

Capítulo 7

Recursão e Iteração

1. Considere a seguinte função:

```
def misterio(x, n):  
    if n == 0:  
        return 0  
    else:  
        return x * n + misterio(x, n - 1)
```

- (a) Explique o que é calculado pela função `misterio`.
 - (b) Mostre a evolução do processo gerado pela avaliação de `misterio(2, 3)`.
 - (c) De que tipo é o processo gerado pela função apresentada? Justifique.
 - (d) Se a função apresentada for recursiva de cauda, defina uma nova função recursiva por transformação da primeira de modo a deixar operações adiadas; se for uma função recursiva com operações adiadas, defina uma função recursiva de cauda.
2. Suponha que as operações de multiplicação (*) e potência (**) não existiam em Python e que pretende calcular o quadrado de um número natural. O quadrado de um número natural pode ser calculado como a soma de todos os números ímpares inferiores ao dobro do número

$$n^2 = \sum_{i=1}^n (2i - 1)$$

Note que o dobro de um número também não pode ser calculado recorrendo à operação de multiplicação. Escreva uma função que calcula o quadrado de um número natural utilizando o método descrito.

- (a) Usando recursão com operações adiadas.
- (b) Usando recursão de cauda.

- (c) Usando um processo iterativo.
3. Escreva a função `numero_digitos` que recebe um número inteiro positivo `n`, e devolve o número de dígitos de `n`. As suas funções não podem usar cadeias de caracteres. As suas funções devem validar a correção do argumento. Por exemplo,

```
>>> numero_digitos(9)
1
>>> numero_digitos(1012)
4
```

- (a) Usando recursão com operações adiadas.
 (b) Usando recursão de cauda.
 (c) Usando um processo iterativo.
4. Um número é uma capicua se se lê igualmente da esquerda para a direita e vice-versa. Escreva a função recursiva de cauda `eh_capicua`, que recebe um número inteiro positivo `n`, e devolve verdadeiro se o número for uma capicua e falso caso contrário. A sua função deve utilizar a função `numero_digitos` do exercício anterior. Por exemplo,

```
>>> eh_capicua(12321)
True
>>> eh_capicua(1221)
True
>>> eh_capicua(123210)
False
```

5. O espelho de um número inteiro positivo é o resultado de inverter a ordem de todos os seus algarismos. Escreva a função recursiva de cauda `espelho`, que recebe um número inteiro positivo `n`, não divisível por 10, e devolve o seu espelho. Por exemplo,

```
>>> espelho(391)
193
>>> espelho(45679)
97654
```

6. Considere a função g , definida para inteiros não negativos do seguinte modo:

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

- (a) Escreva uma função recursiva em Python para calcular o valor de $g(n)$.

- (b) Siga o processo gerado por `g(3)`, indicando todos os cálculos efectuados.
- (c) Que tipo de processo é gerado por esta função?
7. Escreva a função recursiva, `calc_soma`, para calcular o valor da soma.

$$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!}$$

para um dado valor de x e de n . A sua função deve ter em atenção que o i -ésimo termo da soma pode ser obtido do termo na posição $i - 1$, multiplicando-o por x/i .

8. Escreva a função recursiva, `maior_inteiro`, que recebe um inteiro positivo, `limite`, e que devolve o maior inteiro (n) tal que $1 + 2 + \dots + n \leq \text{limite}$. Por exemplo,

```
>>> maior_inteiro(6)
3
>>> maior_inteiro(20)
5
```

9. Um número d é divisor de n se o resto da divisão de n por d for 0. Usando recursão de cauda, escreva a função `soma_divisores` que recebe um número inteiro positivo `n`, e que devolve a soma de todos os divisores de `n`.
10. Um número diz-se perfeito se for igual à soma dos seus divisores (não contando o próprio número). Por exemplo, 6 é perfeito porque $1+2+3 = 6$.
- (a) Usando recursão de cauda, escreva a função `perfeito` que recebe como argumento um número inteiro e tem o valor `True` se o seu argumento for um número perfeito e `False` em caso contrário. Não é necessário validar os dados de entrada.
- (b) Usando recursão com operações adiadas e a função `perfeito` da alínea anterior, escreva a função `perfeitos_entre` que recebe dois inteiros positivos e devolve a lista dos números perfeitos entre os seus argumentos, incluindo os seus argumentos. Por exemplo:

```
>>> perfeitos_entre(6, 30)
[6, 28]
```