MC102—Algoritmos e Programação de Computadores

Prova 2

GABARITO

Segundo Semestre de 2018

Questão	Nota
1	
2	
3	
4	
Total	

Nome: RA:

Importante: As respostas deverão ser escritas nos espaços demarcados, opcionalmente a lápis. Não é permitida a consulta a qualquer material, nem o uso de celulares ou outros dispositivos eletrônicos de comunicação/computação. Não se esqueça de indentar corretamente os códigos solicitados. Boa prova!

1. (3 pontos) João está estudando programação em Python e elaborou vários trechos de código para aumentar sua compreensão sobre alguns conceitos básicos. Em cada série, para cada trecho de código elaborado por ele, indique o que será escrito quando os programas forem executados. Caso o programa não imprima nenhum caractere, indique "Nada será escrito". Caso algum erro seja encontrado, indique o motivo e marque na coluna da esquerda a linha em que ele ocorre.

a) **Listas e sublistas**

Programa	O que será escrito

`lista = [10, 20, 30]`
`lista[1]= "palavra"`
`print(lista)`
	`[10, 'palavra', 30]`

`lista = [10, 20]`
`lista[2] = 4`
`print(lista)`
	`atribuição fora dos limites do vetor`

`lista = [10, 20, 30]`
`lista[0]= [5, 10]`
`print(lista)`
	`[[5, 10], 20, 30]`

`lista = [10, 20, 30]`
`aux = lista`
`aux[0] = 20`
`print(lista)`
	`[20, 20, 30]`

`sublista = [1]`
`lista = [sublista, sublista]`
`print(lista)`
`sublista[0] = 2`
`print(lista)`
	`[[1], [1]]`
	`[[2], [2]]`

`m = [[6, 3, 2],`
`  [7, 2, 0],`
`  [2, 1, 0]]`
`for i in range(len(m)) :`
`  m[0][i] += m[1][i]`
`print(m)`
	`[[13, 5, 2], [7, 2, 0], [2, 1, 0]]`

b) **Funções, passagem de parâmetros e escopo de variáveis**

Programa	O que será escrito

`def soma(a, b):`
`  print(a + b)`
`soma(10, 20)`
	`30`

`def soma(a, b):`
`  return a + b`
`  print("soma =", a + b)`
`soma(10, 20)`
	`Nada será escrito.`

`def soma(a, b, c):`
`  return a + b + c`
`soma(0, 5)`
	`chamada a função soma() com número incorreto de parâmetros.`

`def soma(a, b):`
`  c = a + b`
`  return c`
`soma(10, 20)`
`print(c)`
	`nome c não foi definido`

`def soma(a, b) :`
`  c = a + b`
`  a = 0`
`  a = b`
`  return c`
`a = 10`
`b = 20`
`c = soma(a, b)`
`print(a, b, c)`
	`10 20 30`

`def soma(a, b):`
`  return a + b`
`def subtrai(a, b):`
`  return a - b`
`print(subtrai(soma(10, 20), soma(3, 4)))`
	`23`

c) **Tuplas e dicionários**

Programa	O que será escrito

`tupla = (0,1,2,3)`
`tupla[0] = -1`
`print(tupla)`
	`objeto tupla não permite atribuição`

`numeros = (2,4,6)`
`letras = ("I", "J", "K")`
`tupla = numeros + letras`
`print(tupla)`
	`(2, 4, 6, 'I', 'J', 'K')`

`letra_num = {"A":2, "B":4, "C":6}`
`letra_num["D"] = 8`
`print("A:", letra_num["A"])`
`print("D:", letra_num["D"])`
	`A: 2`
	`D: 8`

`letra_num = {"A":0, "B":1, "C":2}`
`num_letra = {0:"x", 1:"y", 3:"z"}`
`for letra in letra_num:`
`  num = letra_num[letra]`
`  if num in num_letra :`
`  print(letra,":", num_letra[num])`
	`A : x`
	`B : y`

2. (2.5 pontos) Seja $\mathcal{P}$ a propriedade que indica que uma matriz é **quadrada** e que os elementos em sua **diagonal principal** são **iguais** a **1** e os outros são **iguais** a **0**. Maria gostaria de escrever uma função em Python para verficar a propriedade $\mathcal{P}$ em uma matriz representada por uma lista de listas contendo números inteiros.

Inicialmente, Maria precisa identificar a propriedade em algumas matrizes. Indique para cada estrutura de dados apresentada abaixo se esta representa uma matriz que respeita a propriedade $\mathcal{P}$ ou não. Em caso de violação da propriedade, escreva uma justificativa.

