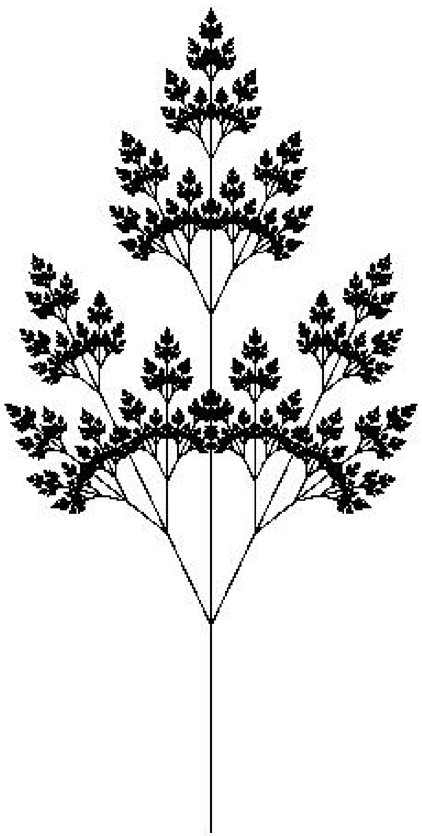


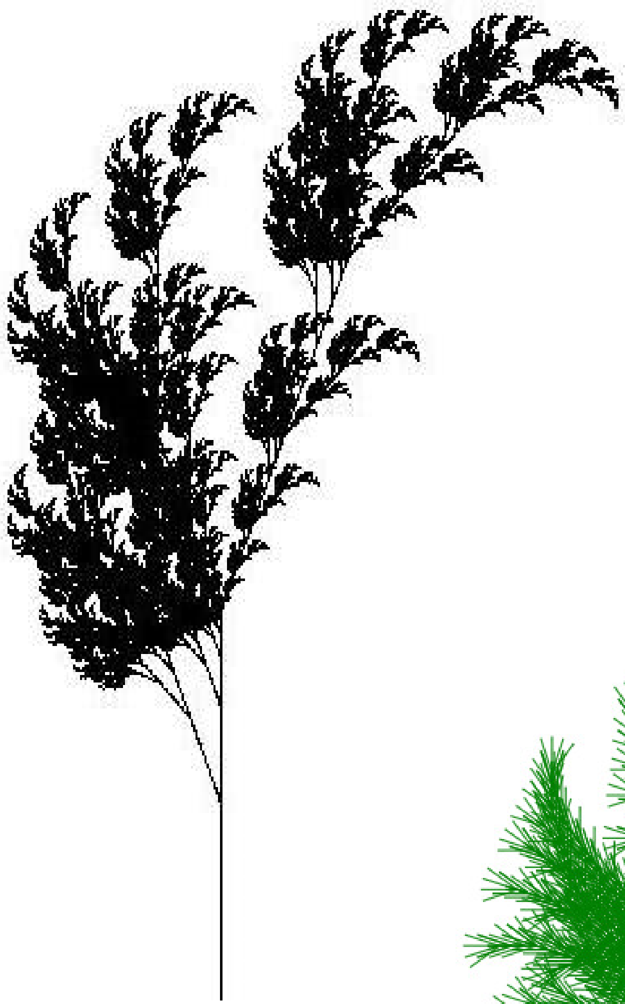


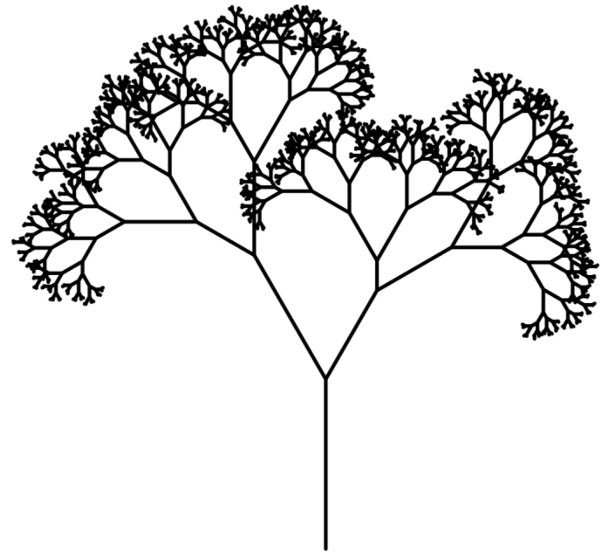
- We can make many special shapes and patterns

