**Lista de Exercícios de Linguagens de Programação II**

**Universidade Federal do Amazonas**

**Departamento de Ciência da Computação**

**Marco Cristo**

**Alunos: Adilson R. A. De Vilar Junior 20510539**

**Crisley Prestes Linhares 20810161**

**Funcional**

1. O que são funções de primeira ordem?

São funções que são tratadas como qualquer outro tipo de dado, podendo inclusive servir como parâmetro de entrada para outras funções e como valor de retorno de outras funções.

1. O que são funções de alta ordem?

Funções como argumentos ou como resultado de outras funções.

1. Quais as três funções de alta ordem mais comumente encontradas em Linguagens Funcionais? Para que servem?

Folding, que é a colocação de um operador entre os elementos de uma lista. Filtering, que signiﬁca ﬁltrar alguns elementos. Map, que é a aplicação de funções a todos os elementos da lista.

1. Quais as características mais importantes do paradigma funcional?

Um programa pode ser visto como um mapeamento de valores de entrada em valores de saída.

O mapeamento é feito de forma direta. O programa é uma função tipicamente composta

por sub-funções mais simples.

O relacionamento entre funções é bastante simples: uma função pode chamar outra ou o

resultado de uma função pode ser usado como argumento de outra.

1. Resolva os seguintes problemas em Python, usando o paradigma funcional (note que não é permitido o uso de métodos da classe *list*, nem instruções de iteração):
2. Escreva uma função que calcule as raízes reais (e complexas) de uma equação quadrática. Os parâmetros para a função são os três coeficientes da equação. Dica: use a função *sqrt*() do pacote *cmath*.

From cmath import sqrt  
 delta = lambda a, b, c: (b\*\*2)-(4\*a\*c)  
 realSqrt = lambda d: sqrt(d).real  
 roots = lambda a, b, c: (  
         ((-b) + realSqrt(delta(a, b, c)) / 2),  
         ((-b) - realSqrt(delta(a, b, c)) / 2),  
         ((-b) + sqrt(delta(a, b, c)) / 2),  
         ((-b) + sqrt(delta(a, b, c)) / 2)  
 )

1. Escreva uma função recursiva que, dados dois parâmetros numéricos, x e y, retorne xy.

une = lambda x, y: x if y == 1 else x\*power(x, y-1)

1. Escreva uma função que retorne o número de zeros em uma lista.

contagemZero = lambda l: len(filter(lambda n: n == 0, l))

1. Escreva uma função que retorne o maior e o menor número de uma lista.

Maior = lambda l: l[0] if len(l) == 1 else l[0] if l[0] > Maior(l[1:]) else Maior(l[1:])  
 Menor = lambda l: l[0] if len(l) == 1 else l[0] if l[0] < Menor(l[1:]) else Menor(l[1:])  
 maioreMenor = lambda l: (Maior(l), Menor(l))

1. Escreva uma função que remova o último elemento de uma lista.

remover = lambda l: [] if len(l) < 2 else l[:-1]

1. Escrevaumafunção que, dada uma lista *l* e um inteiro *n*, é retornada uma lista sem o *n*-ésimo elemento de *l*. Se *n* é maior que o tamanho de *l*, *l* não é modificada.Por exemplo, removaNesimo([1,2,3,4,5,6], 3)  [1,2,4,5,6], removaNesimo([1,2,3,4,5], 42)  [1,2,3,4,5], removaNesimo([2,[1,2],5,[1,[7,[8,2],9],1], 3)  [2,5,[1,[7,[8,2],9],1]

removeN = lambda l, n: l if n-1 > len(l) else l[:n-1] + l[n:]

1. Escreva uma função que verifique se duas listas numéricas têm a mesma estrutura. Duas listas têm a mesma estrutura se cada um dos seus elementos tem a mesma estrutura, não importando os valores numéricos. Por exemplo, as listas [1, [2, [3]], 4] e [5, [6, [7]], 8] enquanto [1, 2, 3, [4]] e [[1], 2, [3, 4]] têm estruturas distintas.

mesmaStruct = lambda l1, l2: False if len(l1) != len(l2) or type(l1[0]) != type(l2[0]) else  
 mesmaStruct (l1[0], l2[0]) if type(l1[0]) == type([]) and type(l2[0]) == type([]) else  
 mesmaStruct (l1[1:], l2[1:]) if len(l1) > 1 and len(l2) > 1 else True if type(l1[0]) == type(l2[0]) else False

1. Escreva uma função que, dada uma lista L, é retornada a lista L em uma forma plana (sem sub-listas). Ou seja, elementos de uma sub-lista se tornam elementos da lista de nível mais alto. Por exemplo, nivelado([2,[1,2],5,[1,[7,[8,2]]])  [2,1,2,5,1,7,8,2], nivelado([1,2,3,4])  [1,2,3,4].

listaPlana = lambda l: [] if len(l) < 1 else l if type(l[0]) != type([]) and len(l) == 1 else [l[0]] +  
 listaPlana (l[1:]) if type(l[0]) != type([]) else   [] + listaPlana(l[0]) + listaPlana(l[1:])

1. Escreva uma função que, dadas duas listas ordenadas de inteiros L1 e L2, retorne a concatenação de L1 e L2, preservando a ordem dos elementos na lista resultante. Por exemplo, concatene([1,3,4,5,7,10],[2,4,6,8,9])  [1,2,3,4,4,5,6,7,8,9,10]; concatene ([1,2],[])  [1,2]; concatene ([],[1,2]) -> [1,2].

concatena = lambda l1, l2: l1 if l2 == [] else l2 if l1 == [] else [l1[0]] + concatena (l1[1:], l2) if l1[0] <=  
 l2[0] else [l2[0]] + concat(l1, l2[1:])