`[[2, 0, 0],`
`[0, 1, 0],`
`[0, 0, 1]]`
	`[[2, 0, 0, 0],`
	`[0, 1, 0, 0],`
	`[0, 0, 1, 0]]`
		`[[1, 0, 0],`
		`[0, 1, 0],`
		`[0, 0, 1]]`
			`[[1, 0, 0],`
			`[0, 1, 0, 0],`
			`[0, 0, 1]]`

`Não garante P.` `m[0][0] != 1`	`Não garante P.` `Não é quadrada.`	`Garante P`	`Não garante P.` `Não é quadrada.`

Em seguida, escreva a **função** `verifica_P(m)` que recebe uma lista de listas de inteiros como parâmetro e retorna `True` se a matriz atende a propriedade $\mathcal{P}$ ou `False` caso contrário.

```
def verifica_P(m) :
    n = len(m)
    for i in range(n) :
        if len(m[i]) != n :
            return False
        for j in range(n) :
            if i == j and m[i][j] != 1 :
                return False
            if i != j and m[i][j] != 0 :
                return False
    return True
```

3. (2 pontos) Rafaela começou a estudar o algoritmo de ordenação QuickSort. Antes de analisar as chamadas recursivas, ela vai explorar o algoritmo que posiciona corretamente o pivô a cada passo.

O elemento escolhido para ser o pivô pode ser, por exemplo, o primeiro elemento do vetor. Após a execução da função `particiona`, à esquerda do pivô ficarão os elementos com valores menores do que o valor do pivô e à direita do pivô ficarão os elementos com valores maiores. Veja o exemplo a seguir:

12	14	6	7	18	2	21

7	2	6	**12**	18	14	21

Observe o código abaixo, em que a partição pode atuar na lista inteira ou em uma sublista, de acordo com os valores dos índices **e** e **d** passados como parâmetro para a função. Para acompanhar os passos, Rafaela introduziu algumas chamadas ao comando `print` em pontos estratégicos, logo após as trocas dos elementos.

```
def particiona(lista, e, d) :
    valor_pivo = lista[e]
    pos_pivo = e
    e += 1
    print("e =", e, "d =", d, lista)
    while e < d :
        while e <= d and valor_pivo >= lista[e] :
            e += 1
        while e <= d and valor_pivo <= lista[d] :
            d -= 1
        if e > d :
            break
        lista[e], lista[d] = lista[d], lista[e]
        print("e =", e, "d =", d, lista)
    lista[pos_pivo], lista[d] = lista[d], lista[pos_pivo]
    print("e =", e, "d =", d, lista)
    return d
```

Para a lista e as chamadas abaixo indique o resultado dos comandos `print` seguindo o modelo. Ao final, indique o valor das variáveis `ponto1` e `ponto2`.

```
lista = [22, 10, 6, 30, 19, 2, 27]
ponto1 = particiona(lista, 0, len(lista) - 1)
ponto2 = particiona(lista, 0, ponto1 - 1)
```

e = [1] d = [6] [[22], [10], [6], [30], [19], [2], [27]]

e = [3] d = [5] [[22], [10], [6], [2], [19], [30], [27]]

e = [5] d = [4] [[19], [10], [6], [2], [22], [30], [27]]

e = [1] d = [3] [[19], [10], [6], [2], [22], [30], [27]]

e = [4] d = [3] [[2], [10], [6], [19], [22], [30], [27]]

ponto1 = [4] ponto2 = [3]

4. (2.5 pontos) João agora está estudando recursão e elaborou nova série de testes. Como na questão 1, você deve escrever a saída do programa ou indicar "Nada será escrito". Caso algum erro seja encontrado, indique o motivo e marque na coluna da esquerda a linha em que ele ocorre. Você também deve indicar claramente se o programa tiver entrado em loop.

Programa	O que será escrito

`def rec(n):`
`  print("n =", n)`
`  if n == 1 or n == 0:`
`  return n`
`  rec(n-3)`
`rec(9)`
	`n = 9`
	`n = 6`
	`n = 3`
	`n = 0`

`def rec(n):`
`  if (n >= 2):`
`  return n`
`  r = rec(n+1) + rec(n+2)`
`  print ("n =", n, "r =", r)`
`  return r`
`rec(0)`
	`n = 1 r = 5`
	`n = 0 r = 7`

`def rec(n):`
`  if n < 2 :`
`  n += 1`
`  return n + rec (n+1)`
`rec(0)`
	`Nada será escrito`
	`Loop infinito (comentário sobre estouro de pilha não é obrigatório)`

João também fez testes para comparar versões recursivas e iterativas (não recursivas) dos mesmos algoritmos. Seguindo esta ideia, complete a tabela abaixo, escrevendo a versão iterativa do código à esquerda.

Versão recursiva	Versão iterativa

`def rec(n):`
`  print(n)`
`  if n == 0:`
`  return 0`
`  else:`
`  return n + rec(n-1)`
	`def iterativa(n) :`
	`  aux = 0`
	`  while (n >= 0) :`
	`  print(n)`
	`  aux += n`
	`  n = n-1`
	`  return aux`
	`ou`
	`def iterativa(n) :`
	`  aux = 0`
	`  for i in range(n,-1,-1) :`
	`  print(i)`
	`  aux += i`
	`  return aux`

MC102—Algoritmos e Programação de Computadores

Prova 2

GABARITO

Segundo Semestre de 2018

Questão	Nota
1	
2	
3	
4	
Total	

Nome: RA:

Importante: As respostas deverão ser escritas nos espaços demarcados, opcionalmente a lápis. Não é permitida a consulta a qualquer material, nem o uso de celulares ou outros dispositivos eletrônicos de comunicação/computação. Não se esqueça de indentar corretamente os códigos solicitados. Boa prova!

