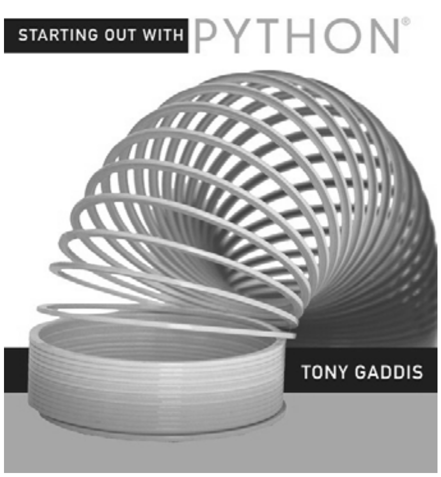
Topic 9: Recursion

To Understand Recursion You Must First Understand Recursion

1

Recommended Readings

- Chapter 11



2

Recursion

- Definition:
  - See Recursion
  - Defining something in terms of itself
    - Generally using a smaller or simpler version
- Recursive Function
  - A function that calls itself

3

A Simple Example

- Compute n factorial:
  - Using a loop
    - Initialize result to 1
    - for i ranging from 1 to n (inclusive)
      - Multiply result by i
  - Another solution
    - By definition, 0! is 1
    - View n! as n * (n-1)!

4

A Simple Example

5

Recursion

- A well formed recursive function normally has two cases
  - Base Case:
    - Does not make a recursive call
    - Permits function to terminate
  - Recursive Case:
    - Function calls itself
    - Generally must be a call to a smaller or simpler version of the problem

6

Useful Examples of Recursion

- Drawing fractals
- Finding a path through a maze
- Flood fill / “paint bucket” tool
- Merge sort, quick sort, binary search
- Finding the total size of all of the files in a directory and its subdirectories
- Parsing / evaluating expressions
- …

7

Greatest Common Divisor

- Finding the greatest common divisor of two positive integers, x and y:
  - If x can be evenly divided by y, then gcd(x,y) is y
  - Otherwise, gcd(x,y) is gcd(y, remainder of x/y)

8

Fibonacci Numbers

- A sequence of values:
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …
- Defined recursively:
  - By definition:
    - fib(0) is 0
    - fib(1) is 1
  - Remaining values:
    - Formed by computing the sum of the previous two values in the sequence

9

Fibonacci Numbers

10

Advantages of Recursion

- Well suited to some problems
  - Tree traversals
  - Flood fill
  - Fractal images
  - Quick sort / merge sort
- Often easier to implement, sometimes faster, than iterative

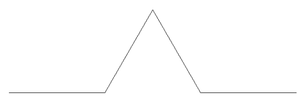
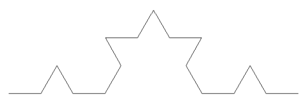
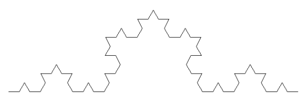
11

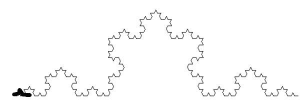
Advantages of Iteration

- Typically
  - Faster (but not always!)
  - Requires less memory (most of the time!)
- But some problems are messy to express iteratively

12

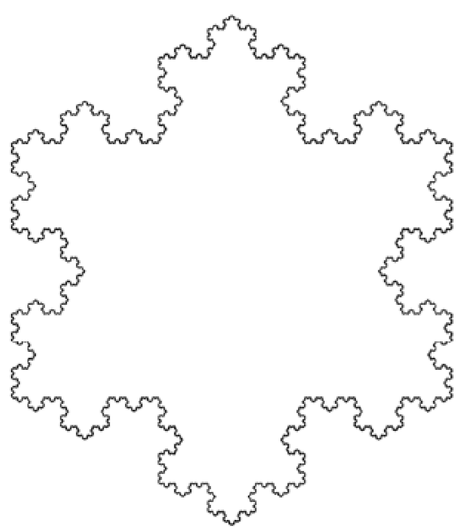
Fractals

- Self similar images
- Often have reasonably simple recursive definitions
  - 1 
  - 2 
  - 3 

