MAC2166 - Introdução a Computação - 2024S1 **Avaliação 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	13	14	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 prova tem duração de 120 minutos. Não desmonte a prova.

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

```
def f(L, x):
    L.append(x)
    i = len(L)-2
    while x < L[i]:
        L[i+1] = L[i]
        i -= 1
    L[i+1] = x
def g(P):
    s = 0
    for i in range(1,len(P)):
        if P[i] > P[i-1]:
            s += 1
    return s
def main():
    L = [28, 34, 72, 10, 82, 56, 63]
    P = [1, 3, 2, 4, 0, 5]
    P.append(len(P))
    print(L[g(P)])
    temp = P[2]
    P[2] = P[5]
    P[5] = temp
    for i in range(len(P)-5):
        print(L[P[i*2]])
    f(L, 42)
    A = L[2:5]
    print(A[1])
    print(A[2])
main()
```

Rascunho

Selecione o primeiro número impresso:

☐ 42 ☐ 34 ☐ 56 ☐ 72 ☐ 63 ☐ 82 ☐ 10 ☐ 28

Selecione o segundo número impresso:

☐ 34 ☐ 82 ☐ 63 ☐ 42 ☐ 28 ☐ 72 ☐ 56 ☐ 10

Selecione o terceiro número impresso:

☐ 56 ☐ 34 ☐ 72 ☐ 42 ☐ 10 ☐ 63 ☐ 28 ☐ 82

Selecione o quarto número impresso:

☐ 72 ☐ 34 ☐ 42 ☐ 10 ☐ 82 ☐ 28 ☐ 63 ☐ 56

Selecione o quinto número impresso:

☐ 10 ☐ 63 ☐ 72 ☐ 28 ☐ 56 ☐ 34 ☐ 42 ☐ 82

Q2 [2,5 pontos]

Implemente uma função chamada `frequencia_relativa(L)` que recebe como parâmetro uma lista L e devolve duas listas indicando a frequência relativa dos elementos de L. Por exemplo, para a lista L = [3, 4, 5, 4, 5, 2, 7, 4, 4, 2], a função frequencia_relativa deve devolver as seguintes listas:

[3, 4, 5, 2, 7], [0.1, 0.4, 0.2, 0.2, 0.1]

indicando que o número 3 representa 10% dos elementos, o número 4 representa 40% dos elementos e assim por diante.

Para facilitar a implementação da função frequencia_relativa, primeiro implemente a função `pertence(elemento, lista)` que verifica se um inteiro `elemento` ocorre na dada lista. Se ele ocorre, a função devolve a posição da primeira ocorrência, caso contrário devolve -1.

```
def pertence(elemento, lista):
    for i in range( L1 ):
        if lista[i] == L2 :
            return L3
    return -1

def frequencia_relativa(L):
    elementos_distintos = []
    L4
    for elemento in L:
        L5
        if indice == -1:
            elementos_distintos.append(elemento)
            L6
        else:
            frequencia[indice] += 1

    for i in range(len(frequencia)):
        L7
    return L8
```

Preencha as lacunas no código acima (L1 até L8), assinalando as respostas correspondentes abaixo.

Consideração: Para cada lacuna, assinale no máximo uma resposta.

L1:
- ☐ `len(lista)`
- ☐ `lista`
- ☐ `len(i)`
- ☐ `elemento`
- ☐ `lista[i]`

L2:
- ☐ `lista[elemento]`
- ☐ `elemento`
- ☐ `0`
- ☐ `lista[i]`
- ☐ `len(lista)`

L3:
- ☐ `lista[i]`
- ☐ `+1`
- ☐ `elemento`
- ☐ `lista`
- ☐ `i`

L4:
- ☐ `frequencia = []`
- ☐ `elementos_distintos.append(frequencia)`
- ☐ `frequencia = [[]]`
- ☐ `elementos_distintos.append(L)`
- ☐ `frequencia = 0`

L5:
- ☐ `indice = frequencia_relativa(L)`
- ☐ `indice = frequencia_relativa(elemento, elementos_distintos)`
- ☐ `elemento = pertence(elementos_distintos, indice)`
- ☐ `indice = pertence(elementos_distintos, elemento)`
- ☐ `indice = pertence(elemento, elementos_distintos)`