1. (3 pontos) João está estudando programação em Python e elaborou vários trechos de código para aumentar sua compreensão sobre alguns conceitos básicos. Em cada série, para cada trecho de código elaborado por ele, indique o que será escrito quando os programas forem executados. Caso o programa não imprima nenhum caractere, indique "Nada será escrito". Caso algum erro seja encontrado, indique o motivo e marque na coluna da esquerda a linha em que ele ocorre.

a) **Listas e sublistas**

Programa	O que será escrito

`lista = [20, 30]`
`lista[2] = 5`
`print(lista)`
	`atribuição fora dos limites do vetor`

`lista = [1, 2, 3]`
`lista[2]= "palavra"`
`print(lista)`
	`[1, 2, 'palavra']`

`lista = [1, 2, 3]`
`aux = lista`
`aux[0] = 0`
`print(lista)`
	`[0, 2, 3]`

`lista = [10, 20, 30]`
`lista[2]= [30, 40]`
`print(lista)`
	`[10, 20, [30, 40]]`

`sublista = [3]`
`lista = [sublista, sublista]`
`print(lista)`
`sublista[0] = 5`
`print(lista)`
	`[[3], [3]]`
	`[[5], [5]]`

`m = [[9, 3, 2],`
`  [3, 2, 1],`
`  [2, 1, 4]]`
`for i in range(len(m)) :`
`  m[0][i] += m[2][i]`
`print(m)`
	`[[11, 4, 6], [3, 2, 1], [2, 1, 4]]`

b) **Funções, passagem de parâmetros e escopo de variáveis**

Programa	O que será escrito

`def subtrai(a, b):`
`  print(a - b)`
`subtrai(20, 10)`
	`10`

`def soma(a, b):`
`  return a + b`
`  print("soma =", a + b)`
`soma(10, 20)`
	`Nada será escrito.`

`def soma(a, b, c):`
`  return a + b + c`
`soma(0, 5)`
	`chamada a função soma() com número incorreto de parâmetros.`

`def soma(a, b):`
`  c = a + b`
`  return c`
`soma(10, 20)`
`print(c)`
	`nome c não foi definido`

`def soma(a, b) :`
`  c = a + b`
`  a = 0`
`  a = b`
`  return c`
`a = 5`
`b = 15`
`c = soma(a, b)`
`print(a, b, c)`
	`5 15 20`

`def soma(a, b):`
`  return a + b`
`def subtrai(a, b):`
`  return a - b`
`print(soma(subtrai(20, 10), soma(3, 4)))`
	`17`

c) **Tuplas e dicionários**

Programa	O que será escrito

`numeros = (1,2,3)`
`letras = ("A", "E", "I")`
`tupla = numeros + letras`
`print(tupla)`
	`(1, 2, 3, 'A', 'E', 'I')`

`tupla = (0,1,2,3)`
`tupla[2] = 3`
`print(tupla)`
	`objeto tupla não permite atribuição`

`letra_num = {"A":10, "B":20, "C":30}`
`letra_num["D"] = 40`
`print("B:", letra_num["B"])`
`print("D:", letra_num["D"])`
	`B: 20`
	`D: 40`

`letra_num = {"A":0, "B":1, "C":2}`
`num_letra = {0:"r", 1:"s", 3:"t"}`
`for letra in letra_num:`
`  num = letra_num[letra]`
`  if num in num_letra :`
`  print(letra,":", num_letra[num])`
	`A : r`
	`B : s`

2. (2.5 pontos) Seja $\mathcal{P}$ a propriedade que indica que uma matriz é **quadrada** e que os elementos em sua **diagonal principal** são **diferentes** de **0** e os outros são **iguais** a **0**. Maria gostaria de escrever uma função em Python para verficar a propriedade $\mathcal{P}$ em uma matriz representada por uma lista de listas contendo números inteiros.

Inicialmente, Maria precisa identificar a propriedade em algumas matrizes. Indique para cada estrutura de dados apresentada abaixo se esta representa uma matriz que respeita a propriedade $\mathcal{P}$ ou não. Em caso de violação da propriedade, escreva uma justificativa.

