



Recursão



# Objetivos



Definição de recursão



Caso base e caso recursivo



Recursão x Laços



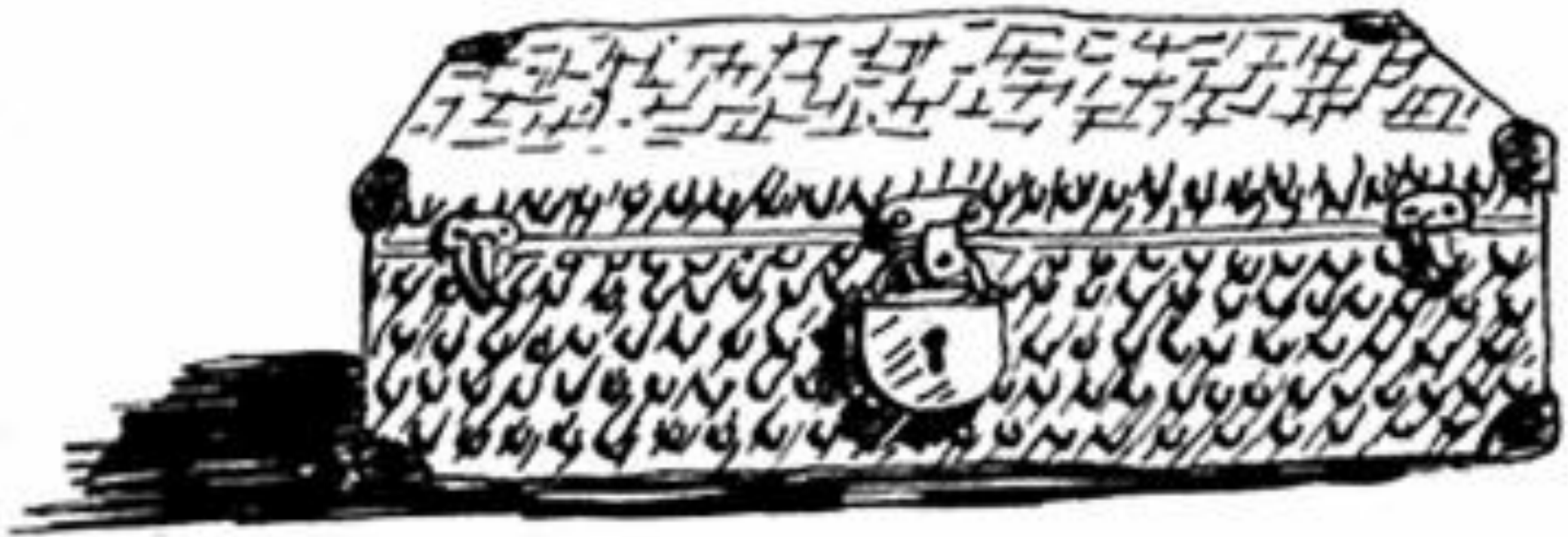
Pilha de chamadas com recursão

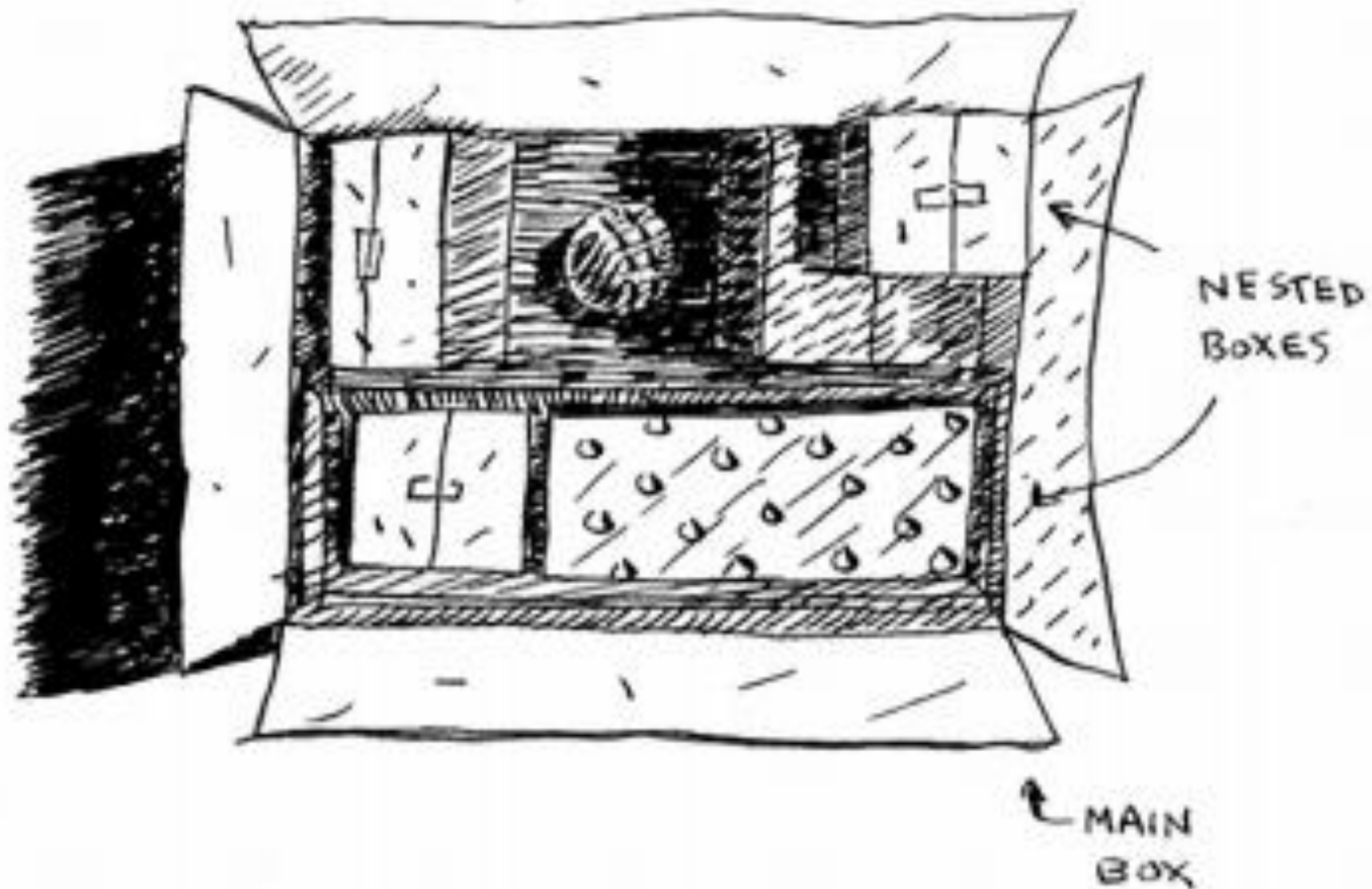


Recursão de cauda

# Recursão

Problema: você precisa encontrar uma chave em uma mala.

