UNIT 5A
Recursion: Basics

Recursion

- A recursive operation is an operation that is defined in terms of itself.

Sierpinski's Gasket

http://fusionanomaly.net/recursion.jpg

Recursive Definitions

- Every recursive definition includes two parts:
  - Base case (non-recursive)
    A simple case that can be done without solving the same problem again.
  - Recursive case(s)
    One or more cases that are "simpler" versions of the original problem.
    - By "simpler", we sometimes mean "smaller" or "shorter" or "closer to the base case".

GCD

```
def gcd2(x, y)
  if y == 0 then
      return x
  else
      return gcd2(y, x % y)
  end
end
```

base case

recursive case
(a "simpler" version of the same problem)

Factorial

- Definition: $n! = n(n-1)(n-2)...(2)(1)$
- Since $(n-1)(n-2)...(2)(1) = (n-1)!$
  - $n! = n(n-1)!$, for $n > 0$
  - $n! = 1$ for $n = 0$ (base case)
- Example:
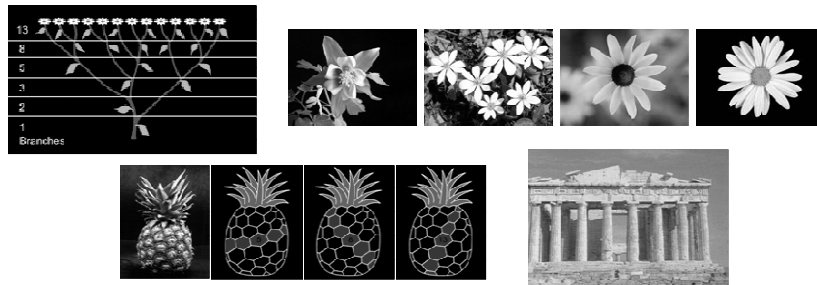 $4! = 4(3!)$ $= 4(6) = 24$
      $3! = 3(2!)$ $= 3(2) = 6$
          $2! = 2(1!)$ $= 2(1) = 2$
              $1! = 1(0!) = 1(1) = 1$

Factorial in Ruby

```ruby
def factorial(n)
   if n == 0 then
     return 1
   else
     return n * factorial(n-1)
   end
end
```

Fibonacci Numbers

- A sequence of numbers such that each number is the sum of the previous two numbers in the sequence, starting the sequence with 0 and 1.
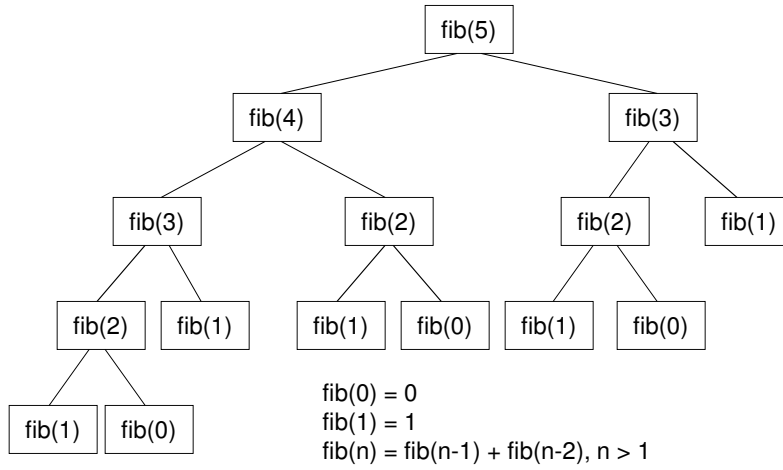
- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, etc.