L6:
- ☐ `frequencia.append(-1)`
- ☐ `frequencia.append(1)`
- ☐ `elemento.next()`
- ☐ `frequencia[indice] = 1`
- ☐ `frequencia.append(0)`

L7:
- ☐ `frequencia[i] /= 1/len(L)`
- ☐ `frequencia[i] +=1`
- ☐ `frequencia[i] *= len(L)`
- ☐ `frequencia[i] /= L`
- ☐ `frequencia[i] /= len(L)`

L8:
- ☐ `elemento, elementos_distintos`
- ☐ `L, frequencia`
- ☐ `frequencia, elementos_distintos`
- ☐ `frequencia, elemento`
- ☐ `elementos_distintos, frequencia`

Q3 [2,5 pontos] A trajetória de uma nave pode ser descrita por uma lista de `t` pontos `T = [P0, P1, ..., Pt-1]`, com as posições `Pi = [xi,yi]`, $0 \leq$ `i` $<$ `t`, amostradas da nave em intervalos fixos de tempo.

Dadas as trajetórias de um conjunto de `n` naves, na forma de uma lista `LN = [T0, T1, ..., Tn-1]`, faça uma função `nave_percorreu_maior_distancia` em Python que calcula qual nave percorreu a maior distância total. A função deve devolver o índice da nave que percorreu a maior distância e o valor desta distância.

Faça uma função `distancia` que calcula a distância de uma nave na posição `P = [x,y]` até a superfície de um astro (corpo celeste) `A = [[xc,yc],R]`, onde `xc` e `yc` são a posição do seu centro e `R` o valor do seu raio. Considere que a nave possui dimensões desprezíveis em relação ao tamanho do astro e que não há colisão.

Dada a trajetória de uma nave `T = [P0, P1, ..., Pt-1]` e um astro `A = [[xc,yc],R]`, faça uma função `distancia_minima` que devolve qual foi a menor distância que a nave esteve do astro ao longo da sua trajetória.

Dadas as trajetórias de `n` naves, numeradas de 0 a `n-1`, na forma de uma lista `LN = [T0, T1, ..., Tn-1]`, e uma lista de `m` astros `LA = [A0, ..., Am-1]`, faça um programa que diz qual nave percorreu a maior distância e quais foram as distâncias mínimas que cada nave assumiu ao longo de sua trajetória em relação a cada um dos astros.

```
def nave_percorreu_maior_distancia(LN):
    im, dm = 0, 0.0
    for i in [L1]:
        d = 0.0
        for j in [L2]:
            [L3]
            [L4]
        [L5]
    return im,dm

def distancia(P, A):
    return [L6]

def distancia_minima(T, A):
    dm = -1
    for i in [L7]:
        [L8]
        if [L9]:
            dm = d
    return dm
```

```
def main():
    T0 = [[150000,214002],[150000,214001],[150000,214000]]
    T1 = [[20000, 20000],[20000, 20001],[20001, 20002]]
    LN = [T0, T1]
    LA = [ [[0,0], 28000], [[150000,150000], 20000] ]
    i,d = nave_percorreu_maior_distancia(LN)
    print("nave %d percorreu a maior distância de %.2f km."%(i,d))
    for i in range(len(LN)):
        for j in [L10]:
            dm = [L11]
            print("Distância (nave %d,astro %d): %.2f km."%(i,j,dm))
main()
```

Exemplo de execução do programa completo:

```
nave 1 percorreu a maior distância de 2.41 km.
Distância (nave 0,astro 0):  233335.03 km.
Distância (nave 0,astro 1):  44000.00 km.
Distância (nave 1,astro 0):  284.27 km.
Distância (nave 1,astro 1):  163845.64 km.
```

Para cada um dos **11** itens a seguir, correspondendo às lacunas no código acima, assinale a única resposta correta.

