Recursividade

Jefersson A. dos Santos
jefersson@dcc.ufmg.br

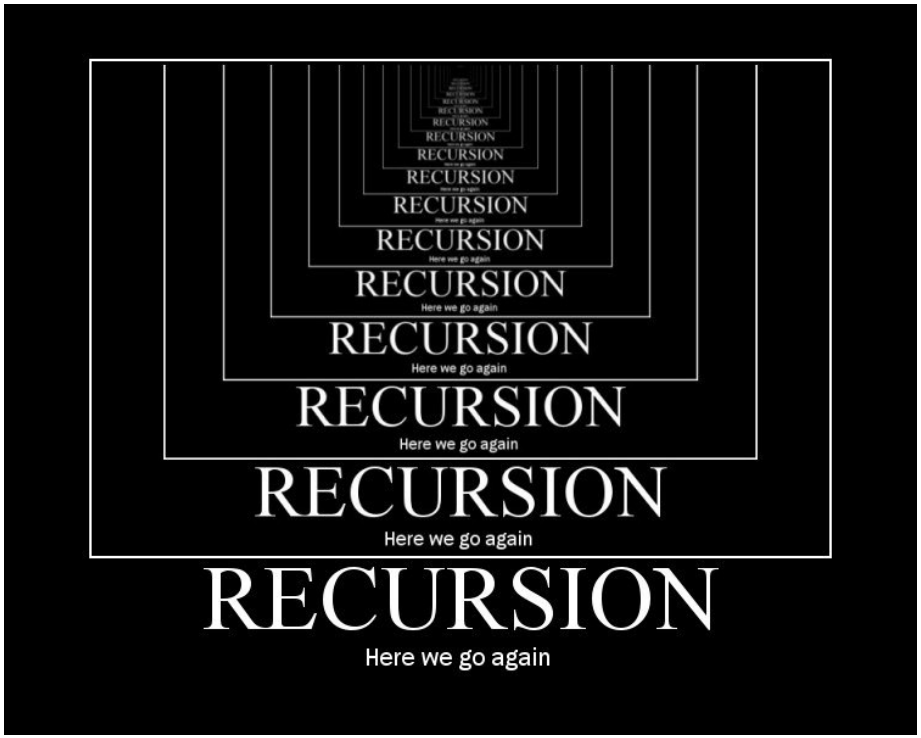
Guilherme Maia
jgmm@dcc.ufmg.br

DCC
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

UFMG

Recursividade

- O que é recursividade?
  - Um objeto é dito recursivo se o mesmo pode ser definido em termos de si próprio.



Recursividade

- Suponha que um colega te pergunte:

Como chego no Mineirão?

- Você pode responder com um conjunto completo de direções e referências.

- Mas se as orientações forem muito complexas, você pode optar por responder:

"Vá até a saída da UFMG, e chegando lá pergunte:
Como chego no Mineirão?"

Recursividade

- Após realizar as suas instruções, seu colega não vai ter resolvido o problema completo, mas ele vai se deparar com uma nova instância do mesmo problema.

- **O problema é idêntico ao original, mas agora está mais próximo de ser solucionado!**

Recursividade

- Componentes da recursão:
  - Caso base (caso trivial): uma instância do problema que pode ser solucionada facilmente.
  - Uma ou mais chamadas recursivas: o objeto define-se em termos de si próprio, tentando convergir para o caso base.

Função fatorial

- A função fatorial de um inteiro não negativo pode ser definida como:

$$n! = \begin{cases} 1 & n = 0 \\ n \times (n-1)! & n > 0 \end{cases}$$

- Esta definição estabelece um processo recursivo para calcular o fatorial de um inteiro n.

- Caso trivial: n = 0. Neste caso: n! = 0! =1

- Método geral: n x (n-1)!

