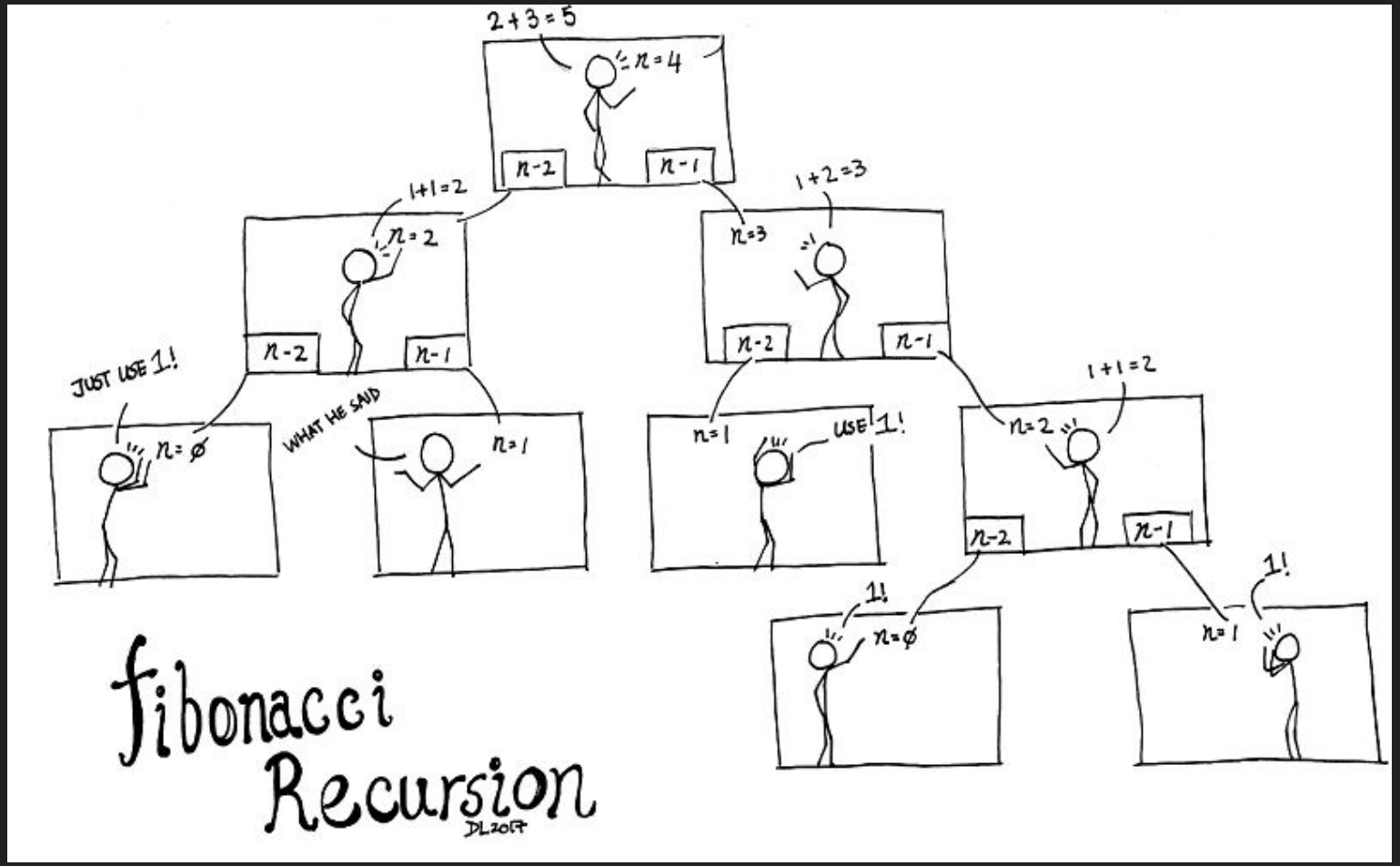
Recursion:

"To loop is human,
to recurse is divine".



