

Topic 9: Recursion

To understand recursion, you must first understand recursion.

Textbook

- Recommended Exercises
 - The Python Workbook, 3rd Edition: 202, 204, 205, 206 and 207; or
 - The Python Workbook, 2nd Edition: 178, 179, 180, 181 and 182
- Recommend Readings
 - The Python Workbook, 2nd or 3rd Edition: Chapter 8

Recursion

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

A Small Example

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

A Small Example

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

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
- ...

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 $\gcd(y, \text{remainder of } x/y)$

Fibonacci Numbers

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

Fibonacci Numbers

Advantages of Recursion

- Very well suited to some problems
 - Tree traversals
 - Flood fill
 - Fractal images
 - Quick sort / merge sort
 - ...
- Easier to implement for some problems, sometimes faster than iterative

Advantages of Iteration

- Typically
 - Faster (but not always)
 - Requires less memory (most of the time)
- Can be more intuitive for some problems / people
- But some problems are messy to express iteratively

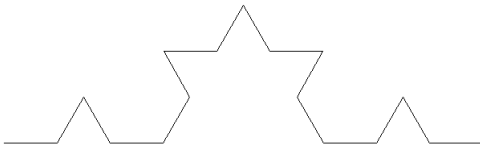
Fractals

- Self similar images
- Often have reasonably simple recursive definitions

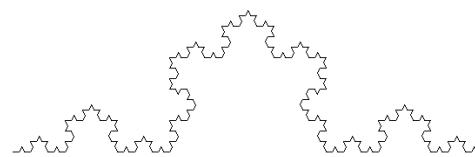
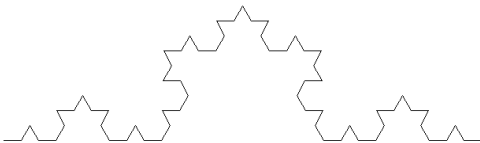
- 1



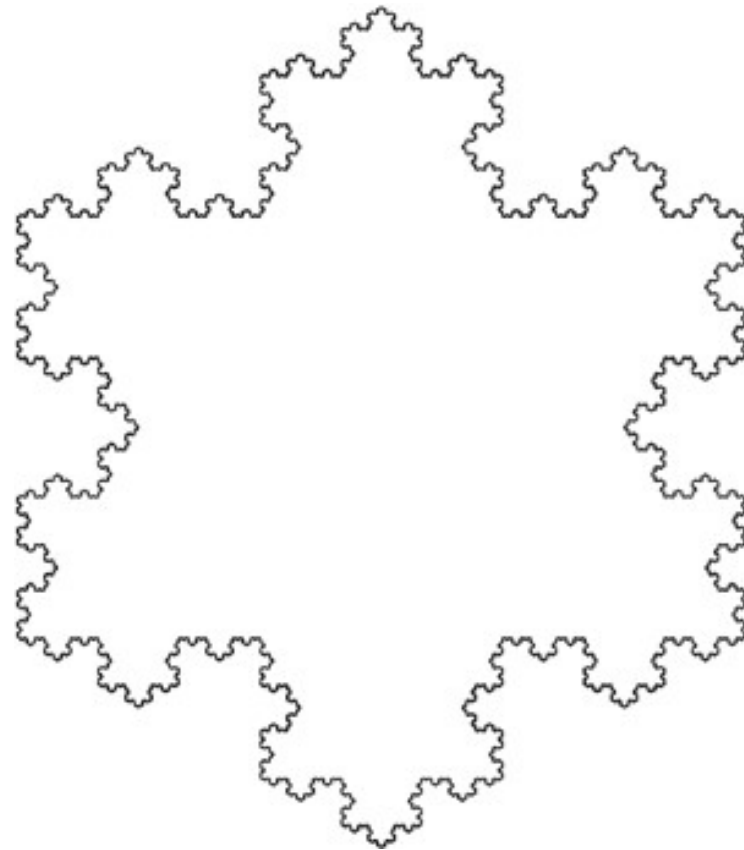
- 2



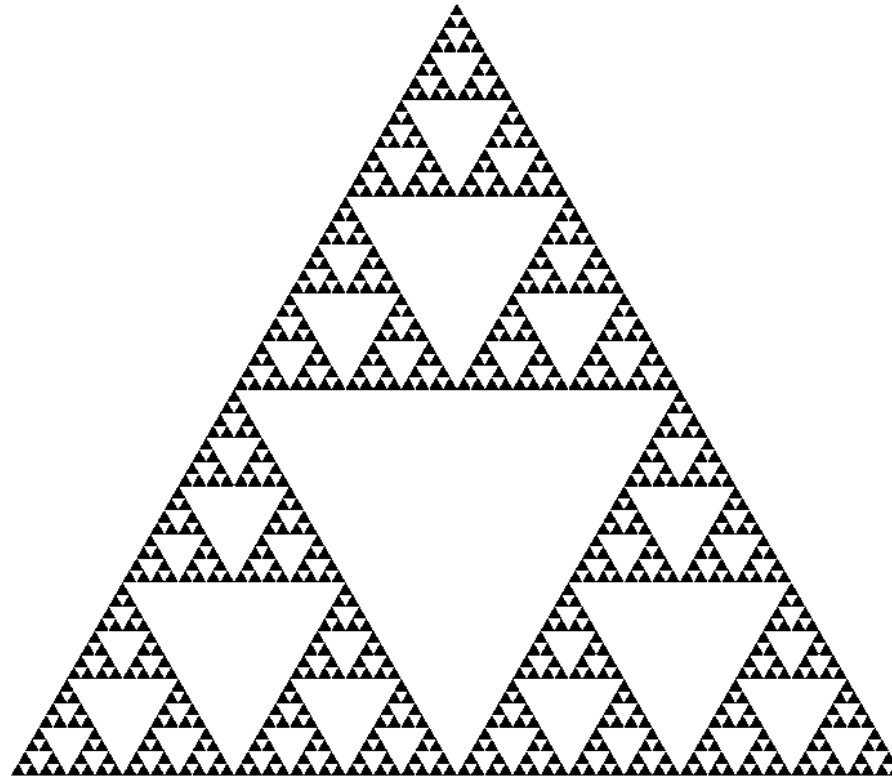
- 3



Koch Snowflake



Sierpinski Triangle



Sierpinski Trianglge
Source: <http://commons.wikimedia.org/wiki/File:Sierpinski-Trigon-7.svg>
License: Public Domain

Fractal T-Square

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

Maze Path Finding

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