+1/1/60+

MAC2166 - Introdução a Computação - 2025S1 **Avaliação P3**

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 [2 pontos] Simule o código abaixo e selecione as opções correspondentes à saída impressa do programa.

```
def funcao(matriz):
    for i in range(4):
        resultado = matriz[i][i] + matriz[i][(i+1)%4]
        print(resultado)

matriz = [ ["a", "b", "c", "d"],
           ["e", "f", "g", "h"],
           ["i", "j", "k", "l"],
           ["m", "n", "o", "p"] ]
funcao(matriz)
```

Rascunho

Selecione o primeiro resultado impresso:

☐ ac ☐ ab ☐ ef ☐ ci ☐ de ☐ ig ☐ fg ☐ da

Selecione o segundo resultado impresso:

☐ de ☐ db ☐ fg ☐ ci ☐ ig ☐ ac ☐ ab ☐ gf

Selecione o terceiro resultado impresso:

☐ ab ☐ ac ☐ ef ☐ de ☐ ca ☐ ci ☐ kl ☐ fg

Selecione o quarto resultado impresso:

☐ ig ☐ mp ☐ ab ☐ fg ☐ pm ☐ de ☐ ef ☐ hb

Selecione o quinto resultado impresso ou N/A (não aplicável) se não há mais que 4 impressões:

☐ hb ☐ N/A ☐ hi ☐ be ☐ fg ☐ de ☐ c ☐ ab ☐ ci

Q2 [2 pontos] Você está desenvolvendo um programa para localizar um aluno em uma lista de alunos matriculados, com base no número de matrícula. O programa deve usar a busca binária em uma matriz, onde cada linha contém duas colunas: a matrícula do aluno na primeira coluna e o nome do aluno na segunda coluna.

Preencha as lacunas no código abaixo para que o programa funcione corretamente utilizando busca binária. A saída do programa deve ser idêntica à saída dos exemplos apresentados.

```
def buscaBinaria(matriz, matricula):
    baixo = 0
    alto = len(matriz) - 1
    while baixo <= alto:
        meio = (baixo + alto) // 2
        L1
            L2
        elif matriz[meio][0] < matricula:
            L3
        else:
            L4
    L5

alunos = [[1001, "Ana"], [1002, "João"], [1003,
"Maria"], [1004, "Pedro"], [1005, "Lucas"]]
matricula = int(input("Digite a matrícula buscada:"))
resultado = buscaBinaria(alunos, matricula)
print("Resultado da busca:", resultado)
```

Exemplos de execução do programa completo:

```
Exemplo 1:
Digite a matrícula buscada:  1002
Resultado da busca:  João

Exemplo 2:
Digite a matrícula buscada:  1004
Resultado da busca:  Pedro

Exemplo 3:
Digite a matrícula buscada:  1005
Resultado da busca:  Lucas

Exemplo 4:
Digite a matrícula buscada:  1006
Resultado da busca:  Aluno não encontrado
```

Para cada um dos **5** itens a seguir, correspondendo as lacunas no código acima, assinale a única resposta correta.

L1:
- ☐ `if matriz[meio][0] < matricula:`
- ☐ `if matriz[meio][1] < matricula:`
- ☐ `if matriz[meio][0] == matricula:`
- ☐ `if matriz[meio][1] == matricula:`
- ☐ `if matriz[meio][0] != matricula:`

L2:
- ☐ `return "Aluno encontrado"`
- ☐ `return matriz[meio][1]`
- ☐ `return "Matrícula encontrada"`
- ☐ `return matriz[meio]`
- ☐ `return matriz[meio][0]`

L3:
- ☐ `alto = meio - 1`
- ☐ `baixo = meio - 1`
- ☐ `alto = meio + 1`
- ☐ `baixo = meio`
- ☐ `baixo = meio + 1`

L4:
- ☐ `alto = meio - 1`
- ☐ `baixo = meio - 1`
- ☐ `baixo = meio`
- ☐ `alto = meio + 1`
- ☐ `baixo = meio + 1`

L5:
- ☐ `"Aluno não encontrado"`
- ☐ `None`
- ☐ `"Matricula inválida"`
- ☐ `"Aluno não encontrado na lista"`
- ☐ `"Matrícula não encontrada"`

Qual é a condição necessária para o correto funcionamento de uma busca binária?

- ☐ Os elementos da lista devem ser pares: chave e valor.
- ☐ Não há restrições de uso, mas seu tempo de execução é menor quando as listas estão ordenadas.
- ☐ A lista deve estar ordenada pelo campo de busca.
- ☐ Somente funciona para números inteiros como campo de busca.
- ☐ O comprimento da lista deve ser uma potência de 2.
- ☐ O comprimento da lista deve ser um número ímpar.