Recursion

A recursive function is a function that calls itself.

Iteration vs. Recursion

In simple terms:

An *iterative* function is one that **loops** to repeat some part of the code.
A *recursive* function is one that **calls itself** again to repeat the code.

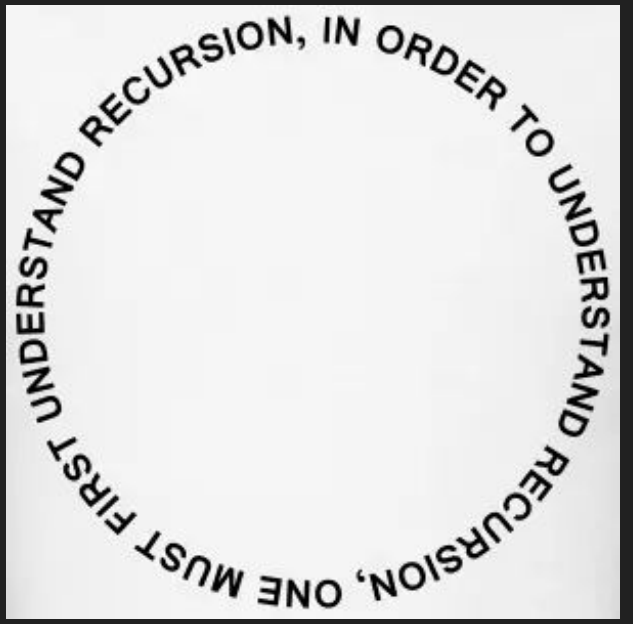
recursion is

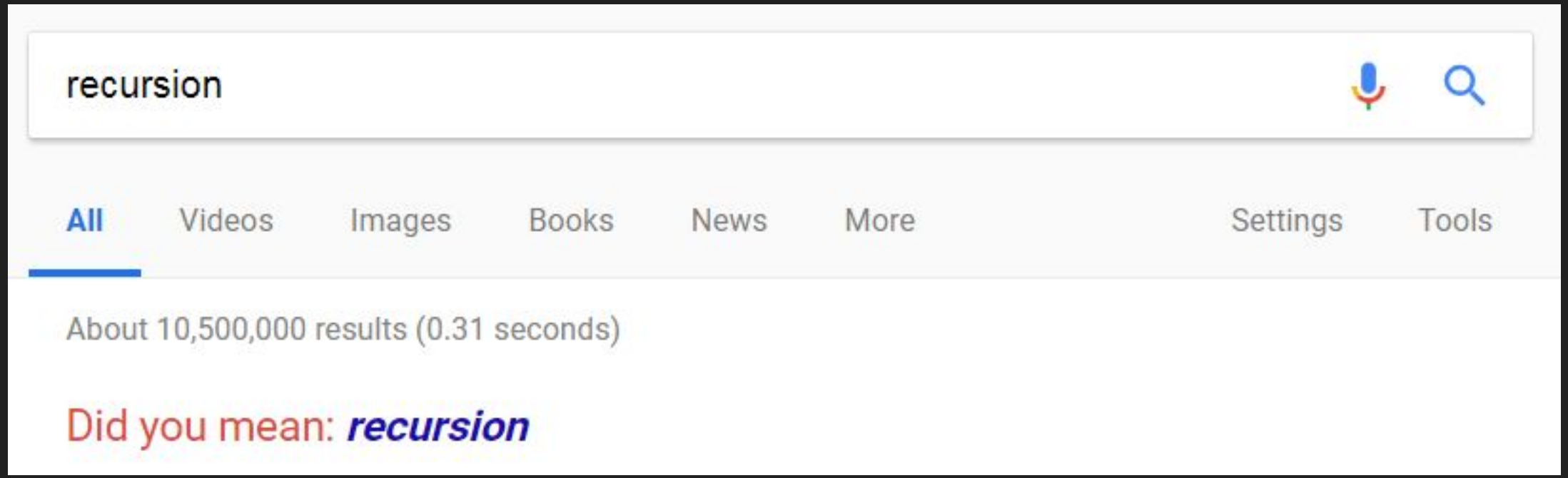
recursion is **memory-intensive because**
recursion is **hard**
recursion is **to the base case as iteration is to what**
recursion is **confusing**
recursion is **bad**
recursion is **magic**
recursion is **slow**
recursion is **a computational technique in which**
recursion is **another name for iteration**
recursion is **similar to which of the following**

