MAC2166 - Introdução a Computação - 2018S1 **Avaliação 3**

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	20

⟵ 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 avaliação tem duração de 120 minutos. Não desmonte o caderno de avaliação.

Q1 [1 ponto] Simule o código abaixo e selecione as opções correspondentes às saídas impressas do programa.

```
def g (L, v, a, b, n) :
   c = (a+b)//2;
   print(a,c,b);
   if (v==L[c])  :  return n-c;
   elif (a==c)   :  return -1;
   elif (v>L[c]) :  return g(L,v,a,c, n);
   elif (v<L[c]) :  return g(L,v,c,b, n);
   else :
     return -2;

def f(L, v, n) :
   return g(L, v, 0, 4, n);

print(f([4,3,1,-5], -5, 4));
print(g([11,5,4,1], 5, 1, 4, 4));
```

Rascunho

Abaixo estão listadas as opções de impressões, na ordem em que ocorrem, para cada linha, selecione a única correta:

☐ 2,3,4	☐ -1	☐ 0,3,4	☐ 3	☐ 0,4,2	☐ 0,2,5	☐ 0,-5,4	■ 0,2,4
☐ -1	☐ 0,4,-5	☐ 1,2,4	☐ 2,2,4	■ 2,3,4	☐ 3	☐ 0,4,4	☐ 0,3,4
☐ 0,2,5	☐ -1	☐ 3	☐ 1,3,4	☐ 0,-5,4	☐ 1,2,4	■ 1	☐ 0,4,2
☐ 3	☐ -1	☐ 2,3,4	☐ 0,2,4	☐ 0,2,5	■ 1,2,4	☐ 0,4,2	☐ 0,4,-5
☐ 0,-5,4	☐ 1,1,4	☐ 3	■ 1,1,2	☐ 0,5,2	☐ -1	☐ 2,3,4	☐ 0,2,4
☐ 0,2,4	☐ 1,1,2	☐ 1,2,4	☐ 0,2,5	■ 3	☐ 1,2,1	☐ 1	☐ -1

Q2 [2.5 pontos] As 5 funções abaixo foram projetadas para ordenar qualquer lista de valores inteiros de modo a resultar em lista em ordem decrescente. Entretanto alguns falham para algumas configurações de lista.

(1)
```
def ordena (lista) :
  for i in range(len(lista)-1) :
    for j in range(len(lista)-1) :
      if (lista[j+1] >= lista[j]) :
        q = lista[j];
        lista[j] = lista[j+1];
        lista[j+1] = q;
```

(2)
```
def ordena (lista) :
  i = 0;
  while (i<len(lista)) :
    x = lista[i];
    j = i-1;
    while (j>=0 and lista[j]<x) :
      lista[j+1] = lista[j];
      j = j-1;
    lista[j+1] = x;
    i += 1;
```

(3)
```
def ordena (lista) :
  for i in range(len(lista)-1) :
    for j in range(len(lista)-2) :
      if (lista[j+1] > lista[j]) :
        q = lista[j];
        lista[j] = lista[j+1];
        lista[j+1] = q;
```

(4)
```
def ordena (lista) :
  for i in range(len(lista)) :
    maior = lista[i];
    indice = i;
    for j in range(i, len(lista)) :
      if (lista[j] > maior) :
        maior = lista[j];
        indice = j;
        lista[indice] = lista[i]
```

(5)
```
def ordena (lista) :
  for i in range(len(lista)) :
    maior = lista[i];
    indice = i;
    for j in range(i, len(lista)) :
      if (lista[j] > maior) :
        maior = lista[j];
        indice = j;
    if (indice != i) :
      x = lista[indice];
      lista[indice] = lista[i];
      lista[i] = x;
```

Rascunho

Assinale abaixo todos os blocos correspondentes a códigos que ordene (*decrescente*), qualquer que seja configuração inicial da lista. **Atenção:**

