



Recursividade

Prof. Sérgio Souza Costa

Recursividade

- Muitas estruturas de dados como listas e árvores são recursivas, então é importante compreender conceito de recursividade.

Recursividade

- A **recursividade** é uma forma de resolver problemas por meio da divisão dos problemas em problemas menores de **mesma natureza**.
- Se a natureza dos subproblemas é a mesma do problema, o mesmo método usado para reduzir o problema pode ser usado para reduzir os subproblemas e assim por diante.
- **Quando devemos parar?** Quando alcançarmos um **caso trivial** que conhecemos a solução.

Recursividade

- Exemplos de **casos triviais**:
 - Qual o fatorial de 0 ?
 - Quanto é um dado X multiplicado por 1 ?
 - Quanto é um dado X multiplicado por 0 ?
 - Quanto é um dado X elevado a 1 ?
 - Quantos elementos tem em uma lista vazia?

Recursividade

- Assim, um **processo recursivo** para a solução de um problema consiste em duas partes:
 - O caso trivial, cuja solução é conhecida;
 - Um método geral que reduz o problema a um ou mais problemas menores (subproblemas) de mesma natureza.
- Muitas funções podem ser definidas recursivamente. Para isso, é preciso identificar as duas partes acima.
- **Exemplo:** fatorial de um número e o n -ésimo termo da seqüência de Fibonacci.

Função fatorial

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

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

- Esta definição estabelece um **processo recursivo** para calcular o fatorial de um inteiro **n**.
- Caso trivial: **n=0**.
- 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! =

Função fatorial

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

$$4! = 4 * 3!$$

Função fatorial

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

$$\begin{aligned} 4! &= 4 * 3! \\ &= 4 * (3 * 2!) \end{aligned}$$

Função fatorial

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

$$\begin{aligned} 4! &= 4 * 3! \\ &= 4 * (3 * 2!) \\ &= 4 * (3 * (2 * 1!)) \end{aligned}$$

Função fatorial

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

$$\begin{aligned} 4! &= 4 * 3! \\ &= 4 * (3 * 2!) \\ &= 4 * (3 * (2 * 1!)) \\ &= 4 * (3 * (2 * (1 * 0!))) \end{aligned}$$

Função fatorial

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

$$\begin{aligned} 4! &= 4 * 3! \\ &= 4 * (3 * 2!) \\ &= 4 * (3 * (2 * 1!)) \\ &= 4 * (3 * (2 * (1 * 0!))) \\ &= 4 * (3 * (2 * (1 * 1))) \end{aligned}$$

Função fatorial

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

$$\begin{aligned}4! &= 4 * 3! \\&= 4 * (3 * 2!) \\&= 4 * (3 * (2 * 1!)) \\&= 4 * (3 * (2 * (1 * 0!))) \\&= 4 * (3 * (2 * (1 * 1))) \\&= 4 * (3 * (2 * 1))\end{aligned}$$

Função fatorial

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

$$\begin{aligned} 4! &= 4 * 3! \\ &= 4 * (3 * 2!) \\ &= 4 * (3 * (2 * 1!)) \\ &= 4 * (3 * (2 * (1 * 0!))) \\ &= 4 * (3 * (2 * (1 * 1))) \\ &= 4 * (3 * (2 * 1)) \\ &= 4 * (3 * 2) \end{aligned}$$

Função fatorial

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

$$\begin{aligned}4! &= 4 * 3! \\&= 4 * (3 * 2!) \\&= 4 * (3 * (2 * 1!)) \\&= 4 * (3 * (2 * (1 * 0!))) \\&= 4 * (3 * (2 * (1 * 1))) \\&= 4 * (3 * (2 * 1)) \\&= 4 * (3 * 2) \\&= 4 * 6\end{aligned}$$

Função fatorial

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

$$\begin{aligned} 4! &= 4 * 3! \\ &= 4 * (3 * 2!) \\ &= 4 * (3 * (2 * 1!)) \\ &= 4 * (3 * (2 * (1 * 0!))) \\ &= 4 * (3 * (2 * (1 * 1))) \\ &= 4 * (3 * (2 * 1)) \\ &= 4 * (3 * 2) \\ &= 4 * 6 \\ &= 24 \end{aligned}$$

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



Mas como uma **função recursiva** é de fato implementada no computador?

