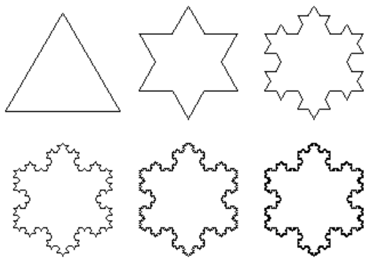
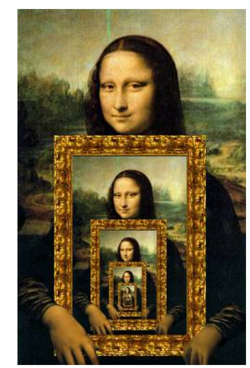


Class 7: Recursion

CSCI 101
Spring 2018

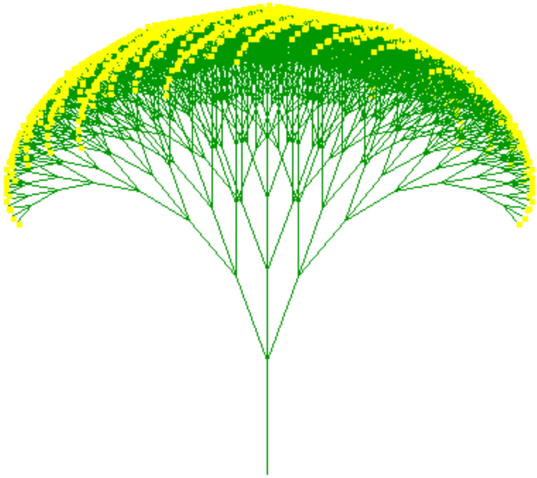


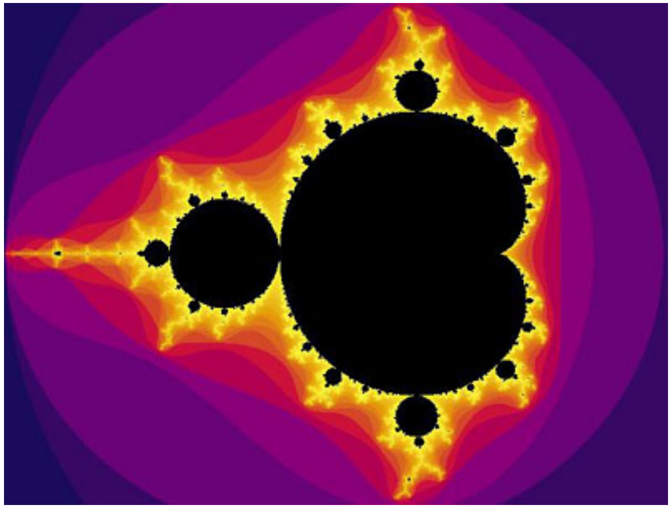
Profs Briggs
and Grant

Class 7: Recursion
CSCI 101
Spring 2018
Profs Briggs
and Grant
Class 7: Recursion
CSCI 101
Spring 2018
Profs Briggs
and Grant
Class 7: Recursion
CSCI 101
Spring 2018
Profs Briggs
and Grant

Recursive Structures

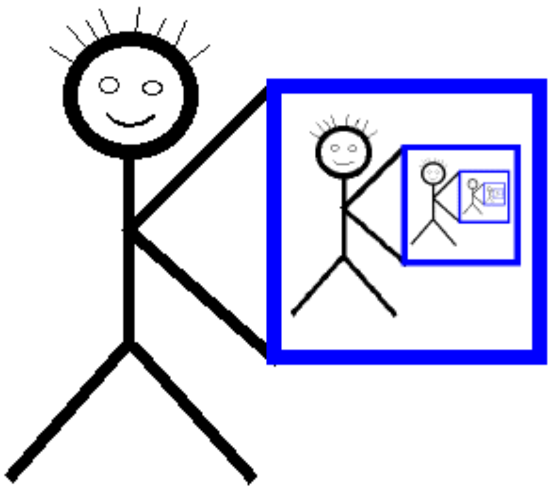
- A recursive structure is one in which part of the structure resembles the whole thing
- Examples:





Recursive Function Definitions

A recursive function definition is a function definition in which an *application of the function itself* makes up part of its definition, i.e. the function is defined in terms of itself.



