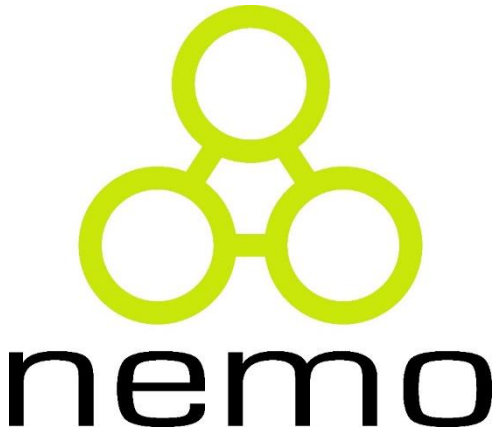


Programação I

Jordana S. Salamon

jssalamon@inf.ufes.br

jordanasalamon@gmail.com

DEPARTAMENTO DE INFORMÁTICA

CENTRO TECNOLÓGICO

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Paradigma Aplicativo

REVISÃO

- Aplicação uma função, de forma cumulativa, à uma coleção de elementos.
- Generalizar a aplicação de uma operação a todos os elementos de uma lista.

- Dada uma lista [x0, x1, ... , xn], deseja-se aplicar uma função f à todos os elementos da lista de forma cumulativa:
  - f(... f (f(x0, x1), x2, ...) , xn)
  - f(... f (f(valor inicial, x0), x1, ...) , xn)

nemo

Paradigma Aplicativo

REVISÃO

Defina o fatorial de um número.

O fatorial de um número natural n>0 é igual ao produto de todos os números naturais de 1 até n.

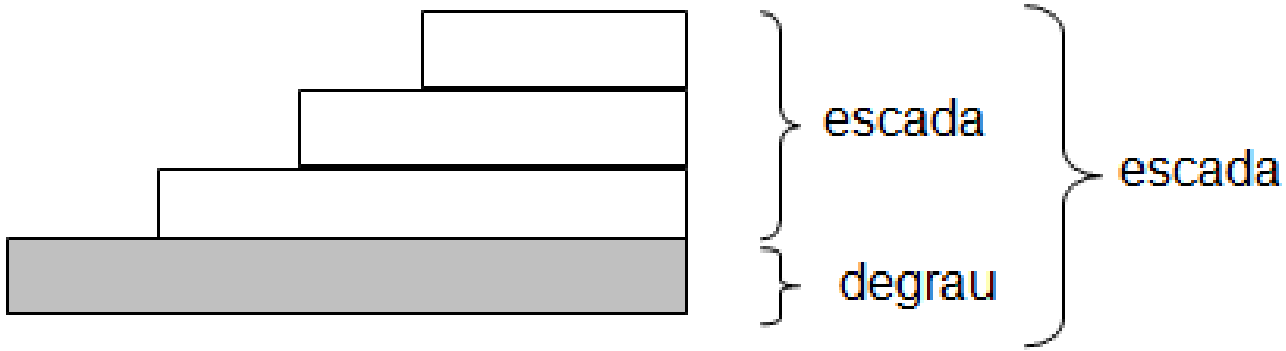
```
>>> def fatorial(n):
        def prod(x,y):  return x * y
        return reduce (prod, range(1,n+1), 1)
```

nemo

Paradigma Recursivo

- Consiste em descrever um conceito de forma recursiva.
- Definir um conceito usando o próprio conceito.

Uma escada é igual a um degrau seguido de uma escada.



Paradigma Recursivo

- Um número natural pode ser definido recursivamente.
- Um número natural é:
  - zero;
  - sucessor de um número natural;
  - 5 = suc(suc(suc(suc(suc(0)))))

- Uma lista também pode ser definida recursivamente.
- Uma lista é:
  - a lista vazia;
  - um elemento seguido de uma lista;
  - [1,2,3] = 1 : [2 : [3 : []]]
    - (elemento 1 seguido da lista definida pelo elemento 2 seguido da lista definida pelo elemento 3 seguido da lista vazia)

Elementos de uma Descrição Recursiva

- **Definição geral:** Toda definição recursiva tem duas partes, uma delas se aplica a um valor qualquer do domínio do problema, denominada de geral. Esta tem uma característica muito importante: o conceito que está sendo definido deve ser utilizado.
  - Por exemplo, para definir fatorial de n, usamos o fatorial do *antecessor de n*. Observe aqui, entretanto que o mesmo conceito foi utilizado, mas não para o mesmo valor. Aplicamos o conceito a um valor mais simples, neste caso o antecessor de n.

- **Definição independente:** A outra parte da definição é destinada ao tratamento de um valor tão simples que a sua definição possa ser dada de forma independente. Este elemento é também conhecido como base da recursão.
  - No caso do fatorial, o valor considerado é o zero.

Elementos de uma Descrição Recursiva

- **Obtenção de valores mais simples:** Para aplicar o conceito a um valor mais simples precisamos de uma função que faça este papel.
  - No caso do fatorial, usamos a subtração de n por 1, obtendo assim o antecessor de n. Em cada caso, dependendo do domínio do problema e do problema em si, precisaremos encontrar a função apropriada.