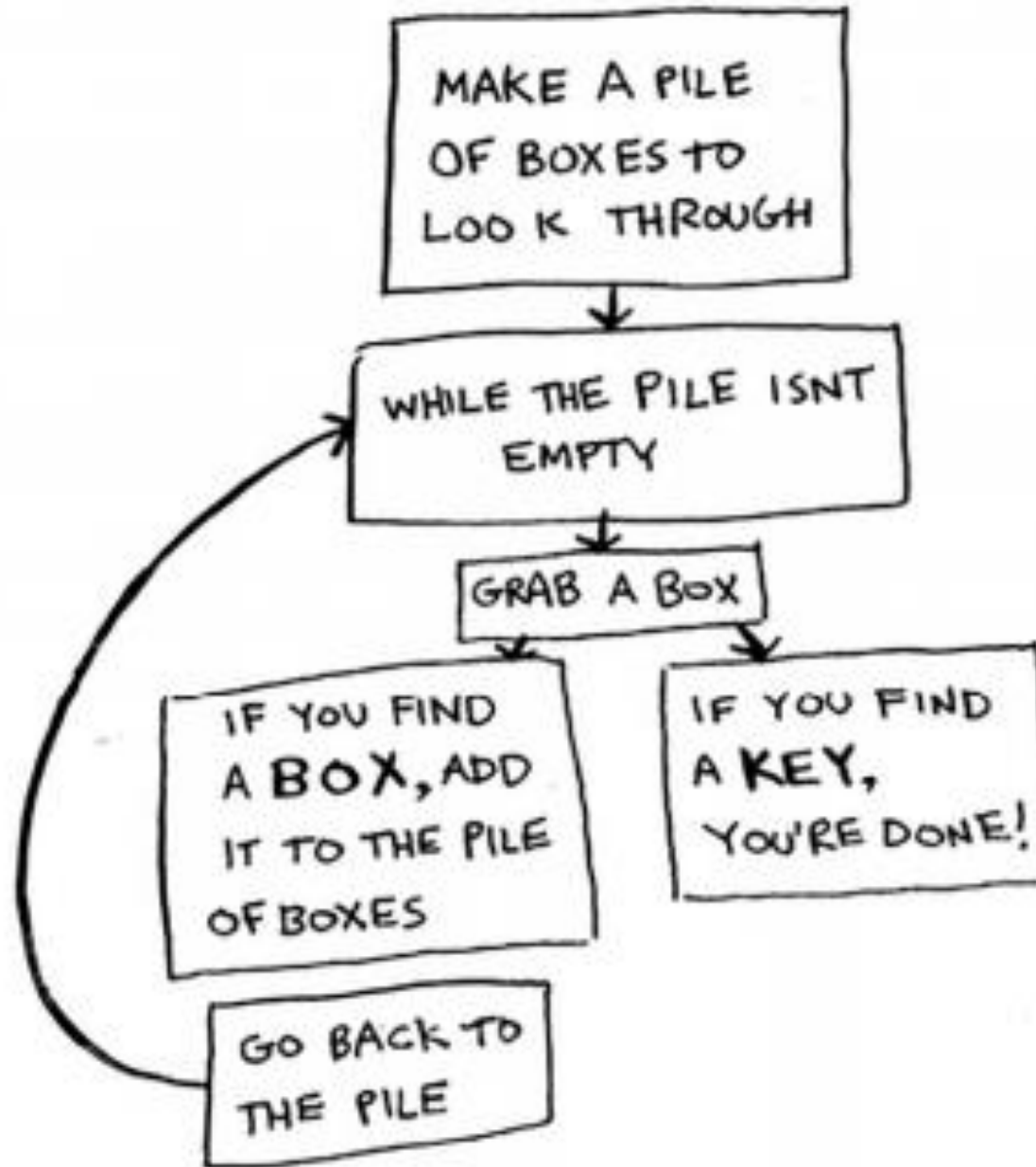




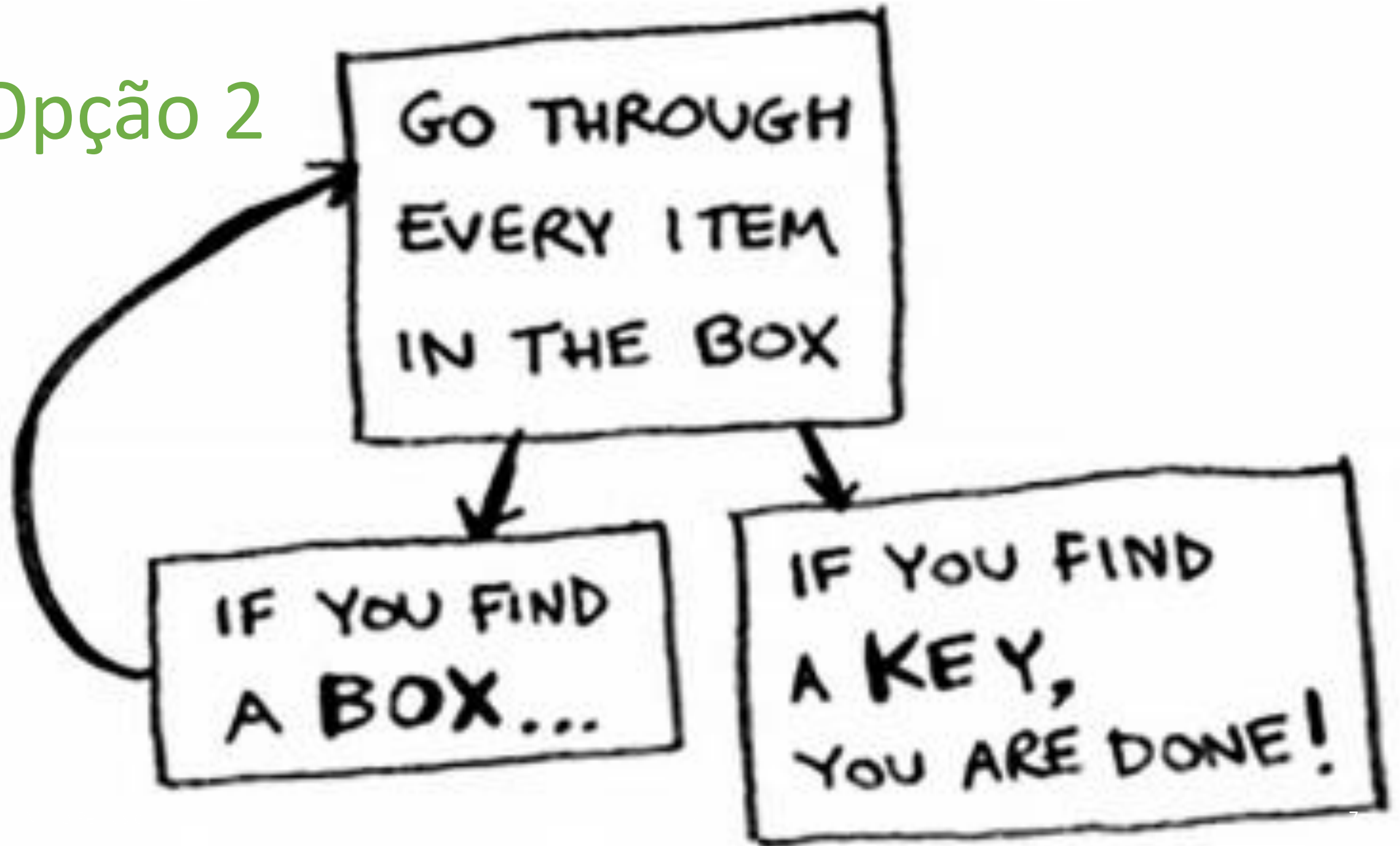
A caixa contém mais caixas com caixas dentro. A chave está em alguma destas caixas.

Qual é o seu algoritmo para procurá-la?

# Opção 1



## Opção 2



Qual é a mais simples?



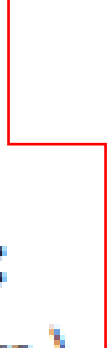
## Solução 1 (laço)

```
def look_for_key(main_box):  
    pile = main_box.make_a_pile_to_look_through()  
    while pile is not empty:  
        box = pile.grab_a_box()  
        for item in box:  
            if item.is_a_box():  
                pile.append(item)  
            elif item.is_a_key():  
                print "found the key!"
