MAC0110 Introdução à Computação

Bacharelado em Estatística, Matemática e Matemática Aplicada

Terceira Prova – 28 de junho de 2018

Nome: _____

Assinatura: _____

Nº USP: _____

Questão	Valor	Nota
1	2,0	
2	2,0	
3	3,0	
4	3,0	
Total	10,0	

Instruções:

1. Não destaque as folhas deste caderno. A prova pode ser feita a lápis.
2. A prova consta de 4 questões. Verifique antes de começar a prova se o seu caderno está completo.
3. As questões podem ser resolvidas em qualquer página. Ao escrever uma solução (ou parte dela) em página diferente do enunciado, escreva QUESTÃO X em letras ENORMES junto da solução.
4. As soluções devem ser em Python. **Você pode usar apenas recursos de Python vistos em aula**.
5. Você pode definir funções auxiliares e usá-las à vontade.
6. Cuidado com a legibilidade e, principalmente, com a TABULAÇÃO.
7. As soluções não precisam verificar consistência de dados.
8. Não é permitido o uso de folhas avulsas para rascunho, a consulta a livros, apontamentos, colegas ou equipamentos eletrônicos.
9. Desligue o seu celular e qualquer equipamento que possa perturbar o andamento da prova;

DURAÇÃO DA PROVA: 100 minutos

Questão 1 (2 pontos)

Suponha que o Python tenha lido as seguintes funções:

```
def a(x, y):
    for i in x:
        x[i] += y
    return len(x)
```

```
def b(x, y):
    x[0] += 'i'
    y = [2]
    print x + y
```

Suponha ainda que fizemos as seguintes atribuições:

```
In [1]: p = ['x', 1, 2.0]
In [2]: q = [-1, [5.1], '6']
In [3]: x = 'ABCD'
In [4]: y = [ p, q, x ]
In [5]: z = {0:'p', 1:'a', 2:'t', 3:'o'}
```

A seguir está uma transcrição de uma seção do Python Shell. Complete as lacunas (tipo e/ou o valor) do resultado da expressão correspondente. Se ocorrer um erro, escreva apenas `ERRO`.

```
In [6]: len(y)
```

tipo: _____ valor: _____

```
In [7]: y[-1][2]
```

tipo: _____ valor: _____

```
In [8]: y[-2][1]
```

tipo: _____ valor: _____

```
In [9]: y[0][2]
```

tipo: _____ valor: _____

```
In [10]: p[1] += 3.1
In [11]: y[0]
```

tipo: _____ valor: _____

```
In [12]: x = 'ijk'
In [13]: y[2]
```

tipo: _____ valor: _____

```
In [14]: y[1][0] = 'jk'
In [15]: q
```

valor: _____

```
In [16]: z[1]
```

valor: _____

```
In [17]: z['t']
```

valor: _____

```
In [18]: j = a(z, 'i')
In [19]: j
```

valor: _____

```
In [20]: z[3]
```

valor: _____

```
In [21]: p = ['x', 1, 2.0]
In [22]: q = [-1, [5.1], '6']
In [23]: j = b(p, q)
In [24]: j
```

valor: _____

```
In 25]: p
```

valor: _____

```
In [26]: q
```

valor: _____

Questão 2 (2 pontos)

Para o trecho de código a seguir indique a saída (= resultado de `print()`).

```
def main():
    nusp = input("Digite o seu número USP: ")
    x = int(nusp[-1])
    y = [ [ int(nusp[0]), int(nusp[1]) ],
          [ int(nusp[2]), int(nusp[3]) ],
          [ int(nusp[4]), int(nusp[5]) ] ]
    print("nusp =", nusp, 'x =', x)
    print("y =", y)
    z = [[10, 20, 30], [40, 50, 60]]
    b = f(x, y, z)
    print("b: ", b, "x: ", x)
    print("y: ", y)
    print("z: ", z)
```

```
def f(x, y, z):
    m = len(y)
    n = len(y[0])
    for i in range(m):
        x += y[i][0]
        if i % 2 == 0:
            print("i: ", i, "x: ", x )
        for j in range(n):
            y[i][j] += z[j][i]
    z[1] = ['a', 'b']
    return x

main()
```

Escreva abaixo a saída do programa:

Questão 3 (3 pontos)

Diremos que uma string é **palíndroma** se lida tanto da esquerda para a direita como da direita para a esquerda é a mesma. Por exemplo, a string `"romametemamor"`, inspirada em `"roma me tem amor"`, é palíndroma.

