# Programação Funcional — Como Definir Funções Recursivas em 5 Etapas

## 1 Etapas

Pode ser difícil saber como começar a escrever uma definição recursiva em Haskell. Para ajudar o processo, pode seguir as seguintes 5 etapas:

(1) *Descrever o objetivo da função* num comentário, i.e. quais os *dados* da função bem como o *resultado* esperado.

(2) *Escrever o tipo da função*, associando cada um dos dados e resultado descritos na etapa anterior um tipo em Haskell.

(3) *Definir equações para os casos de tamanhos pequenos*; não deve preocupar-se em fazer contas, apenas escrever *expressões* para os resultados.

(4) *Generalizar os casos anteriores definindo uma equação de recorrência*; tente escrever resultado para tamanho $n$ usando resultado(s) de tamanho(s) inferiores.

(5) *Eliminar as equações que são redundantes*: verificar se a equação de recorrência geral dá o resultado correto para os exemplo pequenos; se assim for, pode eliminá-lo.

## 2 Exemplos

### 2.1 Factorial

Vamos usar as 5 etapas para definir uma função recursiva para calcular o *factorial* de um inteiro $n \geq 0$, ou seja, $n! = 1 \times 2 \times 3 \times \cdots \times n$.

**Etapas 1 e 2**

```
-- Dado um inteiro n>=0,
-- calcular 1*2*...*n
factorial :: Integer -> Integer
```

**Etapa 3**

```
factorial 0 = 1
factorial 1 = 1
factorial 2 = 2
factorial 3 = 2*3
factorial 4 = 2*3*4
factorial 5 = 2*3*4*5
```

**Etapa 4**

```
factorial 0 = 1
factorial 1 = 1
factorial 2 = 2
factorial 3 = factorial 2 * 3
factorial 4 = factorial 3 * 4
factorial 5 = factorial 4 * 5
factorial n = factorial (n-1) * n
```

**Etapa 5**

```
factorial 0 = 1
~~factorial 1 = 1~~
~~factorial 2 = 2~~
~~factorial 3 = 2*3~~
~~factorial 4 = 2*3*4~~
~~factorial 5 = 2*3*4*5~~
factorial n = factorial (n-1) * n
```

**Versão final**

```
-- Dado um número inteiro n,
-- calcular 1*2*...*n
factorial :: Integer -> Integer
factorial 0 = 1
factorial n = factorial (n-1) * n
```

### 2.2 Soma de quadrados

Definir uma função *somaQ* para somar os quadrados de 1 a $n$, isto é calcular $1^2 + 2^2 + \cdots + n^2$.

**Etapas 1 e 2**

```
-- Dado um inteiro n>=0, calcular
-- 1^2 + 2^2 + 3^2 + ... + n^2
somaQ :: Integer -> Integer
```

**Etapa 3**

```
somaQ 0 = 0
somaQ 1 = 1^2
somaQ 2 = 1^2 + 2^2
somaQ 3 = 1^2 + 2^2 + 3^2
somaQ 4 = 1^2 + 2^2 + 3^2 + 4^2
```

**Etapa 4**

```
somaQ 0 = 0
somaQ 1 = 1^2
somaQ 2 = somaQ 1 + 2^2
somaQ 3 = somaQ 2 + 3^2
somaQ 4 = somaQ 3 + 4^2
somaQ n = somaQ (n-1) + n^2
```

**Etapa 5**

```
somaQ 0 = 0
somaQ 1 = 1^2
somaQ 2 = somaQ 1 + 2^2
somaQ 3 = somaQ 2 + 3^2
somaQ 4 = somaQ 3 + 4^2
somaQ n = somaQ (n-1) + n^2
```

**Versão final**

```
-- Dado um inteiro n>=0, calcular
-- 1^2 + 2^2 + 3^2 + ... + n^2
somaQ :: Integer -> Integer
somaQ 0 = 0
somaQ n = somaQ (n-1) + n^2
```

## 2.3 Somar uma lista

Escrever uma definição recursiva da função *sum* do prelúdio que soma os valores numa lista de números.

**Etapas 1 e 2**

```
-- Dada uma lista [x1,x2...xn]
-- calcular x1+x2+...+xn
sum :: Num a => [a] -> a
```

**Etapa 3**

```
sum [] = 0
sum [x1] = x1
sum [x1,x2] = x1+x2
sum [x1,x2,x3] = x1+x2+x3
```

**Etapa 4**

```
sum [] = 0
sum [x1] = x1
sum [x1,x2] = x1 + sum [x2]
sum [x1,x2,x3] = x1 + sum [x2,x3]
sum (x:xs) = x + sum xs
```

**Etapa 5**

```
sum [] = 0
sum [x1] = x1
sum [x1,x2] = x1+sum [x2]
sum [x1,x2,x3] = x1+sum [x2,x3]
sum (x:xs) = x + sum xs
```

**Versão final**

```
-- Dada uma lista [x1,x2...xn]
-- calcular x1+x2+...+xn
sum :: Num a => [a] -> a
sum [] = 0
sum (x:xs) = x + sum xs
```

## 2.4 Último elemento de uma lista

Escrever uma definição recursiva da função *last* do prelúdio que encontra o último elemento de uma lista.

**Etapas 1 e 2**

```
-- Dada uma lista, determinar
-- o seu último elemento
last :: [a] -> a
```

**Etapa 3**

```
last [x1] = x1
last [x1,x2] = x2
last [x1,x2,x3] = x3
```

Note que não existe último elemento da lista vazia `[]`.

**Etapa 4**

```
last [x1] = x1
last [x1,x2] = last [x2]
last [x1,x2,x3] = last [x2,x3]
last (x : xs) = last xs
```

**Etapa 5**

```
last [x1] = x1
last [x1,x2] = last [x2]
last [x1,x2,x3] = last [x2,x3]
last (x : xs) = last xs
```

**Versão final**

```
-- Dada uma lista, determinar
-- o seu último elemento
last :: [a] -> a
last [x1] = x1
last (x : xs) = last xs
```