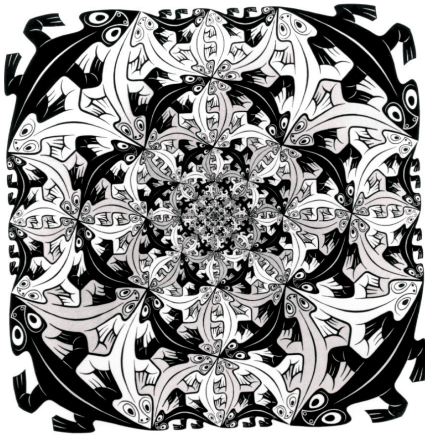
"Recursion" means to define something in terms of itself.

A directory is a collection of files and directories.

Words in dictionaries are defined in terms of other words.



"In order to understand recursion, one must first understand recursion."

– Anonymous

What is 83790 in base 8?

It's easy to find the last digit of a number `n` in base 8: It's simply `n % 8`.

The remaining digits are then the representation of `n // 8`.

But this is an easier version of the same problem!
The Recursion Fairy solves it for us!

```
def print_base_8(n):
  if n >= 8:
    print_base_8(n // 8)
  print(n % 8, end="")
```


We prove that `print_base_8` is correct by induction on k, the number of digits of n in base 8.

Base Case: If $k = 1$, then $n < 8$, and `print_base_8` prints one digit correctly.

Inductive Step: Let $k > 1$, so $n \geq 8$. We make the inductive assumption that `print_base_8` works correctly for numbers with less than k digits. If we call `print_base_8(n)`, then it recursively calls `print_base_8(n//8)`. But `n//8` has $k - 1$ digits in base 8, so this works correctly. Finally, the last digit is printed. It follows that `print_base_8` prints n correctly.

For arbitrary base

```
DIGITS = "0123456789abcdef"
MAX_BASE = len(DIGITS)

# Precondition: n >= 0, 2 <= base <= 16
def print_rec(n, base):
  if n >= base:
    print_rec(n // base, base)
  digit = n % base
  print(digits[digit], end="")
```


The Recursion Fairy

We solve problems by splitting them into simpler subproblems. Then we can hand off each subproblem to a helper.

Recursion happens when the subproblem is exactly the same as the original problem, only smaller (or in some way "easier.")

We can still imagine handing off the subproblem to a helper—the Recursion Fairy. She will "magically" solve the subproblem for us. How she does this is none of our business.

Our task is only to simplify the problem into smaller subproblems, or to solve it directly when simplification is not possible or not necessary.

Factorial

Factorial: $n!$ is $n \times (n-1)!$.

```
# Compute n!
def factorial(n):
  if n <= 1:
    return 1
  else:
    return n * factorial(n-1)
```


Mistakes

Why doesn't this work?

```
def factorial(n):
  return n * factorial(n - 1)
```

And this one?

```
def factorial(n):
  if n <= 1:
    return 1
  else:
    return n * factorial(n)
}
```

Correctness of recursive methods

The Recursion Fairy takes care of the simpler subproblems.

There has to be a base case.

And we need to be sure that we will reach the base case eventually—there has to be some progress in each recursive call.

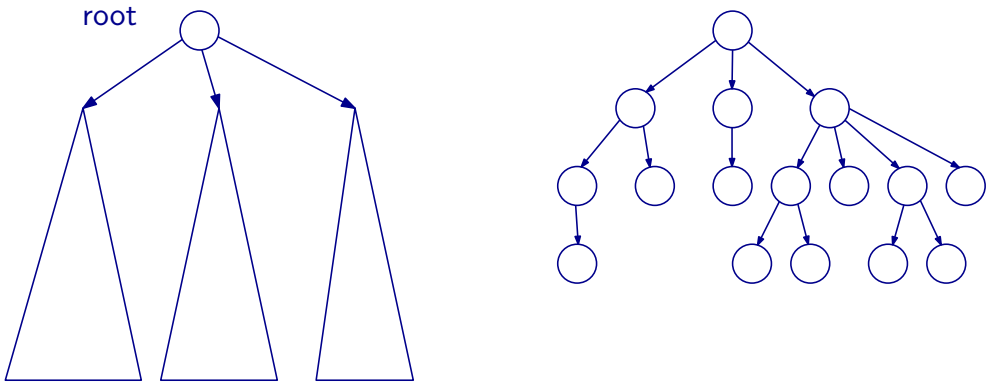
In other words, the version given to the Recursion Fairy has to be easier than the original problem.

