

Questão	1	2	3	4	5	Total
Valor	2	2	2	2	2	10,0
Nota						

Nome: _____ NUSP: _____

Questão 1 (2 pontos):

- a) A função abaixo é recursiva. Escreva uma função equivalente não recursiva.

```
def fr():  
    x = int(input())  
    if x == 0: return 0  
    return 1 / x + fr()
```

A função acima lê um conjunto de valores terminados por zero e devolve a soma dos inversos dos valores lidos.

```
# versão não recursiva  
def fnr():  
    soma = 0.0  
    while True:  
        x = int(input())  
        if x == 0: return soma  
        soma = soma + 1 / x
```

- b) A função abaixo é não recursiva. Escreva uma função equivalente recursiva.

```
def fnr(n):  
    s = 0  
    for k in range(n):  
        x = int(input())  
        s = s + x  
    return s
```

A função acima recebe n, lê n valores e devolve a soma dos valores lidos.

versão recursiva

```
def fr(n):  
    if n == 0: return 0  
    x = int(input())  
    return x + fr(n - 1)
```

Questão 2 (2 pontos):

A prioridade de alguns dos operadores em Python é a seguinte (1 é a maior 15 é a menor):

1	**	exponenciação
3	*, /, %, //	mult., divisão, resto, divisão inteira
4	+, -	soma, subtração
5	<<, >>	operador de bit – deslocamento a dir. ou esq.
6	&	operador de bit – and
7	~, ^	operador de bit – or e or exclusivo
8	>, <, >=, <=	comparações
9	==, !=	comparações – igual e diferente
10	=	atribuição
13	not	operador lógico - not
14	and	operador lógico – and
15	or	operador lógico – or

Supondo que as variáveis A, B, C, D, E sejam numéricas inteiras, traduzir as seguintes expressões para a notação pós-fixa.

a) `A + B // C * D - E`

A	B	C	//	D	*	+	E	-							
---	---	---	----	---	---	---	---	---	--	--	--	--	--	--	--

b) `A + B >= C or A == C and B > D`

A	B	+	C	>=	A	C	==	B	D	>	and	or			
---	---	---	---	----	---	---	----	---	---	---	-----	----	--	--	--

c) `not A + B or C < D`

A	B	+	not	C	D	<	or								
---	---	---	-----	---	---	---	----	--	--	--	--	--	--	--	--

d) `E = A < B ** C or D == A`

(não foi corrigida - atribuição = com precedência incorreta)

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

e) `A << B & C >> D | A ^ B`

A	B	<<	C	D	>>	&	A		B	^					
---	---	----	---	---	----	---	---	--	---	---	--	--	--	--	--

Questão 3 (2 pontos):

Considere as seguintes expressões que já estão em notação pós-fixa. Calcule o valor de cada expressão, supondo que os valores das variáveis A, B, C, D, são os 4 últimos algarismos de seu NUSP:

Exemplo: NUSP = 3254376. Então: A = 4, B = 3, C = 7, D = 6

Para o NUSP acima:

A = 4 B = 3 C = 7 D = 6

a) A B C D + * +

Equivalente a: $A + B * (C + D)$

Resposta: 43

b) A B D A - ** *

Equivalente a: $A * B ** (D - A)$

Resposta: 36

c) A B > C A D * < A D C - == and or

Equivalente a: $A > B \text{ or } C < A * D \text{ and } A == D - C$

Resposta: True

d) A B > A C < not or

Equivalente a: $A > B \text{ or not } A < C$

Resposta: True

e) A B & C | D ^

Equivalente a: $A \& B | C \wedge D$

A: 0100

B: 0011

C: 0111

D: 0110

Resposta: 1

Questão 4 (2 pontos):

Um polinômio pode ser representado por uma lista de valores float, onde o elemento i é o coeficiente de x^i . Por exemplo, a lista:

[1.2, -3.0, 4.5] representa o polinômio: $1.2 - 3.0 x + 4.5 x^2$

Escreva uma classe Polinômio que implementa um Tipo de Dado Abstrato. O método construtor `__init__()` recebe uma lista de valores float e cria um Polinômio com essa lista.

