www.estudar.com.vc

# Lista de Exercícios

## P2 2016

Programação em Python Poli

USP

## 1. Tema

Um polinômio de uma variável pode ser representado pela lista de seus coeficientes. Por exemplo, o polinômio $3x^4 + 2x^2 + x + 5$ é representado pela lista $[5, 1, 2, 0, 3]$. Nesta representação:

**I.** O tamanho da lista é o grau do polinômio mais $1$.
**II.** O número na posição $0$ é o coeficiente do monômio de grau $0$, o número na posição $1$ é o coeficiente do monômio de grau $1$. Em geral, o número na posição $k$ é o coeficiente do monômio $x^k$.
**III.** O último elemento da lista é sempre diferente de zero. Assim, o polinômio zero é representado pela lista vazia []. Lembre-se também de que o grau do polinômio zero é $-1$ por convenção, assim o item $1$ acima também se aplica ao polinômio zero.

Esta questão consiste na implementação de $2$ funções:

**a.** Escreva uma função $calcula\_derivada$ que recebe uma lista que representa um polinômio $p$ e devolve a lista que representa sua derivada $p'$.

```
def calcula_derivada(p):
    '''list -> list

    Recebe uma lista p que representa um polinômio e
    devolve a lista que representa sua derivada.

    Exemplos:
    calcula_derivada([1]) devolve [].
```

```
    calcula_derivada([1,2,3]) devolve [2,6].
    '''
```

**b.** Escreva uma função *calcula_polinomio* que recebe a lista que representa um polinômio $p$ e um número real $x$ e devolve $p(x)$.

```
def calcula_polinomio(p, x):
    '''list, float -> float

    Recebe uma lista p que representa um polinômio e um
    número real x e devolve o valor do polinômio
    calculado em x.

    Exemplos:
    calcula_polinomio([], 5) devolve 0.
    calcula_polinomio([1,2,3]) devolve 17.
    '''
```

## 2. Tema

Sequências de DNA são *strings* de caracteres $A, C, G$ e $T$. Um alinhamento de duas sequências de DNA $s$ e $t$ é obtido adicionando-se *gaps*, representados pelo caractere '_' (*underscore*), a $s$ e $t$, de modo que as *strings* resultantes tenham o mesmo tamanho. Por exemplo, se $s = TCGTAC$ e $t = ATCG$, então um alinhamento pode ser:

$$s' = T\_CGTAC$$

$$t' = ATCG\_\_\_$$

Dados números inteiros não-negativos $m, d$ e $g$, a pontuação de um alinhamento é calculada da seguinte forma: duas letras iguais alinhadas contam $m$ pontos, duas letras diferentes alinhadas contam $-d$ pontos e uma letra alinhada com um gap ou dois gaps alinhados contam $-g$ pontos. Assim, se $m = 5$, $d = 5$ e $g = 3$, a pontuação do alinhamento acima é:

$$-5 - 3 + 5 + 5 - 3 - 3 - 3 = -7$$

Uma tarefa importante em biologia computacional é calcular um alinhamento entre duas sequências de DNA que tenha pontuação máxima. Nesta questão você dará um primeiro passo para resolver este problema, escrevendo uma função que, dados números inteiros não-negativos $m, d$ e $g$ e duas *strings* de mesmo tamanho contendo apenas as letras $A, C, G$ e $T$ e o caractere *underscore* '_', devolve a pontuação do alinhamento.

```
def pontuacao(m, d, g, s, t):
    '''int, int, int, str, str -> int

    Recebe inteiros não-negativos m, d e g e duas strings
    s e t de
    mesmo tamanho contendo apenas os caracteres A, C, G,
    T e _ e
    devolve a pontuacao do alinhamento representado
    pelas strings.

    Exemplos:
```

```
    pontuacao(5, 5, 3, 'T_CGTAC', 'ATCG___') devolve -
    7.
    '''
```

## 3. TEMA

Considere um jogo de *PacMan* no qual o labirinto é representado por uma matriz com os seguintes caracteres:

**I.** '+' representa uma parede;
**II.** '.' representa um *pac-dot* (a pastilha que quando comida pelo *PacMan* dá 1 ponto a ele);
**III.** Um espaço em branco representa uma posição vazia.