```

Solução 2  
(recursão)

```
def look_for_key(box):  
    for item in box:  
        if item.is_a_box():  
            look_for_key(item)  
        elif item.is_a_key():  
            print "found the key!"
```

Recursão



# Recursão x Iteração

“Os **laços** podem melhorar o desempenho do seu **programa**. A **recursão** melhora o desempenho do seu **programador**. Escolha o que for mais importante para a sua situação.” (Leigh Caldwell) [1]

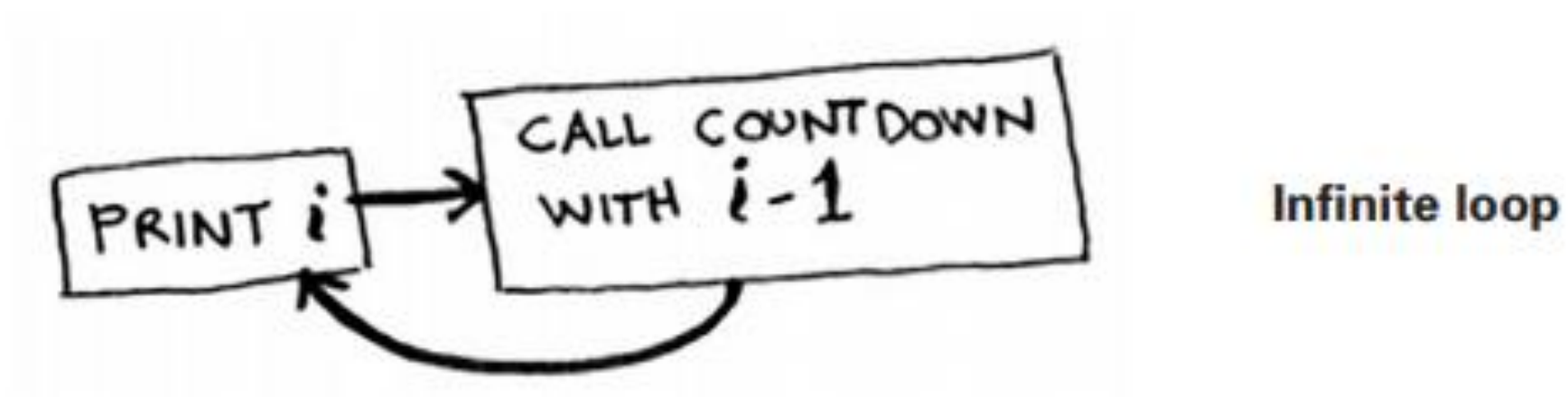
# Problema

Imprima uma contagem regressiva (3...2...1...)

```
def countdown(i):  
    print i  
    countdown(i-1)
```

Qual o problema com essa solução?

