MAC2166 - Introducao a Computacao - 2023S1 **Avaliacao P2**

0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9

Utilize caneta azul ou preta e preencha completamente a quadrícula.
Exemplo: ■. **Não use ☒.**

Turma: (somente um número; consulte a pessoa responsável se não souber)

4 5 6 7 8 9 10 11 12

⟵ Marque as quadrículas ao lado para formar o seu número USP e escreva seu nome completo em letra legível na linha pontilhada abaixo. **Se seu número possui menos que 8 dígitos complete com zeros à esquerda.**

Nome:

.....

Esta prova tem duração de 120 minutos. Não desmonte a prova.

Q1 [1,5 pontos] Simule o código abaixo e selecione as opções correspondentes a saída impressa do programa.

```
def f(A, B):
    for i in range(len(B)):
        d = A[B[i]]
        print( 7*d )

def g(A, B, p):
    d = len(B) // 2
    k = 0
    soma = 0
    for i in range(p-d, p+d+1):
        soma += A[i]*B[k]
        k += 1
    return soma

def main():
    L = [2,1,3,5,7,4,6]
    K = [3,-2,1]
    C = [2, 4]
    L.append(8)
    print(L[len(L)-4])
    f(L, C)
    for elem in C:
        d = g(L, K, elem)
        print(7*d)
main()
```

Rascunho

Selecione o primeiro número impresso:

☐ 8 ☐ 2 ☐ 6 ☐ 3 ☐ 7 ☐ 5 ☐ 4 ☐ 1

Selecione o segundo número impresso:

☐ 21 ☐ 42 ☐ 56 ☐ 63 ☐ 14 ☐ 28 ☐ 49 ☐ 35

Selecione o terceiro número impresso:

☐ 49 ☐ 21 ☐ 28 ☐ 63 ☐ 35 ☐ 56 ☐ 14 ☐ 42

Selecione o quarto número impresso:

☐ 28 ☐ 21 ☐ 63 ☐ 35 ☐ 42 ☐ 14 ☐ 56 ☐ 49

Selecione o quinto número impresso:

☐ 35 ☐ 56 ☐ 28 ☐ 14 ☐ 49 ☐ 21 ☐ 63 ☐ 42

Q2 [2,5 pontos] Queremos fazer um programa que converte uma frase de entrada em outra frase, "secretamente codificada", de saída. Ou seja, por meio de uma lista secreta de codificação de caracteres, podemos especificar como um subconjunto de caracteres da frase de entrada deve ser convertido em outros caracteres. Por exemplo, se queremos fazer a conversão do caracter 't' pelo caracter 'n', especificamos em nossa lista o par de códigos dado pela tupla (ord('t'), ord('n')). Para tanto, definimos uma função `converte(entrada, lista_pares_codigo )`, que recebe uma string contendo a frase de entrada e uma lista de pares `(chave, saida)`, ou seja, tuplas de tamanho 2, em que a `chave` é um inteiro contendo o código Unicode do caracter que deve ser substituído, e `saida` é o código Unicode do caracter pelo qual a chave deve ser substituída; a lista está ordenada crescentemente de acordo com a `chave`. A função `converte` percorre a frase de entrada e converte cada caracter chamando a função `codifica(codigo, lista_pares_codigo)`, que por sua vez faz uma busca binária em `lista_pares_codigo` para encontrar a `saida` (código) para a `chave` de entrada. Caso o código de `entrada` não seja encontrado na lista secreta, a função `codifica` devolve o próprio código de entrada, indicando que não houve alteração para aquele caracter.

```
def converte(frase_entrada, lista_pares_codigo):
    frase_saida = ""
    for letra in L1:
        L2
        cod_out = codifica(cod_in, lista_pares_codigo)
        frase_saida += L3
    return frase_saida

def codifica(codigo, lista_pares_codigo):
    ini, fim = 0, L4
    while ini <= fim:
        L5
        chave, saida = lista_pares_codigo[meio]
        if L6:
            return saida
        elif L7:
            ini = meio + 1
        else:
            L8
    L9
```

Execução: Considere a seguinte execução do programa completo:

```
Entre com a frase a ser convertida:
Batata doce
FRASE CONVERTIDA:
banana loca
```

```
def main():
    L_secreta = [(ord('B'),ord('b')), (ord('d'),ord('l')), (ord('e'),ord('a')), (ord('t'),ord('n'))]
    frase = input("Entre com a frase a ser convertida:  ")
    frase_codificada = L10
    print("FRASE CONVERTIDA:")
    print(frase_codificada)
main()
```

Preencha as lacunas no código acima (L1 até L10), de forma a obter um programa em Python que realize a conversão de uma frase lida. OBS: Para cada lacuna, assinale no máximo uma resposta.