`[[2, 0, 0],`
`[0, 1, 0],`
`[0, 0, 1]]`
	`[[1, 0, 0],`
	`[0, 1, 0, 0],`
	`[0, 0, 1]]`
		`[[2, 0, 0, 0],`
		`[0, 1, 0, 0],`
		`[0, 0, 1, 0]]`
			`[[0, 1, 0],`
			`[0, 1, 0],`
			`[0, 0, 1]]`

`Garante P`	`Não garante P.` `Não é quadrada.`	`Não garante P.` `Não é quadrada.`	`Não garante P.` `m[0][0] == 0`

Em seguida, escreva a **função** `verifica_P(m)` que recebe uma lista de listas de inteiros como parâmetro e retorna `True` se a matriz atende a propriedade $\mathcal{P}$ ou `False` caso contrário.

```
def verifica_P(m) :
    n = len(m)
    for i in range(n) :
        if len(m[i]) != n :
            return False
        for j in range(n) :
            if i == j and m[i][j] == 0 :
                return False
            if i != j and m[i][j] != 0 :
                return False
    return True
```

3. (2 pontos) Rafaela começou a estudar o algoritmo de ordenação QuickSort. Antes de analisar as chamadas recursivas, ela vai explorar o algoritmo que posiciona corretamente o pivô a cada passo.

O elemento escolhido para ser o pivô pode ser, por exemplo, o primeiro elemento do vetor. Após a execução da função `particiona`, à esquerda do pivô ficarão os elementos com valores menores do que o valor do pivô e à direita do pivô ficarão os elementos com valores maiores. Veja o exemplo a seguir:

12	14	6	7	18	2	21

7	2	6	**12**	18	14	21

Observe o código abaixo, em que a partição pode atuar na lista inteira ou em uma sublista, de acordo com os valores dos índices **e** e **d** passados como parâmetro para a função. Para acompanhar os passos, Rafaela introduziu algumas chamadas ao comando `print` em pontos estratégicos, logo após as trocas dos elementos.

```
def particiona(lista, e, d) :
    valor_pivo = lista[e]
    pos_pivo = e
    e += 1
    print("e =", e, "d =", d, lista)
    while e < d :
        while e <= d and valor_pivo >= lista[e] :
            e += 1
        while e <= d and valor_pivo <= lista[d] :
            d -= 1
        if e > d :
            break
        lista[e], lista[d] = lista[d], lista[e]
        print("e =", e, "d =", d, lista)
    lista[pos_pivo], lista[d] = lista[d], lista[pos_pivo]
    print("e =", e, "d =", d, lista)
    return d
```

Para a lista e as chamadas abaixo indique o resultado dos comandos `print` seguindo o modelo. Ao final, indique o valor das variáveis `ponto1` e `ponto2`.

```
lista = [25, 12, 5, 40, 18, 4, 30]
ponto1 = particiona(lista, 0, len(lista) - 1)
ponto2 = particiona(lista, 0, ponto1 - 1)
```

e = [1] d = [6] [[25], [12], [5], [40], [18], [4], [30]]

e = [3] d = [5] [[25], [12], [5], [4], [18], [40], [30]]

e = [5] d = [4] [[18], [12], [5], [4], [25], [40], [30]]

e = [1] d = [3] [[18], [12], [5], [4], [25], [40], [30]]

e = [4] d = [3] [[4], [12], [5], [18], [25], [40], [30]]

ponto1 = [4] ponto2 = [3]

4. (2.5 pontos) João agora está estudando recursão e elaborou nova série de testes. Como na questão 1, você deve escrever a saída do programa ou indicar "Nada será escrito". Caso algum erro seja encontrado, indique o motivo e marque na coluna da esquerda a linha em que ele ocorre. Você também deve indicar claramente se o programa tiver entrado em loop.

Programa	O que será escrito

`def rec(n):`
`  print("n =", n)`
`  if n == 0:`
`  return 0`
`  rec(n-1)`
`rec(3)`
	`n = 3`
	`n = 2`
	`n = 1`
	`n = 0`

`def rec(n):`
`  if (n >= 3):`
`  return n`
`  r = rec(n+1) + rec(n+2)`
`  print ("n =", n, "r =", r)`
`  return r`
`rec(1)`
	`n = 2 r = 7`
	`n = 1 r = 10`

`def rec(n):`
`  if n < 5 :`
`  n += 1`
`  return n + rec (n+1)`
`rec(0)`
	`Nada será escrito`
	`Loop infinito (comentário sobre estouro de pilha não é obrigatório)`

João também fez testes para comparar versões recursivas e iterativas (não recursivas) dos mesmos algoritmos. Seguindo esta ideia, complete a tabela abaixo, escrevendo a versão iterativa do código à esquerda.

Versão recursiva	Versão iterativa

`def rec(n):`
`  print(n)`
`  if n == 0:`
`  return 0`
`  else:`
`  return n + rec(n-1)`
	`def iterativa(n) :`
	`  aux = 0`
	`  while (n >= 0) :`
	`  print(n)`
	`  aux += n`
	`  n = n-1`
	`  return aux`
	`ou`
	`def iterativa(n) :`
	`  aux = 0`
	`  for i in range(n,-1,-1) :`
	`  print(i)`
	`  aux += i`
	`  return aux`

MC102—Algoritmos e Programação de Computadores

Prova 2

GABARITO

Segundo Semestre de 2018

Questão	Nota
1	
2	
3	
4	
Total	

Nome: RA:

Importante: As respostas deverão ser escritas nos espaços demarcados, opcionalmente a lápis. Não é permitida a consulta a qualquer material, nem o uso de celulares ou outros dispositivos eletrônicos de comunicação/computação. Não se esqueça de indentar corretamente os códigos solicitados. Boa prova!