Usando-se o mecanismo conhecido como **Pilha de Execução!**

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



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

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



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

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



fatorial(2)	-> return 2*fatorial(1)
fatorial(3)	-> return 3*fatorial(2)
fatorial(4)	-> return 4*fatorial(3)

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



fatorial(1)	-> return 1*fatorial(0)
fatorial(2)	-> return 2*fatorial(1)
fatorial(3)	-> return 3*fatorial(2)
fatorial(4)	-> return 4*fatorial(3)

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



fatorial(0)	-> return 1 (caso trivial)
fatorial(1)	-> return 1*fatorial(0)
fatorial(2)	-> return 2*fatorial(1)
fatorial(3)	-> return 3*fatorial(2)
fatorial(4)	-> return 4*fatorial(3)

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



fatorial(0)	-> return 1 (caso trivial)
fatorial(1)	-> return 1*fatorial(0)
fatorial(2)	-> return 2*fatorial(1)
fatorial(3)	-> return 3*fatorial(2)
fatorial(4)	-> return 4*fatorial(3)

Desempilha fatorial(0)

Função fatorial

- Considere, novamente, o exemplo para 4!:

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

Desempilha fatorial(1)

fatorial(1)	-> return 1*1
fatorial(2)	-> return 2*fatorial(1)
fatorial(3)	-> return 3*fatorial(2)
fatorial(4)	-> return 4*fatorial(3)

Função fatorial

- Considere, novamente, o exemplo para 4!:

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

Desempilha fatorial(2)

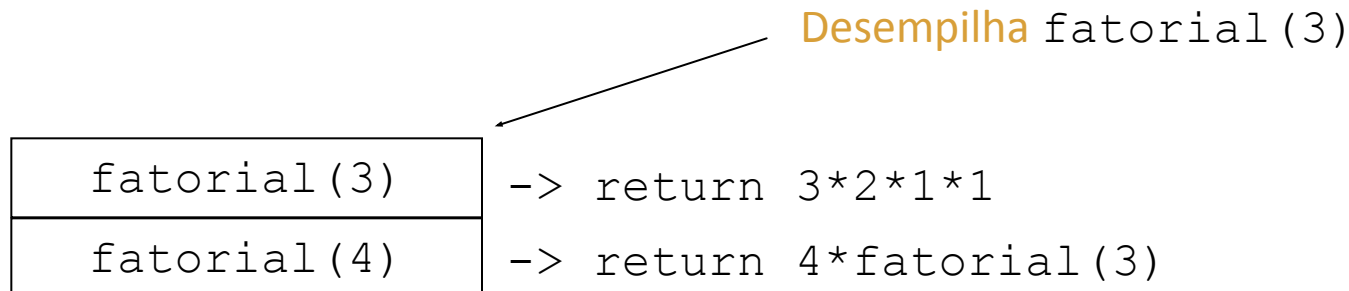
fatorial(2)	-> return 2*1*1
fatorial(3)	-> return 3*fatorial(2)
fatorial(4)	-> return 4*fatorial(3)

Função fatorial

- Considere, novamente, o exemplo para 4!:

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

Desempilha fatorial(3)



fatorial(3)	-> return 3*2*1*1
fatorial(4)	-> return 4*fatorial(3)

Função fatorial

- Considere, novamente, o exemplo para 4!:

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

Desempilha fatorial(4)

fatorial(4)

-> return 4*3*2*1*1

Função fatorial

- Considere, novamente, o exemplo para 4!:

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

Resultado = 24

Recursividade

"Para fazer um procedimento recursivo é preciso ter fé."

prof. Siang Wun Song

"Ao tentar resolver o problema, encontrei obstáculos dentro de obstáculos. Por isso, adotei uma solução recursiva." Aluno S.Y., 1998