Além do método construtor `__init__()`, escreva também métodos para suportar a soma e a multiplicação de dois Polinômios.

a)

```
class Polinômio:
    def __init__(self, coef):

        def __init__(self, coef):
            # copia a lista de coeficientes
            self._cf = [0] * len(coef)
            for k in range(len(coef)):
                self._cf[k] = coef[k]
```

b)

```
def __add__(pa, pb):

    def __add__(pa, pb):
        lpa = len(pa._cf)
        lpb = len(pb._cf)
        if lpa > lpb:
            # soma contém o Polinomio de maior grau pa
            soma = Polinomio(pa._cf)
            # soma pb
            for k in range(lpb):
                soma._cf[k] += pb._cf[k]
        else:
            # soma contém o Polinomio de maior grau pb
            soma = Polinomio(pb._cf)
            # soma pa
            for k in range(lpa):
                soma._cf[k] += pa._cf[k]
        # parte opcional
        # verifica se grau do polinomio diminuiu
        for k in range(max(lpa, lpb) - 1, -1, -1):
            if soma._cf[k] == 0: soma._cf.pop(k)
        # retorna a soma
        return soma
```

Outra solução:

```
def __add__(pa, pb):
    lpa = len(pa._cf)
    lpb = len(pb._cf)
```

```

# cria polinomio soma com o tamanho do maior
soma = Polinomio([0] * max(lpa, lpb))
for k in range(max(lpa, lpb)):
    if k >= lpa: cfa = 0
    else: cfa = pa._cf[k]
    if k >= lpb: cfb = 0
    else: cfb = pb._cf[k]
    soma._cf[k] = cfa + cfb
# parte opcional
# verifica se grau do polinomio diminuiu
for k in range(max(lpa, lpb) - 1, -1, -1):
    if soma._cf[k] == 0: soma._cf.pop(k)
# retorna a soma
return soma

```

c)

```

def __mul__(pa, pb):

    def __mul__(pa, pb):
        lpa = len(pa._cf)
        lpb = len(pb._cf)
        # define o Polinomio produto
        mult = Polinomio([0] * (lpa + lpb - 1))
        for i in range(lpa):
            for j in range(lpb):
                mult._cf[i + j] += pa._cf[i] * pb._cf[j]
        return mult

```

Questão 5 (2 pontos):

Escreva as duas funções abaixo que manipulam uma fila circular em alocação sequencial:
Considere as seguintes variáveis globais com os valores iniciais abaixo:

```
MAXF = 10          # Tamanho máximo da fila
Fila = [None] * MAXF # lista que conterà a fila
InicioF = FimF = -1 # indicadores de início e fim da fila
```

```
a) # Adiciona x no final da fila.
# Devolve False se fila cheia ou True caso contrário
def Enfila(x):
    global MAXF, Fila, InicioF, FimF
```

```
# Adiciona x no final da fila.
# Devolve False se fila cheia ou True caso contrário
def Enfila(x):
    global MAXF, Fila, InicioF, FimF
    # caso particular - primeiro elemento
    if FimF == -1:
        InicioF = FimF = 0
        Fila[FimF] = x
        return True
    # caso geral
    # verifica se fila está cheia
    if (FimF + 1) % MAXF == InicioF: return False
    # adiciona x no final da fila
    FimF = (FimF + 1) % MAXF
    Fila[FimF] = x
    return True
```

```
b) # Remove elemento do início da fila.
# Devolve False se fila vazia ou True caso contrário
def Desenfila():
    global MAXF, Fila, InicioF, FimF
```

```
# Remove elemento do início da fila.
# Devolve False se fila vazia ou True caso contrário
def Desenfila():
    global MAXF, Fila, InicioF, FimF
    # caso particular - fila vazia
    if InicioF == -1: return False
    # remove elemento
    x = Fila[InicioF]
    Fila[InicioF] = None # não é necessário
    # verifica se fila ficou vazia
    if InicioF == FimF: InicioF = FimF = -1
    else: InicioF = (InicioF + 1) % MAXF
    return True
```