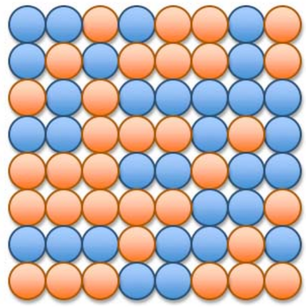


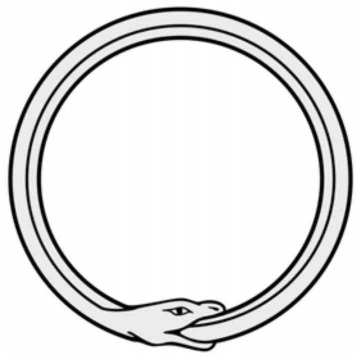
UNIT 5A
Recursion: Basics

Feedback in Autolab

- When your CAs grade your programming assignments, they should be leaving feedback if you don't get full credit on a problem.
- Click on the score to see the feedback.
- If you still have questions about why an answer wasn't correct, ask your CA.

Recursion

- A "recursive" function is one that calls itself.
- Infinite loop? Not necessarily.



Not like this.

15110 Principles of Computing,
Carnegie Mellon University - CORTINA

- The recursive function calls itself on a *smaller* version of the problem to be solved.
- Recursion looks more like this:

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”.

Example: Factorial

- $N! = N \times (N-1) \times (N-2) \times \cdots \times 1$

- $5! = \quad 5 \times 4 \times 3 \times 2 \times 1$
- $6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1$
- So $6! = 6 \times 5!$
- And $5! = 5 \times 4!$
- And $4! = 4 \times 3!$
- What is the base case? $0! = 1$

Factorial in Ruby (Recursive)

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


Tracing Factorial

```
factorial(5) = 5 * factorial(4)
  factorial(4) = 4 * factorial(3)
    factorial(3) = 3 * factorial(2)
      factorial(2) = 2 * factorial(1)
        factorial(1) = 1 * factorial(0)
          factorial(0) =                  1
```

Tracing Factorial

factorial(5) = 5 * factorial(4)
  factorial(4) = 4 * factorial(3)
    factorial(3) = 3 * factorial(2)
      factorial(2) = 2 * factorial(1)
        factorial(1) = 1 * factorial(0) = 1 * 1 = **1**
          factorial(0) = 1

Tracing Factorial

factorial(5) = 5 * factorial(4)
  factorial(4) = 4 * factorial(3)
    factorial(3) = 3 * factorial(2)
      factorial(2) = 2 * factorial(1) = 2 * 1 = **2**
        factorial(1) = 1 * factorial(0) = 1 * 1 = 1
          factorial(0) = 1

Tracing Factorial

factorial(5) = 5 * factorial(4)
  factorial(4) = 4 * factorial(3)
    factorial(3) = 3 * factorial(2) =   3 * 2 = **6**
      factorial(2) = 2 * factorial(1) =   2 * 1 = 2
        factorial(1) = 1 * factorial(0) = 1 * 1 = 1
          factorial(0) =   1

Tracing Factorial

factorial(5) = 5 * factorial(4)
  factorial(4) = 4 * factorial(3) =   4 * 6 = **24**
    factorial(3) = 3 * factorial(2) =   3 * 2 = 6
      factorial(2) = 2 * factorial(1) =   2 * 1 = 2
        factorial(1) = 1 * factorial(0) = 1 * 1 = 1
          factorial(0) =   1

Tracing Factorial

factorial(5) = 5 * factorial(4) = 5 * 24 = **120**
factorial(4) = 4 * factorial(3) = 4 * 6 = 24
factorial(3) = 3 * factorial(2) = 3 * 2 = 6
factorial(2) = 2 * factorial(1) = 2 * 1 = 2
factorial(1) = 1 * factorial(0) = 1 * 1 = 1
factorial(0) = 1

Recursive vs. Iterative Solutions

- For every recursive function, there is an equivalent iterative solution.
- For every iterative function, there is an equivalent recursive solution.
- But some problems are easier to solve one way than the other way.

Factorial Function (Iterative)

```
def factorial (n)
   result = 1
   for i in 1..n do
      result = result * i
   end
   return result
end
```