"To understand recursion, we must first understand recursion." —anônimo

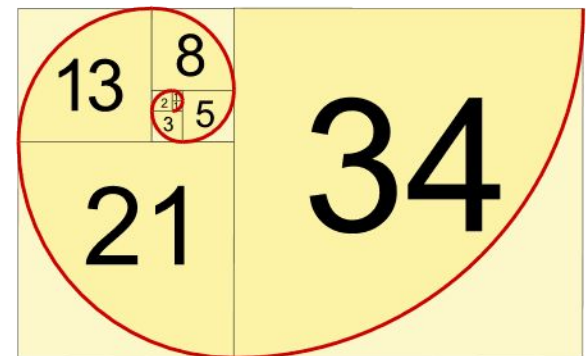
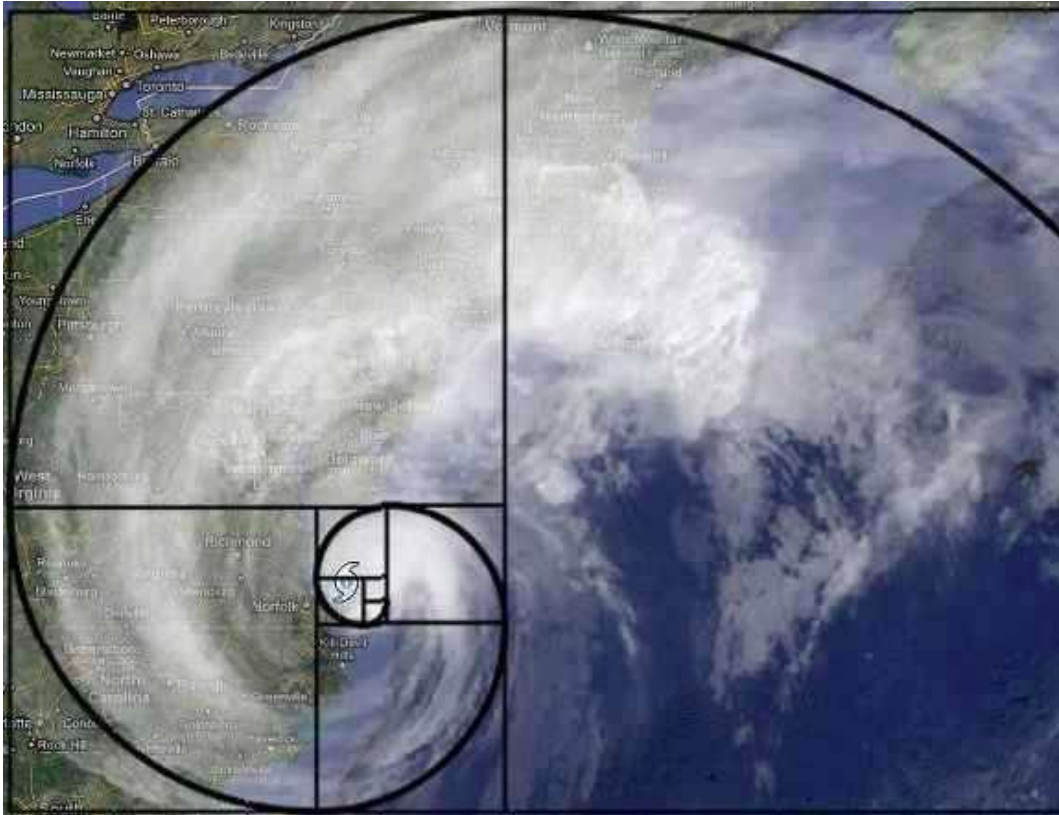
Funções recursivas

- Base do paradigma funcional.
 - ○ Lisp, Haskell, Miranda, F#, Scheme, Erland
 - Influenciou diversas: JavaScript, Python, Ruby, Lua
- Presente em praticamente todas as linguagens, mesmo as imperativas, como
 - C, Java, Pascal, ADA ...