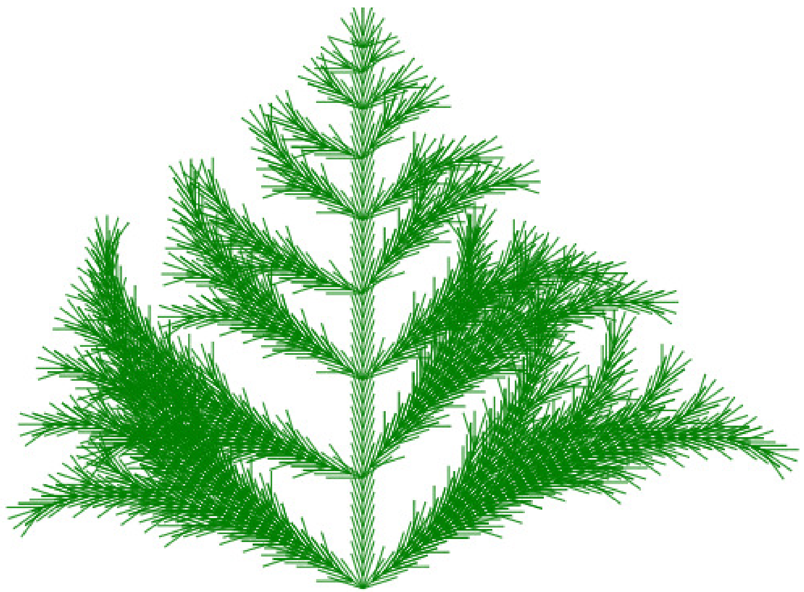




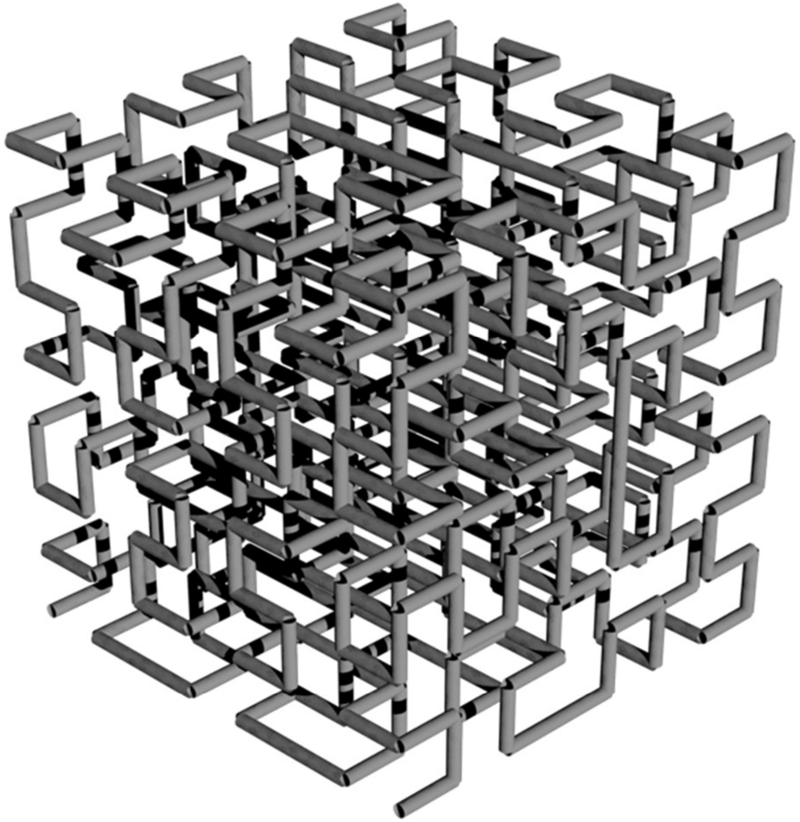


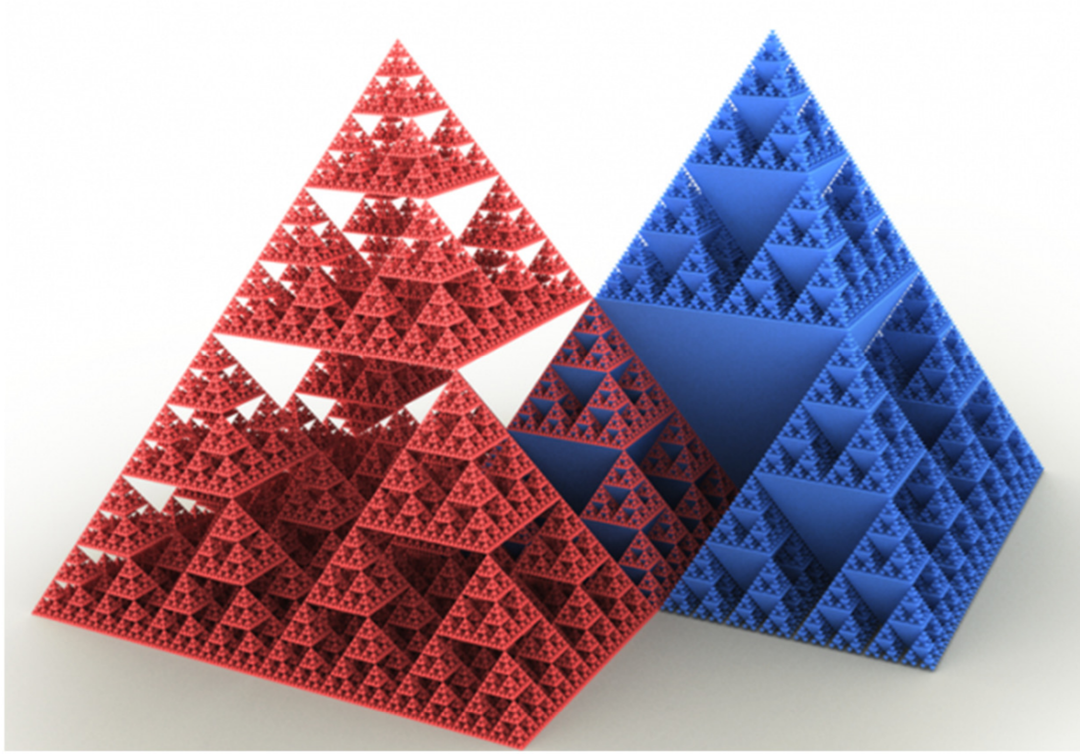




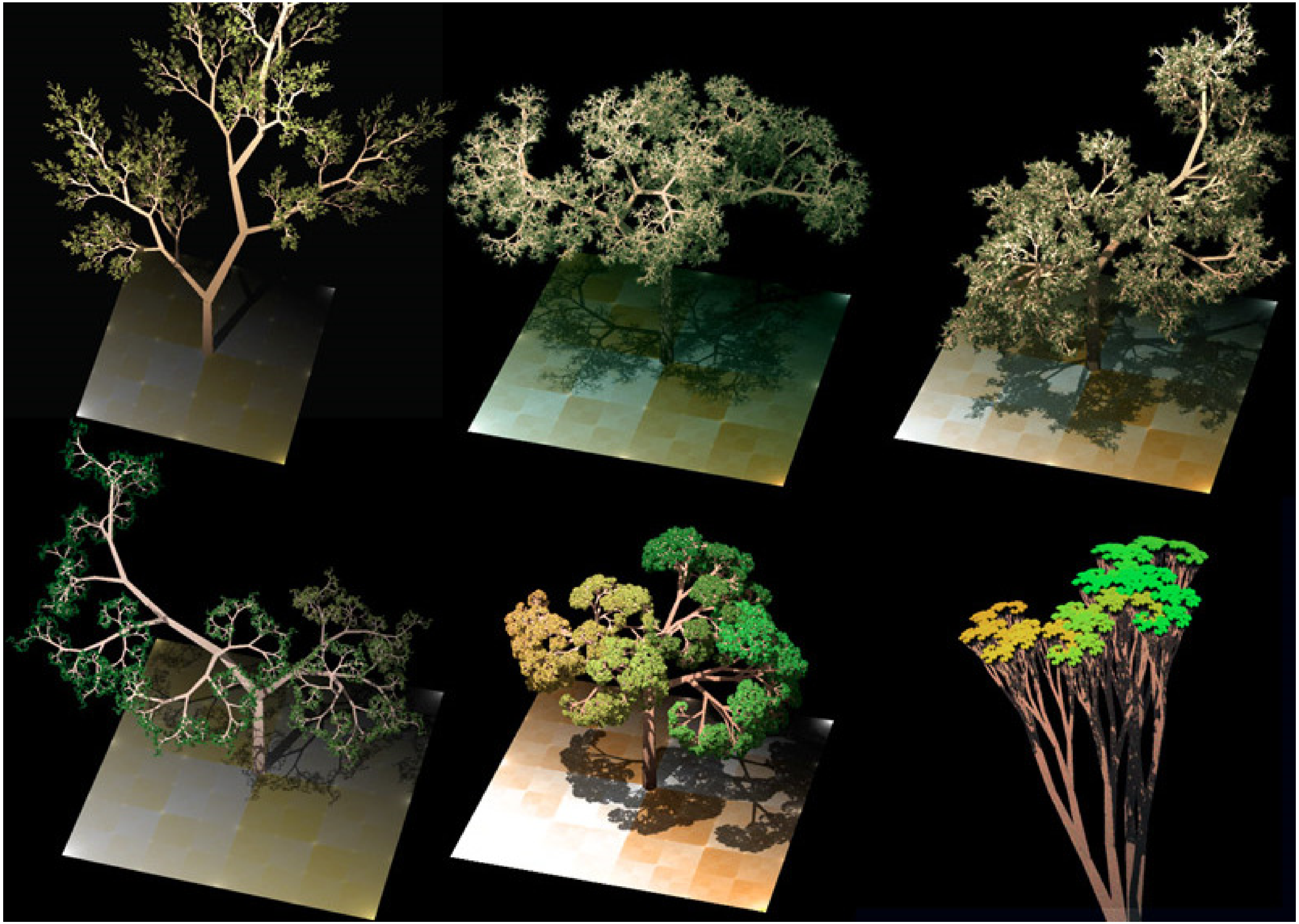


- If we add the ability to branch, we can make trees and plants (not covered in this presentation)