1. (3 pontos) João está estudando programação em Python e elaborou vários trechos de código para aumentar sua compreensão sobre alguns conceitos básicos. Em cada série, para cada trecho de código elaborado por ele, indique o que será escrito quando os programas forem executados. Caso o programa não imprima nenhum caractere, indique "Nada será escrito". Caso algum erro seja encontrado, indique o motivo e marque na coluna da esquerda a linha em que ele ocorre.

a) **Listas e sublistas**

Programa	O que será escrito

`lista = [3, 4, 5]`
`lista[0]= "palavra"`
`print(lista)`
	`['palavra', 4, 5]`

`lista = [1, 2]`
`lista[2] = 7`
`print(lista)`
	`atribuição fora dos limites do vetor`

`lista = [10, 20, 30]`
`lista[2]= [20, 10]`
`print(lista)`
	`[10, 20, [20, 10]]`

`lista = [1, 2, 3]`
`aux = lista`
`aux[2] = 5`
`print(lista)`
	`[1, 2, 5]`

`sublista = [5]`
`lista = [sublista, sublista]`
`print(lista)`
`sublista[0] = 3`
`print(lista)`
	`[[5], [5]]`
	`[[3], [3]]`

`m = [[1, 3, 2],`
`[5, 2, 0],`
`[4, 1, 1]]`
`for i in range(len(m)) :`
`m[1][i] += m[2][i]`
`print(m)`
	`[[1, 3, 2], [9, 3, 1], [4, 1, 1]]`

b) **Funções, passagem de parâmetros e escopo de variáveis**

Programa	O que será escrito

`def soma(a, b, c):`
`  return a + b + c`
`soma(10, 15)`
	`chamada a função soma() com número incorreto de parâmetros.`

`def soma(a, b):`
`  print(a + b)`
`soma(10, 20)`
	`30`

`def soma(a, b):`
`  return a + b`
`  print("soma =", a + b)`
`soma(15, 2)`
	`Nada será escrito.`

`def soma(a, b):`
`  c = a + b`
`  return c`
`soma(10, 20)`
`print(c)`
	`nome c não foi definido`

`def soma(a, b) :`
`  c = a + b`
`  a = 0`
`  a = b`
`  return c`
`a = 4`
`b = 5`
`c = soma(a, b)`
`print(a, b, c)`
	`4 5 9`

`def soma(a, b):`
`  return a + b`
`def subtrai(a, b):`
`  return a - b`
`print(soma(soma(10, 20), subtrai(4, 2)))`
	`32`

c) **Tuplas e dicionários**

Programa	O que será escrito

`tupla = (0,1,2,3)`
`tupla[0] = 4`
`print(tupla)`
	`objeto tupla não permite atribuição`

`numeros = (10,20,30)`
`letras = ("A", "B", "C")`
`tupla = letras + numeros`
`print(tupla)`
	`('A', 'B', 'C', 10, 20, 30)`

`letra_num = {"A":0, "B":1, "C":2}`
`letra_num["D"] = 3`
`print("A:", letra_num["A"])`
`print("D:", letra_num["D"])`
	`A: 0`
	`D: 3`

`letra_num = {"A":0, "B":1, "C":2}`
`num_letra = {0:"i", 1:"j", 3:"k"}`
`for letra in letra_num:`
`  num = letra_num[letra]`
`  if num in num_letra :`
`  print(letra,":", num_letra[num])`
	`A : i`
	`B : j`

2. (2.5 pontos) Seja $\mathcal{P}$ a propriedade que indica que uma matriz é **quadrada** e que os elementos em sua **diagonal principal** são **iguais** a **1** e os outros são **iguais** a **0**. Maria gostaria de escrever uma função em Python para verficar a propriedade $\mathcal{P}$ em uma matriz representada por uma lista de listas contendo números inteiros.

Inicialmente, Maria precisa identificar a propriedade em algumas matrizes. Indique para cada estrutura de dados apresentada abaixo se esta representa uma matriz que respeita a propriedade $\mathcal{P}$ ou não. Em caso de violação da propriedade, escreva uma justificativa.

