Introdução à Programação em Python

(2) Exercícios de programação recursiva

Carlos Caleiro, Jaime Ramos

Dep. Matemática, IST - 2016

(versão de 21 de Fevereiro de 2019)

Os exercícios a seguir listados devem ser resolvidos recorrendo a funções *definidas por recursão*. Todas as funções devem ser testadas para garantir que funcionam como é esperado.

1. Defina a função `soma_nat` que recebe como argumento um número natural n e devolve a soma de todos os números naturais até n.

```
In [2]: soma_nat(5)
Out[2]: 15
```

2. Defina a função `div` que recebe como argumentos dois números naturais m e n e devolve o resultado da divisão inteira de m por n. Neste exercício não pode recorrer às operações aritméticas de multiplicação, divisão e resto da divisão inteira.

```
In [4]: div(7,2)
Out[4]: 3

In [5]: div(3,4)
Out[5]: 0
```

3. Defina a função `prim_alg` que recebe como argumento um número natural e devolve o primeiro algarismo (o mais significativo) na representação decimal de n.

```
In [8]: prim_alg(5629)
Out[8]: 5

In [9]: prim_alg(7)
Out[9]: 7
```

4. Defina a função `media_digitos` que recebe como argumento um número natural e devolve a média dos seus digitos.

```
In [34]: media_digitos(1234)
Out[34]: 2.5
```

5. Defina a função `num_perf` que recebe como argumento um número inteiro positivo e devolve `True` se esse número for um número perfeito e `False` em caso contrário. Recorde que um *número perfeito* é um número natural que é igual à soma de todos os seus divisores próprios, isto é, a soma de todos os divisores excluindo o próprio número. Pode, se assim o entender, definir funções auxiliares.

```
In [7]: num_perf(6)
Out[7]: True

In [16]: num_perf(5)
Out[16]: False
```

6. Considere a função $f : \mathbb{N} \to \mathbb{N}$ tal que

$$f(x) = \begin{cases} x/2 & \text{se } x \text{ for um número par} \\ 3x+1 & \text{caso contrário} \end{cases}$$

Defina a função `num_it` que recebe como argumento um número natural n e devolve o número de vezes que f tem de ser aplicada (recursivamente) a n até se atingir o número 1, i.e., devolve o número k tal que

$$\underbrace{f(f(\ldots f(n)))}_{k-\text{vezes}} = 1.$$

A *conjectura de Collatz* afirma que tal programa termina sempre, facto de que não se conhece nenhuma prova ou contraexemplo.

```
In [19]: num_it(5)
Out[19]: 5

In [20]: num_it(21)
Out[20]: 7

In [21]: num_it(11)
Out[21]: 14
```

7. Defina a função `comb` que recebe como argumentos dois naturais m e q, com $m \geq q$, tal que $\mathtt{comb}(m, q) = \binom{m}{q}$, as combinações de m, q a q.

Recorde que as combinações satisfazem a relação

$$\binom{m}{q} = \binom{m-1}{q-1} + \binom{m-1}{q}$$

para $0 < q < m$.

Claro que se soubermos que $\binom{m}{q} = \frac{m!}{q!(m-q)!}$ podemos definir esta função facilmente. No entanto, não é esse o objectivo aqui.

Note ainda que a solução óbvia para este problema, embora simples, é extremamente ineficiente. Tente perceber porquê.

```
In [26]: comb(3,2)
Out[26]: 3
```

8. Defina a função `quadrados` que recebe como argumento um número natural n e devolve a lista dos n primeiros quadrados perfeitos.

```
In [28]: quadrados(6)
Out[28]: [1, 4, 9, 16, 25, 36]
```

9. Defina a função `quadrados_inv` que recebe como argumento um número natural n e devolve a lista dos quadrados perfeitos até n, por ordem decrescente.

```
In [33]: quadrados_inv(6)
Out[33]: [36, 25, 16, 9, 4, 1]
```