Google Search I'm Feeling Lucky

Report inappropriate predictions

To understand recursion,
you must first understand recursion.




recursion
All
Videos
Images
Books
News
More
Settings
Tools
About 10,500,000 results (0.31 seconds)
Did you mean: recursion

Toothpaste For Dinner.com


I'M SO META, EVEN THIS ACRONYM

I.S.M.E.T.A.

Recursion: A working definition

- Recursion solves a big problem by the solving increasingly smaller examples of the same problem and building up a solution.

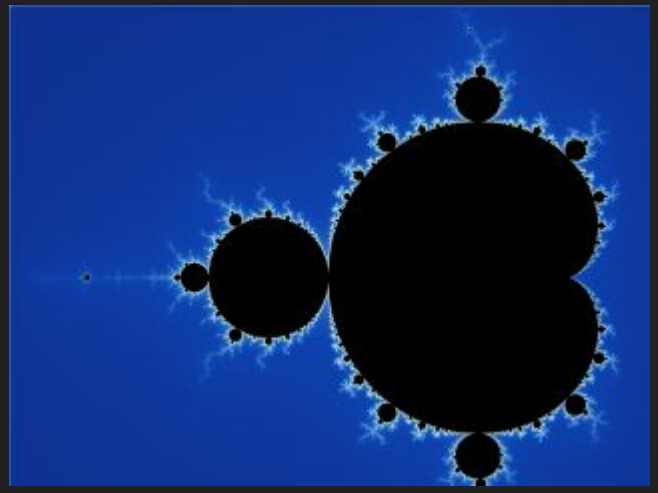
Recursion is All Around Us

- Nature: Cauliflower, ferns, rivers
- Math & Art: Fractals
- Computer Science…

Recursion in Nature: Cauliflower

Recursion in Art: Fractals

Koch Snowflake:
https://upload.wikimedia.org/wikipedia/commons/6/65/Kochsim.gif



Mandelbrot set:
https://en.wikipedia.org/wiki/File:Mandelbrot_sequence_new.gif

Recursion

- A recursive solution must have **base** case
  - A simplest problem where recursion is **not** needed
- A recursive function also has a **recursive** case, which calls same function with reduced version of the problem