`[[2, 0, 0],`
`[0, 1, 0],`
`[0, 0, 1]]`
	`[[2, 0, 0, 0],`
	`[0, 1, 0, 0],`
	`[0, 0, 1, 0]]`
		`[[1, 0, 0],`
		`[0, 1, 0],`
		`[0, 0, 1]]`
			`[[1, 0, 0],`
			`[0, 1, 0, 0],`
			`[0, 0, 1]]`

`Não garante P.` `m[0][0] != 1`	`Não garante P.` `Não é quadrada.`	`Garante P`	`Não garante P.` `Não é quadrada.`

Em seguida, escreva a **função** `verifica_P(m)` que recebe uma lista de listas de inteiros como parâmetro e retorna `True` se a matriz atende a propriedade $\mathcal{P}$ ou `False` caso contrário.

```
def verifica_P(m) :
    n = len(m)
    for i in range(n) :
        if len(m[i]) != n :
            return False
        for j in range(n) :
            if i == j and m[i][j] != 1 :
                return False
            if i != j and m[i][j] != 0 :
                return False
    return True
```

3. (2 pontos) Rafaela começou a estudar o algoritmo de ordenação QuickSort. Antes de analisar as chamadas recursivas, ela vai explorar o algoritmo que posiciona corretamente o pivô a cada passo.

O elemento escolhido para ser o pivô pode ser, por exemplo, o primeiro elemento do vetor. Após a execução da função `particiona`, à esquerda do pivô ficarão os elementos com valores menores do que o valor do pivô e à direita do pivô ficarão os elementos com valores maiores. Veja o exemplo a seguir:

12	14	6	7	18	2	21

7	2	6	**12**	18	14	21

Observe o código abaixo, em que a partição pode atuar na lista inteira ou em uma sublista, de acordo com os valores dos índices **e** e **d** passados como parâmetro para a função. Para acompanhar os passos, Rafaela introduziu algumas chamadas ao comando `print` em pontos estratégicos, logo após as trocas dos elementos.

```
def particiona(lista, e, d) :
    valor_pivo = lista[e]
    pos_pivo = e
    e += 1
    print("e =", e, "d =", d, lista)
    while e < d :
        while e <= d and valor_pivo >= lista[e] :
            e += 1
        while e <= d and valor_pivo <= lista[d] :
            d -= 1
        if e > d :
            break
        lista[e], lista[d] = lista[d], lista[e]
        print("e =", e, "d =", d, lista)
    lista[pos_pivo], lista[d] = lista[d], lista[pos_pivo]
    print("e =", e, "d =", d, lista)
    return d
```

Para a lista e as chamadas abaixo indique o resultado dos comandos `print` seguindo o modelo. Ao final, indique o valor das variáveis `ponto1` e `ponto2`.

```
lista = [21, 10, 3, 27, 13, 1, 25]
ponto1 = particiona(lista, 0, len(lista) - 1)
ponto2 = particiona(lista, 0, ponto1 - 1)
```

e = [1] d = [6] [21 , 10 , 3 , 27 , 13 , 1 , 25]

e = [3] d = [5] [21 , 10 , 3 , 1 , 13 , 27 , 25]

e = [5] d = [4] [13 , 10 , 3 , 1 , 21 , 27 , 25]

e = [1] d = [3] [13 , 10 , 3 , 1 , 21 , 27 , 25]

e = [4] d = [3] [1 , 10 , 3 , 13 , 21 , 27 , 25]

ponto1 = [4] ponto2 = [3]

4. (2.5 pontos) João agora está estudando recursão e elaborou nova série de testes. Como na questão 1, você deve escrever a saída do programa ou indicar "Nada será escrito". Caso algum erro seja encontrado, indique o motivo e marque na coluna da esquerda a linha em que ele ocorre. Você também deve indicar claramente se o programa tiver entrado em loop.

Programa	O que será escrito

`def rec(n):`
`  print("n =", n)`
`  if n == 1 or n == 0:`
`  return 0`
`  rec(n-2)`
`rec(6)`
	`n = 6`
	`n = 4`
	`n = 2`
	`n = 0`

`def rec(n):`
`  if (n >= 1):`
`  return n`
`  r = rec(n+1) + rec(n+2)`
`  print ("n =", n, "r =", r)`
`  return r`
`rec(-1)`
	`n = 0 r = 3`
	`n = -1 r = 4`

`def rec(n):`
`  if n < 2 :`
`  n += 1`
`  return n + rec (n+1)`
`rec(0)`
	`Nada será escrito`
	`Loop infinito (comentário sobre estouro de pilha não é obrigatório)`

João também fez testes para comparar versões recursivas e iterativas (não recursivas) dos mesmos algoritmos. Seguindo esta ideia, complete a tabela abaixo, escrevendo a versão iterativa do código à esquerda.

Versão recursiva	Versão iterativa

`def rec(n):`
`  print(n)`
`  if n == 0:`
`  return 0`
`  else:`
`  return n + rec(n-1)`
	`def iterativa(n) :`
	`  aux = 0`
	`  while (n >= 0) :`
	`  print(n)`
	`  aux += n`
	`  n = n-1`
	`  return aux`
	`ou`
	`def iterativa(n) :`
	`  aux = 0`
	`  for i in range(n,-1,-1) :`
	`  print(i)`
	`  aux += i`
	`  return aux`

MC102—Algoritmos e Programação de Computadores

Prova 2

GABARITO

Segundo Semestre de 2018

Questão	Nota
1	
2	
3	
4	
Total	

Nome: RA:

Importante: As respostas deverão ser escritas nos espaços demarcados, opcionalmente a lápis. Não é permitida a consulta a qualquer material, nem o uso de celulares ou outros dispositivos eletrônicos de comunicação/computação. Não se esqueça de indentar corretamente os códigos solicitados. Boa prova!