Recursive Definition

- Let fib(n) = the nth Fibonacci number, n ≥ 0
  - fib(0) = 0   (base case)
  - fib(1) = 1   (base case)
  - fib(n) = fib(n-1) + fib(n-2),   n > 1

Recursive Definition

```
                        fib(5)
                      /        \
                fib(4)          fib(3)
               /      \         /      \
          fib(3)    fib(2)   fib(2)   fib(1)
          /    \    /    \   /    \
     fib(2) fib(1) fib(1) fib(0) fib(1) fib(0)
      /  \
  fib(1) fib(0)
```

fib(0) = 0
fib(1) = 1
fib(n) = fib(n-1) + fib(n-2), n > 1

Recursive Definition

```
                        5
                      /    \
                    3        2
                  /    \    /    \
               2        1   1      1
              /  \     / \  / \
             1    1   1   0 1   0
            /  \
           1    0
```

fib(0) = 0
fib(1) = 1
fib(n) = fib(n-1) + fib(n-2), n > 1
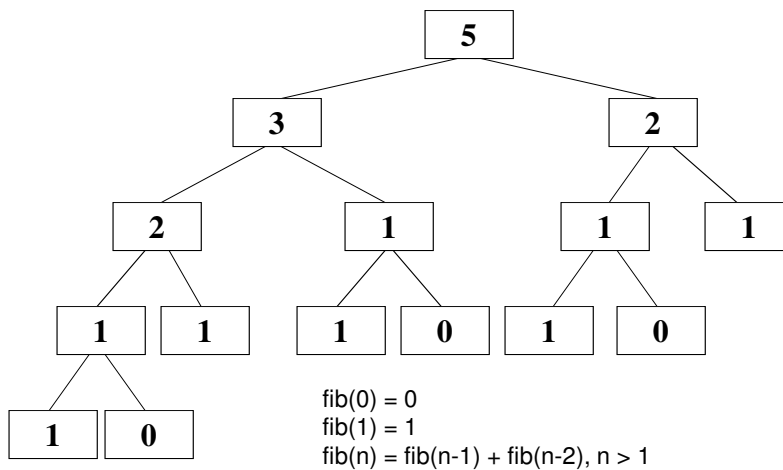
Fibonacci Numbers in Ruby

```
def fib(n)
    if n == 0 or n == 1 then
      return n
    else
      return fib(n-1) + fib(n-2)
    end
end
```

In irb:

```
for i in 0..30 do puts fib(i) end
```
Why does it take longer to print each subsequent value?

Computing the sum of a list

```
def sumlist(list)
    n = list.length
    if n == 0 then
      return 0
    else
      return list[0] + sumlist(list[1..n-1])
    end
end
```
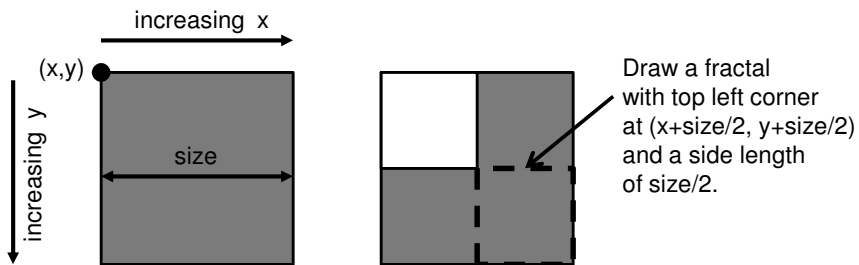
Base case:
The sum of an empty list is 0.

Recursive case:
The sum of a list is the first element +
the sum of the rest of the list.

Simple Fractal

To draw a fractal with top-left corner (x,y) and a side length of size:
- Draw a white square with top-left corner (x,y) and a side length of size/2.
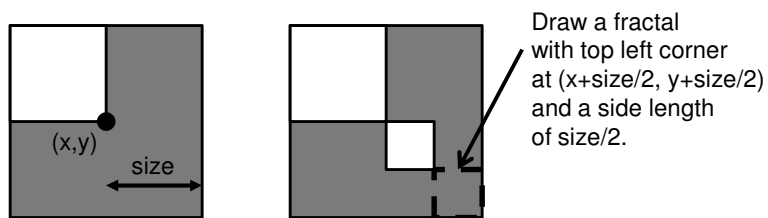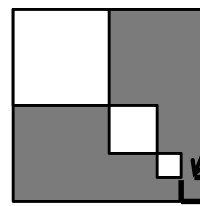- Draw another fractal with top-left corner (x+size/2, y+size/2) and a side length of size/2. [recursive step]

increasing x

(x,y)

increasing y

size

Draw a fractal with top left corner at (x+size/2, y+size/2) and a side length of size/2.