Função fatorial

- Assim, usando-se este processo recursivo, o cálculo de 4!, por exemplo, é feito como a seguir:

```
4! = 4 * 3!
   = 4 * (3 * 2!)
   = 4 * (3 * (2 * 1!))
   = 4 * (3 * (2 * (1 * 0!)))
   = 4 * (3 * (2 * (1 * 1)))
   = 4 * (3 * (2 * 1))
   = 4 * (3 * 2)
   = 4 * 6
   = 24
```

Recursividade

- No contexto de funções, recursividade significa uma **função que chama ela mesma.**

```
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```

Recapitulando

- Para que x receba um valor, é necessário que todas as expressões à direita da atribuição sejam processadas.
  - Isso inclui as chamadas de função

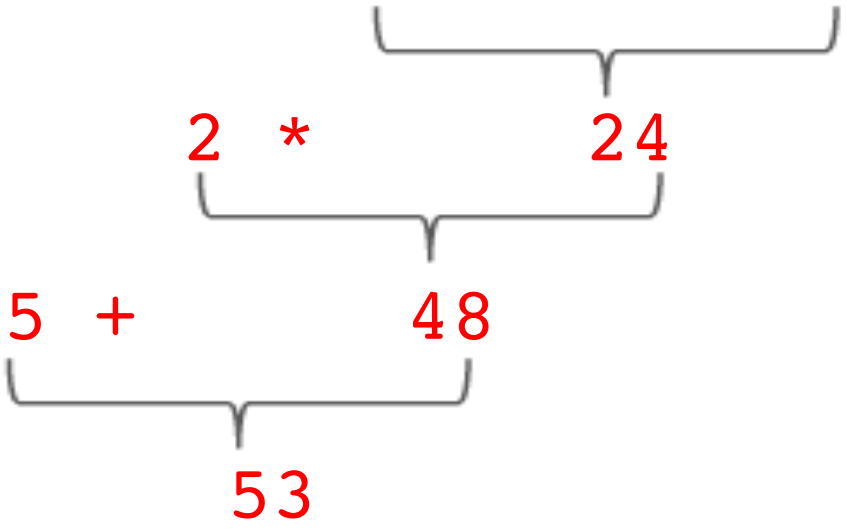
```
def fatorial(n):
    fat = 1
    for i in range(1, n+1):
        fat = fat * i
    return fat


x = 5 + 2 * fatorial(4)
```

Recapitulando

```
def fatorial(n):
    fat = 1
    for i in range(1, n+1):
        fat = fat * i
    return fat

x = 5 + 2 * fatorial(4)
```



Recapitulando

- Em uma função recursiva, a mesma regra se aplica, o que resulta na construção de uma **pilha** de chamadas de função, que para o computador representa uma **pilha de execução**.

```
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)

x = 5 + 2 * fatorial(4)
```

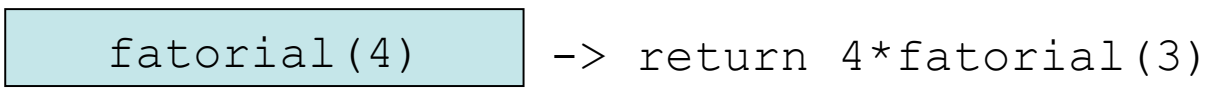
Função fatorial

- Considere, novamente, o exemplo para 4!:

```
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```

Pilha de Execução

Empilha `fatorial(4)`



Passo a passo: https://goo.gl/f8Ur5B

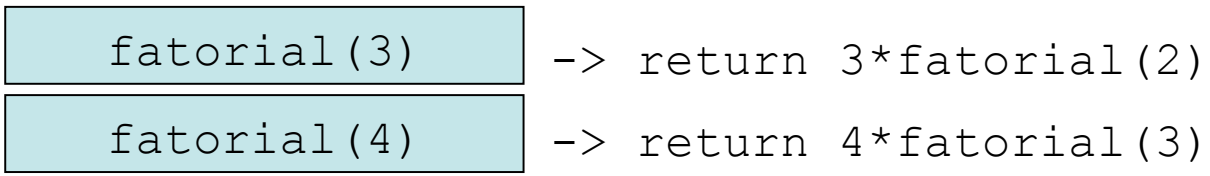
Função fatorial

- Considere, novamente, o exemplo para 4!:

```
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```

Pilha de Execução

Empilha `fatorial(3)`



Passo a passo: https://goo.gl/f8Ur5B

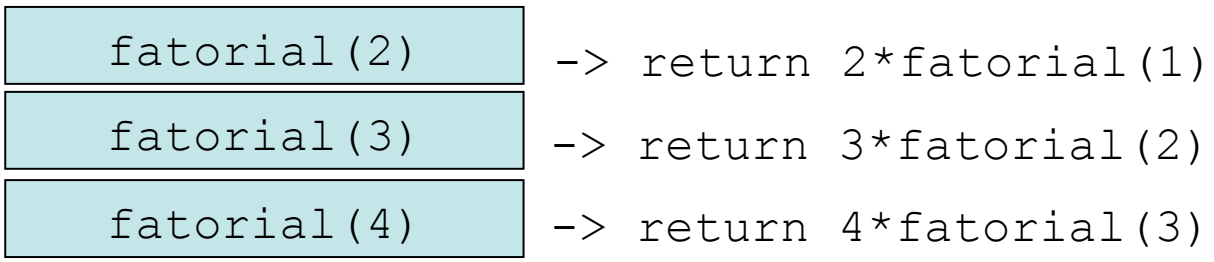
Função fatorial

- Considere, novamente, o exemplo para 4!:

```
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```

Pilha de Execução

Empilha `fatorial(2)`



Passo a passo: https://goo.gl/f8Ur5B

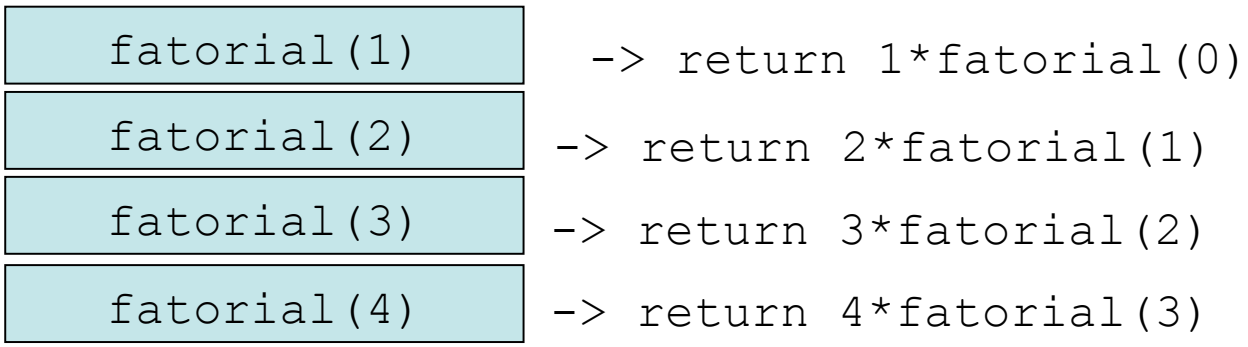
Função fatorial

- Considere, novamente, o exemplo para 4!:

```
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```

Pilha de Execução

Empilha `fatorial(1)`

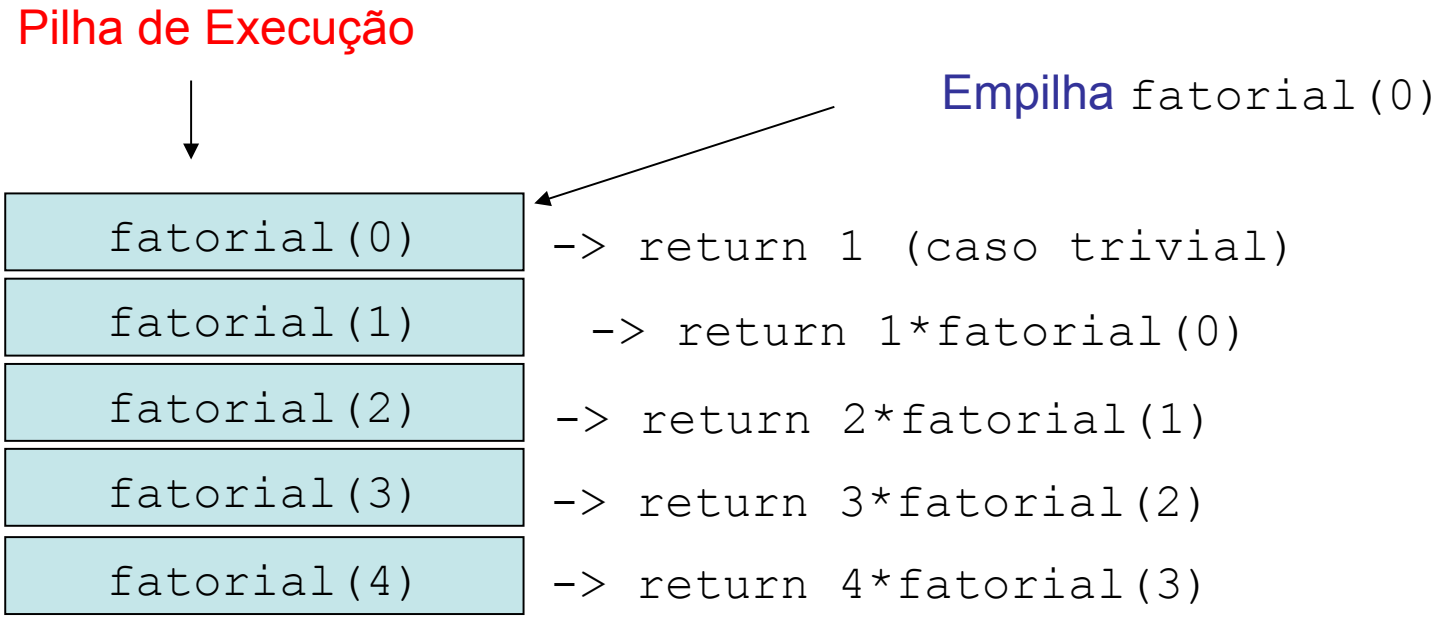


Passo a passo: https://goo.gl/f8Ur5B

Função fatorial

- Considere, novamente, o exemplo para 4!:

```
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```

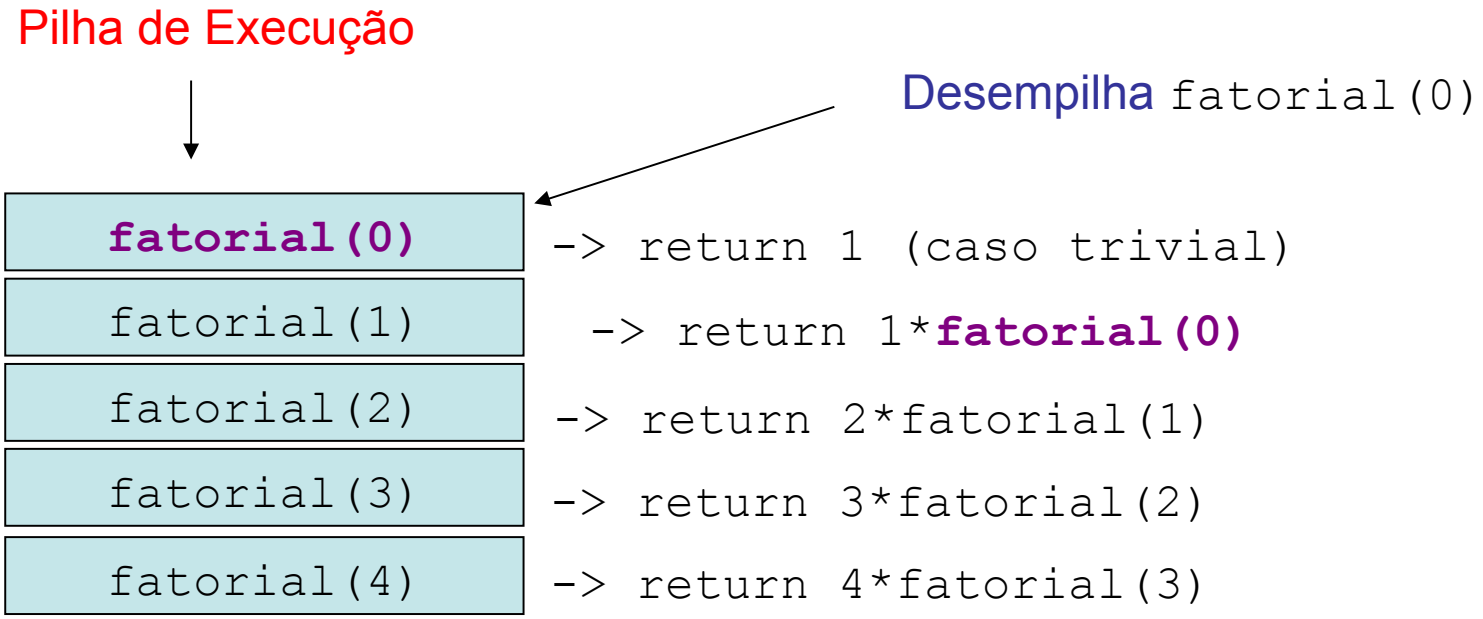


Passo a passo: https://goo.gl/f8Ur5B

Função fatorial

- Considere, novamente, o exemplo para 4!:

```
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```

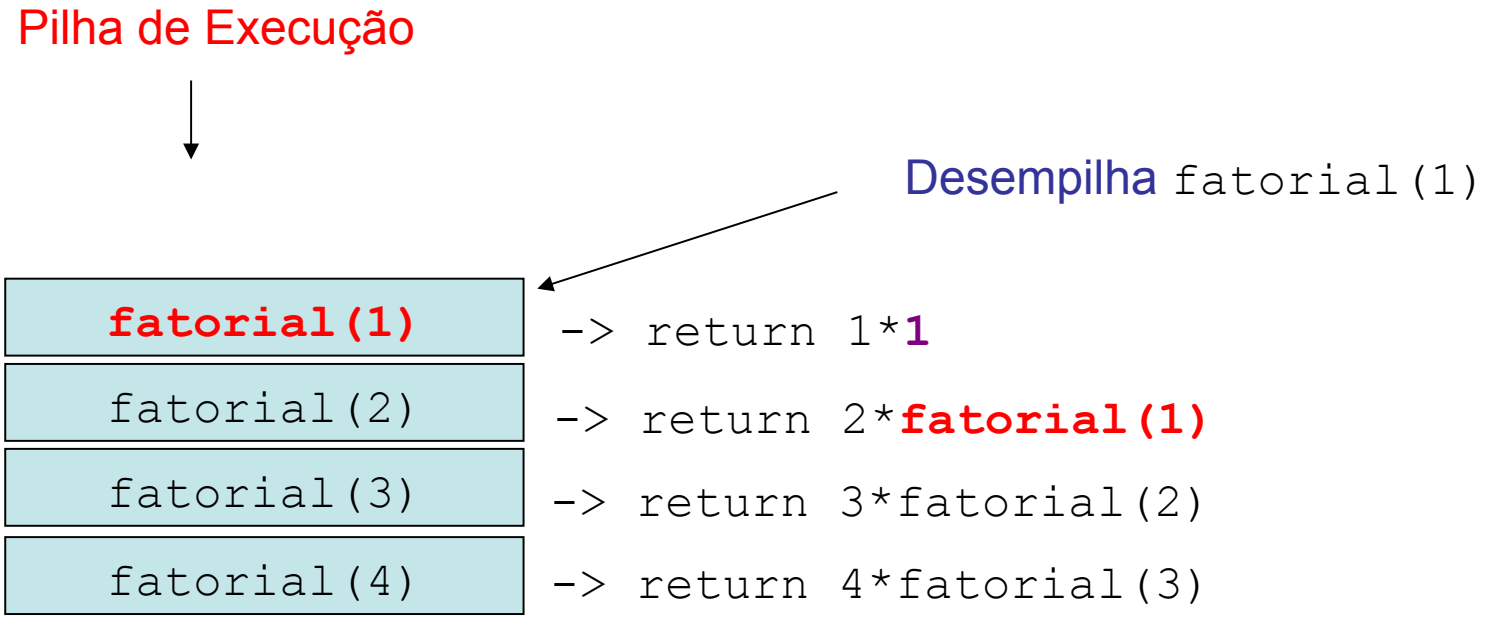


Passo a passo: https://goo.gl/f8Ur5B

Função fatorial

- Considere, novamente, o exemplo para 4!:

```
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```



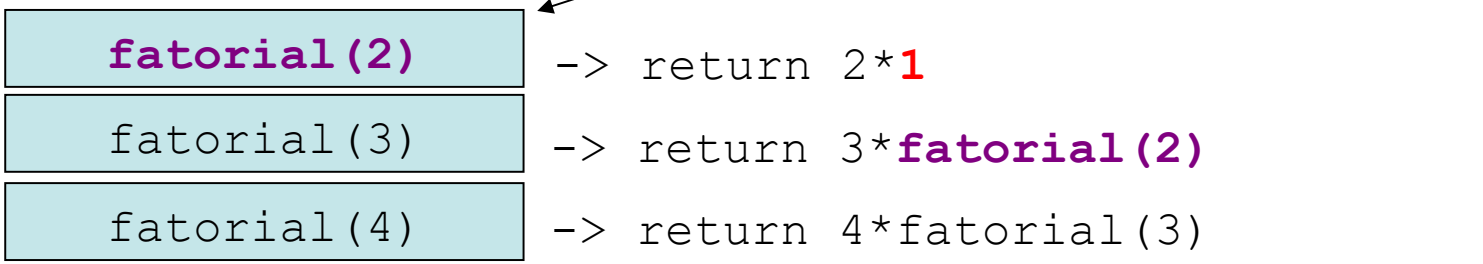
Passo a passo: https://goo.gl/f8Ur5B

Função fatorial

- Considere, novamente, o exemplo para 4!:

```
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```

Pilha de Execução



Passo a passo: https://goo.gl/f8Ur5B

Função fatorial

- Considere, novamente, o exemplo para 4!:

```
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```

Pilha de Execução

Desempilha `fatorial(3)`

`fatorial(3)`	`-> return 3*`**`2`**
`fatorial(4)`	`-> return 4*`**`fatorial(3)`**

Passo a passo: https://goo.gl/f8Ur5B

Função fatorial

- Considere, novamente, o exemplo para 4!:

```
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```

Pilha de Execução

Desempilha `fatorial(4)`

`fatorial(4)` `-> return 4*6`

Passo a passo: https://goo.gl/f8Ur5B

Função fatorial

- Considere, novamente, o exemplo para 4!:

```
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```

Resultado = 24

Passo a passo: https://goo.gl/f8Ur5B

Função Fibonacci

- A função Fibonacci retorna o n-ésimo número da seqüência: 1, 1, 2, 3, 5, 8, 13, ...

- Os dois primeiros termos são iguais a 1 e cada um dos demais números é a soma dos dois números imediatamente anteriores.

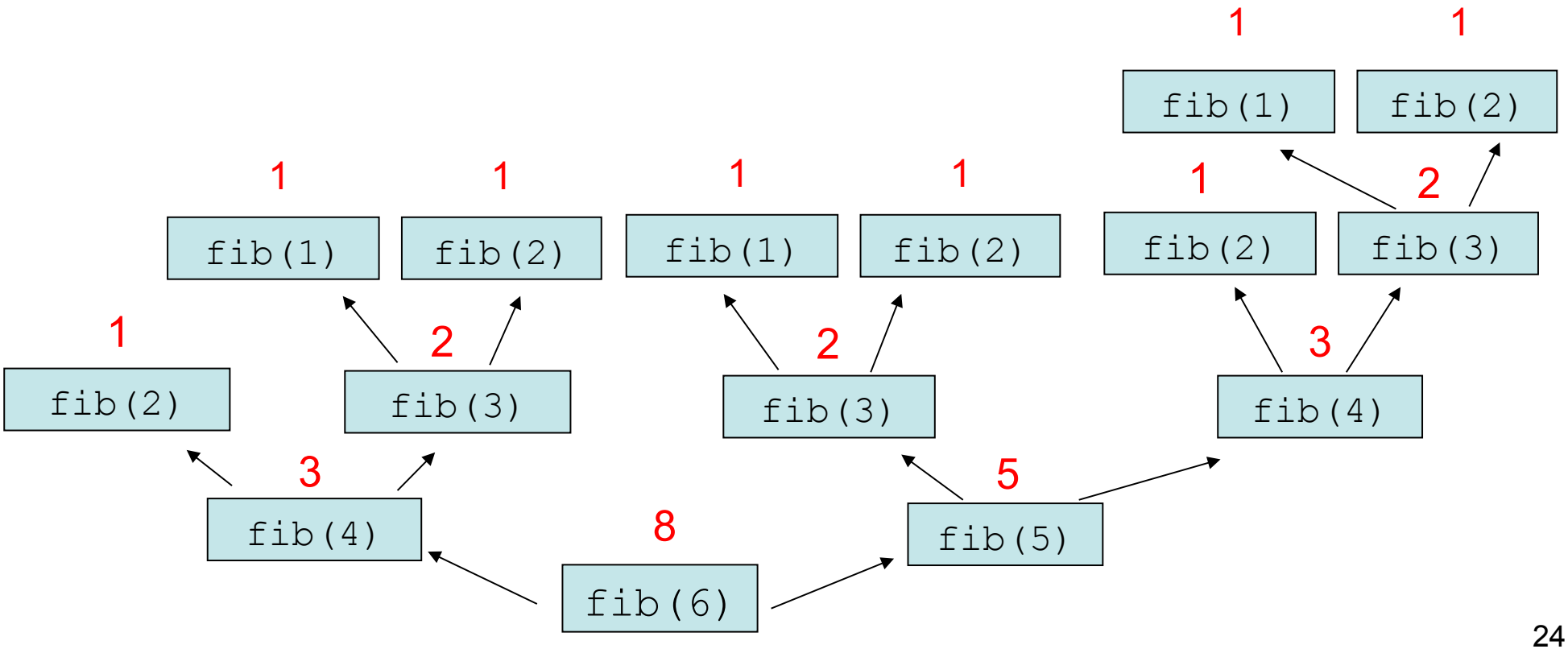
- Sendo assim, o n-ésimo número *fib(n)* é dado por:

$$fib(n) = \begin{cases} 1 & n = 1 \\ 1 & n = 2 \\ fib(n-2) + fib(n-1) & n > 2 \end{cases}$$