- If we apply some 3D ideas (i.e. growing in the z axis as well as the x and y axis/ using light & shadows) to L-systems, we can make some great realistic images
- However, we don't have enough time to do any 3D

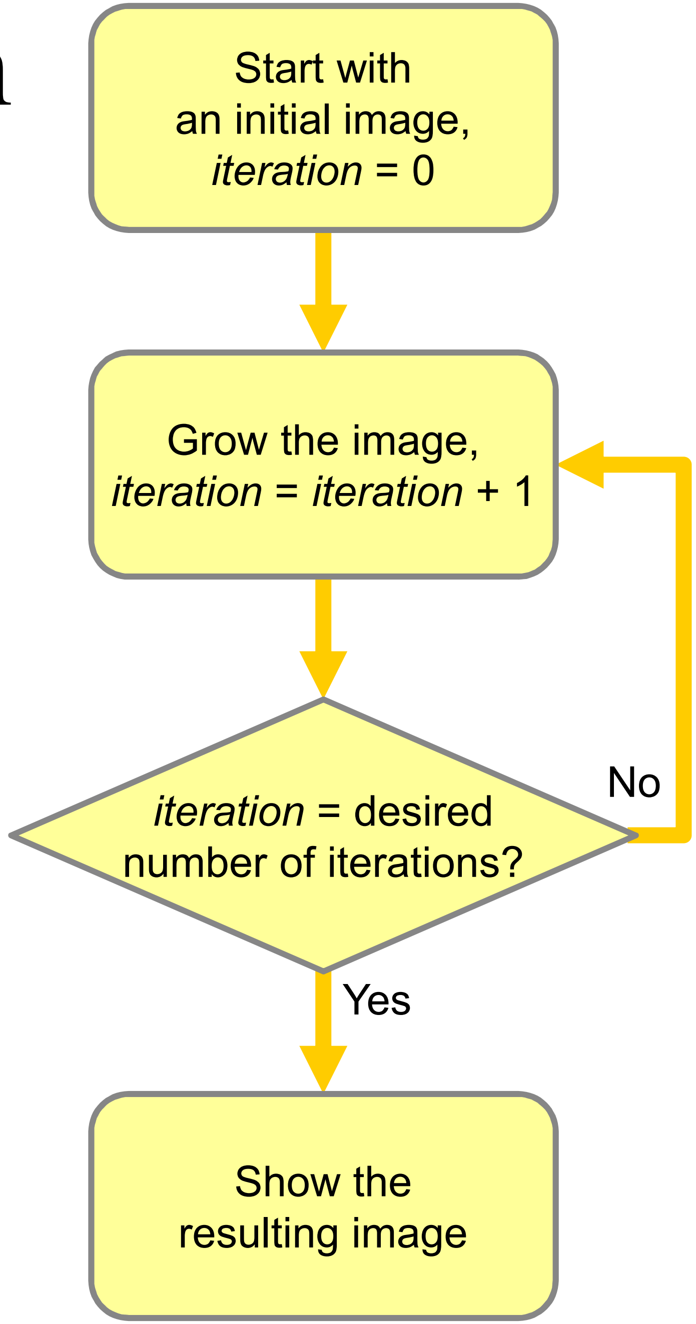


L-System trees in 3D

from http://en.wikipedia.org/wiki/L-system

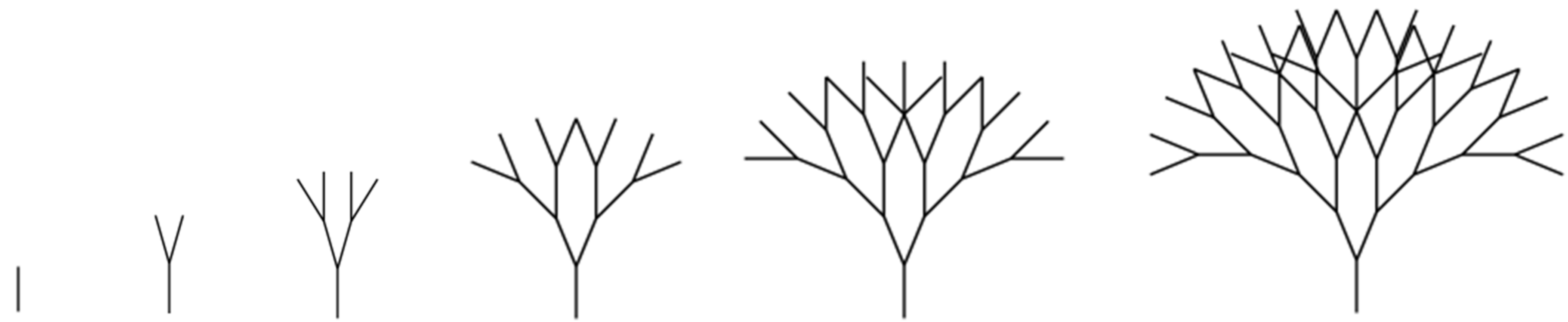
Basic Idea of L-System

- The system starts from an initial image, which is typically a simple one
- The system then repeatedly lets the image grow into a slightly more complex one based on some given rules
  - Each time the image grows, we say it has completed an iteration
- The system stops when the desired number of iterations has been reached



Example Growing of a Tree

- Here is an example of growing a tree
- The tree starts with a branch (i.e. the trunk)
- At each iteration, any branch without child branches grows two child branches out of the branch
- The tree is fully grown in 5 iterations



Starting image | After iteration 1 | After iteration 2 | After iteration 3 | After iteration 4 | After iteration 5

L-System Strings

- Although the results of L-systems are usually some form of images, they are represented using simple text in the system, which we call a *string*
- An L-system string can have letters and symbols
- Here are some commonly used ones:
  - Capital letters such as A, B, F, X and Y
  - Symbols such as + and -
- These letters and symbols have some associated drawing actions so that images can be drawn by reading the string

An Example L-System String

- Here is a simple L-system string:

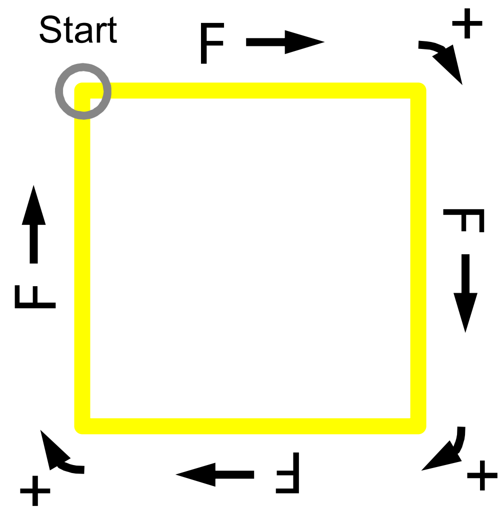
  F+F+F+F

- The letter 'F' and the symbol '+' represent the following actions:

  F : moving forward

  + : turning right

- Then by reading the string from left to right, we can perform the associated actions and draw a square as a result



Using Turtle Graphics for L-Systems

- Turtle graphics is very useful in drawing L-System strings
- For example, the L-System string on the previous slide can be translated to the turtle graphics code shown on the right:

F+F+F+F

```
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
```

L-System Rules

- Rules are used to tell the system how the L-system string grows
- Each rule is a simple replacement of a particular letter or symbol
- For example, a rule can say, from a given L-system string, replacing every occurrence of a letter F with a string FF, which can be written like this:

$$F \rightarrow FF$$

- Let's see how the above rule works with our example L-system string F+F+F+F

Using an L-System Rule

- Given the following example string and rule:

  The L-system string: F+F+F+F

  The L-system rule: F → FF

- Applying the rule to the string means replacing every matching letter / symbol, i.e.:

F+F+F+F

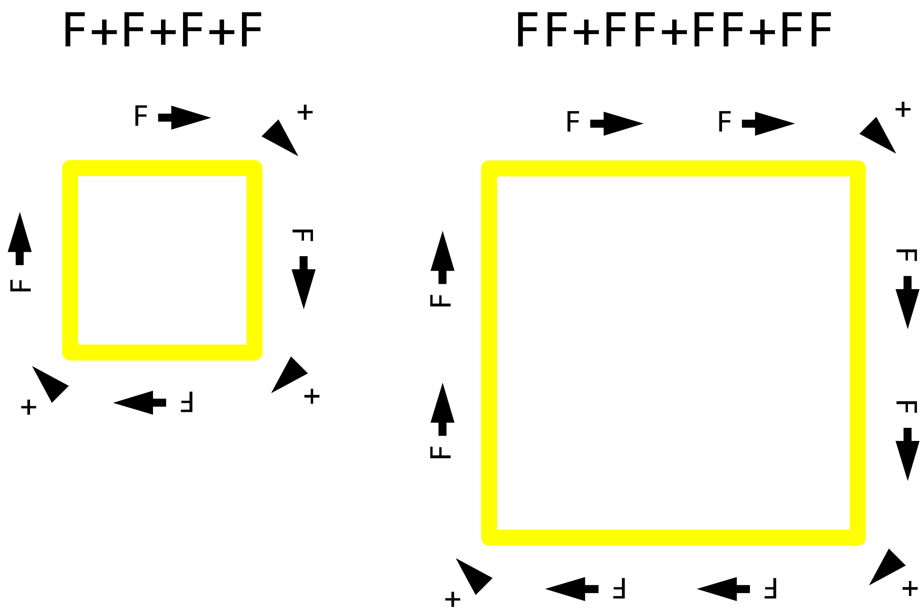
$\boxed{FF} + \boxed{FF} + \boxed{FF} + \boxed{FF}$

After the First Iteration

- After using the L-system rule once, we say the system has completed one iteration
- In the example, the L-system string has become:

FF+FF+FF+FF

- It is easy to see the square produced by the above string will double the size of the initial one



After the Second Iteration

- If you want to you can continue to grow the L-system string after the first iteration
- At iteration 1, the L-system string has become:

FF+FF+FF+FF

- Applying the L-system rule again will result in the following string (each F has become FF):

FFFF+FFFF+FFFF+FFFF

- The square drawn using the above string will be four times the size of the initial one

Stopping the L-System

- You can keep on applying the L-system rule repeatedly for many more iterations
- At some point, you may want to stop growing the image
- You can do that by simply asking the L-system to stop at a particular iteration
- For our example, if we stop at iteration n, we will get a square with a size of 2^n times of the initial one

A Python Program for the Example 1/3

- We can put together some Python code to create our example L-system:

```
import turtle

turtle.speed(0)
turtle.width(2)

iterations = int(input("How many iterations
                                  do you want? "))
string = "F+F+F+F"
```

This is the number of iterations that the program uses for the L-system

This is the initial L-system string

A Python Program for the Example 2/3

- This part of the code applies the rule repeatedly to the L-system string for the chosen iterations:

```
for _ in range(iterations):
    result = ""

    for letter in string:
        if letter == "F":
            result = result + "FF"
        else:
            result = result + letter

    string = result
```

This is the rule:
F → FF

Read each letter and replace if necessary

This makes sure the resulting string is repeatedly grown

A Python Program for the Example 3/3

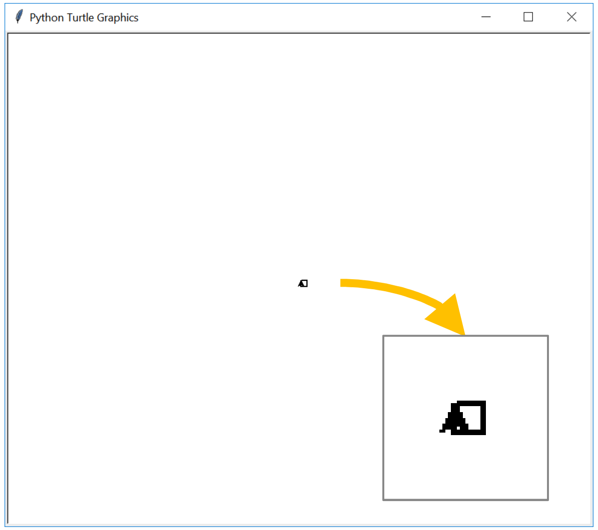
- Finally, after applying the rule, the code draws the image by reading each letter/symbol from the L-system string

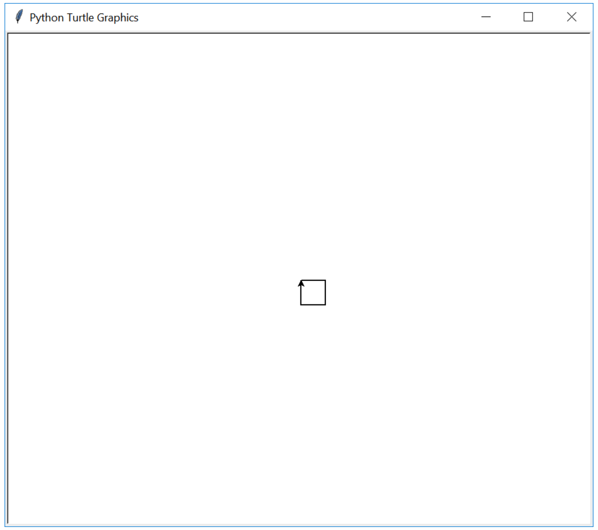
```
for letter in string:
    if letter == "F":
        turtle.forward(10)
    elif letter == "+":
        turtle.right(90)

turtle.done()
```

In this example, ‘`F`’ means moving forward by 10 pixels and ‘+’ means turning right by 90 degrees

Example Output

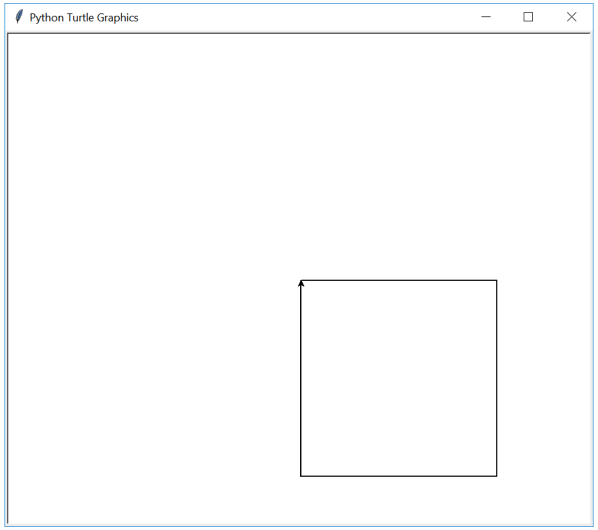
How many iterations do you want? 0





How many iterations do you want? 2

How many iterations do you want? 5



Moving Distance

- Some letters in the L-system strings mean moving forward
- You can control how far to move each time

```
if letter == "F":
    turtle.forward(10)
```