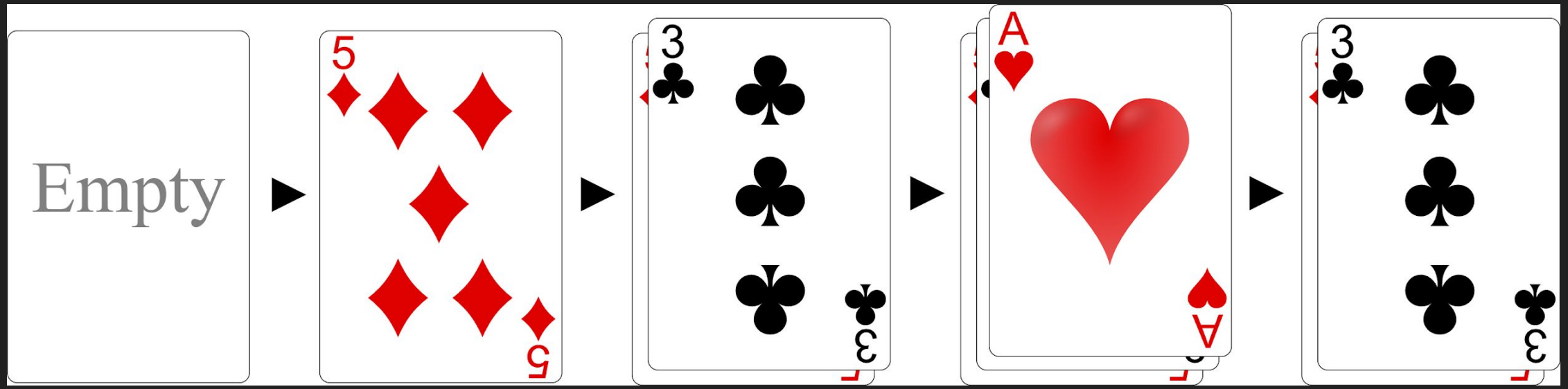
The Three Laws of Recursion

1. A recursive algorithm must have a base case.
2. A recursive algorithm moves toward the base case (recursive case).
3. A recursive algorithm must call itself -> the very definition of recursion.

A simple recursive function

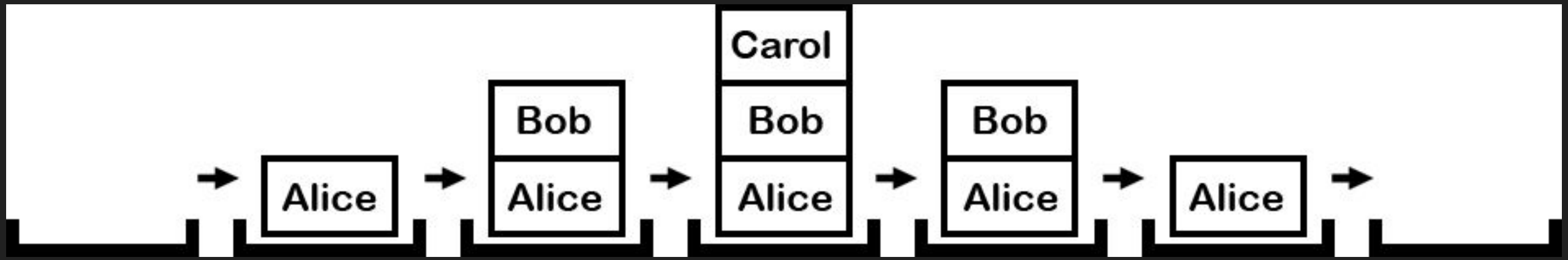
```
def message (times):
   if times==0:
      print('this is the base case ', times)
   else:
      message(times-1)
      print('this is the recursive case ', times)

message(5)
```

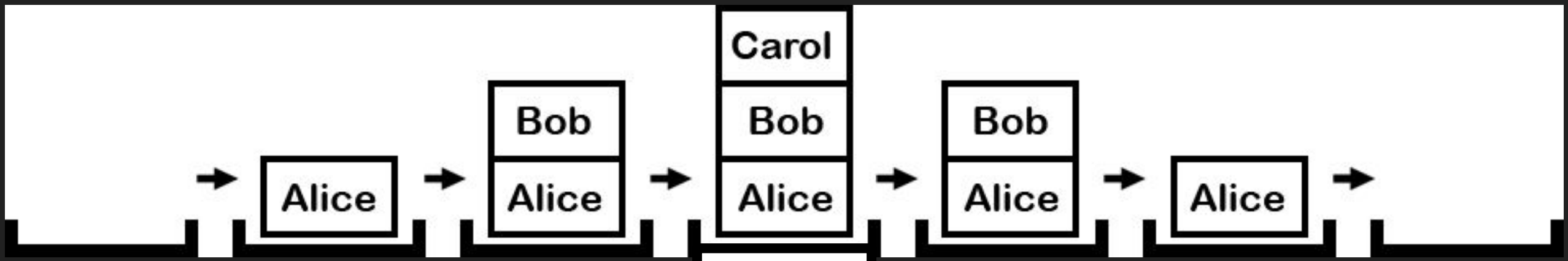
A stack is a data structure that holds a sequence of data and only lets you interact with the topmost item.