10. Defina a função `prod_lista` que recebe como argumento uma lista de inteiros e devolve o produto dos seus elementos.

```
In [3]: prod_lista([1,2,3,4,5,6])
Out[3]: 720
```

11. Defina a função `contem_parQ` que recebe como argumento uma lista de números inteiros w e devolve `True` se w contém um número par e `False` em caso contrário.

```
In [5]: contem_parQ([2,3,1,2,3,4])
Out[5]: True

In [6]: contem_parQ([1,3,5,7])
Out[6]: False

In [7]: contem_parQ([])
Out[7]: False
```

12. Defina a função `todos_imparesQ` que recebe como argumento uma lista de números inteiros w e devolve `True` se w contém apenas números ímpares e `False` em caso contrário.

```
In [18]: todos_imparesQ([1,3,5,7])
Out[18]: True

In [19]: todos_imparesQ([])
Out[19]: True

In [20]: todos_imparesQ([1,2,3,4,5])
Out[20]: False
```

13. Defina a função `pertenceQ` que recebe como argumentos uma lista de números inteiros w e um número inteiro n e devolve `True` se n ocorre em w e `False` em caso contrário.

```
In [9]: pertenceQ([1,2,3],1)
Out[9]: True

In [10]: pertenceQ([1,2,3],2)
Out[10]: True

In [11]: pertenceQ([1,2,3],3)
Out[11]: True

In [12]: pertenceQ([1,2,3],4)
Out[12]: False
```

14. Defina a função `negpos` que recebe como argumento uma lista de números inteiros w e devolve a diferença entre o número de números positivos e o número de números negativos de w.

```
In [2]: negpos([1,-2,-1,4,6,3])
Out[2]: 2

In [3]: negpos([-1,-2,-3,4])
Out[3]: -2

In [4]: negpos([1,-1])
Out[4]: 0
```

15. Defina a função `indices_impar` que recebe como argumento uma lista de números inteiros w e devolve a lista dos elementos de w em posições de índice ímpar. Recorde que a primeira posição de uma lista tem índice 0, que é um número par.

```
In [20]: indices_impar([0,1,2,3,4,5,6])
Out[20]: [1, 3, 5]

In [21]: indices_impar([0,1,2,3,4,5])
Out[21]: [1, 3, 5]

In [22]: indices_impar([])
Out[22]: []

In [24]: indices_impar([1,2])
Out[24]: [2]
```

16. Defina a função `escolhe_pares` que recebe como argumento uma lista de números inteiros w e devolve a lista dos elementos pares de w.

```
In [26]: escolhe_pares([1,2,3,4,4,2,6,8,9])
Out[26]: [2, 4, 4, 2, 6, 8]

In [27]: escolhe_pares([])
Out[27]: []

In [28]: escolhe_pares([1,3,5,7,9])
Out[28]: []
```

17. Defina a função `retira_negativos` que recebe como argumento uma lista de números inteiros w e devolve a lista resultante de retirar todos os números negativos de w.

```
In [31]: retira_negativos([1,-2,-3,7,0])
Out[31]: [1, 7, 0]
```

18. Defina a função `supremo` que recebe como argumento uma lista de números inteiros w e devolve o supremo de w. Note que o supremo do conjunto vazio é $-\infty$. Para representar ∞ em Python pode recorrer à seguinte extensão:

```
In [62]: from math import inf
```

```
In [37]: supremo([1,2,3,-2,5,-7])
Out[37]: 5

In [35]: supremo([])
Out[35]: -inf

In [36]: supremo([3,2,1])
Out[36]: 3
```

19. Defina a função `conta` que recebe como argumentos uma lista de números inteiros w e um número inteiro k e devolve o número de vezes que k ocorre em w.

```
In [44]: conta([1,2,3,2,1,2],2)
Out[44]: 3

In [45]: conta([1,2,3,2,1,2],4)
Out[45]: 0

In [46]: conta([],5)
Out[46]: 0
```