Computing Factorial

- Example from last time: What is n factorial?
  - $n! = 1 \times 2 \times 3 \times \ldots \times n$
- Recursive definition of factorial:
  - $$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

base case (if n = 0)

recursive case (if n > 0)

n! is defined in terms of (n-1)!

Recursive Algorithms in Python

- A recursive algorithm is an algorithm whose definition involves calling itself (with “simpler” or “smaller” parameters)

- Example:

```
def factorial(n):
    if n == 0:
      return 1                      ] base case
    else:
      return n * factorial(n-1)     ] recursive case
```

Example: Computing 6!

- 6!
- 6 * 5!
- 6 * (5 * 4!)
- 6 * (5 * (4 * 3!))
- 6 * (5 * (4 * (3 * 2!)))
- 6 * (5 * (4 * (3 * (2 * 1!))))
- 6 * (5 * (4 * (3 * (2 * (1 * 0!)))))
- 6 * (5 * (4 * (3 * (2 * (1 * 1)))))
- 6 * (5 * (4 * (3 * (2 * 1))))
- 6 * (5 * (4 * (3 * 2)))
- 6 * (5 * (4 * 6))
- 6 * (5 * 24)
- 6 * 120
- 720

Creating a recursive solution

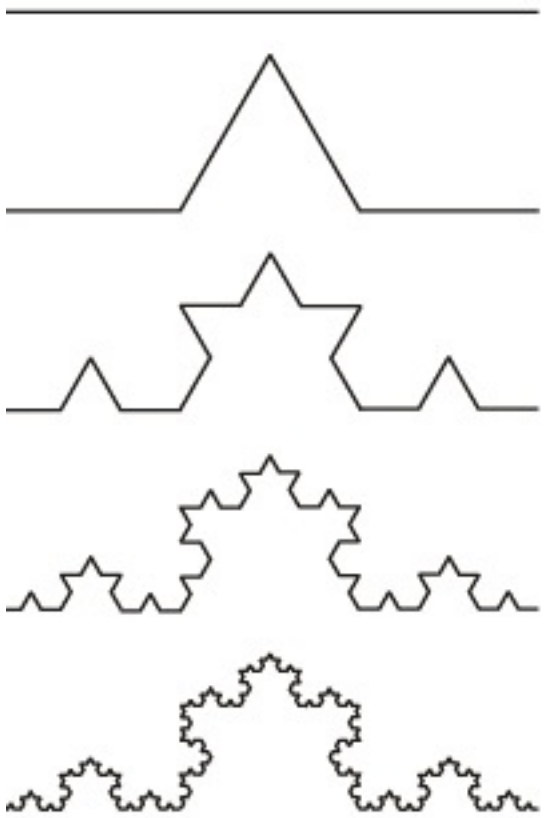
Base case:

- A trivial and easily solvable instance of the problem

Recursive case:

- Break the problem up into solvable problems and smaller versions of the same problem [*must make progress toward the base case]*
- Make the problem smaller by looking at smaller numbers, less data, or fewer choices
- Figure out how to combine the solutions to smaller problems to get the solution to the overall problem