Simple Fractal

To draw a fractal with top-left corner (x,y) and a side length of size:
- Draw a white square with top-left corner (x,y) and a side length of size/2.
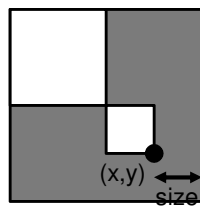- Draw another fractal with top-left corner (x+size/2, y+size/2) and a side length of size/2. [recursive step]

(x,y)

size

Draw a fractal with top left corner at (x+size/2, y+size/2) and a side length of size/2.

Simple Fractal

To draw a fractal with top-left corner (x,y) and a side length of size:

• Draw a white square with top-left corner (x,y) and a side length of size/2.

• Draw another fractal with top-left corner (x+size/2, y+size/2) and a side length of size/2. [recursive step]

Draw a fractal with top left corner at (x+size/2, y+size/2) and a side length of size/2.

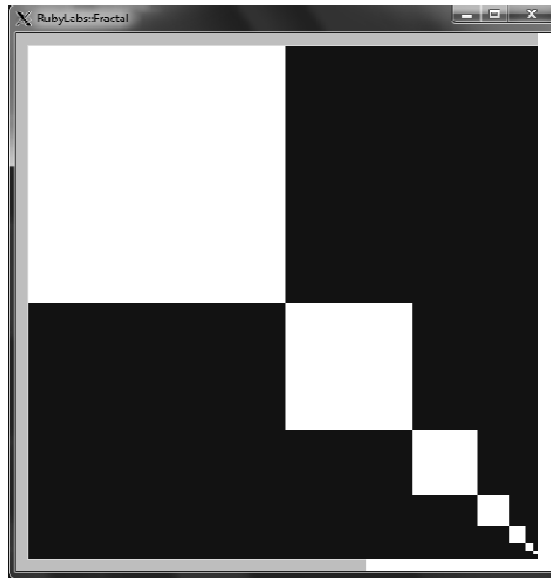# Simple Fractal in Ruby
(not all code shown)

```ruby
def fractal(x, y, size)
    return if size < 2        # base case
    draw_square(x, y, size/2)
    fractal(x+size/2, y+size/2, size/2)
end


def draw_fractal()
    # initial top-left (x,y) and size
    fractal(0, 0, 512)
end
```

Towers of Hanoi

Towers of Hanoi with 8 discs.

- A puzzle invented by French mathematician Edouard Lucas in 1883.
- At a temple far away, priests were led to a courtyard with three pegs and 64 discs stacked on one peg in size order.
  - Priests are only allowed to move one disc at a time from one peg to another.
  - Priests may not put a larger disc on top of a smaller disc at any time.
- The goal of the priests was to move all 64 discs from the leftmost peg to the rightmost peg.
- According to the story, the world would end when the priests finished their work.
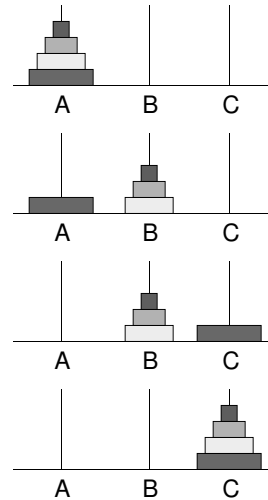
Towers of Hanoi

Problem: Move n discs
from peg A to peg C using peg B.

1. Move n-1 discs from peg A to peg B using peg C. (recursive step)

2. Move 1 disc from peg A to peg C.

3. Move n-1 discs from peg B to C using peg A. (recursive step)

Towers of Hanoi in Ruby

```ruby
def towers(n, from_peg, to_peg, using_peg)
  if n >= 1 then
    towers(n-1, from_peg, using_peg, to_peg)
    puts "Move disc from " + from_peg
        + " to " + to_peg
    towers(n-1, using_peg, to_peg, from_peg)
  end
end
```

In irb: **towers(4, "A", "C", "B")**
How many moves do the priests need to move 64 discs?