As seguintes regras devem ser respeitadas no jogo:

**I.** Paredes não podem ser atravessadas;
**II.** Se o *PacMan* chegar numa posição com um *pac-dot*, ele ganha 1 ponto e no lugar do *pac-dot* é colocado um espaço em branco;
**III.** Se o *PacMan* se chocar com um fantasma, ele é comido. Se havia um *pac-dot* naquela posição, ele não é comido;
**IV.** O labirinto é cíclico.

Escreva a função $movimentaPacMan$ com protótipo a seguir. A função recebe uma matriz representando o labirinto, uma lista que representa o *PacMan* e uma matriz com as posições dos fantasmas, realiza o movimento do *PacMan* de acordo com as regras do jogo, atualizando o

labirinto e a lista que representa o *PacMan*, e devolve *True* se o *PacMan* não morreu após o movimento e *False* caso contrário.

A lista que representa o *PacMan* tem o formato $[linha, coluna, direcao, pontos]$, onde:
**I.** $linha$ é um inteiro representando a linha da matriz onde o *PacMan* se encontra;
**II.** $coluna$ é um inteiro representando a coluna da matriz onde o *PacMan* se encontra;
**III.** $direcao$ é um inteiro representando para onde o *PacMan* deve se mover (esquerda = 0, direita = 1, cima = 2 e baixo = 3);
**IV.** $pontos$ é um inteiro representando a quantidade de pontos do *PacMan* na partida.

A matriz que representa os fantasmas contém, para cada fantasma, sua posição no labirinto, no formato:

$$[\,[linha, coluna], [linha, coluna], \ldots\,]$$

De modo que cada linha da matriz contém a posição de um dos fantasmas.

```
def movimentaPacMan(lab, pacman, fantasmas):
    '''(matriz, lista, matriz) -> bool

        Recebe uma matriz lab que representa um labirinto,
        uma lista pacman que representa o PacMan e que contém
        a direção na qual ele deve ser movido e uma matriz
        fantasmas que contém as posições dos fantasmas no
```

```
labirinto, uma por linha. A função realiza o
movimento do PacMan, atualizando a lista que o
representa e o labirinto, e devolve True se o PacMan
não morreu após seu movimento; False caso contrário.
'''
```

## Gabarito

**1.**

**a.**
```
def calcula_derivada(p)
    derivada = []
    for i in range(1, len(p)):
      derivada.append(i*p[i])
    return derivada
```

**b.**
```
def calcula_polinomio(p, x)
    resultado = 0
    for i in range(len(p)):
      resultado += p[i]*x**i
    return resultado
```

**2.**
```
def pontuacao(m, d, g, s, t):
    pontos = 0
    for i in range(len(s)):
        if s[i] == '_' or t[i] == '_':
            pontos -= g
        elif s[i] == t[i]:
            pontos += m
        else:
            pontos -= d
    return pontos
```

**3.**
```
def movimentaPacMan(lab, pacman, fantasmas):
    i, j = pacman[0], pacman[1]
    direcao = pacman[2]
```

```
if direcao == 0: #esquerda
    j -= 1
elif direcao == 1:
    j += 1
elif direcao == 2:
    i -= 1
elif direcao == 3:
    i += 1

i %= len(lab)
j %= len(lab[0])

if lab[i][j] != '+':
    for fantasma in fantasmas:
        if fantasma[0] == pacman[0] and fantasma[1]
        == pacman[1]:
            return False

    if lab[i][j] == '.':
        pacman[3] += 1
        lab[i][j] = ' '

        pacman[0] = i
        pacman[1] = j

return True
```