Função Fibonacci

- Veja uma implementação recursiva para esta função:

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



Função Fibonacci

- A função recursiva para cálculo do n-ésimo termo da seqüência é extremamente ineficiente, uma vez que recalcula o mesmo valor várias vezes

Observe agora uma versão iterativa da função fib:

```
def fib(n):
        if n == 1 or n == 2:
                return 1
        x = y = 1
        for i in range(2, n):
                z = x + y
                x = y
                y = z
        return y
```

Função Fibonacci

- O livro dos autores Brassard e Bradley (*Fundamentals of Algorithmics*, 1996, pág. 73) apresenta um quadro comparativo de tempos de execução das versões iterativa e recursiva:

n	10	20	30	50	100
recursivo	8 ms				
iterativo	0.17 ms				

- Portanto: um algoritmo recursivo nem sempre é o melhor caminho para se resolver um problema.
- No entanto, a recursividade muitas vezes torna o algoritmo mais simples.

Função Fibonacci

- O livro dos autores Brassard e Bradley (*Fundamentals of Algorithmics*, 1996, pág. 73) apresenta um quadro comparativo de tempos de execução das versões iterativa e recursiva:

n	10	20	30	50	100
recursivo	8 ms	1 s			
iterativo	0.17 ms	0.33 ms			

- Portanto: um algoritmo recursivo nem sempre é o melhor caminho para se resolver um problema.
- No entanto, a recursividade muitas vezes torna o algoritmo mais simples.

Função Fibonacci

- O livro dos autores Brassard e Bradley (*Fundamentals of Algorithmics*, 1996, pág. 73) apresenta um quadro comparativo de tempos de execução das versões iterativa e recursiva:

n	10	20	30	50	100
recursivo	8 ms	1 s	2 min		
iterativo	0.17 ms	0.33 ms	0.50 ms		

- Portanto: um algoritmo recursivo nem sempre é o melhor caminho para se resolver um problema.
- No entanto, a recursividade muitas vezes torna o algoritmo mais simples.

Função Fibonacci

- O livro dos autores Brassard e Bradley (*Fundamentals of Algorithmics*, 1996, pág. 73) apresenta um quadro comparativo de tempos de execução das versões iterativa e recursiva:

n	10	20	30	50	100
recursivo	8 ms	1 s	2 min	21 dias	
iterativo	0.17 ms	0.33 ms	0.50 ms	0.75 ms	

- Portanto: um algoritmo recursivo nem sempre é o melhor caminho para se resolver um problema.
- No entanto, a recursividade muitas vezes torna o algoritmo mais simples.

Função Fibonacci

- O livro dos autores Brassard e Bradley (*Fundamentals of Algorithmics*, 1996, pág. 73) apresenta um quadro comparativo de tempos de execução das versões iterativa e recursiva:

n	10	20	30	50	100
recursivo	8 ms	1 s	2 min	21 dias	?????
iterativo	0.17 ms	0.33 ms	0.50 ms	0.75 ms	1,50 ms

- Portanto: um algoritmo recursivo nem sempre é o melhor caminho para se resolver um problema.
- No entanto, a recursividade muitas vezes torna o algoritmo mais simples.

Função Fibonacci

- O livro dos autores Brassard e Bradley (*Fundamentals of Algorithmics*, 1996, pág. 73) apresenta um quadro comparativo de tempos de execução das versões iterativa e recursiva:

n	10	20	30	50	100
recursivo	8 ms	1 s	2 min	21 dias	10^9 anos
iterativo	0.17 ms	0.33 ms	0.50 ms	0.75 ms	1,50 ms

- Portanto: um algoritmo recursivo nem sempre é o melhor caminho para se resolver um problema.
- No entanto, a recursividade muitas vezes torna o algoritmo mais simples.

Exemplos

- Exemplo 1: https://tinyurl.com/y3ponot9
- Exemplo 2: https://tinyurl.com/y3ueeurj