# Qual o problema com essa solução?



3...2...1...0...-1...-2...

# Caso-base e caso recursivo

Uma função recursiva chama a si mesma.



Erro comum

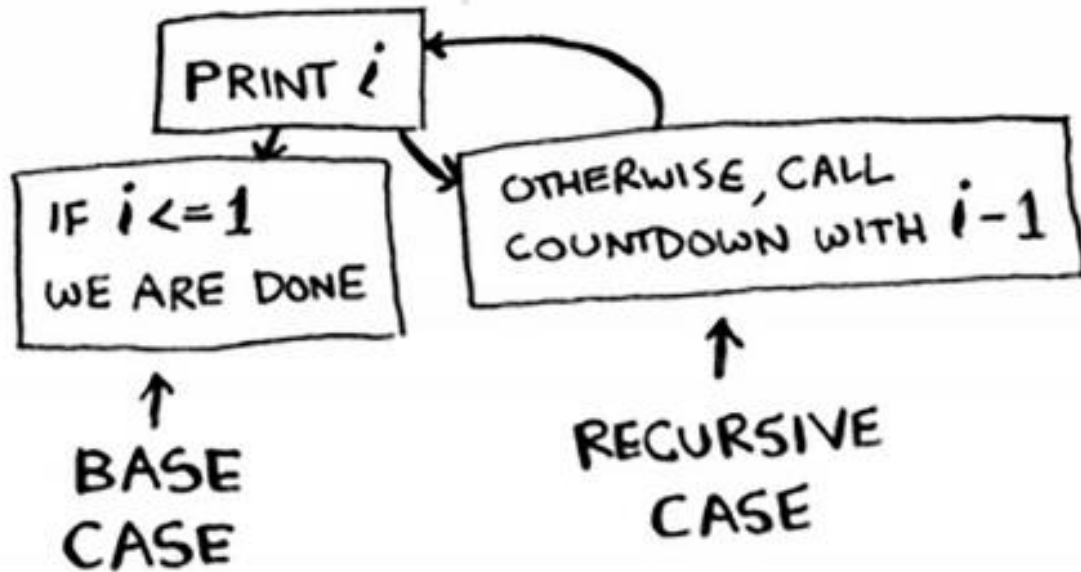
Acabar em um loop infinito!



Solução

Informar quando a recursão deve parar!

# Caso base e caso recursivo

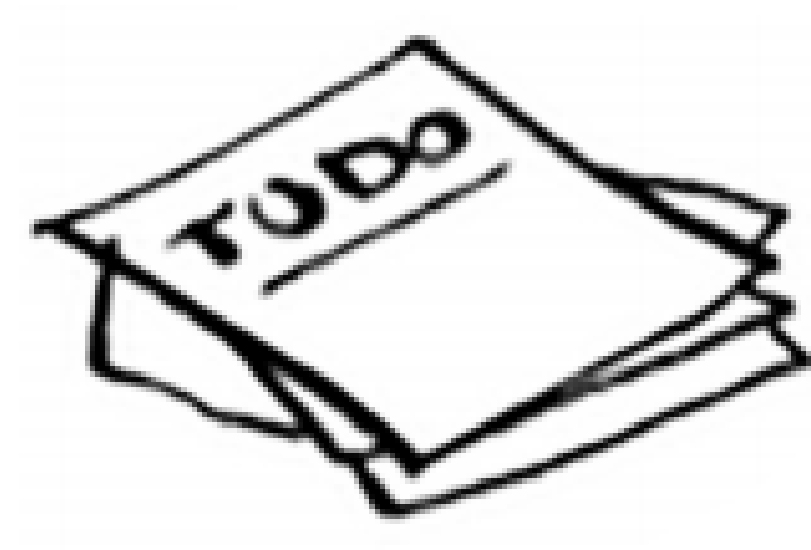


```
def countdown(i):  
    print i  
    if i <= 0: ← ..... Base case  
        return  
    else: ← ..... Recursive case  
        countdown(i-1)
```

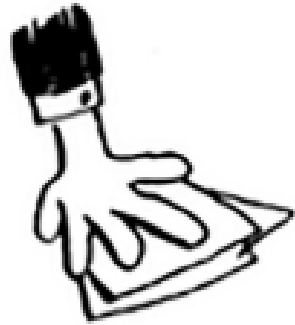


# A pilha de chamada (*call stack*)

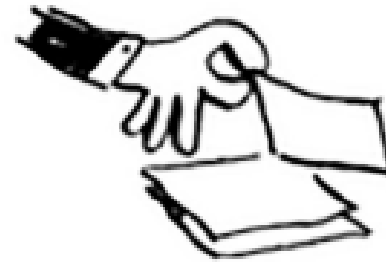
# A pilha



# A pilha (ações)



**PUSH**  
(ADD A NEW ITEM  
TO THE TOP)



**POP**  
(REMOVE THE TOPMOST  
ITEM AND READ IT)

# A pilha (exemplos)



POP A TODO  
OFF THE STACK



IT SAYS "GET FOOD".  
YOU NEED TO GET  
BUNS, BURGERS AND  
BAKE A CAKE.