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

Q3 [3 pontos] Considere a seguinte sequência de números inteiros: **[5, 2, 9, 1, 5, 6]**. A simulação do algoritmo *Insertion Sort* ordena essa sequência em ordem crescente, para isso o algoritmo trabalha iterando da esquerda para a direita, comparando o elemento atual com os anteriores e inserindo-o na posição correta dentro da parte já ordenada da sequência. Complete o algoritmo e depois simule o algoritmo e selecione as alternativas que correspondem à sequência dos números após cada inserção (**não à sequência final, mas a sequência após cada passo do algoritmo**).

```
def insertion_sort(lista):
    for i in range(1, len(lista)):
        L1
        j = i - 1
        L2
            lista[j + 1] = lista[j]
            j -= 1
        L3
        print(lista)
def main():
    lista = [5, 2, 9, 1, 5, 6]
    print("Sequência inicial:", lista)
    insertion_sort(lista)

main()
```

Rascunho

Para cada um dos **3** itens a seguir, correspondendo as lacunas no código acima, assinale a única resposta correta.

L1:
- ☐ `lista[i]=chave`
- ☐ `chave = lista[-1]`
- ☐ `chave = lista[i+1]`
- ☐ `chave = lista[i-1]`
- ☐ `chave = lista[i]`

L2:
- ☐ `while j >= 0:`
- ☐ `while j == 0 and lista[j] != chave:`
- ☐ `while j >= 1 and lista[j] < chave:`
- ☐ `while lista[j] > chave:`
- ☐ `while j >= 0 and lista[j] > chave:`

L3:
- ☐ `lista[chave] = chave+1`
- ☐ `lista[j + 1] = chave`
- ☐ `lista[chave + 1] = j`
- ☐ `lista[j - 1] = chave`
- ☐ `lista[j] = chave`

Agora que você já completou o código. Assinale o que será impresso em cada iteração do for da função insertion_sort.
Após a **primeira iteração do laço for da função de ordenação por inserção**, qual é a sequência resultante?

- ☐ [2, 5, 9, 5, 1, 6]
- ☐ [5, 2, 9, 6, 1, 5]
- ☐ [2, 9, 5, 1, 5, 6]
- ☐ [2, 5, 9, 1, 5, 6]
- ☐ [5, 2, 9, 1, 5, 6]

Após a **segunda iteração do laço for da função de ordenação por inserção**, qual é a sequência resultante?

- ☐ [2, 9, 5, 1, 5, 6]
- ☐ [2, 5, 9, 1, 5, 6]
- ☐ [5, 2, 9, 1, 5, 6]
- ☐ [1, 5, 9, 2, 5, 6]
- ☐ [5, 2, 9, 6, 1, 5]

Após a **terceira iteração do laço for da função de ordenação por inserção**, qual é a sequência resultante?

- ☐ [5, 2, 9, 1, 5, 6]
- ☐ [9, 2, 5, 1, 5, 6]
- ☐ [9, 2, 1, 5, 5, 6]
- ☐ [2, 5, 9, 1, 5, 6]
- ☐ [1, 2, 5, 9, 5, 6]

Após a **quarta iteração do laço for da função de ordenação por inserção**, qual é a sequência resultante?

- ☐ [2, 5, 9, 1, 5, 6]
- ☐ [5, 2, 9, 6, 1, 5]
- ☐ [1, 2, 5, 9, 5, 6]
- ☐ [1, 2, 5, 5, 9, 6]
- ☐ [5, 2, 9, 6, 5, 1]

Após a **quinta iteração do laço for da função de ordenação por inserção**, qual é a sequência resultante?

- ☐ [1, 2, 5, 5, 6, 9]
- ☐ [1, 2, 5, 9, 5, 6]
- ☐ [1, 2, 5, 9, 1, 6]
- ☐ [2, 5, 9, 1, 5, 6]
- ☐ [1, 2, 5, 9, 6, 5]

Q4 [3 pontos] Preencha as lacunas no código abaixo (L1 até L8), de forma a obter um programa que lê vários arquivos de texto e cria um dicionário que relaciona cada palavra encontrada com o nome dos arquivos em que ela ocorre. Se uma palavra ocorrer duas vezes no mesmo arquivo, ela deverá ser incluída apenas uma vez.

```
def indexaArquivo( ind, arq ):
    f = L1
    for l in f:
        ps = L2
        L3 :
            if L4 :
                L5
            elif L6 :
                L7
    L8
def main():
    arquivos = [ "arq1.txt", "arq2.txt", "arq3.txt"]
    I = {}
    for a in arquivos:
        indexaArquivo( I, a )
    print(I)
main()
```

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.

L1	`file.open( arq )`	`open( arq, 'r' )`	`open( arq, 'w' )`	`open( arq, 'a' )`
L2	`l=l[:]`	`l.join(' ')`	`l.split().strip()`	`l.strip().split()`
L3	`for i in range(len(ps))`	`for p in range(len(ps))`	`while i < len(ps)`	`for p in ps`
L4	`ind[p] != arq`	`ps[i] in ind`	`p not in ind`	`p in ind`
L5	`ps[i] = [ arq ]`	`ind[p] = [ arq ]`	`ps[p] = [ arq ]`	`ind[i] = [ arq ]`
L6	`ind[i]`	`arq not in ind[p]`	`not( ind[p][arq] )`	`ind not in arq[i]`
L7	`ind[p] = arq`	`ind[i].append(arq)`	`ind[p].append(arq)`	`ind[i] = arq`
L8	`return ind`	`f.close()`	`f.delete()`	`return arq`