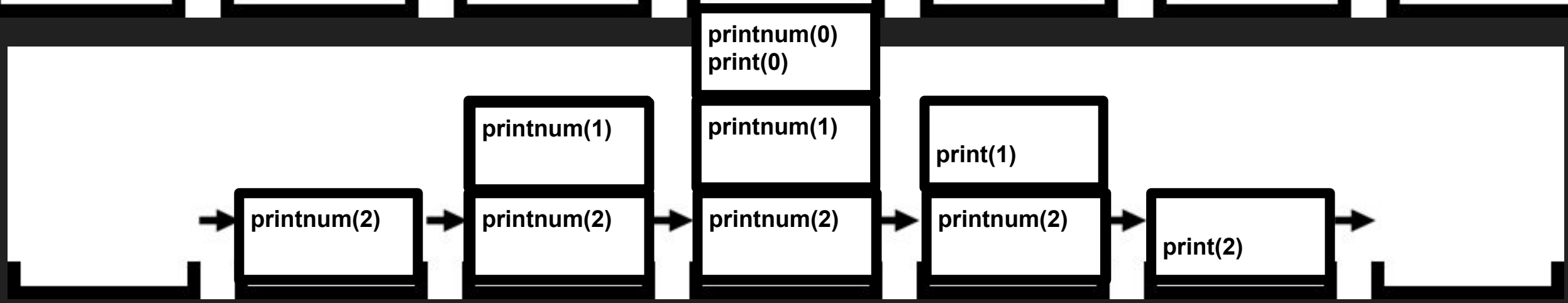


First-In, Last-Out (FILO)

A stack is a data structure that holds a sequence of data and only lets you interact with the topmost item.








```
def printnum(n):
    if n == 0:        # Base case
        print(n)
    else:             # Recursive case
        printnum(n-1)
        print(n)
```

Recursion is a lot easier to understand if you understand stacks and the call stack.

To understand recursion, you must first understand ~~recursion~~.
STACKS (just a bit).

The secret is that the base case starts the first ‘pop’ off the stack

A simple recursive function

```
def message (times):
   if times==0:
      print('this is the base case ', times)
   else:
      message(times-1)
      print('this is the recursive case ', times)

message(5)
```

functions calls in code

```
message(5)
        message(4)
                message(3)
                        message(2)
                                message(1)
                                        message(0)
                                        print(‘this is the base case’, 0 )
                                print(‘this is the recursive case’, 1)
                        print(‘this is the recursive case’, 2)
                print(‘this is the recursive case’, 3)
        print(‘this is the recursive case’, 4)
print(‘this is the recursive case’, 5)
```

A simple recursive function

```
def message (times):
   if times==0:
      print('this is the base case ', times)
   else:
      message(times-1)
      print('this is the recursive case ', times)

message(5)
```

You can try reversing the lines in the recursive case to see what happens.

A simple recursive function

```
def printnum(n):
    if n == 0:    # Base case
        print(n)
    else:         # Recursive case
        printnum(n-1)
        print(n)

def main():
    printnum(4)

main()
```

fucntion calls in code

```
printnum(5)
        printnum(4)
                printnum(3)
                        printnum(2)
                                printnum(1)
                                        printnum(0)
                                        print(0)
                                print(1)
                        print(2)
                print(3)
        print(4)
print(5)
```

function calls in code

```
printnum(5)
        printnum(4)
                printnum(3)
                        printnum(2)
                                printnum(1)
                                        printnum(0)
                                        print(0)
                                print(1)
                        print(2)
                print(3)
        print(4)
```

The Stack of Calls

```
def printnum(n):
    if n == 0:
        print(n)
    else:
        printnum(n-1)
        print(n)

printnum(5)
```

How do you learn recursion?

- Learn the concept.
- Practice the concept with **lots** of coding practice.
- Honestly just keep doing it. You will get/"grok" it at some point.

sum(n) - adds numbers from 1 to n

Let’s try a simple function sum(n) that adds up all numbers from 1 to n.