L1: ☐ frase_entrada ☐ frase_saida ☐ cod_in ☐ cod_out ☐ lista_pares_codigo

L2: ☐ letra=ord(cod_in) ☐ cod_in=ord(letra) ☐ cod_in=chave ☐ cod_in=chr(letra) ☐ cod_in=letra

L3: ☐ chr(frase_saida) ☐ chr(frase_entrada) ☐ chr(cod_in) ☐ ord(cod_out) ☐ chr(cod_out)

L4: ☐ len(pares_codigo)-1 ☐ n-1 ☐ codigo-1 ☐ len(pares_codigo)*2.5 ☐ len(codigo)-1

L5: ☐ meio+=1 ☐ meio=fim-1 ☐ meio=ini+1 ☐ 2*meio=ini+fim ☐ meio=(ini+fim)//2

L6: ☐ fim>=codigo ☐ chave==codigo ☐ ini==codigo ☐ meio<=codigo ☐ chave!=meio

L7: ☐ saida>chave ☐ meio>chave ☐ ini>chave ☐ fim>chave ☐ codigo>chave

L8: ☐ fim=meio/2 ☐ meio=ini+fim ☐ fim=ini-1 ☐ fim=ini/2 ☐ fim=meio-1

L9: ☐ print(saida) ☐ return codigo ☐ print(codigo) ☐ return meio ☐ return saida

L10: ☐ converte(frase, L_secreta) ☐ codifica(frase, L_secreta) ☐ converte(frase+L_secreta) ☐ converte(frase) ☐ converte(L_secreta)

Q3 [3,0 pontos] No EP2, um tabuleiro $n \times n$ é representado com n^2 dígitos de 0 a $n^2 - 1$. Ao listar as casas de um tabuleiro da esquerda para a direita e de cima para baixo, a cada casa corresponde um índice do número da casa na sequência formada. Por exemplo, no tabuleiro 3×3 a seguir, cada casa do tabuleiro contém como número o seu próprio índice na lista `[0,1,2,3,4,5,6,7,8]`. Parametrizando o tabuleiro

0	1	2
3	4	5
6	7	8

por

a	b	c
d	e	f
g	h	i

,

codificamo-lo pela sequência de dígitos ihgfedcba e montanos o inteiro $876543210_9 = \sum_{i=0}^{8}(i \times 9^i) = 381367044_{10}$. Uma etapa importante para verificar se um movimento é válido é o de verificar se ele envolve a troca de duas casas vizinhas. Assim, especificamos a função `casas_vizinhas(i1,i2,n)`, que devolve `True` se as casas de índices `i1` e `i2` em tabuleiros `n` por `n` forem vizinhas ou `False` caso contrário. Dentre as opções de trecho a seguir, selecione cada uma que, quando substituída na [LACUNA] no esqueleto abaixo, forma o corpo de uma implementação correta da função `casas_vizinhas`.

Considerações:

- Pode haver mais de uma opção de trecho correta; você deve selecionar todas as corretas.
- Haverá um **desconto na nota** a cada item **errôneo** marcado.

Dica: Tente primeiro codificar o programa num rascunho, isso auxiliará a escolha dos trechos adequados.

```
def casas_vizinhas( i1, i2, n )

    [LACUNA]

    return r
```

Rascunho

☐
```
d = abs( i1 - i2 )
m = max( i1, i2 ) % n
v = (d == n)
h = (d == 1 and m != 0)
r = (v or h)
```

☐
```
d = abs( i1 - i2 )
v = (d == n)
h = (d == 1)
r = (v or h)
```

☐
```
d = abs( i1 - i2 )
m = max( i1, i2 ) % n
v = (d == n)
h = (d == 1 and m == 0)
r = (v or h)
```

☐
```
d = abs( i1 - i2 )
S = (i1 + i2 + 1) % (2*n)
v = (d == n)
h = (d == 1 and S == 0)
r = (v or h)
```

☐
```
d = abs( i1 - i2 )
S = (i1 + i2 + 1) % (2*n)
v = (d == n)
h = (d == 1 and S != 0)
r = (v or h)
```

☐
```
d = abs( i1 - i2 )
m = max( i1, i2 ) % n
S = (i1 + i2 + 1) % (2*n)
t = (m != 0 or S != 0)
v = (d == n)
h = (d == 1 and t)
r = (v or h)
```

☐
```
d = abs( i1 - i2 )
s = (i1 + i2) % (2*n)
v = (d == n)
h = (d == 1 and s!=2*n-1)
r = (v or h)
```

☐
```
d = abs( i1 - i2 )
M = max( i1, i2 ) % (2*n)
v = (d == n)
h = (d == 1 and M != 0)
r = (v or h)
```