Koch Curves



Level 0

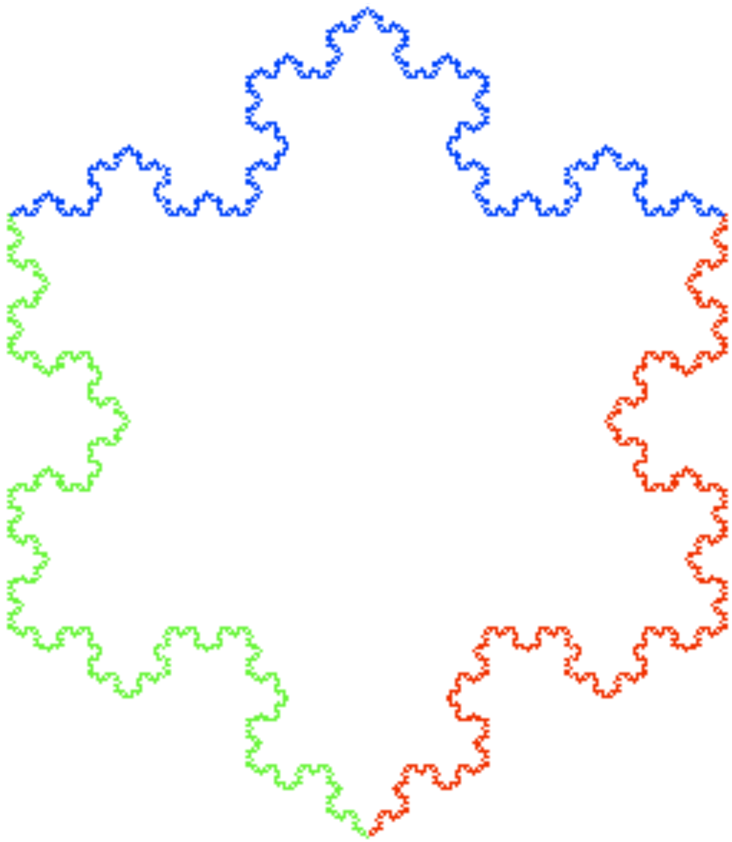
Level 1

Level 2

Level 3

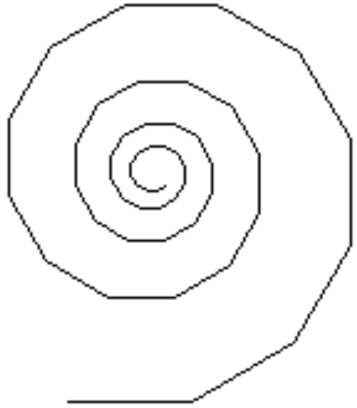
Level 4

The Koch Snowflake



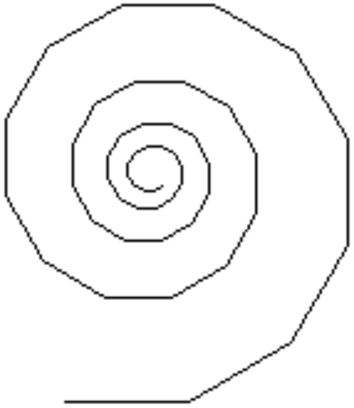
An inward folding curve

```
def curve(len, level):
  if level > 0:
    turtle.forward(len)
    turtle.left(30)
    curve(len * 0.95, level-1)
```