1. (3 pontos) João está estudando programação em Python e elaborou vários trechos de código para aumentar sua compreensão sobre alguns conceitos básicos. Em cada série, para cada trecho de código elaborado por ele, indique o que será escrito quando os programas forem executados. Caso o programa não imprima nenhum caractere, indique "Nada será escrito". Caso algum erro seja encontrado, indique o motivo e marque na coluna da esquerda a linha em que ele ocorre.

a) **Listas e sublistas**

Programa	O que será escrito

`lista = [4, 8]`
`lista[2] = 10`
`print(lista)`
	`atribuição fora dos limites do vetor`

`lista = [0, 5, 10]`
`lista[2]= "palavra"`
`print(lista)`
	`[0, 5, 'palavra']`

`lista = [5, 10, 15]`
`aux = lista`
`aux[2] = 20`
`print(lista)`
	`[5, 10, 20]`

`lista = [10, 20, 30]`
`lista[0]= [10, 15]`
`print(lista)`
	`[[10, 15], 20, 30]`

`sublista = [10]`
`lista = [sublista, sublista]`
`print(lista)`
`sublista[0] = 20`
`print(lista)`
	`[[10], [10]]`
	`[[20], [20]]`

`m = [[2, 3, 0],`
`[7, 4, 1],`
`[2, 1, 5]]`
`for i in range(len(m)) :`
`m[0][i] = m[1][i] + m[2][i]`
`print(m)`
	`[[9, 5, 6], [7, 4, 1], [2, 1, 5]]`

b) **Funções, passagem de parâmetros e escopo de variáveis**

Programa	O que será escrito

`def soma(a, b):`
`  return a + b`
`  print("soma =", a + b)`
`soma(10, 20)`
	`Nada será escrito.`

`def soma(a, b, c):`
`  return a + b + c`
`soma(0, 5)`
	`chamada a função soma() com número incorreto de parâmetros.`

`def soma(a, b):`
`  print(a + b)`
`soma(5, 20)`
	`25`

`def subtrai(a, b):`
`  c = a - b`
`  return c`
`soma(20, 10)`
`print(c)`
	`nome soma não foi definido`

`def soma(a, b) :`
`  c = a + b`
`  a = 0`
`  a = b`
`  return c`
`a = 7`
`b = 8`
`c = soma(a, b)`
`print(a, b, c)`
	`7 8 15`

`def soma(a, b):`
`  return a + b`
`def subtrai(a, b):`
`  return a - b`
`print(subtrai(soma(10, 20), subtrai(4, 2)))`
	`28`

c) **Tuplas e dicionários**

Programa	O que será escrito

`numeros = (0,1,2)`
`letras = ("X", "Y", "Z")`
`tupla = numeros + letras`
`print(tupla)`
	`(0, 1, 2, 'X', 'Y', 'Z')`

`tupla = (0,1,2,3)`
`tupla[1] = 2`
`print(tupla)`
	`objeto tupla não permite atribuição`

`letra_num = {"X":0, "Y":1, "W":2}`
`letra_num["Z"] = 3`
`print("Z:", letra_num["Z"])`
`print("Y:", letra_num["Y"])`
	`Z: 3`
	`Y: 1`

`letra_num = {"A":0, "B":1, "C":2}`
`num_letra = {0:"a", 1:"b", 3:"c"}`
`for letra in letra_num:`
`  num = letra_num[letra]`
`  if num in num_letra :`
`  print(letra,":", num_letra[num])`
	`A : a`
	`B : b`

2. (2.5 pontos) Seja $\mathcal{P}$ a propriedade que indica que uma matriz é **quadrada** e que os elementos em sua **diagonal principal** são **diferentes** de **0** e os outros são **iguais** a **0**. Maria gostaria de escrever uma função em Python para verficar a propriedade $\mathcal{P}$ em uma matriz representada por uma lista de listas contendo números inteiros.

Inicialmente, Maria precisa identificar a propriedade em algumas matrizes. Indique para cada estrutura de dados apresentada abaixo se esta representa uma matriz que respeita a propriedade $\mathcal{P}$ ou não. Em caso de violação da propriedade, escreva uma justificativa.