L1: ☐ `range(1,len(LN))` ☐ `range(len(LN))` ☐ `range(LN)` ☐ `len(LN)` ☐ `range(len(LN),0,-1)`

L2: ☐ `range(1,len(LN[i]))` ☐ `range(len(LN[i]))` ☐ `range(LN[i],1,-1)` ☐ `len(LN[i])` ☐ `range(1,LN[i]-1)`

L3: ☐ `P,Pa = LN[i][j],LN[i][j+1]` ☐ `P,Pa = [i,j],[i,j-1]` ☐ `P,Pa = LN[i][j],LN[i][j-1]` ☐ `P,Pa = LN[i],LN[j]` ☐ `P,Pa = LN[j][i],LN[j][i+1]`

L4: ☐ `d += ((P-Pa)**2)**1/2` ☐ `d += ((P[0]-Pa[1])**2 + (P[0]-Pa[1])**2)**0.5` ☐ `d += ((P[0]-P[1])**2 + (Pa[0]-Pa[1])**2)**0.5` ☐ `d += ((P[0]-Pa[0])**2 + (P[1]-Pa[1])**2)**0.5` ☐ `d += (P[0]-Pa[0]) - (P[1]-Pa[1])`

L5: ☐ `if d < dm: im,dm = i,d` ☐ `if d**0.5 < dm: dm = d**0.5` ☐ `if d > dm: im = i` ☐ `if d > dm: return i,d` ☐ `if d > dm: im,dm = i,d`

L6: ☐ `((P[0]-A[0][0])*2+(P[1]-A[1][0])*2)**0.5-A[1]` ☐ `((P[0]-A[0][0])**2+(P[1]-A[1][0])**2)**0.5` ☐ `((P[0]-A[0][0])**2+(P[1]-A[0][1])**2)**0.5-A[1]` ☐ `((P[0]-A[0])**2+(P[1]-A[1])**2)-A[1]` ☐ `((P[0]-A[0][0])**2+(P[1]-A[1][0])**2)**1/2-A[2]`

L7: ☐ `range(T)` ☐ `range(1,len(T)+1)` ☐ `range(len(T))` ☐ `range(len(T)-1)` ☐ `range(len(T),1,-1)`

L8: ☐ `d = distancia(T[i], A)` ☐ `d = distancia(T[i][0], A)` ☐ `d = distancia(T[i], A[0])` ☐ `d = distancia(T[i+1], A[1])` ☐ `d = distancia(T, A[i])`

L9: ☐ `dm == -1 or d > dm` ☐ `not(dm == -1 and d < dm)` ☐ `dm == -1 and d < dm` ☐ `dm == -1 or d < dm` ☐ `dm != -1 or d-dm > 0`

L10: ☐ `range(1,len(LA))` ☐ `range(LA)` ☐ `range(len(LA)-1,0,-1)` ☐ `range(len(LA))` ☐ `range(len(LA)-1)`

L11: ☐ `distancia_minima(LN[i], LA[j][0])` ☐ `distancia(LN[i][j], LA[j][0])` ☐ `distancia_minima(LN[i], LA[j])` ☐ `distancia_minima(LN[j], LA[i])` ☐ `distancia(LN[i], LA[j])`

Q4 [2,5 pontos] Uma matriz M é dita Matriz de Hadamard se (1) M é quadrada; (2) todas as suas células contém 1 ou -1; e (3) todas as suas linhas são ortogonais entre si, ou seja, que o produto escalar de quaisquer duas linhas é 0. Lembre-se que o produto escalar de dois vetores $(a_1, \ldots, a_n)$ e $(b_1, \ldots, b_n)$ é o somatório $\sum_{i=1}^{n} a_i * b_i = a_1 \cdot b_1 + a_2 \cdot b_2 + \cdots + a_n \cdot b_n$.

Queremos fazer uma função (`matriz_hadamard`) que decide se uma dada matriz é uma Matriz de Hadamard. Para isso, vamos fazer uma função auxiliar (`ortogonais`) que decide se duas linhas são ortogonais.

```
def ortogonais(linha1, linha2):
    produto_escalar = 0
    for i in range(len(linha1)):
        produto_escalar += linha1[i]*L1
    return L2

def matriz_hadamard(M):
    n,m = len(M),L3

    if n != m:
        return L4

    L5
    for i in range(n):
        for j in L6:

            if L7:
                L8

            if i != j and L9:
                é_hadamard = False

    return L10
```

Preencha as lacunas no código acima (L1 até L10), de forma a obter as funções descritas.