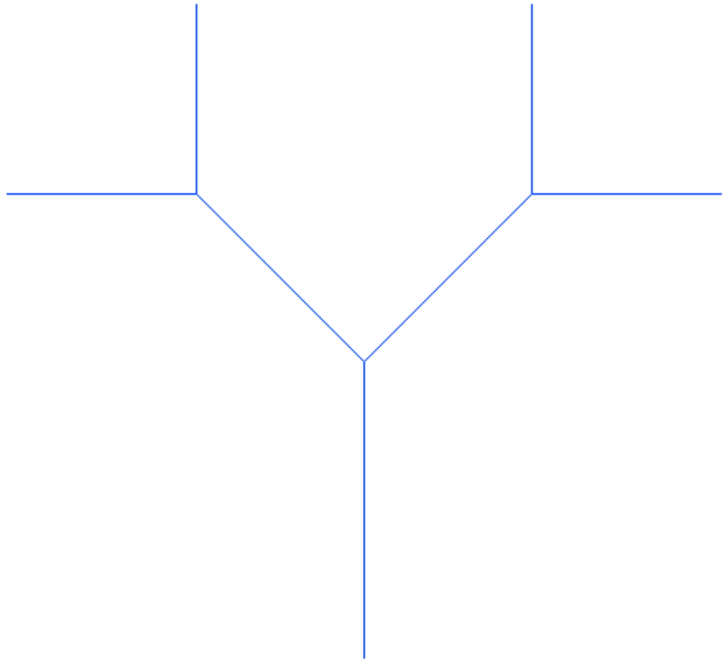


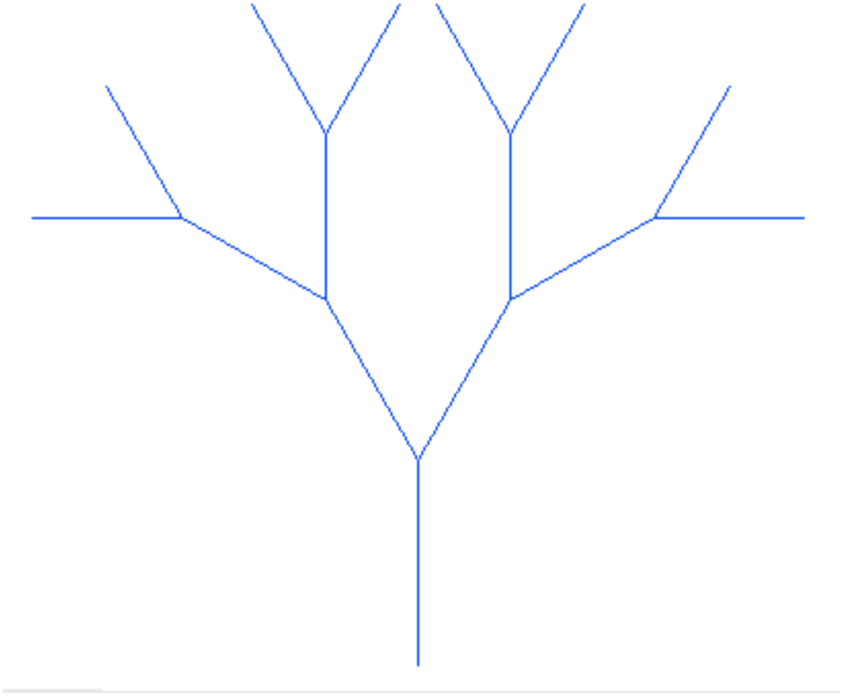
Getting the Turtle back home

```
def curve(len, level):
    if level > 0:
        turtle.forward(len)
        turtle.left(30)
        curve(len * 0.95, level-1)
        turtle.right(30)
        turtle.backward(len)
```

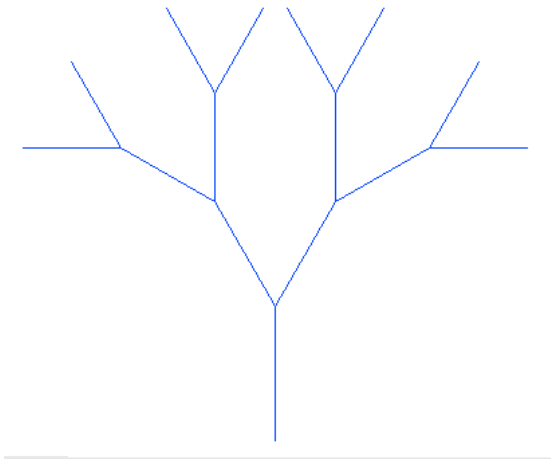


Drawing a Tree





Drawing a Tree



```
def drawTree(levels, len, angle, shrink):
    if levels > 0:
      t.forward(len)
      t.left(angle)
      drawTree(levels-1, shrink * len, angle, shrink)
      t.right(2*angle)
      drawTree(levels-1, shrink * len, angle, shrink)
      t.left(angle)
      t.backward(len)
```

Towers of Hanoi

Move *n* disks from pole A to pole B:

- move 1 disk at a time
- never place a larger disk on top of a smaller one
- use the extra pole for "temporary storage"

Towers of Hanoi

Move *n* disks from pole A to pole B:

1. Move top n-1 disks from A to C
2. Move largest disk from A to B
3. Move n-1 disks from C to B

Towers of Hanoi

How many moves to solve puzzle for n disks?

n	# moves
1	1
2	3
3	7
4	15

$= 2^n - 1$