`[[2, 0, 0],`
`[0, 1, 0],`
`[0, 0, 1]]`
	`[[1, 0, 0],`
	`[0, 1, 0, 0],`
	`[0, 0, 1]]`
		`[[2, 0, 0, 0],`
		`[0, 1, 0, 0],`
		`[0, 0, 1, 0]]`
			`[[0, 1, 0],`
			`[0, 1, 0],`
			`[0, 0, 1]]`

`Garante P`	`Não garante P.` `Não é quadrada.`	`Não garante P.` `Não é quadrada.`	`Não garante P.` `m[0][0] == 0`

Em seguida, escreva a **função** `verifica_P(m)` que recebe uma lista de listas de inteiros como parâmetro e retorna `True` se a matriz atende a propriedade $\mathcal{P}$ ou `False` caso contrário.

```
def verifica_P(m) :
    n = len(m)
    for i in range(n) :
        if len(m[i]) != n :
            return False
        for j in range(n) :
            if i == j and m[i][j] == 0 :
                return False
            if i != j and m[i][j] != 0 :
                return False
    return True
```

3. (2 pontos) Rafaela começou a estudar o algoritmo de ordenação QuickSort. Antes de analisar as chamadas recursivas, ela vai explorar o algoritmo que posiciona corretamente o pivô a cada passo.

O elemento escolhido para ser o pivô pode ser, por exemplo, o primeiro elemento do vetor. Após a execução da função `particiona`, à esquerda do pivô ficarão os elementos com valores menores do que o valor do pivô e à direita do pivô ficarão os elementos com valores maiores. Veja o exemplo a seguir:

12	14	6	7	18	2	21

7	2	6	**12**	18	14	21

Observe o código abaixo, em que a partição pode atuar na lista inteira ou em uma sublista, de acordo com os valores dos índices **e** e **d** passados como parâmetro para a função. Para acompanhar os passos, Rafaela introduziu algumas chamadas ao comando `print` em pontos estratégicos, logo após as trocas dos elementos.

```
def particiona(lista, e, d) :
    valor_pivo = lista[e]
    pos_pivo = e
    e += 1
    print("e =", e, "d =", d, lista)
    while e < d :
        while e <= d and valor_pivo >= lista[e] :
            e += 1
        while e <= d and valor_pivo <= lista[d] :
            d -= 1
        if e > d :
            break
        lista[e], lista[d] = lista[d], lista[e]
        print("e =", e, "d =", d, lista)
    lista[pos_pivo], lista[d] = lista[d], lista[pos_pivo]
    print("e =", e, "d =", d, lista)
    return d
```

Para a lista e as chamadas abaixo indique o resultado dos comandos `print` seguindo o modelo. Ao final, indique o valor das variáveis `ponto1` e `ponto2`.

```
lista = [24, 11, 6, 35, 15, 1, 25]
ponto1 = particiona(lista, 0, len(lista) - 1)
ponto2 = particiona(lista, 0, ponto1 - 1)
```

e = [1] d = [6] [[24], [11], [6], [35], [15], [1], [25]]

e = [3] d = [5] [[24], [11], [6], [1], [15], [35], [25]]

e = [5] d = [4] [[15], [11], [6], [1], [24], [35], [25]]

e = [1] d = [3] [[15], [11], [6], [1], [24], [35], [25]]

e = [4] d = [3] [[1], [11], [6], [15], [24], [35], [25]]

ponto1 = [4] ponto2 = [3]

4. (2.5 pontos) João agora está estudando recursão e elaborou nova série de testes. Como na questão 1, você deve escrever a saída do programa ou indicar "Nada será escrito". Caso algum erro seja encontrado, indique o motivo e marque na coluna da esquerda a linha em que ele ocorre. Você também deve indicar claramente se o programa tiver entrado em loop.

Programa	O que será escrito

`def rec(n):`
`  print("n =", n)`
`  if n == 1:`
`  return 1`
`  rec(n-1)`
`rec(4)`
	`n = 4`
	`n = 3`
	`n = 2`
	`n = 1`

`def rec(n):`
`  if (n >= 2):`
`  return n`
`  r = rec(n+1) + rec(n+2)`
`  print ("n =", n, "r =", r)`
`  return r`
`rec(0)`
	`n = 1 r = 5`
	`n = 0 r = 7`

`def rec(n):`
`  if n < 2 :`
`  n += 1`
`  return n + rec (n+1)`
`rec(0)`
	`Nada será escrito`
	`Loop infinito (comentário sobre estouro de pilha não é obrigatório)`

João também fez testes para comparar versões recursivas e iterativas (não recursivas) dos mesmos algoritmos. Seguindo esta ideia, complete a tabela abaixo, escrevendo a versão iterativa do código à esquerda.

Versão recursiva	Versão iterativa

`def rec(n):`
`  print(n)`
`  if n == 0:`
`  return 0`
`  else:`
`  return n + rec(n-1)`
	`def iterativa(n) :`
	`  aux = 0`
	`  while (n >= 0) :`
	`  print(n)`
	`  aux += n`
	`  n = n-1`
	`  return aux`
	`ou`
	`def iterativa(n) :`
	`  aux = 0`
	`  for i in range(n,-1,-1) :`
	`  print(i)`
	`  aux += i`
	`  return aux`