LET'S PUSH THESE  
TODOS ONTO THE STACK

A pilha de  
chamada  
(*call stack*)

```
def greet(name):  
    print "hello, " + name + "!"  
    greet2(name)  
    print "getting ready to say bye..."  
    bye()
```

- Considere o exemplo acima.
  - greet () chama greet2() e bye()

A pilha de  
chamada  
(*call stack*)

```
def greet2(name):  
    print "how are you, " + name + "?"  
  
def bye():  
    print "ok bye!"
```

# A pilha de chamada (*call stack*)

```
def greet(name):  
    print "hello, " + name + "!"  
    greet2(name)  
    print "getting ready to say bye..."  
    bye()
```

```
def greet2(name):  
    print "how are you, " + name + "?"
```

```
def bye():  
    print "ok bye!"
```

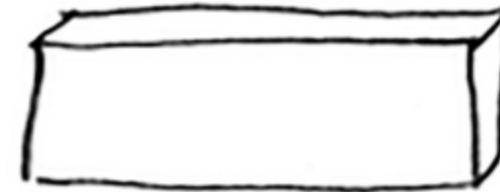
# A pilha de chamada (*call stack*)

Uma chamada de função `greet("maggie")` é realizada

```
def greet(name):  
    print "hello, " + name + "!"  
    greet2(name)  
    print "getting ready to say bye..."  
    bye()
```

```
def greet2(name):  
    print "how are you, " + name + "?"
```

```
def bye():  
    print "ok bye!"
```





# A pilha de chamada (*call stack*)

```
def greet(name):  
    print "hello, " + name + "!"  
    greet2(name)  
    print "getting ready to say bye..."  
    bye()
```

```
def greet2(name):  
    print "how are you, " + name + "?"
```

```
def bye():  
    print "ok bye!"
```



# A pilha de chamada (*call stack*)

```
def greet(name):  
    print "hello, " + name + "!"  
    greet2(name)  
    print "getting ready to say bye..."  
    bye()
```

```
def greet2(name):  
    print "how are you, " + name + "?"
```

```
def bye():  
    print "ok bye!"
```



# A pilha de chamada (*call stack*)

```
def greet(name):  
    print "hello, " + name + "!"  
    greet2(name)  
    print "getting ready to say bye..."  
    bye()
```

```
def greet2(name):  
    print "how are you, " + name + "?"
```

```
def bye():  
    print "ok bye!"
```

CURRENT  
FUNCTION  
CALL



# A pilha de chamada (*call stack*)

```
def greet(name):  
    print "hello, " + name + "!"  
    greet2(name)  
    print "getting ready to say bye..."  
    bye()
```

- def greet2(name):  
 print "how are you, " + name + "?"

```
def bye():  
    print "ok bye!"
```

CURRENT  
FUNCTION  
CALL



# A pilha de chamada (*call stack*)

```
def greet(name):  
    print "hello, " + name + "!"  
    greet2(name)  
    print "getting ready to say bye..."  
    bye()
```

```
def greet2(name):  
    print "how are you, " + name + "?"
```

```
def bye():  
    print "ok bye!"
```

CURRENT  
FUNCTION  
CALL



# A pilha de chamada (*call stack*)

```
def greet(name):  
    print "hello, " + name + "!"  
    greet2(name)  
    print "getting ready to say bye..."  
    bye()
```

```
def greet2(name):  
    print "how are you, " + name + "?"
```

```
def bye():  
    print "ok bye!"
```

Quando greet2() foi chamada, greet() estava *parcialmente completa*



# A pilha de chamada (*call stack*)

```
def greet(name):  
    print "hello, " + name + "!"  
    greet2(name)  
    print "getting ready to say bye..."  
    bye()
```

```
def greet2(name):  
    print "how are you, " + name + "?"
```

```
def bye():  
    print "ok bye!"
```



# A pilha de chamada (*call stack*)

```
def greet(name):  
    print "hello, " + name + "!"  
    greet2(name)  
    print "getting ready to say bye..."  
    bye()
```

```
def greet2(name):  
    print "how are you, " + name + "?"
```

```
● def bye():  
    print "ok bye!"
```





# A pilha de chamada (*call stack*)

```
def greet(name):  
    print "hello, " + name + "!"  
    greet2(name)  
    print "getting ready to say bye..."  
    bye()
```

```
def greet2(name):  
    print "how are you, " + name + "?"
```

```
def bye():  
    print "ok bye!"
```



# A pilha de chamada (*call stack*)

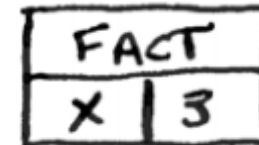
```
def greet(name):  
    print "hello, " + name + "!"  
    greet2(name)  
    print "getting ready to say bye..."  
    bye()  
  
def greet2(name):  
    print "how are you, " + name + "?"  
  
def bye():  
    print "ok bye!"
```



# A pilha de chamada com recursão

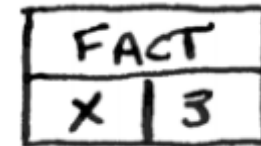
Uma chamada de função `fact(3)` é realizada

- ```
def fact(x):  
    if x == 1:  
        return 1  
    else:  
        return x * fact(x-1)
```



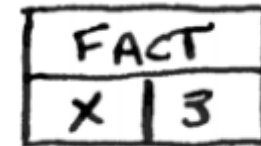
# A pilha de chamada com recursão

```
def fact(x):  
    if x == 1:  
        return 1  
    else:  
        return x * fact(x-1)
```



# A pilha de chamada com recursão

```
def fact(x):  
    if x == 1:  
        return 1  
    else:  
        return x * fact(x-1)
```