Funções recursivas

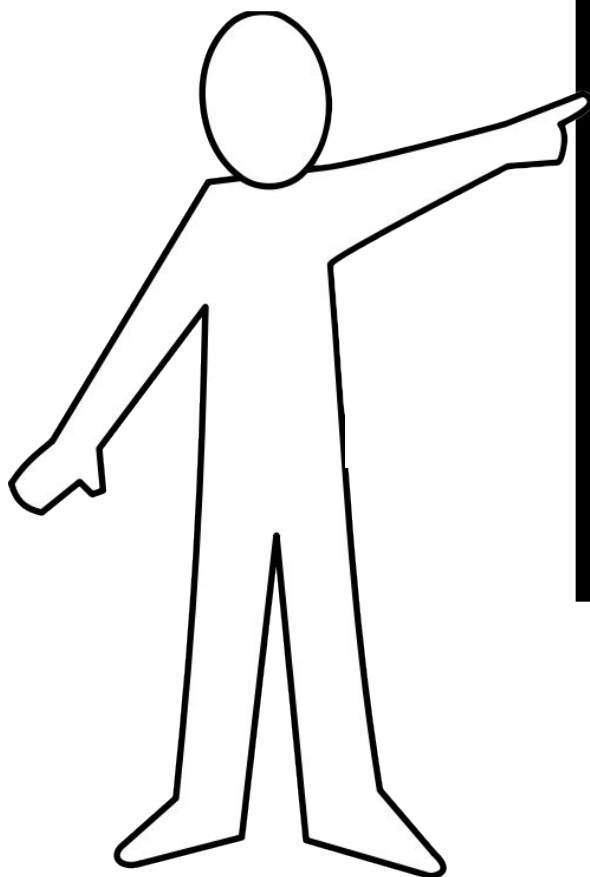
- Muitos algoritmos complexos são resolvidos através de soluções recursivas
 - Quick-sort (ordenação rápida)
 - Ordenação por intercalação (merge sort)
 - Busca binária
- Muitas estruturas de dados são recursivas
 - Listas encadeadas
 - Árvores binárias, n-árias ...
- Tendem a gerar códigos menores

Recursividade - Natureza



Recursividade - Natureza





Vamos fazer mais um exercício. Na atividade de aprofundamento tem mais exercícios. Tentem fazer e postem suas dúvidas.

Exercício resolvido

- Codifiquem uma função que multiplica um dado inteiro “a” por um inteiro “b”, usando somas sucessivas. Exemplo:
- $4 * 3 = 3 + 3 + 3 + 3 = 12$

Exercício resolvido

```
def mult (a, b):
```

Exercício resolvido

```
def mult (a, b):  
    if a == 0:  
        return 0;
```

Exercício resolvido

```
def mult (a, b):  
    if a == 0:  
        return 0;  
    elif a == 1:  
        return b;
```

Exercício resolvido

```
def mult (a, b):  
    if a == 0:  
        return 0;  
    elif a == 1:  
        return b;  
    else:  
        return b + mult (a-1,b)
```

Vamos testar ?

Vamos testar ?

```
mult (3, 5)
```


Vamos testar ?

```
mult(3, 5) = 5 + mult(2, 5)
```

Vamos testar ?

```
mult(3, 5) = 5 + mult(2, 5)  
           = 5 + 5 + mult(1, 5)
```

Vamos testar ?

```
mult(3, 5) = 5 + mult(2, 5)  
            = 5 + 5 + mult(1, 5)  
            = 5 + 5 + 5
```

Vamos testar ?

```
mult(3, 5) = 5 + mult(2, 5)  
            = 5 + 5 + mult(1, 5)  
            = 5 + 5 + 5  
            = 15
```

Exercícios

1. Faça uma função recursiva que calcula a divisão usando subtrações sucessivas

int div (int a, int b);

2. Faça uma função recursiva que calcula o resto de uma divisão usando subtrações sucessivas.

int mod (int a, int b);

3. Faça uma função recursiva que calcula a potencia de um numero usando multiplicações sucessivas.

int pot (int base, int exp);

4. Faça uma função recursiva que calcula a quantidade de caracteres em uma string.

int tamanho (char *str)

5. Fazer uma função recursiva que calcule o valor da série S descrita a seguir para um valor $n > 0$ a ser fornecido como parâmetro para a mesma:

$S = 1 + 1/2! + 1/3! + \dots 1/n!$