The distance used by our example is 10

- The resulting image will structurally look the same regardless of the size of this distance
- If you use a small / large distance, you will get a small / large resulting image

Koch Triangle

- The previous square example is not very interesting
- Let's look at a more interesting example – the Koch triangle
- The Koch triangle uses one letter and two symbols:

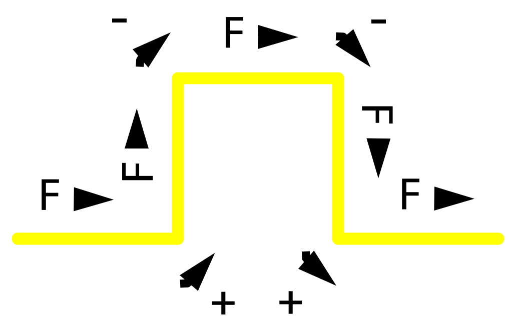
  F : moving forward

  \+ : turning left

  \- : turning right

- The initial L-system string is: F
- The rule is: F → F+F-F-F+F

F+F-F-F+F



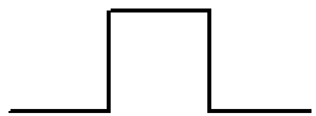
Koch Triangle at Iteration 1

- The Koch triangle
  at the start (iteration 0)

  - The L-system string: F

F -> F+F-F-F+F

- The Koch triangle
  at iteration 1

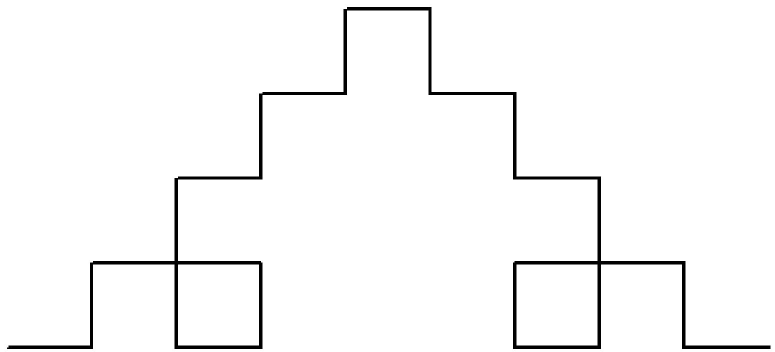


  - The L-system string: F+F-F-F+F

Koch Triangle at Iteration 2

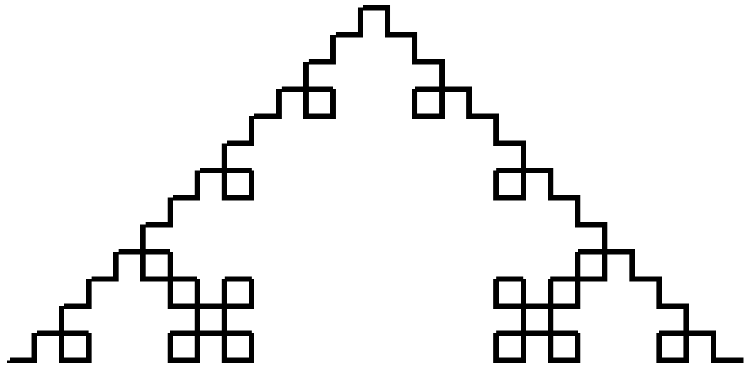
- The L-system string at iteration 1: F+F-F-F+F
- The Koch triangle at iteration 2
  - The L-system string:

F+F-F-F+F +
F+F-F-F+F -
F+F-F-F+F -
F+F-F-F+F +
F+F-F-F+F



Koch Triangle at Iteration 3

- The Koch triangle at iteration 3

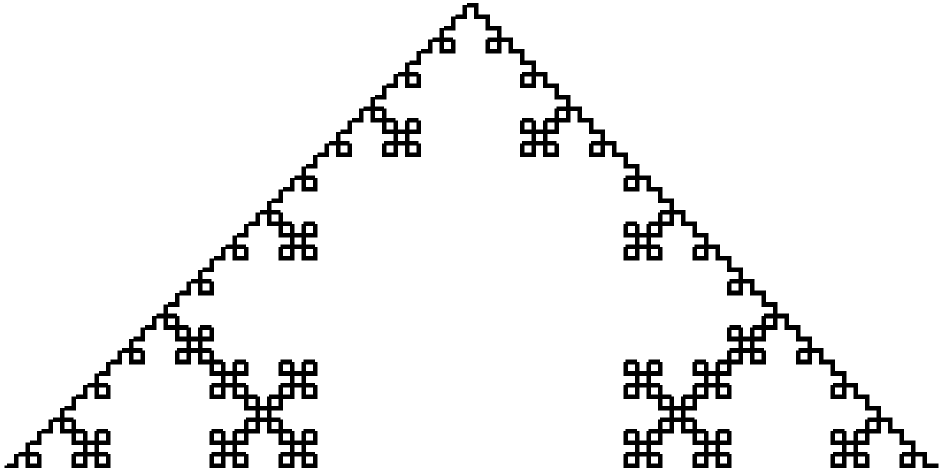


- The L-system string:

F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+
F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-
F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-
F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+
F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F

Koch Triangle at Iteration 4

- The Koch triangle at iteration 4
- The L-system string:



```
F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-
F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-
F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+
F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-
F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-
F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-
F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-
F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-
F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-
F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-
F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-
F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+
F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-
F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-
F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F
```

Python Code for Koch Triangle 1/2

- It is easy to make a Python program for the Koch triangle by modifying the previous example
- First, change the initial string:

from:
```
string = "F+F+F+F"
```

to:
```
string = "F"
```

- Then, change the rule:

from:
```
if letter == "F":
    result = result + "FF"
```

to:
```
if letter == "F":
    result = result + "F+F-F-F+F"
```

Python Code for Koch Triangle 2/2

- Finally, change and extend the drawing code:

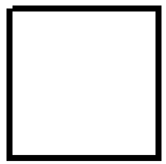
from:
```
if letter == "F":
    turtle.forward(10)
elif letter == "+":
    turtle.right(90)
```

to:
```
if letter == "F":
    turtle.forward(10)
elif letter == "+":
    turtle.left(90)
elif letter == "-":
    turtle.right(90)
```

Note that the meaning of ‘+’ has been changed

Rings

- Here is another example:
  - Initial string: F-F-F-F

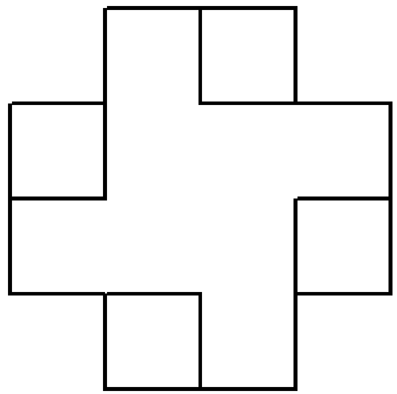


  - Rule: F → FF-F-F-F-F-F+F
  - Letters and symbols:

    F : moving forward

    \+ : turning left

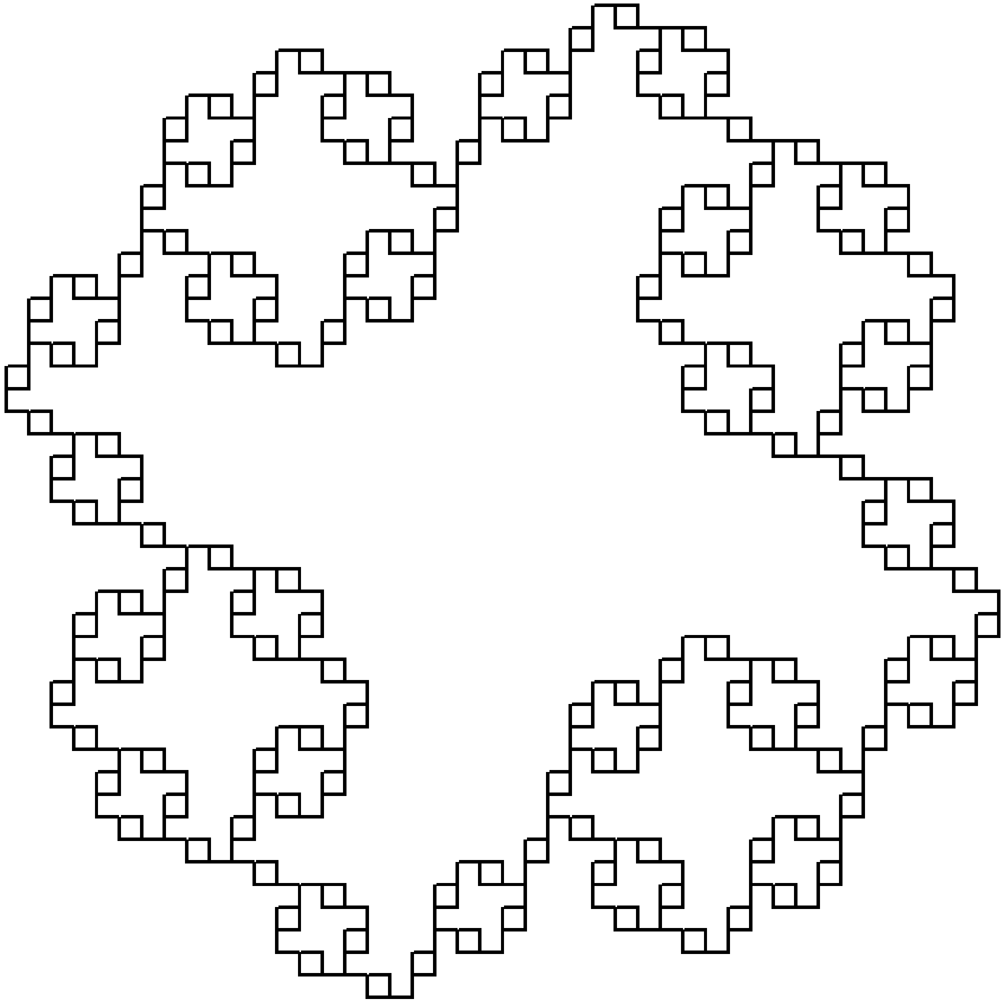
    \- : turning right



- At iteration 1, the string becomes:

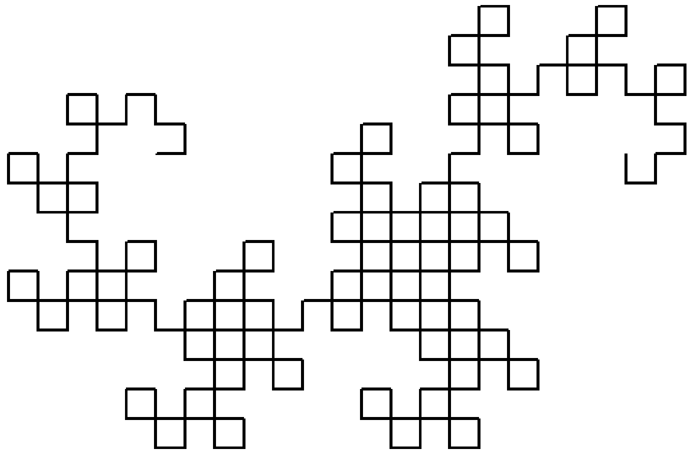
  FF-F-F-F-F-F+F-FF-F-F-F-F-F+F-
  FF-F-F-F-F-F+F-FF-F-F-F-F-F+F

Rings at Iteration 3



Using Multiple Rules

- The L-systems we have shown so far use only one L-system rule
- You can use more than one rules to create L-system images such as the dragon curve
- Each rule should then describe the replacement of a unique letter/symbol in the L-system string



Dragon Curve

- Here is the dragon curve:
  - Initial string: FX
  - Rules: X → X+YF+

    Y → -FX-Y
  - Letters and symbols:

    F : moving forward

    \+ : turning left

    \- : turning right

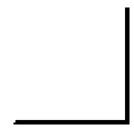
    X : no action

    Y : no action

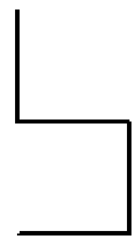
Letters/symbols can have no associated drawing action