# A pilha de chamada com recursão

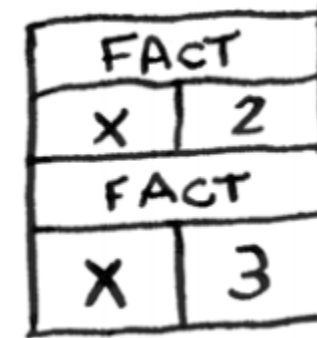
```
def fact(x):  
    if x == 1:  
        return 1  
    else:  
        return x * fact(x-1)
```

Recursão

| FACT |   |
|------|---|
| x    | 2 |
| FACT |   |
| x    | 3 |

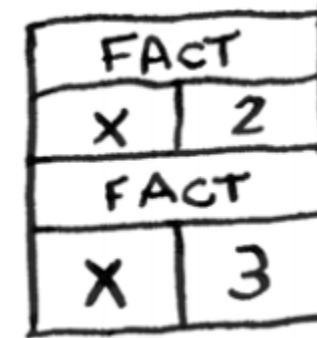
# A pilha de chamada com recursão

- ```
def fact(x):  
    if x == 1:  
        return 1  
    else:  
        return x * fact(x-1)
```



# A pilha de chamada com recursão

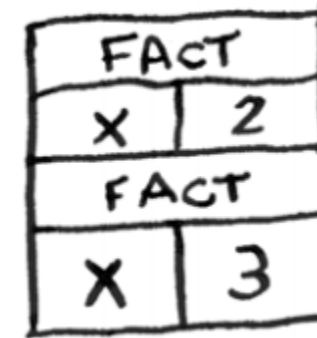
```
def fact(x):  
    if x == 1:  
        return 1  
    else:  
        return x * fact(x-1)
```





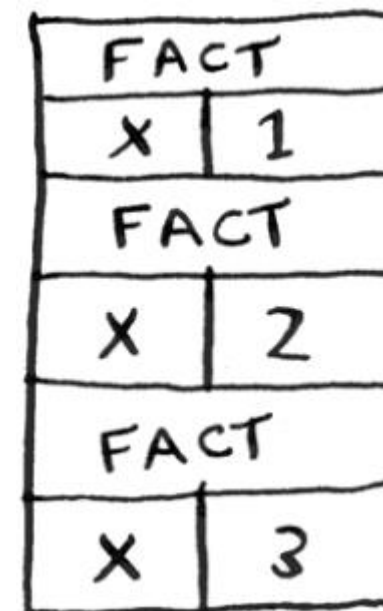
# A pilha de chamada com recursão

```
def fact(x):  
    if x == 1:  
        return 1  
    else:  
        return x * fact(x-1)
```



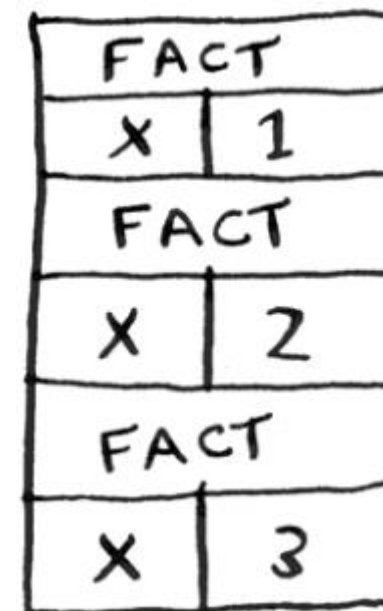
# A pilha de chamada com recursão

```
def fact(x):  
    if x == 1:  
        return 1  
    else:  
        return x * fact(x-1)
```



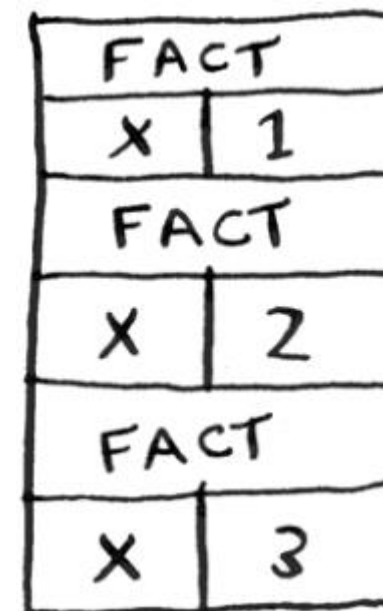
# A pilha de chamada com recursão

- ```
def fact(x):  
    if x == 1:  
        return 1  
    else:  
        return x * fact(x-1)
```



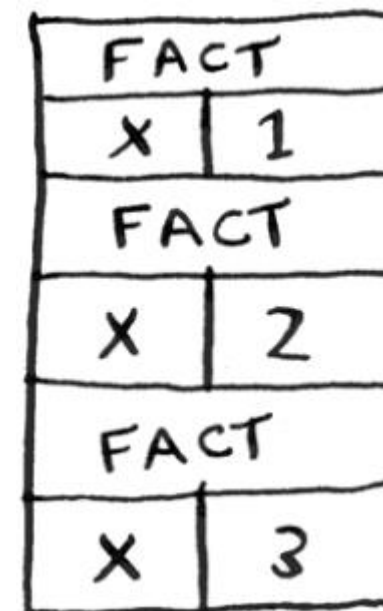
# A pilha de chamada com recursão

```
def fact(x):  
    if x == 1:  
        return 1  
    else:  
        return x * fact(x-1)
```