OBS: Para cada lacuna, assinale no máximo uma resposta.

L1:
- ☐ `linha2[j]`
- ☐ `2`
- ☐ `linha2[i]`
- ☐ `i`
- ☐ `len(linha1)`

L2:
- ☐ `(produto_escalar != 0)`
- ☐ `(produto_escalar == 0)`
- ☐ `produto_escalar`
- ☐ `True`
- ☐ `False`

L3:
- ☐ `n`
- ☐ `len(M[0])`
- ☐ `M[0][0]`
- ☐ `len(M)`
- ☐ `M[0]`

L4:
- ☐ `m`
- ☐ `True`
- ☐ `False`
- ☐ `n != m`
- ☐ `n`

L5:
- ☐ `i,j = 0,0`
- ☐ `n = m`
- ☐ `é_hadamard = False`
- ☐ `i = 0`
- ☐ `é_hadamard = True`

L6:
- ☐ `range(i+1)`
- ☐ `M[i]`
- ☐ `range(n)`
- ☐ `range(n+1)`
- ☐ `range(2*i)`

L7:
- ☐ `M[i][j] == -1 or M[i][j] == 1`
- ☐ `M[i][j] != 1 or M[i][j] != -1`
- ☐ `M[i][j] == 1 and M[i][j] == -1`
- ☐ `M[i][j] == 1 or M[i][j] == -1`
- ☐ `M[i][j] != 1 and M[i][j] != -1`

L8:
- ☐ `é_hadamard = False`
- ☐ `j += 1`
- ☐ `i += 1`
- ☐ `é_hadamard = True`
- ☐ `é_hadamard = not é_hadamard`

L9:
- ☐ `not ortogonais(M[i],M[i])`
- ☐ `ortogonais(M[j],M[i])`
- ☐ `not ortogonais(M[j],M[j])`
- ☐ `ortogonais(M[i],M[j])`
- ☐ `not ortogonais(M[i],M[j])`

L10:
- ☐ `é_hadamard`
- ☐ `True`
- ☐ `not é_hadamard`
- ☐ `False`
- ☐ `é_hadamard == False`

Q5 [1 ponto] Linguagens de programação permitem a definição de funções, o que ajuda na organização (modularização) do código. Algumas das principais vantagens do uso de funções são:

- ☐ (1) permitem resolver alguns problemas que seriam impossíveis de resolver computacionalmente sem funções; (2) permitem reaproveitar código existente minimizando a duplicação de código e (3) permitem esconder detalhes de implementação dentro de um trecho de código, permitindo uma visão de mais alto nível, viabilizando programas mais complexos.
- ☐ (1) permitem quebrar tarefas grandes em menores facilitando o entendimento e manutenção do código; (2) permitem reaproveitar código existente minimizando a duplicação de código e (3) permitem o uso de depuradores (debuggers), que facilitam a correção de erros.
- ☐ (1) permitem quebrar tarefas grandes em menores facilitando o entendimento e manutenção do código; (2) permitem implementar soluções mais eficientes, que chegam na solução mais rapidamente e (3) permitem esconder detalhes de implementação dentro de um trecho de código, permitindo uma visão de mais alto nível, viabilizando programas mais complexos.
- ☐ (1) permitem quebrar tarefas grandes em menores facilitando o entendimento e manutenção do código; (2) permitem reaproveitar código existente minimizando a duplicação de código e (3) permitem que o código fique mais extenso, aproveitando melhor a memória disponível.
- ☐ (1) permitem quebrar tarefas grandes em menores facilitando o entendimento e manutenção do código; (2) permitem reaproveitar código existente minimizando a duplicação de código e (3) permitem esconder detalhes de implementação dentro de um trecho de código, permitindo uma visão de mais alto nível, viabilizando programas mais complexos.

Com base nas alternativas acima, assinale a única alternativa correta.