At the start:
FX

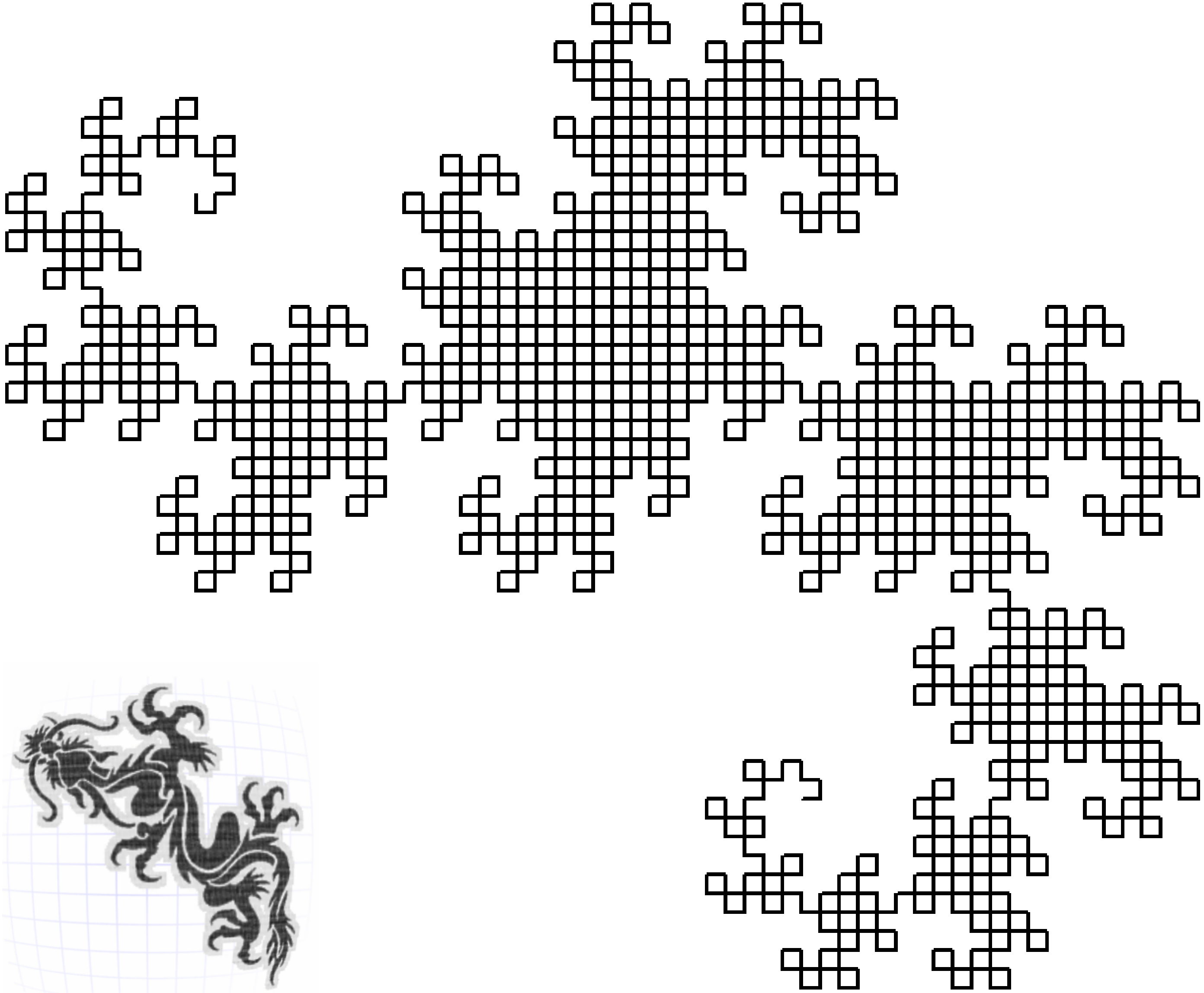
At iteration 1:
FX+YF+



At iteration 2:
FX+YF++
-FX-YF+



Dragon Curve at Iteration 11



Extending the Code for Dragon Curve

- Using the previous program code, the code can be similarly extended to draw the dragon curve
- In the dragon curve L-system, the code will need to do replacements using two rules, i.e.:

```
if letter == "X":
    result = result + "X+YF+"
elif letter == "Y":
    result = result + "-FX-Y"
```

X → X+YF+

Y → -FX-Y

- It works but it is not very efficient to keep on adjusting the rules in the if statements for different systems

Using a List of Lists

- A better way to write the L-system program is to put the rules in a 'list of lists'
- For example, the list on the right represents the rules in the dragon curve system
- By putting the rules near the top of the program, you can easily change the L-system by replacing the content of the list

```
rules = [
        ["X", "X+YF+"] ,
        ["Y", "-FX-Y"]
]
```

This is a list, inside a list and the whole data structure is called a 'list of lists'

Another Example of a List of Lists

- Here's another example of a list of lists
- It stores some lecture information

```
lecture_events = [
    ["Monday", "9:30am", "L2 lecture", 113],
    ["Monday", "1:30pm", "L1 lecture", 107],
    ["Wednesday", "9:30am", "L2 lecture", 113],
    ["Friday", "9:00am", "L1 lecture", 107]
]
```

- On the next slide we show some examples of how to get information from this list of lists
- Note that the first item in a Python list is item 0 (not item 1)

```
>>> lecture_events = [
        ["Monday", "9:30am", "L2 lecture", 113],
        ["Monday", "1:30pm", "L1 lecture", 107],
        ["Wednesday", "9:30am", "L2 lecture", 113],
        ["Friday", "9:00am", "L1 lecture", 107]
]
>>> len(lecture_events)
4
>>> print(lecture_events[0])
['Monday', '9:30am', 'L2 lecture', 113]
>>> print(lecture_events[1])
['Monday', '1:30pm', 'L1 lecture', 107]
>>> print(lecture_events[2])
['Wednesday', '9:30am', 'L2 lecture', 113]
>>>
>>> one_lecture_event = lecture_events[2]
>>> print(one_lecture_event[0])
Wednesday
>>> print(one_lecture_event[1])
9:30am
>>> print(one_lecture_event[2])
L2 lecture
>>> len(one_lecture_event)
4
>>>
>>> print(lecture_events[3][0])
Friday
>>>
```

The L-System Lab

- In the coming lab, you will need to write an L-system program which uses a list of lists to store the rules
- Using the program, you can then easily change the L-system by replacing the content of the list at the top of the program

Changing the L-System Angle

- So far, '+' and '-' have always used 90 degrees
- We can make more creative images if we use other angles instead of 90 degrees
- You can easily adjust the angle in the drawing code for '+' and '-'
- The following L-systems all use 60 degrees in the drawing stage:
  - Koch snowflake
  - Sierpinski triangle
  - Peano-Gosper curve

```
...
elif letter == "+":
    turtle.left( 60 )
elif letter == "-":
    turtle.right( 60 )

...
```

Koch Snowflake

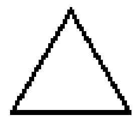
- Here is the L-system for the Koch snowflake:
  - Initial string: F++F++F
  - Rule: F → F-F++F-F
  - Letters and symbols:

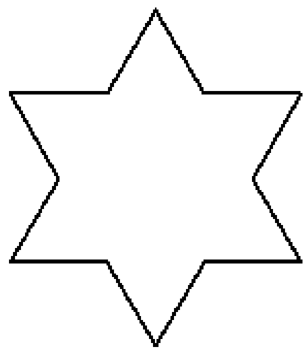
    F : moving forward

    + : turning left 60 degrees

    - : turning right 60 degrees

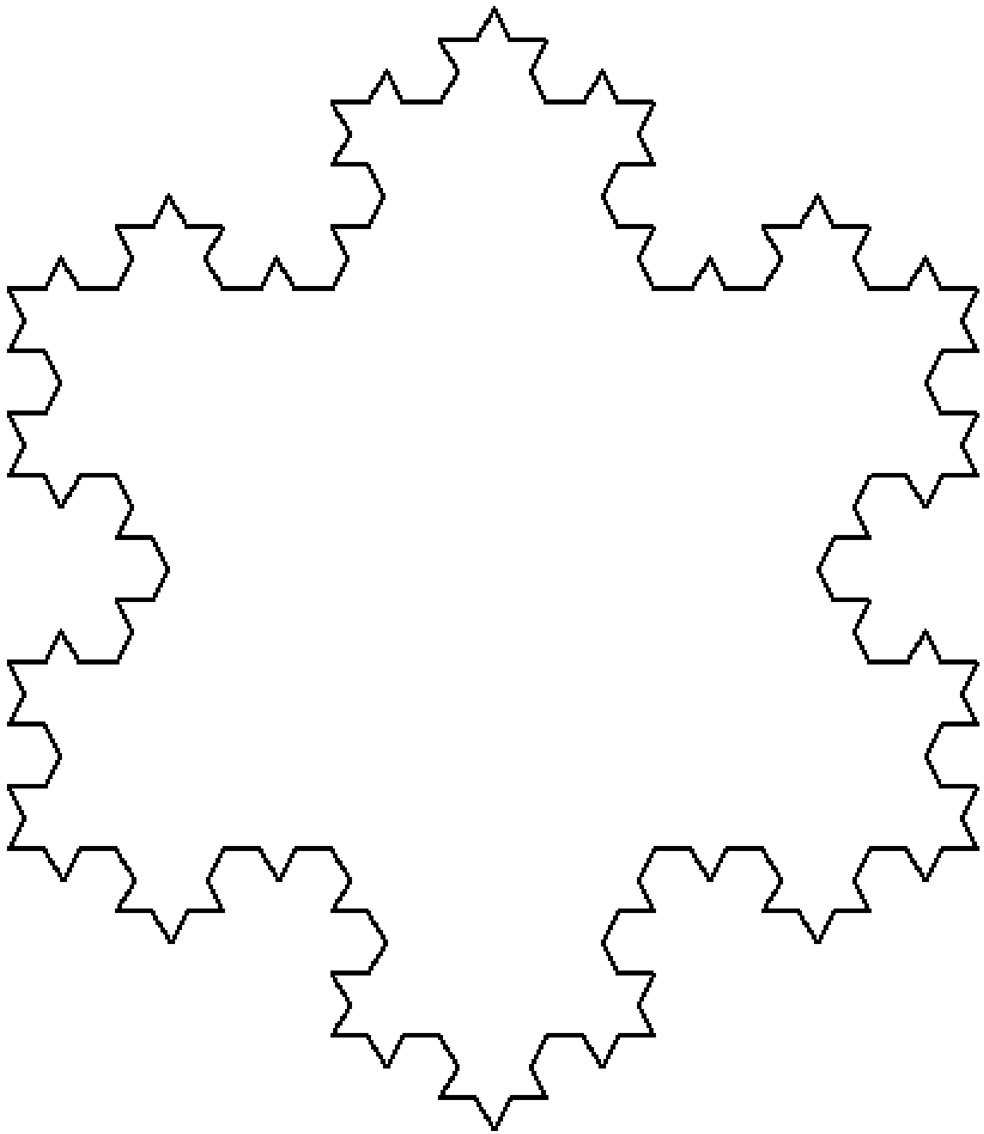
At the start:
F++F++F





At iteration 1:
F-F++F-F++F-F++F-F++F-F++F-F

Koch Snowflake at Iteration 3

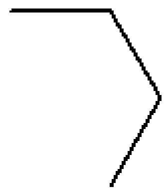


Sierpinski Triangle

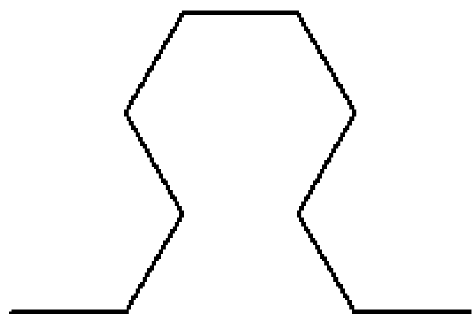
- Here is the L-system for the Sierpinski triangle:
  - Initial string: A
  - Rules: A → B-A-B
    B → A+B+A
  - Letters and symbols:
    A : moving forward
    B : moving forward
    \+ : turning left 60 degrees
    \- : turning right 60 degrees

At the start:
A

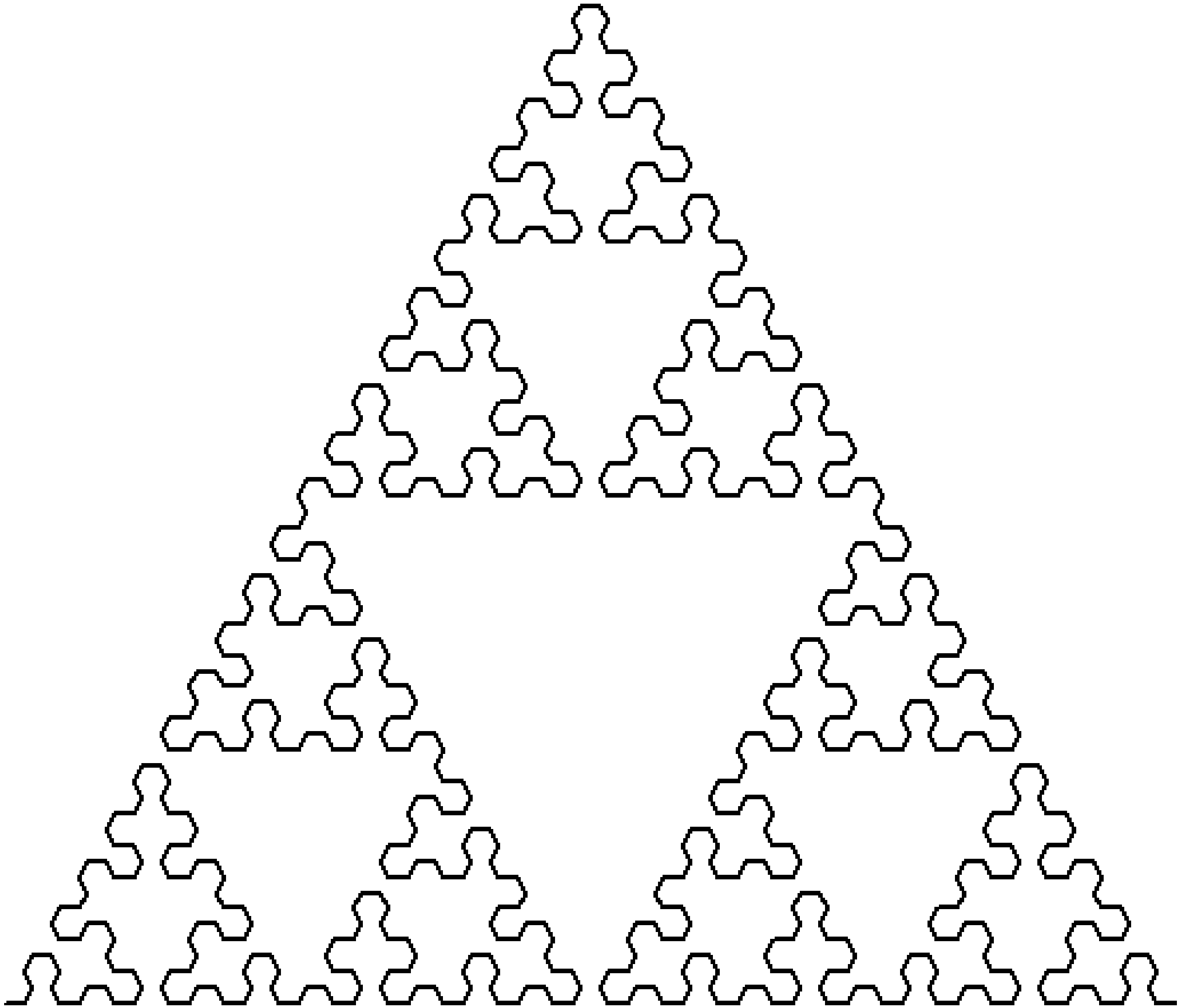
At iteration 1:
B-A-B



At iteration 2:
A+B+A-B-A-B+A+B+A



Sierpinski Triangle at Iteration 6

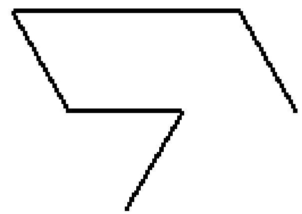


Peano-Gosper Curve

- Here is the L-system for the Peano-Gosper curve:
  - Initial string: X
  - Rule: X → X+YF++YF-FX--FXFX-YF+
    Y → -FX+YFYF++YF+FX--FX-Y
  - Letters and symbols:
    F : moving forward
    + : turning left 60 degrees
    - : turning right 60 degrees
    X : no action
    Y : no action

At the start: X
Nothing is shown (no action)

At iteration 1:
X+YF++YF-FX--FXFX-YF+



Peano-Gosper Curve at Iteration 4