15110 Principles of Computing, Carnegie Mellon University - CORTINA

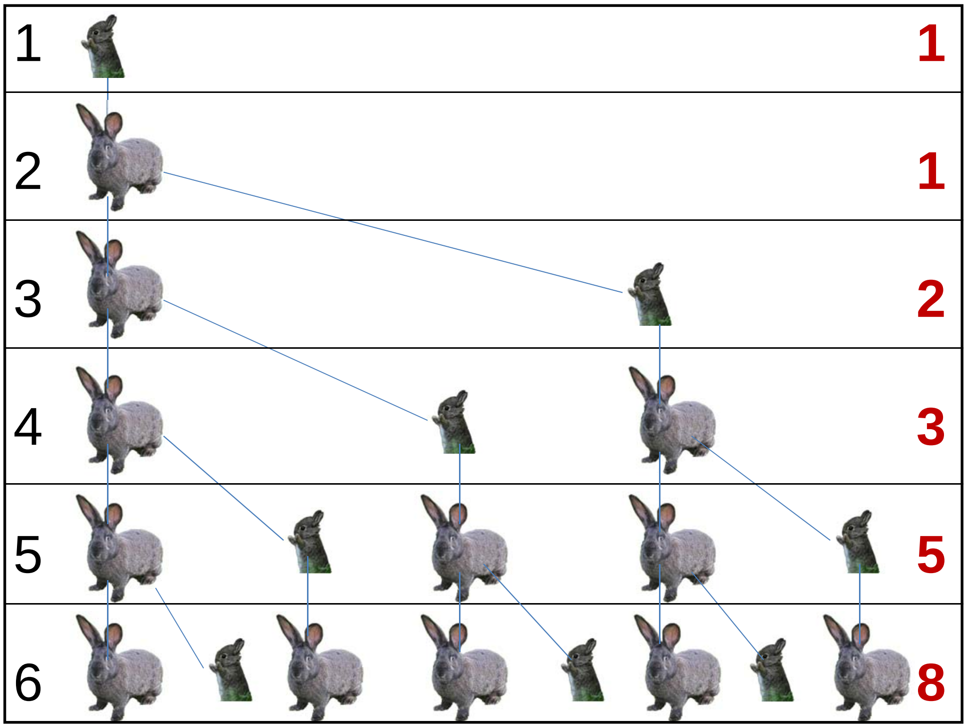
15

Fibonacci Sequence

- Start with 1 pair of baby rabbits.
- Babies take 1 month to reach maturity.
- Mature rabbits produce 1 new pair of babies every month.
- After a year, how many rabbits do you have?

15110 Principles of Computing, Carnegie Mellon University - CORTINA

16



Recursive Fibonacci

- Base case: we start with nothing.
  - fib(0) is 0
- In the first month we have 1 baby rabbit:
  - fib(1) is 1
- At n>1 months, the number of rabbits is:
  # of rabbits from last month fib(n-1)
  +
  # of babies born this month ???

Recursive Fibonacci

- How many babies are born in month n?
  - One baby for every *adult* who was alive at n-1
- How many adults were alive in month n-1?
  - As many as the total number of rabbits at n-2
- Therefore:

fib(n) = fib(n-1) + fib(n-2)

fib(n-1): Adults in month *n*

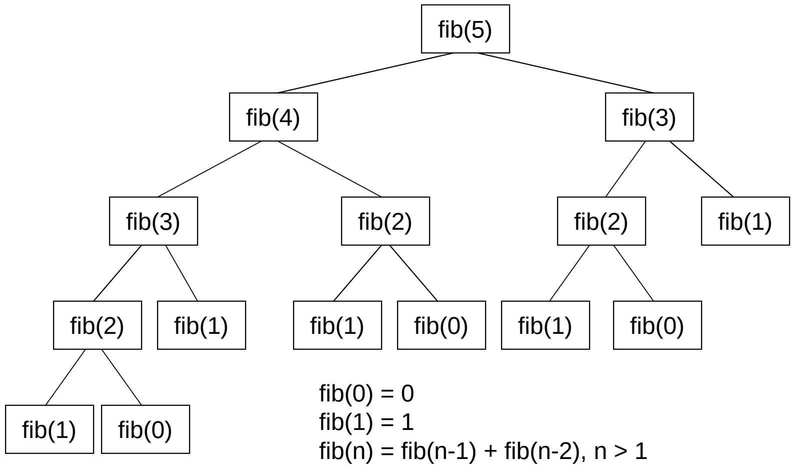
fib(n-2): Babies in month *n*

Recursive Fibonacci in Ruby

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

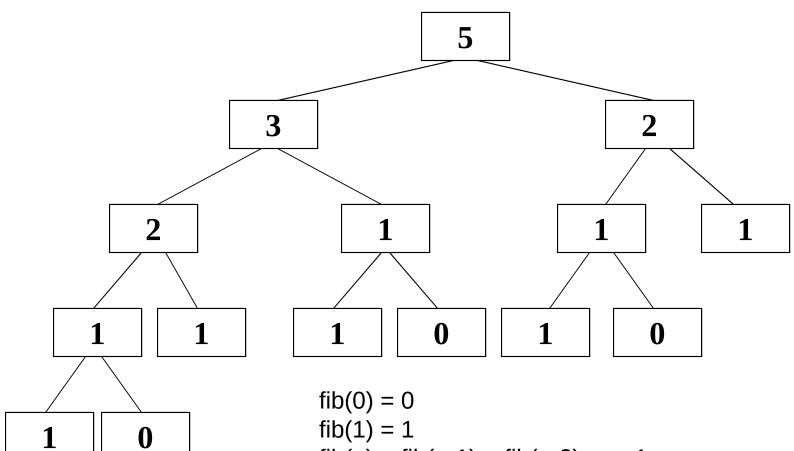
```
>> (0..20).each { |i| p [i, fib(i)] }
```