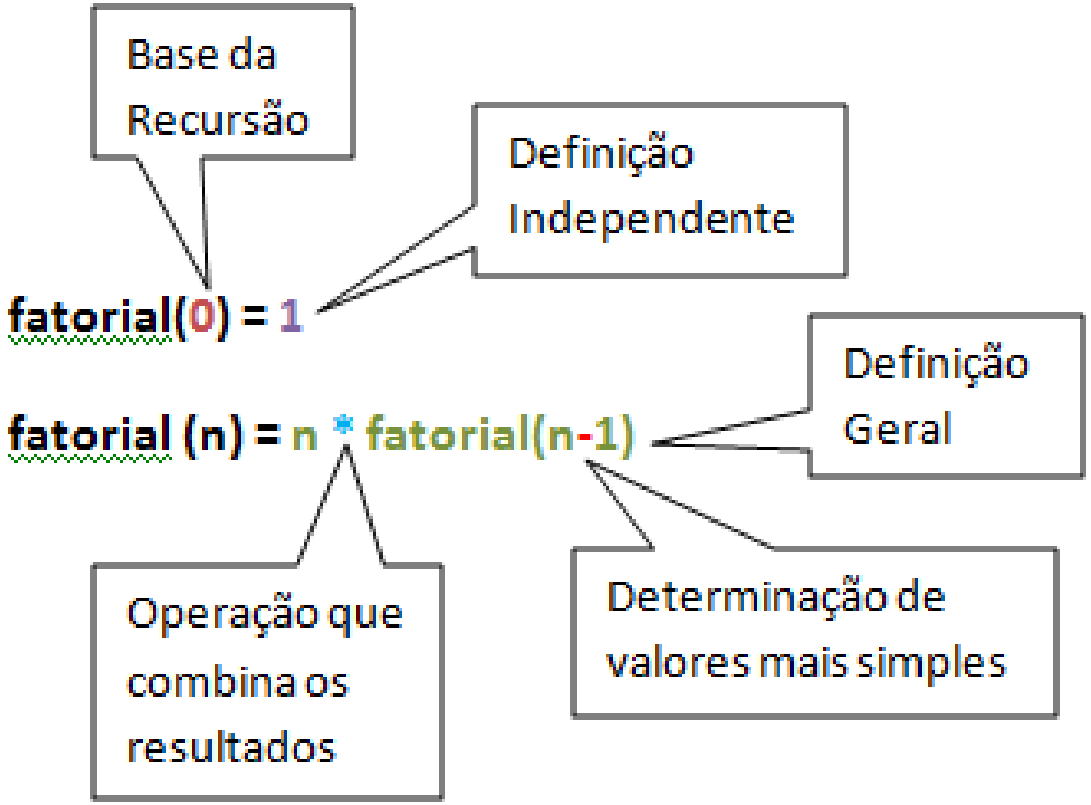
- **Função auxiliar**: Na definição geral, para obter um valor usando o valor considerado e o valor definido recursivamente, em geral faz-se necessário o uso de uma função auxiliar. Algumas vezes esta função pode ser originada a partir de um conceito aplicável a dois elementos e que desejamos estender aos elementos de uma lista.
  - No caso do fatorial esta função é a multiplicação.

- **Garantia de atingir o valor independente**: É fundamental que a aplicação sucessiva da função que obtém valores mais simples garanta a determinação do valor mais simples. Este valor é também denominado de base da recursão.

Paradigma Recursivo

- Forma recursiva de definição do fatorial:

O Fatorial de um número natural n é igual ao produto deste número pelo fatorial de seu antecessor.



nemo

Paradigma Recursivo

- Forma recursiva de definição do fatorial:

O Fatorial de um número natural n é igual ao produto deste número pelo fatorial de seu antecessor.

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

>>> fatorial(5)
120
```

nemo

Exemplos

- Faça uma função recursiva que calcule o valor do fibonacci de n:
  - F(0) = 0
  - F(1) = 1
  - F(n) = F(n-1) + F(n-2)

Exemplos

- Faça uma função recursiva que calcule o valor do fibonacci de n:

```
>>> def fibonacciRec(n):
        if n in [0,1]: return n
        else: return fibonacciRec(n-1) + fibonacciRec(n-2)
```

```
>>> fibonacciRec(0)
0
>>> fibonacciRec(1)
1
```

```
>>> fibonacciRec(2)
 1
>>> fibonacciRec(3)
     2
```

nemo

Solução Aplicativa

- Calcule a sequência de fibonacci:

```
>>> def fibonacciAplic(n):
        def f ( (penultimo, ultimo,),  elem):
            return (ultimo, ultimo + penultimo)
        def seg((x,y)):  return y
        if n <= 1: return n
        else:  return seg( reduce(f,  range(2,n+1),  (0,1) ))
```

Exemplo

- Verifique se uma lista está em ordem não-decrescente

Solução Recursiva

- Verifique se uma lista está em ordem não-decrescente

```
>>> def  listaOrdemNaoDecresc(lista):
    if len(lista) <= 1: return True
    else: return lista[0] <= lista[1] and
            listaOrdemNaoDecresc(lista[1:])
```

Solução Aplicativa

- Verifique se uma lista está em ordem não-decrescente

```
>>> def listaOrdemNaoDecrescente(lista):
    def f((avaliacao, elemAnterior), elemAtual):
        def novaAvaliacao():
            return avaliacao and elemAnterior <= elemAtual
        return (novaAvaliacao(), elemAtual)
    def prim((x,y)): return x
    if len(lista) <= 1: return True
    else: return prim(reduce(f, lista[1:], (True, lista[0])))
```

Exercícios

- Identifique o menor elemento de uma lista
- Calcule o somatório dos elementos de uma lista
- Some os elementos pares e subtraia os impares de uma lista
- Elimine os elementos repetidos de uma lista
- Inserir um elemento de forma ordenada numa lista ordenada.
- Ordenar uma lista.

That’s all Folks!