20. Defina a função `lposicoes` que recebe como argumentos uma lista de números inteiros w e um número inteiro k e devolve a lista das posições em que k ocorre em w.

```
In [49]: lposicoes([1,2,3,4,2,2],2)
Out[49]: [1, 4, 5]

In [50]: lposicoes([1,2,3,4,2,2],7)
Out[50]: []

In [51]: lposicoes([],3)
Out[51]: []
```

21. Defina a função `pos_max` que recebe como argumento uma lista de números inteiros e devolve o índice da primeira posição onde ocorre o máximo da lista. No caso da lista vazia, devolve -1.

```
In [37]: pos_max([1,2,3,3,2,1,3])
Out[37]: 2
```

22. Defina a função `car_pares` que recebe como argumento uma lista de números inteiros w e devolve uma lista com `True` nas posições onde ocorre em w um número par e `False` nas outras.

```
In [2]: car_par([2,3,4,3,2,2])
Out[2]: [True, False, True, False, True, True]

In [3]: car_par([3,3,3])
Out[3]: [False, False, False]

In [4]: car_par([])
Out[4]: []
```

23. Defina a função `apaga1` que recebe como argumentos uma lista de números inteiros w e um número inteiro k e devolve a lista que resulta de apagar de w a primeira ocorrência de k (caso exista).

```
In [9]: apaga1([1,2,3,4,3,2,1],3)
Out[9]: [1, 2, 4, 3, 2, 1]

In [10]: apaga1([1,2,3,4,3,2,1],1)
Out[10]: [2, 3, 4, 3, 2, 1]

In [11]: apaga1([1,2,3,4,3,2,1],5)
Out[11]: [1, 2, 3, 4, 3, 2, 1]

In [12]: apaga1([],3)
Out[12]: []
```

24. Defina a função `apaga` que recebe como argumentos uma lista de números inteiros w e um número inteiro k e devolve a lista que resulta de apagar de w todas as ocorrências de k.

```
In [14]: apaga([1,2,3,4,3,2,1],3)
Out[14]: [1, 2, 4, 2, 1]

In [15]: apaga([1,2,3,4,3,2,1],5)
Out[15]: [1, 2, 3, 4, 3, 2, 1]

In [16]: apaga([],3)
Out[16]: []
```

25. Defina a função `seleccao` que recebe como argumentos uma lista de números inteiros w e um predicado p e devolve a lista de todos os elementos de w que verificam p.

```
In [28]: seleccao([],primoQ)
Out[28]: []

In [31]: seleccao([1,2,3,4,5,6,7,8,9,10],primoQ)
Out[31]: [2, 3, 5, 7]
```

26. Defina a função `mapeia` que recebe como argumentos uma função f e uma lista w e devolve a lista que resulta de aplicar f a cada um dos elementos de w.

```
In [34]: def suc(x):
             return x+1
```

```
In [37]: mapeia(suc,[0,1,2,3,4,5])
Out[37]: [1, 2, 3, 4, 5, 6]

In [39]: mapeia(suc,[])
Out[39]: []
```

27. Uma lista s diz-se sufixo de uma lista w se s for um segmento do fim de w. Defina a função `sufixoQ` que recebe como argumentos duas listas s e w e devolve `True` se s for sufixo de w e `False` em caso contrário.

```
In [45]: sufixoQ([3,4],[1,2,3,4])
Out[45]: True

In [46]: sufixoQ([3,4],[1,2,3,4,5])
Out[46]: False

In [47]: sufixoQ([1,2,3],[1,2])
Out[47]: False

In [48]: sufixoQ([],[1])
Out[48]: True

In [49]: sufixoQ([1,2,3],[1,2,3])
Out[49]: True
```

