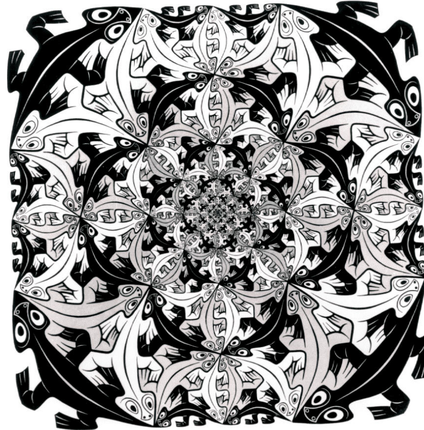
"Recursion" means to define something in terms of itself.

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

How to print a number in any base

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="")
```


Why it works (without recursion fairy)

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="")
```


Correctness of recursive methods

When arguing about the correctness of a recursive method, always assume that the recursive call works.

The Recursion Fairy takes care of that.

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.

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?

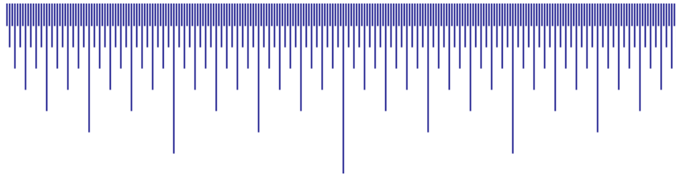
```
def factorial(n):
  return n * factorial(n - 1)
```

And this one?

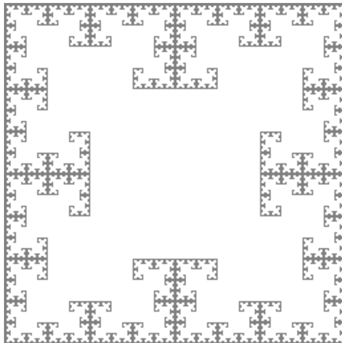
```
def factorial(n):
  if n <= 1:
    return 1
  else:
    return n * factorial(n)
}
```

Recursive drawings

Ruler

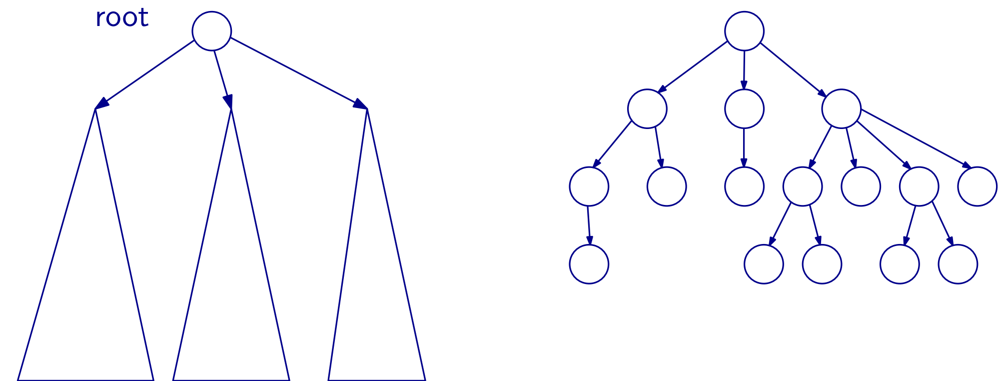


Fractal star



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.

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)
```

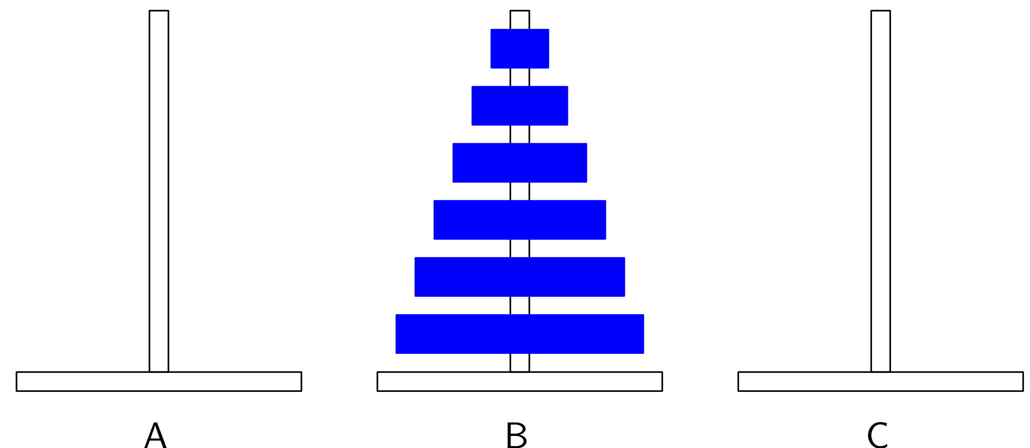
Recursion is not useful when recursive calls duplicate work. Don't solve the same subproblem in separate recursive calls.

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.



When uncovering a cell with no neighboring bombs, we can immediatly open all neighbors.

A good implementation does this automatically:

```
What cell do you want to check? F14
                   1 1 1 1 1 1 1 1 1 1 2
  1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
A . . . . . . . . . . . . . . 1   1 . 1
B . . . . . . . . . . . . . . 2   2 . 2
C . . . . . . . . . . . . . . 2   1 . 1
D . . . . . 1 . . . . . 1 1 1 1   1 1 1
E . . . . . . . . . . . 1
F . . . . . . . . . 1 . 2 1           1 1 1
G . . . . . . . . . . . . 1 1 2 2 1 1 . .
H . . . . . . . . . . . . . . . . . . . .
```