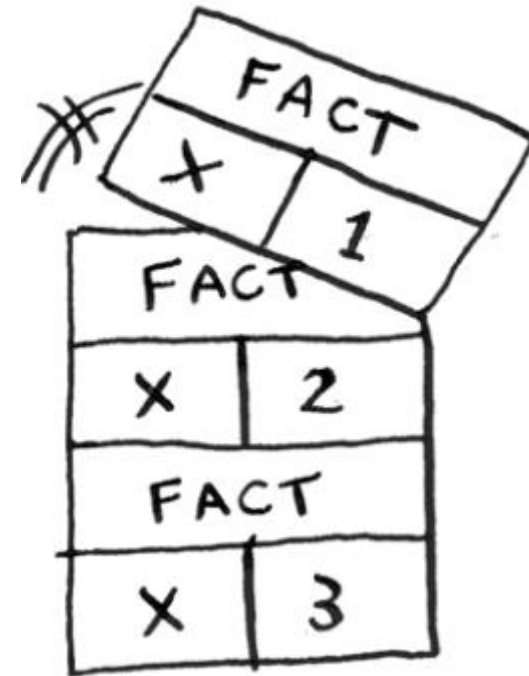
# A pilha de chamada com recursão

```
def fact(x):  
    if x == 1:  
        return 1  
    else:  
        return x * fact(x-1)
```



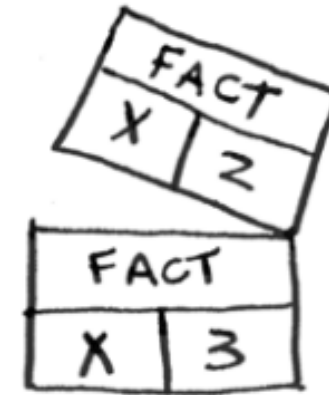
# A pilha de chamada com recursão

```
def fact(x):  
    if x == 1:  
        return 1  
    else:  
        return x * fact(x-1)
```



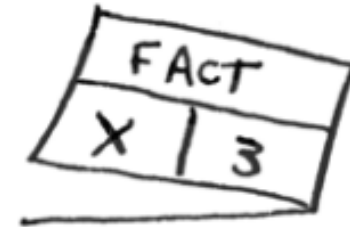
# A pilha de chamada com recursão

```
def fact(x):  
    if x == 1:  
        return 1  
    else:  
        return x * fact(x-1)
```



# A pilha de chamada com recursão

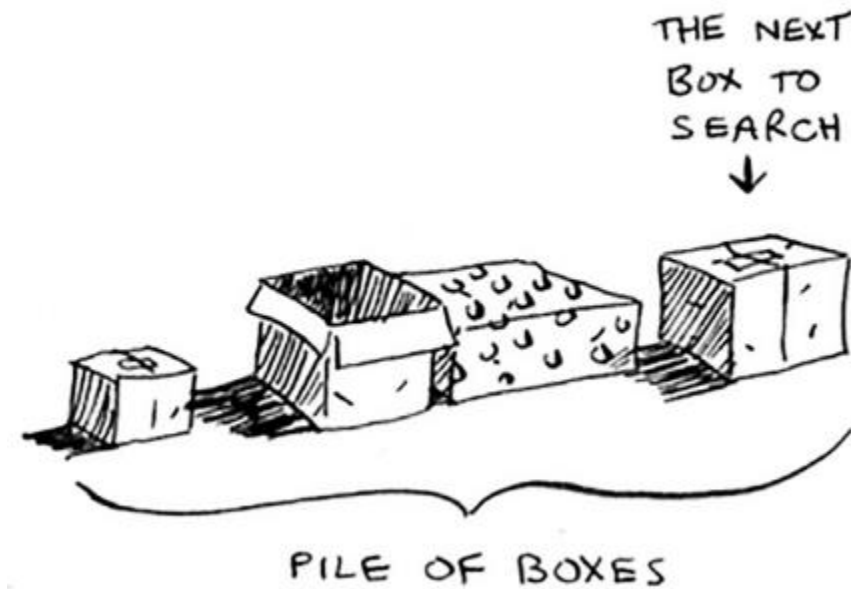
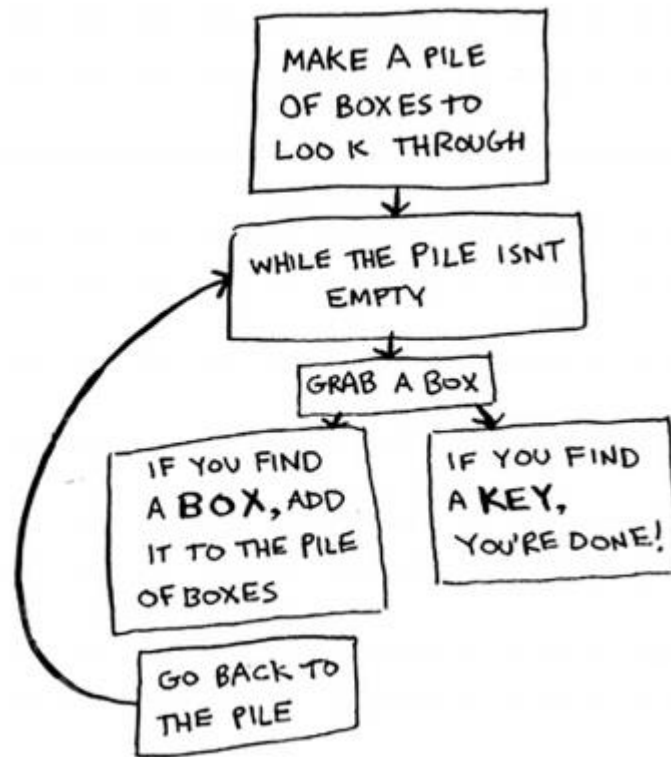
```
def fact(x):  
    if x == 1:  
        return 1  
    else:  
        return x * fact(x-1)
```