Do you know the formal name of the recursion fairy?

She is the induction hypothesis in mathematical induction.

A recursive definition: trees

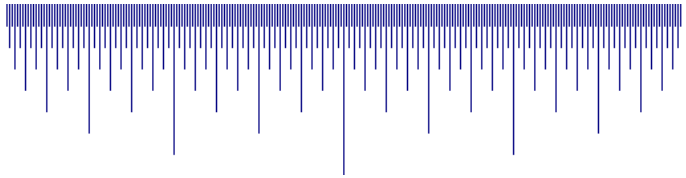
A tree consists of a root and zero or more subtrees, each of whose roots are connected to the root.



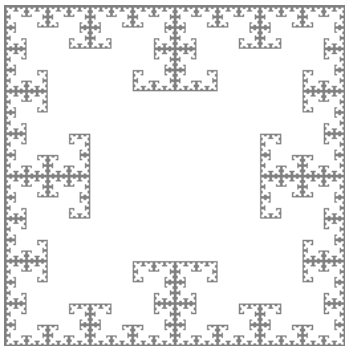
Each edge goes from the parent to the child.

Recursive drawings

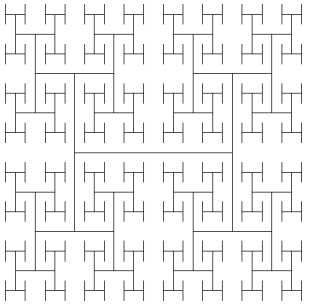
Ruler



Fractal star



H-tree



Too much recursion

The Fibonacci numbers $F_0, F_1, F_2, \ldots$ are defined as follows: $F_0 = 0, F_1 = 1$, and $F_i = F_{i-1} + F_{i-2}$ for $i > 1$.

```
def fib(n):
  if n == 0:
    return 0
  elif n == 1:
    return 1
  else:
    return fib(n - 1) + fib(n - 2)
```

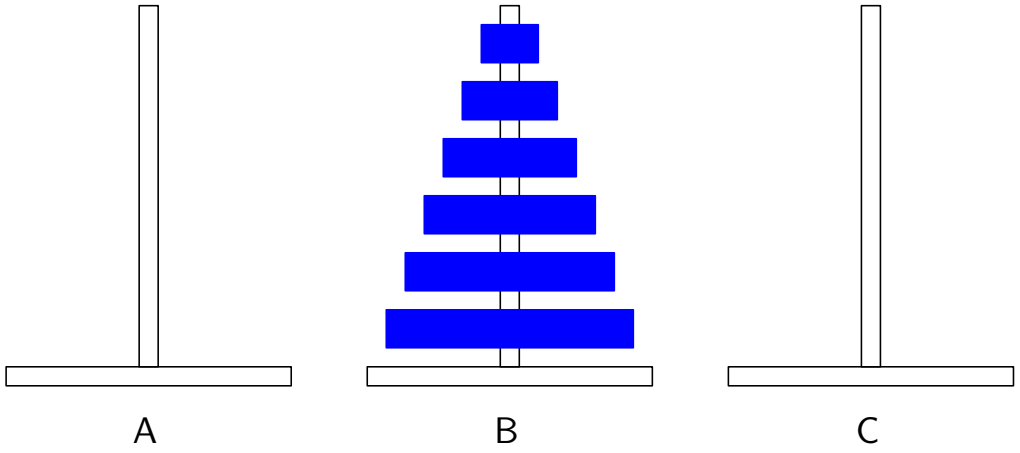
Recursion is not useful when recursive calls duplicate work.
Don't solve the same subproblem in separate recursive calls.

Towers of Hanoi

Three poles, n discs.

One move: take the top disc from one pole and move it to another pole.

Goal: Move all discs from pole A to pole B.



Indirect recursion

For an explanation of indirect recursion, please see the previous slide.

Sine and cosine can be computed using the following identities:

$$\sin x = 2 \sin \frac{x}{2} \cos \frac{x}{2}$$

$$\cos x = 1 - 2(\sin \frac{x}{2})^2$$

Your computer uses *indirectly recursive* methods `sin(x)` and `cos(x)` that compute $\sin x$ and $\cos x$ using these identities. The base case occurs when x is so small that a direct approximation is possible.

Indirect recursion

All the previous examples are examples of direct recursion: A function calls itself.

Indirect recursion happens when there are two (or more) functions `f` and `g`, such that `f` calls `g` and `g` calls `f`.

If you want to understand indirect recursion in more detail, please see the next slide.