1. Pode existir desde todas as opções corretas até nenhuma opção correta.

2. A cada opção errada que for selecionada, desconta-se nota do exercício.

- [x] (5)
- [ ] (4)
- [ ] (3)
- [x] (2)
- [x] (1)

Q3 [3.5 pontos] Supondo que uma matriz $M_{m\times n}$ tenha, em cada linha, seus elementos ordenados crescente, montar a função acrescenta(.) para que ela seja invocada pela funação ordena_matriz(.) de modo a gerar uma ordenação crescente completa dos elementos da matriz, ou seja, ao final de execução de ordena_matriz(.) gere um vetor v com todos os dados iniciais de M $(m_{00}, m_{01}, \ldots, m_{10}, m_{11} \ldots, m_{m-1,n-1})$, de modo que $v_0 \leqq v_1 \leqq v_2 \leqq \ldots \leqq v_{m\times n-1}$.
Por exemplo, para a matriz 3×4 abaixo, a saída seria o vetor $v = (0, 1, 1, 2, 3, 5, 6, 7, 9, 10, 15, 18)$.

$$M = \begin{bmatrix} 1 & 2 & 10 & 18 \\ 1 & 3 & 9 & 15 \\ 0 & 5 & 6 & 7 \end{bmatrix}$$

```
def ordena_matriz (M, nlnhs, ncols) :
  v1 = [0]*(nlnhs*ncols);
  v2 = [0]*(nlnhs*ncols);
  prontos = 0;
  for linha in range(nlnhs):
    if (linha %2 == 0) :
      acrescenta(M, linha, v1, v2, prontos, ncols);
    else :
      acrescenta(M, linha, v2, v1, prontos, ncols);
    prontos += ncols;
  if (nlnhs % 2 == 0):  print(v1);
  else :  print(v2);
```

```
def acrescenta (M, lnh, vetAnt,
                vetNov, prontos, ncols) :
  c_ant = 0; c_nov = 0; c_mat = 0;
  while ( L1 ):
    if ( L2 ):
       L3
       c_mat += 1; c_nov += 1;
    elif ( L4 ):
       L5
       c_ant += 1; c_nov += 1;
    elif ( L6 ):
       L7
       c_mat += 1; c_nov += 1;
    else :
       L8
       c_ant += 1; c_nov += 1;
  print("parcial:", vetNov);
```

Rascunho

Para cada um dos **8** itens a seguir, correspondendo as lacunas no código acima, assinale a única resposta que torna o programa acima correto. A cada opção errada que for selecionada, desconta-se até 0.3.

L1:
- [ ] M[c_ant][c_nov]
- [x] c_mat < ncols or c_ant < prontos
- [ ] M[c_ant][c_mat]
- [ ] vetAnt[c_ant]<vetNov[c_nov]
- [ ] c_mat < prontos or c_ant < ncols
- [ ] c_mat < ncols
- [ ] c_ant < prontos

L2:
- [ ] prontos < c_ant
- [ ] c_ant < prontos
- [ ] prontos < c_nov
- [x] c_ant >= prontos
- [ ] c_ant < c_nov
- [ ] c_ant <= prontos
- [ ] c_ant > prontos

L3:
- [ ] vetNov[c_mat]=vetAnt[c_nov]
- [ ] vetNov[c_mat]=M[lnh][c_nov]
- [ ] vetAnt[c_nov]=M[lnh][c_mat]
- [ ] vetAnt[c_ant]=vetAnt[c_nov]
- [ ] vetAnt[c_ant]=vetAnt[c_mat]
- [x] vetNov[c_nov]=M[lnh][c_mat]
- [ ] vetNov[c_nov]=vetAnt[c_ant]

L4:
- [ ] c_mat >= c_nov
- [ ] c_nov > ncols
- [ ] c_mat > ncols
- [x] c_mat >= ncols
- [ ] c_ant < prontos
- [ ] c_nov >= ncols
- [ ] c_ant < ncols