4 

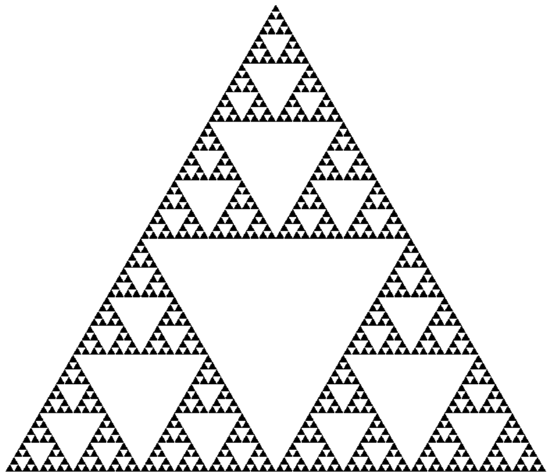
13

Koch Snowflake



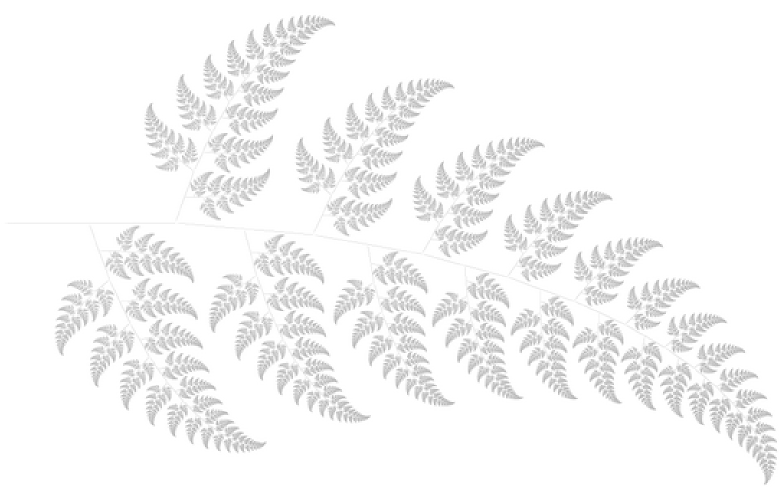
14

Sierpinski Triangle



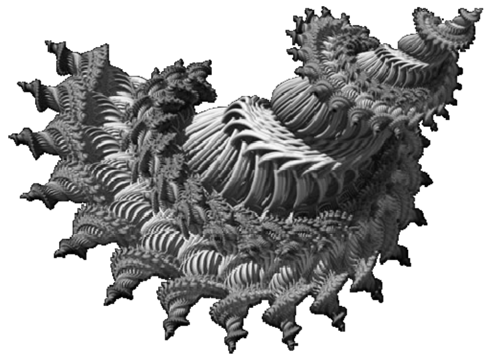
15

Fractal Fern



16

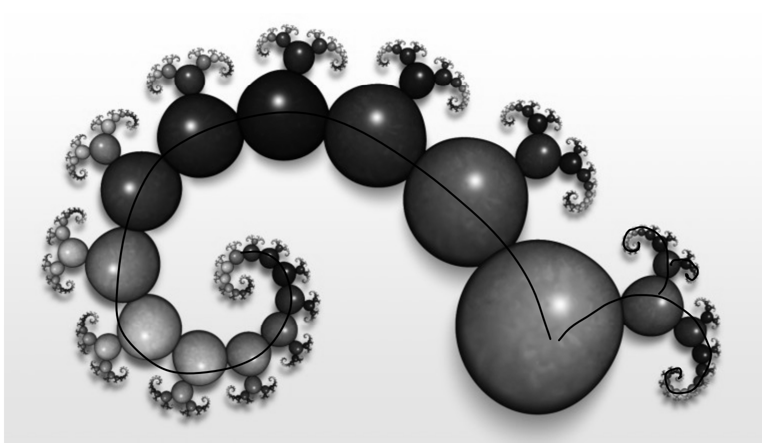
Fractal Art



Spiral Fantasy by Alfred Laing

17

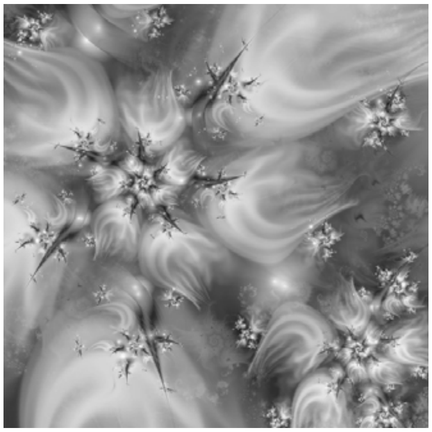
Fractal Art



Simple Thing by Philip Taylor

18

Fractal Art



Edenesque by Helen Grainge (top)
Rose by Keith Mackay (right)

19

Fractal T-Square

20

Fractal T-Square

Maze Path Finding

- Consider a two dimensional list containing 4 different values
  - Entrance for the maze
  - Exit for the maze
  - Open spaces
  - Walls
- Assume that the maze is fully enclosed

Maze Path Finding

- Algorithm solve(map, x, y)
  - If the current square is a wall or a space we have already visited, return failure
  - If the current square is the exit point, mark it as part of the solution and return success
  - Mark the current square as part of the solution
  - If solve(map, x, y+1) is successful, return success
  - If solve(map, x, y-1) is successful, return success
  - If solve(map, x+1, y) is successful, return success
  - If solve(map, x-1, y) is successful, return success
  - Mark the current square as visited but not part of the solution
  - Return failure

23

Maze Path Finding

24

Recursion

- Recursion: See Recursion
  - Very useful for some problems
  - Caution:
    - Can be inefficient
    - Not a good solution for all problems – Use it when appropriate, don't abuse it