☐
```
d = abs( i1 - i2 )
s = (i1 + i2) % (2*n)
v = (d == n)
h = (d == 1 and s != 0)
r = (v or h)
```

☐
```
i = abs( i1//n - i2//n )
j = abs( i1%n - i2%n )
V = (i == 1)
H = (j == 1)
r = (V or H)
```

Q4 [3,0 pontos] Escreva um programa em Python, completando as lacunas, que calcula as médias ponderadas das notas fornecidas de alunos e que exibe os nomes e as médias dos três alunos com melhores médias (em ordem decrescente). As notas são fornecidas em uma matriz M, com uma linha para cada aluno no formato $[\text{Nome}, A_1, A_2, \ldots, A_N]$, com o nome dos alunos na primeira coluna e as notas das várias atividades realizadas nas demais colunas. Cada atividade A_i possui um peso atribuído correspondente W_i para efeito do cálculo da média ponderada. Os pesos são fornecidos em uma lista $W = [W_1, \ldots, W_N]$. As médias computadas deverão ser adicionadas em uma nova coluna à direita da matriz. Por exemplo, para a matriz M e lista W fornecidas na função main abaixo, ao final do cálculo das médias (realizado pela função CalculaMedias), teremos o novo estado da matriz exibido no quadro ao lado. A saída esperada do programa, que indica os três melhores alunos, é também exibida no segundo quadro ao lado.

Estado da matriz M após cálculo das médias:

```
[['Ricardo', 10.0, 7.1, 6.2, 7.133333333333333],
 ['Renato',   2.0, 1.5, 3.4, 2.533333333333333],
 ['Paulo',    7.5, 8.3, 9.2, 8.616666666666667],
 ['Heitor',   5.0, 6.0, 4.0, 4.833333333333333],
 ['Davi',     4.4, 7.2, 5.6, 5.933333333333333]]
```

Saída impressa pelo programa:

```
Paulo:  8.62
Ricardo:  7.13
Davi:  5.93
```

```
def CalculaMedias(M, W):
    for i in L1:
        sN,sW = 0,0
        for j in L2:
            sN += L3
            sW += L4
        L5

def main():
    W = [1, 2, 3]
    M = [ ["Ricardo",10.0, 7.1, 6.2],
          ["Renato",  2.0, 1.5, 3.4],
          ["Paulo",   7.5, 8.3, 9.2],
          ["Heitor",  5.0, 6.0, 4.0],
          ["Davi",    4.4, 7.2, 5.6] ]
    CalculaMedias(M, W)
    I = []   #Lista de índices das linhas selecionadas
    n = len(M[0])
    for k in range(3):
        im = -1   #Índice da linha de maior média ainda não selecionada
        for i in range(len(M)):
            if i not in I:
                if im == -1 or L6:
                    L7
        L8
    for i in L9:
        print("%s:  %.2f"%(M[i][0], M[i][n-1]))
main()
```

Preencha as lacunas no código acima (L1 até L9), assinalando as respostas correspondentes abaixo.
OBS: A cada marcação errada, você poderá ter uma penalização na questão. Deixe em branco se não souber.
Consideração: Para cada lacuna, assinale no máximo uma resposta.

L1:	☐ range(len(W)) ☐ range(1,len(M)) ☐ range(3) ☐ range(len(M)) ☐ range(len(M[0]))
L2:	☐ range(len(M[0])) ☐ range(1,len(M[i])) ☐ range(len(M)) ☐ range(len(W)) ☐ range(3) ☐ range(1,len(M))
L3:	☐ M[j][i]*W[i] ☐ M[i][j]*W[j] ☐ M[i][j]*W[j+1] ☐ M[j][i]*W[j] ☐ M[i][j]*W[j-1]
L4:	☐ W[j-1] ☐ W[j+1] ☐ W[i] ☐ W[i-1] ☐ sum(W) ☐ W[i+1] ☐ W[j]
L5:	☐ M[i].append(sW/sN) ☐ M[i] += sN/sW ☐ M[i].append(sN/sW) ☐ M[i][len(M[0])] = sN/sW ☐ M.append(sW/sN) ☐ M[i].append(sN//sW)
L6:	☐ M[i][n] > M[im][n] ☐ M[i][n-1] < M[im][n-1] ☐ M[n-1][i] > M[n-1][im] ☐ M[i][n-1] > M[im][n-1] ☐ M[i][n] < M[im][n]
L7:	☐ im += 1 ☐ im = k ☐ im = M[i][n-1] ☐ im = n-1 ☐ im = i
L8:	☐ I.append(im) ☐ I = [im] + I ☐ I = im + I ☐ I = [i] + I ☐ I.append(i) ☐ I += im
L9:	☐ range(3) ☐ range(1,len(I)) ☐ range(len(I)) ☐ range(len(M)) ☐ I