Ciências ULisboa
Faculdade de Ciências da Universidade de Lisboa
DEPARTAMENTO·DE·INFORMÁTICA

# Folha 11 - Recursão

Python 3.*

## Exercícios

**1.** Considere a seguinte função recursiva:

```
def recursiva(n):
    if n == 0:
        return n
    else:
        return n + recursiva(n-1)
```

**a)** Qual o valor de *recursiva(3)*? E de *recursiva (5)*?

**b)** O que faz esta função?

**c)** O que caracteriza uma função recursiva?

**d)** Escreva uma função iterativa que faça o mesmo que a função *recursiva.*

**2.** Considere a seguinte função que (supostamente) calcula o fatorial de um número inteiro:

```
def fatorial(n):
    """
    Fatorial de um número dado

    Requires: n >=0
    Ensures: fatorial de n
    """

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

**a)** Qual o problema desta função? Corrija-o.

**b)** Simule a execução de *fatorial(5).* Quantas vezes é invocada a função *fatorial* nesse caso?

**c)** Há mais algum caso base? Altere a função de forma a inclui-lo.

**d)** Escreva uma função iterativa que faça o mesmo que a função *fatorial.*

**3.** Considere a função matemática *potencia*. Imagine que pretende escrever uma solução recursiva que recebe dois argumentos, a base e o expoente*:*

**a)** Qual o caso base da função recursiva *potencia?*

**b)** Qual o caso recursivo?

**c)** Escreva a solução recursiva da função *potencia*.

**d)** Defina o contrato da função *potencia.* Use uma *docstring.*

**4.** Considere a seguinte função que calcula, de forma recursiva, a divisão de um número *n* por 2, calculando para isso o número de vezes que se consegue subtrair 2 a *n*.

```
def div2(n):
   if (n == 0):
       return 0
   else:
       return 1 + div2(n - 2)
```

**a)** Simule as seguintes chamadas à função *div2:*

```
div2(16)
div2(6)
div2(7)
```

**b)** Corrija o caso base de forma a evitar o problema encontrado na alínea anterior.

**5.** Considere o algoritmo de Euclides, que calcula o máximo divisor comum de dois números: *mdc(a, b) =mdc(b, a%b) para b != 0 e mdc(a, 0) = a.*

**a)** Implemente o algoritmo de Euclides de forma recursiva.

**b)** Simule a execução de mdc(4,2) e de mdc(66,42).

**6.** O que faz cada uma das seguintes funções?

**a)**
```
def funA(count):
    if count < 1:
        return
    else:
        print("Hello World!")
        funA(count - 1)
```

**b)**

```
def funB(n):
    if n == 1:
        return 3
    else:
        return funB(n-1) + 3
```

**c)**

```
def funC(n):
    if n== 0:
        return 0
    else:
        return n + funC(n-1)
```

**d)**

```
def funD(n):
  if n == 0:
    print('Blastoff!')
  else:
    print(n)
    funD (n - 1)
```

**e)**

```
def funE(n):
    if n < 10:
        return n
    else:
        return n%10 + funE(n//10)
```

**7.** Considere as seguintes funções e exemplos de uso:

```
def f1(n):
    if n == 0:
        return 0
    else:
        return n + f1(n - 1)

def f2(n, result):
    if n == 0:
        return result
    else:
        return f2(n - 1, n + result)

print(f1(3))
print(f2(3, 0))
```

**a)** Simule a execução das duas últimas linhas do programa apresentado.

**b)** O que é *tail-recursion?* Qual das duas funções (*f1* ou *f2)* aplica *tail-recursion?*

**8.** Qual o resultado do seguinte programa?

```
def f2(n, result):
    if n == 0:
        return 0
    else:
        return f2(n - 1, n + result)

print(f2(2, 0))
```

**9.** Indique quais das seguintes afirmações são verdadeiras:

a) As funções recursivas correm mais rápido que as funções não-recursivas
b) As funções recursivas normalmente consomem mais memória que as funções não-recursivas
c) Uma função recursiva pode sempre ser substituida por uma função não-recursiva
d) Em alguns casos, no entanto, usar recursão permite encontrar uma solução simples para um problema, que seria muito difícil de solucionar de outra forma

**10.** Analise os seguinte programas (A e B) e escolha qual das opções abaixo é a correcta:

```
#A
def xfunction(length):
    if length > 1:
        print(length – 1)
        xfunction(length - 1)

xfunction(5)

#B
def xfunction(length):
    while length > 1:
        print (length – 1)
        xfunction(length - 1)

xfunction(5)
```

a) Os dois programas produzem o mesmo resultado: 5 4 3 2 1
b) Os dois programas produzem o mesmo resultado: 1 2 3 4 5
c) Os dois programas produzem o mesmo resultado: 4 3 2 1
d) Os dois programas produzem o mesmo resultado: 1 2 3 4
e) O programa A produz o resultado 4 3 2 1 enquanto o programa B não termina.

# Problemas

**1.** Escreva uma função que calcule o produto de dois inteiros positivos de forma recursiva. Note que a x 1 = a e $a \times (b + 1) = a + a \times b$.

**2.** Elabore uma versão iterativa do exercício anterior.

**3.** Escreva uma função que calcule a potência de base *a* e expoente *b*, onde *b* é um inteiro maior ou igual a zero, de forma recursiva. Utilize a função desenvolvida no exercício 1.

**4.** Elabore uma versão distinta da função pedida no exercício anterior que seja muito mais rápida a calcular potências, sempre de forma recursiva. Note que $a^b = a^{b/2} \times a^{b/2}$ caso *b* seja um número par (e se fosse ímpar?).

**5.** Escreva uma função que lide com potências de expoente inteiro (positivo, zero e negativo). Recorra à função anterior e utilize uma expressão condicional.

**6.** Escreva uma função que calcule o n-ésimo número da sucessão de Fibonacci. Relembre que *fib(0) = fib(1) = 1* e *fib(n) = fib(n-1)+fib(n-2).*

**7.** Escreve uma função recursiva que recebe um inteiro positivo *n* e calcule o n-ésimo valor da seguinte sequência: 1, 3, 5, 7, 15, 31, 63, 127, .....

**8.** Escreva uma função que determine, de forma recursiva, o comprimento de uma *string*.

**9.** Escreva uma função que determine, de forma recursiva, se uma cadeia de caracteres (*string*) é prefixo de outra. Por exemplo: “abc” é prefixo de “abcdef” mas não de “acbdef”.

**10.** Escreva uma função que determine, de forma recursiva, se uma *string* é subcadeia de outra (é substring). Utilize a função anterior. Por exemplo, “abc" é substring de “ababcdef” mas não de “abacbdef”.