28. Defina a função `temPrimoQ` que recebe como argumento uma lista de listas de números inteiros w e devolve `True` se alguma das sublistas w tem um número primo e `False` em caso contrário.

```
In [26]: temPrimoQ([[4,4,4,4],[5,4,6,7],[2,4,3]])
Out[26]: True

In [27]: temPrimoQ([[4,4,4,4],[4,4,4],[],[4]])
Out[27]: False
```

29. Defina a função `inverteLista` que recebe como argumento uma lista w e devolve a mesma lista mas invertida.

```
In [82]: inverteLista([1,2,3,4,5])
Out[82]: [5, 4, 3, 2, 1]

In [83]: inverteLista([])
Out[83]: []
```

30. Defina a função `lista_igualQ` que recebe como argumentos duas lista e devolve `True` se as listas forem iguais e `False` em caso contrário. Não pode usar a comparação == entre listas.

```
In [86]: lista_igualQ([1,2,3],[1,2,3])
Out[86]: True

In [87]: lista_igualQ([1,2,3],[1,3,3])
Out[87]: False

In [88]: lista_igualQ([1,2],[1,2,3])
Out[88]: False

In [89]: lista_igualQ([1,2,3],[1,2])
Out[89]: False
```

31. Defina a função `sup_listas1` que recebe como argumento uma lista de listas de números inteiros w e devolve o supremo de w.

```
In [99]: sup_listas1([[1,2,3],[2,3,4],[2]])
Out[99]: 4

In [100]: sup_listas1([[7,3,2],[],[1,2,3]])
Out[100]: 7
```

Os exercícios seguintes são de um nível de dificuldade mais elevado. Caso considere necessário, utilize algumas das funções definidas anteriormente.

32. Defina a função `sup_listas2` que recebe como argumento uma lista de listas de números inteiros w e devolve uma lista de comprimento igual a w contendo em cada posição o supremo de cada uma das listas de w na posição correspondente.

```
In [104]: sup_listas2([[1,2,3],[4,3,2],[],[7,7,7,7]])
Out[104]: [3, 4, -inf, 7]
```

33. Defina a função `permutacao` que recebe como argumentos duas listas $w1$ e $w2$ e devolve `True` se $w1$ for uma permutação de $w2$ e `False` em caso contrário.

```
In [109]: permutacao([1,2,3],[1,2,3])
Out[109]: True

In [110]: permutacao([1,2,3],[2,3,1])
Out[110]: True

In [111]: permutacao([1,1,1,2,3],[1,2,3])
Out[111]: False
```

34. Defina a função `intercala` que recebe como argumentos duas listas $w1$ e $w2$ e devolve a lista resultante de intercalar os elementos de $w1$ com os de $w2$.

```
In [115]: intercala([1,2,3],[4,5,6])
Out[115]: [1, 4, 2, 5, 3, 6]

In [116]: intercala([1,3,5,7,9],[2,4])
Out[116]: [1, 2, 3, 4, 5, 7, 9]
```

35. Defina a função `indPrimos` que recebe como argumento uma lista de listas de números inteiros $w = \{w_1, \ldots, w_k\}$ e devolve a lista $r = \{r_1, \ldots, r_n\}$, em que r_i é uma lista composta pelos índices das posições dos números primos em w_i.

```
In [144]: indPrimos1([[1,2,3,4,5],[11,12,13,14,15],[],[22,33,44]])
Out[144]: [[1, 2, 4], [0, 2], [], []]
```

36. Defina a função `separaMult3e5` que recebe como argumento uma lista de números inteiros w e devolve um par (uma lista) formado por duas listas: a dos múltiplos de 5 que ocorrem em w e a dos múltiplos de 3 que ocorrem em w.

```
In [146]: mult3e5([1,2,3,4,5,9,15])
Out[146]: [[5, 15], [3, 9, 15]]

In [147]: mult3e5([1,2,3,4,6,7,8,9])
Out[147]: [[], [3, 6, 9]]
```

37. Defina a função `max_soma_linha` que recebe como argumento uma matriz de números inteiros e devolve o índice da primeira linha cuja a soma dos elementos é máxima.