L5:
- [ ] vetNov[c_ant] = vetAnt[c_nov]
- [ ] vetAnt[c_ant] = M[c_ant][c_nov]
- [ ] vetAnt[c_ant] = vetNov[c_mat]
- [ ] vetNov[c_nov] = M[c_ant][c_nov]
- [ ] vetNov[c_nov] = M[c_ant][c_nov]
- [x] vetNov[c_nov] = vetAnt[c_ant]
- [ ] vetAnt[c_ant] = vetNov[c_nov]

L6:
- [ ] vetAnt[c_ant] < vetNov[c_nov]
- [ ] vetAnt[c_ant] < M[lnh][c_mat]
- [ ] vetAnt[c_nov] < M[lnh][c_mat]
- [ ] vetAnt[c_nov] < vetNov[c_ant]
- [ ] vetAnt[c_ant] < vetNov[c_nov]
- [x] vetAnt[c_ant] > M[lnh][c_mat]
- [ ] vetAnt[c_nov] > M[lnh][c_mat]

L7:
- [ ] vetNov[c_mat]=vetAnt[c_nov]
- [ ] vetAnt[c_nov]=M[lnh][c_mat]
- [x] vetNov[c_nov]=M[lnh][c_mat]
- [ ] vetNov[c_nov]=vetAnt[c_ant]
- [ ] vetAnt[c_ant]=vetAnt[c_mat]
- [ ] vetNov[c_mat]=M[lnh][c_nov]
- [ ] vetAnt[c_ant]=vetAnt[c_nov]

L8:
- [ ] vetNov[c_ant] = vetAnt[c_nov]
- [ ] vetAnt[c_ant] = M[c_ant][c_nov]
- [ ] vetAnt[c_ant] = vetNov[c_nov]
- [ ] vetNov[c_nov] = M[c_ant][c_nov]
- [x] vetNov[c_nov] = vetAnt[c_ant]
- [ ] vetAnt[c_ant] = vetNov[c_mat]
- [ ] vetNov[c_nov] = M[c_ant][c_nov]

Q4 [3 pontos] Nesta questão você deve elaborar um programa que, dada uma matriz $A_{m\times n}$, um vetor $x_{n\times 1}$ e um vetor $b_{m\times 1}$, determina se $Ax = b$, nesse caso devolve $(-1, None)$. Em caso contrário, devolve (i, Ax), sendo i o primeiro índice tal que $A_i x \neq b_i$. Pode-se supor que os dados sejam inteiros.