Escreva um programa que leia uma string `s` e imprima uma substring palíndroma de `s` de maior comprimento; no caso de haver mais de uma substring nessas condições, qualquer uma pode ser impressa. Por exemplo, para:

- `s = "xyz"`, o programa deve imprimir `"x"` ou `"y"` ou `"z"`.
- `s = "xyzabcdcbaijk"`, o programa deve imprimir `"abcdcba"`.
- `s = "youwillborroworrobthehat"` (inspirada em `"you will borrow or rob the hat"`), o programa deve imprimir `"borroworrob"`.
- `s = "vamoscomeraposasopadecebola"` (inspirada em `"vamos comer apos a sopa de cebola"`), o programa deve imprimir `"aposasopa"`.
- `s = "vimosabandapassardasacadadacasaamarela"` (inspirada em `"vimos a banda passar da sacada da casa amarela"`), o programa deve imprimir `"asacadadacasa"`.

Questão 4 (3 pontos)

Esta questão consiste na implementação de duas funções.

Por um **monte de areia** entenderemos uma coleção de grãos de areia distribuídos entre as casas de um tabuleiro retangular. Um monte de areia é representado por uma matriz em que cada posição `[i][j]` contém o número de grãos na casa. A matriz a seguir representa um monte de areia.

	0	1	2	3
0	1	1	1	0
1	1	12	2	1
2	1	1	2	4

Nessa questão, considere a **vizinhança** de uma casa como o número de casas vizinhas. Na matriz a seguir o número em cada casa indica o número de vizinhos da casa.

	0	1	2	3
0	3	5	5	3
1	5	8	8	5
2	3	5	5	3

Se o número de grãos em uma casa é maior ou igual a quantidade de vizinhos que ela possui, então dizemos que a casa está **instável** . Um monte de areia evolui ao longo do tempo da seguinte maneira:

- A cada instante, cada casa instável deve espalhar seus grãos entre as suas casas vizinhas, **apenas 1 grão** para cada casa vizinha.
- Todas as casas instáveis devem espalhar seus grãos **simultaneamente**.

No monte de areia mostrado abaixo à esquerda a casa `[1][1]` está instável pois ela possui 12 grãos e tem 8 vizinhos. De maneira semelhante, a casa `[2][3]` está instável pois possui 4 grãos e tem 3 vizinhos. Após o espalhamento simultâneo dos grãos das casas instáveis obtemos o monte de areia mostrado abaixo à direita. Nesse novo monte de areia todas as casas estão estáveis.

	0	1	2	3
0	1	1	1	0
1	1	12	2	1
2	1	1	2	4

	0	1	2	3
0	2	2	2	0
1	2	4	4	2
2	2	2	4	1

(a) (vale 1,0 ponto)

Escreva uma função

```
def vizinhos(i, j, nlin, ncol):
    ''' (int, int, int, int) -> list '''
```

que recebe como parâmetro a posição `[i][j]` de uma matriz de dimensão `nlin` $\times$ `ncol` e cria e retorna uma lista com as posições vizinhas de `[i][j]`.

Exemplos de execuções da função no Python Shell:

```
>>> vizinhos(0, 0, 3, 4)
[[0, 1], [1, 0], [1, 1]]
>>> vizinhos(2, 2, 4, 4)
[[1, 1], [1, 2], [1, 3], [2, 1], [2, 3], [3, 1], [3, 2], [3, 3]]
>>> vizinhos(0, 2, 4, 4)
[[0, 1], [0, 3], [1, 1], [1, 2], [1, 3]]
>>> vizinhos(3, 2, 4, 4)
[[2, 1], [2, 2], [2, 3], [3, 1], [3, 3]]
>>> vizinhos(0, 3, 4, 4)
[[0, 2], [1, 2], [1, 3]]
>>> vizinhos(0, 0, 1, 1)
[]
>>>
```

Para escrever a função pedida no **item (b)** você **deve** usar a função `clone()` a seguir **sem escrevê-la**. Suponha que lhe é dada uma função de protótipo

```
def clone(mat):
    ''' (matriz) -> matriz '''
```

que recebe como parâmetro uma matriz `mat` e **cria** e **retorna** um clone da matriz `mat`.

No **item (b)** você **deve** utilizar ainda a função `vizinhos` do **item (a) sem reescrevê-la**. Você pode usar a função do item (a) mesmo que não a tenha feito.

(b) (vale 2,0 pontos)

Escreva uma função

```
def espalhe(monte):
    ''' (matriz) -> matriz '''
```

que recebe uma matriz `monte` que representa um monte de areia e **cria** e **retorna** uma matriz `novo_monte` que representa o monte de areia resultante após todas as casas instáveis de `monte` espalharem os seus grãos entre os seus vizinhos.

Por exemplo, se `monte` é a matriz

	0	1	2	3
0	5	5	0	0
1	1	5	0	1

então a função deve retornar a matriz

	0	1	2	3
0	4	2	2	0
1	4	2	2	1

Atenção, a moldura e os índices não fazem parte das matrizes.