Sugestão: Comece por definir uma função recursiva para somar os elementos de uma linha da matriz.

```
In [2]: m=[[1,2,3],[11,11,11],[6,7,8],[10,11,12]]
```

```
In [3]: max_soma_linha(m)
Out[3]: 1
```

38. Defina a função `potencia` que recebe como argumento um dígito k (que não zero) e devolve o menor natural n tal que 2^n começa por k.

```
In [9]: potencia(2)
Out[9]: 1

In [10]: potencia(3)
Out[10]: 5
```

39. Defina a função `repete` que recebe como argumento uma lista w e devolve uma lista em que o primeiro elemento de w aparece repetido 1 vez, o segundo elemento aparece repetido duas vezes e assim sucessivamente.

```
In [29]: repete([1,2,3])
Out[29]: [1, 2, 2, 3, 3, 3]
```

40. Defina versões iterativas das funções definidas nos exercícios **1**, **7**, **8**, **14**, **15**, **16**, **17**, **18**, **19** e **20**.

Exercícios complementares

1. Defina em Python uma função recursiva `g1` que recebe como argumento uma lista de listas de números inteiros w e devolve o número de elementos pares que existem nas sublistas de w.
Por exemplo, no caso da lista `[[2,4,3,1],[3,5,7],[],[8,0,6]]` a função deve devolver 5. Pode, se achar conveniente, definir funções auxiliares desde que sejam funções recursivas.

2. Defina em Python uma função recursiva `g2` que recebe como argumento uma lista de listas de números inteiros w e devolve `True` se em todas as sublistas de w existir um número positivo.
Por exemplo, no caso da lista `[[2,4,3,1],[3,-5,-7],[],[8,0,-6]]` a função deve devolver `False` porque em `[]` não existe nenhum número positivo. Pode, se achar conveniente, definir funções auxiliares desde que sejam funções recursivas.

3. Defina em Python uma função recursiva `g3` que recebe como argumento uma lista de listas de números inteiros w e devolve `True` se alguma das listas em w tiver mais números pares do que ímpares e `False` em caso contrário.
Por exemplo, no caso da lista `[[2,4,3,1],[3,5,7],[8,1,6]]` a função deve devolver `True` pois a lista `[8,1,6]` tem mais pares que ímpares. Pode, se achar conveniente, definir funções auxiliares desde que sejam funções recursivas.

4. Defina em Python uma função recursiva `g4` que recebe como argumento uma lista de listas de números inteiros w e devolve `True` se todas as listas em w verificam a propriedade de mais de metade dos seus elementos serem 0 e `False` em caso contrário.
Por exemplo, no caso da lista `[[0,0,3,0],[0,5,0],[0,0,0]]` a função deve devolver `True` pois em todas as listas mais de metade dos elementos são 0. Pode, se achar conveniente, definir funções auxiliares desde que sejam funções recursivas.

5. Defina em Python uma função recursiva `g5` que recebe como argumento uma lista de listas de números inteiros w e devolve `True` se cada lista em `w` tiver igual número de elementos positivos e negativos, e `False` em caso contrário.
Por exemplo, no caso da lista `[[2,4,-3,-1],[-3,0,7],[-8,6]]` a função deve devolver `True` pois todas as listas têm igual número de elementos positivos e negativos. Pode, se achar conveniente, definir funções auxiliares desde que sejam funções recursivas.

6. Defina em Python uma função recursiva `g6` que recebe como argumento uma lista de listas de números inteiros w e devolve o número de listas em w ordenadas por ordem crescente em sentido lato, ou seja, em que cada elemento é menor ou igual que o seguinte.
Por exemplo, no caso da lista `[[2,2,3,0],[1,2,5,4],[2,4,4]]`a função deve devolver 1 pois só a terceira lista está ordenada. Pode, se achar conveniente, definir funções auxiliares desde que sejam funções recursivas.