Por exemplo, para a matriz $A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \end{bmatrix}$ e vetor $x = [4; 3; 2; 1]$, cujo produto é $Ax = \begin{bmatrix} 20 \\ 30 \end{bmatrix}$. Assim, usando como vetor lado direito $b = [20; 30]$ ou $b = [20; 31]$, a resposta seria, respectivamente, $(-1, None)$ ou $(1, [20; 30])$.

```
#Trecho-A - ID 0
main()
#Trecho-B - ID 0
def produto_escalar (v1, v2) :
  n = len(v1[0])
#Trecho-C - ID 0
def produto_escalar (v1, v2) :
  n = len(v1)
#Trecho-D - ID 0
def produto_matriz_vetor (A, x, b) :
  m = len(A[])
  n = len(A[][])
#Trecho-E - ID 0
def produto_matriz_vetor (A, x, b) :
  m = len(A)
  n = len(A[0])
#Trecho-F - ID 0
def produto_matriz_vetor (A, x, b) :
  m = len(A[0])
  n = len(A[0][0])
#Trecho-G - ID 1
  soma = 0
#Trecho-H - ID 1
  erro = True;
#Trecho-I - ID 1
  erro = False;
#Trecho-J - ID 1
  ind = -1
#Trecho-K - ID 1
  ind = 0
#Trecho-L - ID 1
  y = [];
```

```
#Trecho-M - ID 2
    esc = produto_escalar(A[i], x);
#Trecho-N - ID 3
      erro = False
      ind = i
#Trecho-O - ID 3
      erro = True
      ind = i
#Trecho-P - ID 2
    if (soma !=0):
      soma += v1[i] * v2[i];
#Trecho-Q - ID 2
    soma += v1[i] * v2[i];
#Trecho-R - ID 2
    y = y + esc;
#Trecho-S - ID 2
    y.append(esc);
#Trecho-T - ID 2
    if (ind == -1 and esc != b[i]) :
#Trecho-U - ID 2
    if (ind == 0 and esc != b[i]) :
#Trecho-V - ID 1
  if (erro) :
#Trecho-X - ID 1
  for i in range(1,n) :
#Trecho-Y - ID 1
  for i in range(0,n+1) :
#Trecho-Z - ID 1
  for i in range(0,n) :
#Trecho-AA - ID 1
  for i in range(1,m) :
```

```
#Trecho-BB - ID 1
  for i in range(0,m) :
#Trecho-CC - ID 1
  for i in range(1,m+1) :
#Trecho-DD - ID 1
  return soma;
#Trecho-EE - ID 1
  if (soma ==0 ):
    return True
  else:
    return False
#Trecho-FF - ID 2
    return y, ind
#Trecho-GG - ID 1
  return -1, None
#Trecho-HH - ID 1
  return None, -1
#Trecho-II - ID 2
  return False, None
#Trecho-JJ - ID 2
    return ind, y
#Trecho-KK - ID 2
    return i, y
#Trecho-LL - ID 0
def main() :  # A*x = [ 20; 30]
    A = [ [1, 2, 3, 4], [2, 3, 4, 5]];
    x = [ 4, 3, 2, 1 ]; b = [ 20, 31 ];
    resp, y = produto_matriz_vetor(A, x, b)
    if (resp == -1):
      print("Confere, resultado=",y)
    else:
      print("Errado, linha=", resp)
```

Assinale a ÚNICA alternativa que contém os blocos corretos na ORDEM correta. Marcar mais de uma alternativa implica em ZERO.

- ☐ A,C,G,Z,Q,EE,E,L,I,K,BB,M,S,U,O,V,JJ,GG,LL
- ☐ A,G,X,Q,DD,E,L,I,J,AA,M,S,T,O,V,KK,II,LL
- ☐ A,G,Z,P,DD,E,L,H,K,BB,M,S,U,O,V,JJ,GG,LL,A
- ☐ C,G,Z,Q,EE,E,L,I,J,BB,M,R,T,O,V,JJ,GG,LL,A
- ☐ B,G,Y,Q,DD,F,L,I,J,CC,M,S,T,O,V,JJ,GG,LL,A
- ☐ A,G,Z,Q,DD,E,L,H,K,BB,M,S,T,N,V,KK,GG,LL,A
- ☐ C,G,X,Q,DD,E,L,I,J,AA,M,S,T,O,V,KK,II,LL,A
- ☐ A,C,G,Z,Q,EE,E,L,I,J,BB,M,R,T,O,V,JJ,GG,LL
- ☐ C,G,Z,Q,DD,E,L,H,J,BB,M,S,T,O,V,KK,II,LL,A
- ☐ A,B,G,X,Q,DD,F,L,I,J,AA,M,S,T,O,V,JJ,GG,LL
- ☐ C,G,Y,Q,DD,E,L,I,J,CC,M,S,T,O,V,JJ,GG,LL,A
- ■ C,G,Z,Q,DD,E,L,I,J,BB,M,S,T,O,V,JJ,GG,LL,A
- ☐ B,G,X,Q,DD,F,L,I,J,AA,M,S,T,O,V,JJ,GG,LL,A
- ☐ B,G,Z,Q,DD,F,L,I,J,BB,M,S,T,O,V,JJ,GG,LL,A
- ☐ C,G,Z,Q,EE,DD,L,I,K,BB,M,S,U,O,V,JJ,GG,LL,A
- ☐ C,G,X,Q,EE,E,L,I,J,AA,M,R,T,O,V,JJ,GG,LL,A
- ☐ C,G,X,Q,DD,E,L,I,J,AA,M,S,T,O,V,JJ,GG,LL,A

Rascunho