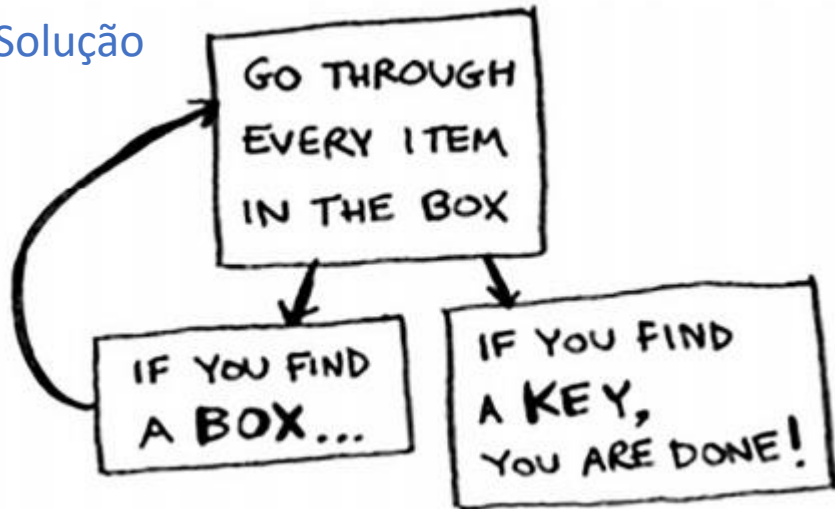
Voltando ao problema inicial...

# Solução 1 (laço)

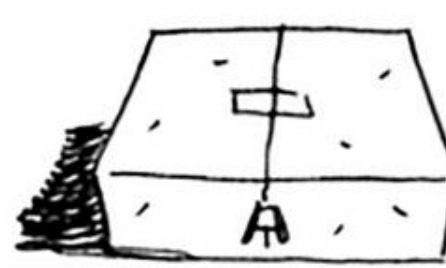
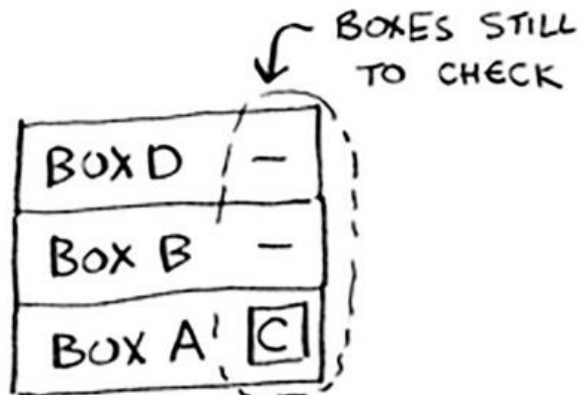


# Solução 1 (recursão)

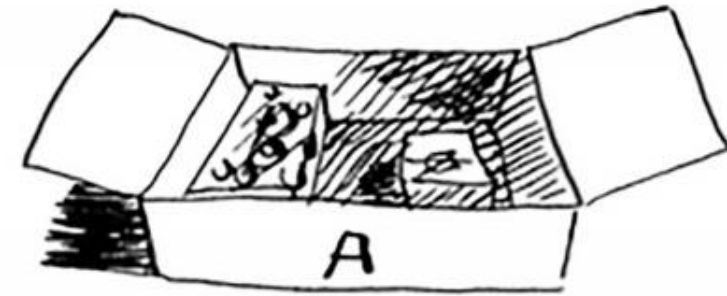
Solução



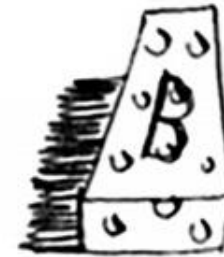
Call stack



YOU LOOK THROUGH  
BOX A



INSIDE YOU FIND  
BOXES B AND C



YOU CHECK  
BOX B



IT CONTAINS  
BOX D



YOU CHECK  
BOX D



IT IS  
EMPTY

# Limitações da recursividade

# Limitações da recursividade

Cada uma das funções de chamada ocupa memória, e pode resultar em estouro de memória.

O que fazer?

Reescrever seu código  
utilizando laços

Utilizar *tail recursion*  
(recursão de cauda)

# Recursão de cauda

# Recursão de cauda

- A chamada recursiva é a última tarefa executada pela função.

```
def countdown_print(n):  
    if (n < 0)  
        return  
    print(n)  
    countdown_print(n-1)
```

# É recursão de cauda?

```
def fact(x):  
    if x == 1:  
        return 1  
    else:  
        return x * fact(x-1)
```



# É recursão de cauda?

```
def fact(x):  
    if x == 1:  
        return 1  
    else:  
        return x * fact(x-1)
```

Não, o resultado de `fact(x-1)` é utilizado em `fact(x)`!

# Transformando em recursão de cauda

```
def factTR(x, a):  
    if (x == 0):  
        return a  
    return factTR(x-1, x*a)
```

```
def fact(x):  
    return factTR(x, 1)
```

Adiciona um  
parâmetro para  
guardar o valor  
acumulado.

# Otimização de recursão de cauda

```
def countdown_print(n):  
    if (n < 0)  
        return  
    print(n)  
    countdown_print(n-1)
```

```
def countdown_print(n):  
    start:  
        if (n < 0)  
            return  
        print(n)  
        n = n - 1  
        goto start;
```

# Resumo

**Recursão** é quando uma função chama a si mesmo

Toda chamada recursão tem dois casos: o **caso base** e o **caso recursivo**

A pilha tem duas operações básicas: *push* e *pop*

Todas as chamadas de função vão para a **pilha de chamadas** (*call stack*)

Uma pilha de chamada pode ficar muito grande, consumindo muita memória

Recursão de cauda como meio para otimização

# Referências

- [1] <https://stackoverflow.com/questions/72209/recursion-or-iteration/72694#72694>
- Imagens tiradas de **Entendendo Algoritmos**, Bhargava: <https://novatec.com.br/livros/entendendo-algoritmos/>