Recursive Definition



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

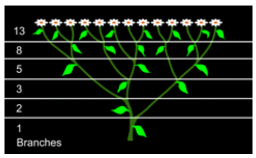
Recursive Definition

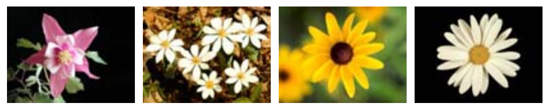


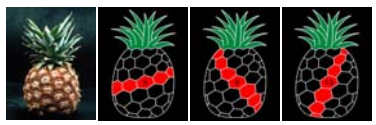
fib(0) = 0
fib(1) = 1
fib(n) = fib(n-1) + fib(n-2), n > 1

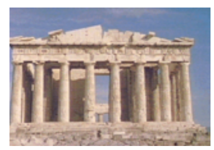
Fibonacci Numbers in Nature

- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, etc.
- Number of branches on a tree.
- Number of petals on a flower.
- Number of spirals on a pineapple.









Iterative Fibonacci

```
def fib(n)
   x = 0
   next_x = 1
   for i in 1..n do
      x, next_x = next_x, x+next_x
   end
   return x
end
```

Much faster than the recursive version. Why?

GCD

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

base case (if y == 0 then / return x)

recursive case (a "simpler" version of the same problem) (return gcd2(y, x % y))

Recursive sum of a list

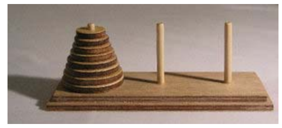
```
def sumlist(list)
    n = list.length
    if n == 0 then
      return 0
    else
      return list[0] + sumlist(list[1..n-1])
    end
end
```

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.

Towers of Hanoi

- A puzzle invented by French mathematician Edouard Lucas in 1883.



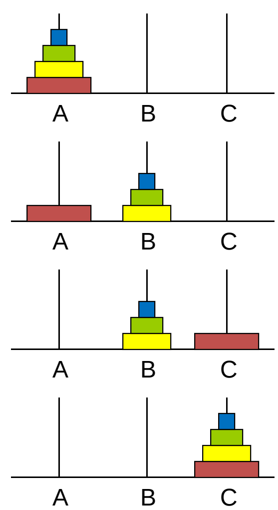
Towers of Hanoi with 8 discs.

- 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

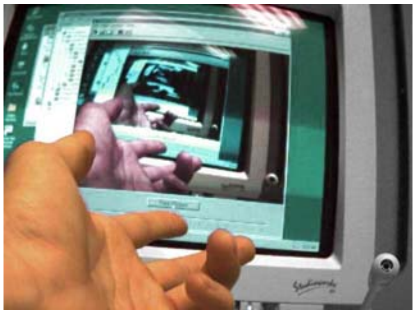
```
def hanoi (disks, from, temp, to)
  n = disks.length
  if n > 1 then
    towers(disks[1..n-1], from, to, temp)
  end
  print "Move ",disks[0]," from ", from,
       " to ", to, "\n"
  if n > 1 then
    hanoi(disks[1..n-1], temp, from, to)
  end
end
```

In irb: `towers([4,3,2,1], "A", "B", "C")`

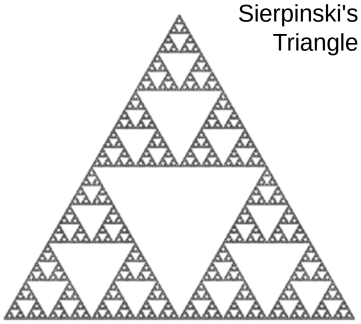
How many moves do the priests need to move 64 discs?

Geometric Recursion (Fractals)

- A recursive operation performed on successively smaller regions.

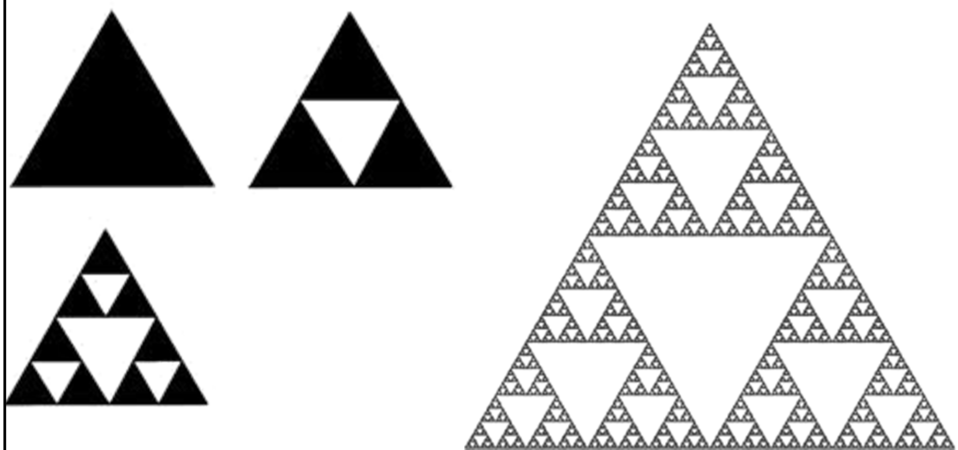


http://fusionanomaly.net/recursion.jpg



Sierpinski's Triangle

Sierpinski's Triangle



Sierpinski's Carpet