First create an iterative function sum(n) that uses a *for loop* and returns the sum of all numbers 1 to n.

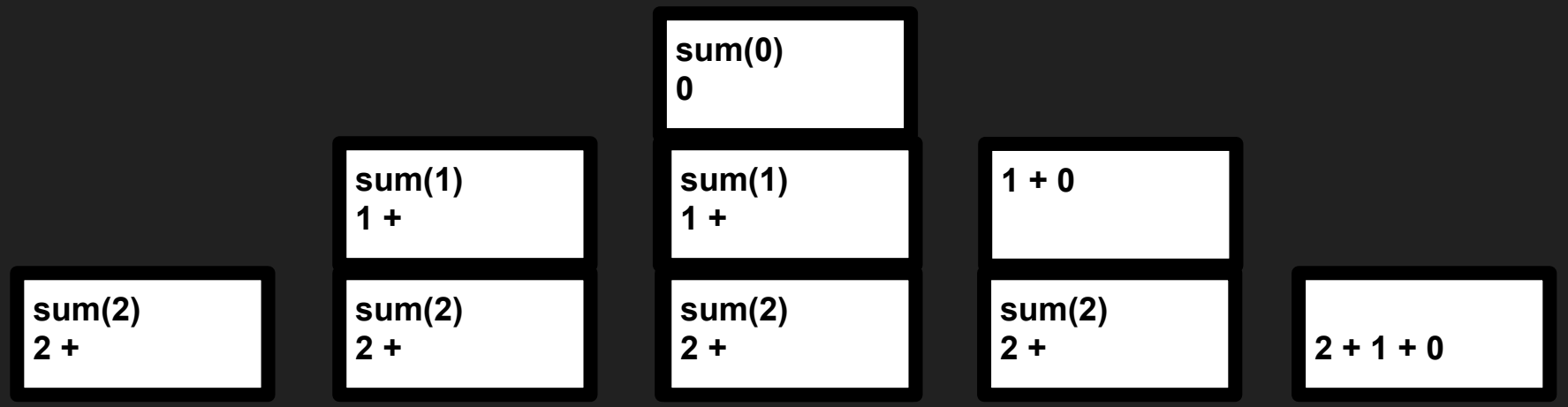
Iterative version of sum(n)

```
def sum_for(n):
   total =0
   for x in range(n+1):
      total+=x
   return total

print(sum_for(5))
```

Recursive sum of numbers from 1 to n

```
def sum(n):
   if n==0:
      return 0
   else:
      return n + sum (n-1)

print(sum(5))
```

```
def sum(n):
    if n==0:
        return 0
    else:
        return n + sum (n-1)

print(sum(2))
```



sum(5)

sum(4)

sum(3)

sum(2)

sum(1)

sum(0)

0

1 + 0

2 + 1 + 0

3 + 2 + 1 + 0

4 + 3 + 2 + 1 + 0

5 + 4 + 3 + 2 + 1 + 0

```
def sum(n):
    if n==0:
        return 0
    else:
        return n + sum (n-1)

print(sum(5))
```

Example: sum of numbers from 1 to n

- What is the base case?
  - When you get to n=0 you return 0 which stops the recursing of the function
- How do we move towards the base case?
  - You start at the highest number and subtract one each time you recurse

Another Example: Factorial

- n! = n x (n-1) x (n-2) x (n-3) … x 1
- 1! = 1 # Base Case
- 2! = 2 x 1! = 2 x 1 = 2
- 3! = 3 x 2! = 3 x (2 x 1!) = 6
- 4! = 4 x 3! = 4 x (3 x 2!) = 24 # Move towards base case
- n! = n x (n -1)!

The Stack of Calls

factorial stack calls

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

Note: 0! =1
You will need to account for this in your code

In-Class Exercise: Factorial Functions

- Code it!
  - with ***while*** loop
  - with ***for*** loop
  - with ***recursion***