INSTITUTO FEDERAL
Catarinense | Campus Rio do Sul

Curso: Bacharel em Ciência da Computação

Disciplina: Programação Lógica e Funcional

Semestre: 2023/1

Professor: Daniel Gomes Soares

Acadêmico: Rafael José Camargo Bekhauser

LISTA DE EXERCÍCIOS II

Código fonte disponível em: <https://github.com/rafandoo/Programacao-Logica-e-Funcional/tree/main/Funcional/ATV_2_LISTA_2>.

1.1

```haskell
maiorLista :: [Int] -> Int
maiorLista [] = 0
maiorLista (x:xs) = max x (maiorLista xs)
```

1.2

```python
def maiorLista(lista):
    if len(lista) == 1:
        return lista[0]
    else:
        return max(lista[0], maiorLista(lista[1:]))
```

2.1

```haskell
menorLista :: [Int] -> Int
menorLista [] = 1
menorLista (x:xs) = min x (menorLista xs)
```

2.2

```python
def menorLista(lista):
    if len(lista) == 1:
        return lista[0]
```

```
    else:
        return min(lista[0], menorLista(lista[1:]))
```

3.1

```
insereElemento lista valor = if notElem valor lista then lista ++
[valor] else lista
```

3.2

```
def insere(lista, valor):
    if not valor in lista:
        lista.append(valor)
    return lista
```

4.1

```
listaSup lista valor = [x | x <- lista, x > valor]
```

4.2

```
def listaSup(lista, valor):
    novaLista = [x for x in lista if x > valor]
    return novaLista
```

5.1

```
ehPrimo :: Int -> Bool
ehPrimo num = if (num <= 1) then False else if (num <= 3) then True
else if (num `mod` 2 == 0 || num `mod` 3 == 0) then False else
ehPrimo' num 5

ehPrimo' :: Int -> Int -> Bool
ehPrimo' num i = if (i * i <= num) then if (num `mod` i == 0 || num
`mod` (i + 2) == 0) then False else ehPrimo' num (i + 6) else True

main = do
    print(filter ehPrimo [0..100])
```

5.2

```python
def ehPrimo(num):
    if (num <= 1):
        return False
    elif (num <= 3):
        return True

    if (num % 2 == 0 or num % 3 == 0):
        return False

    i = 5
    while(i * i <= num):
        if (num % i == 0 or num % (i + 2) == 0):
            return False
        i = i + 6
    return True

lista = list(range(0, 100))

print(list(filter(ehPrimo, lista)))
```

6.a.1

```haskell
main = do
    let lista = [x | x <- [0..16], x `mod` 3 == 0]
    print lista
```

6.a.2

```python
lista = list(range(0, 16, 3))
```

6.b.1

```haskell
ehMultiploDe2Ou3 :: Int -> Bool
ehMultiploDe2Ou3 num = num `mod` 2 == 0 || num `mod` 3 == 0

main = do
    let lista = [x | x <- [0..20], ehMultiploDe2Ou3 x]
    print lista
```

6.b.2

```
def ehMultiploDe2Ou3(num):
    return num % 2 == 0 or num % 3 == 0

lista = list(range(0, 21))
lista = list(filter(ehMultiploDe2Ou3, lista))
print(lista)
```

6.c.1

```
listaDeLista :: [[Int]] -> Int -> Int -> [[Int]]
listaDeLista lista l c = if l == 0 then lista else listaDeLista
(lista ++ [[c]]) (l-1) (c+1)

main = do
    print(listaDeLista [] 5 1)
```

6.c.2

```
lista = []

def listaDeLista(lista, l, c):
    cont = 1
    for i in range(l):
        lista.append([])
        for j in range(c):
            lista[i].append(cont)
        cont += 1
    return lista

print(listaDeLista(lista, 5, 1))
```

6.d.1

```
listaDeUm = [replicate x 1 | x <- [1..5]]
```

6.d.2

```
lista = []

def listaDeLista(lista, l, elem):
```

```
    for i in range(l):
        lista.append([])
        for j in range(i + 1):
            lista[i].append(elem)
    return lista

print(listaDeLista(lista, 5, 1))
```

6.e.1

```
listaDeTuplas = [(x, y) | x <- [1..3], y <- [3,2..1]]
```

6.e.2

```
lista = []

def listaDeTuplas(lista, l, c):
    for i in range(l):
        op = 3
        for j in range(c):
            x = (i+1, op)
            lista.append(x)
            op -= 1
    return lista

print(listaDeTuplas(lista, 3, 3))
```

7.1

```
import Data.Char
maiuscula :: [Char] -> [Char]
maiuscula = map toUpper
```

7.2

```
def maiuscula(lista):
    return list(map(lambda x: x.upper(), lista))
```

8.1

```
maiorDeIdade :: [(String, Int)] -> [(String, Int)]
maiorDeIdade lista = [pessoa | pessoa <- lista, snd pessoa >= 18]
```

8.2

```python
def maiorDeIdade(lista):
    newLista = []
    for pessoa in lista:
        if int(pessoa[1]) >= 18:
            newLista.append(pessoa)
    return newLista
```

9.1

```haskell
filtraA :: [String] -> [String]
filtraA lista = [string | string <- lista, head string == 'a']
```

9.2

```python
def filtraA(lista):
    return list(filter(lambda x: x[0] == 'a', lista))
```

10.1

```haskell
listaIdades :: [(String, Int)] -> [Int]
listaIdades lista = [idade | (_, idade) <- lista]
```

10.2

```python
def listaIdades(lista):
    newLista = []
    for pessoa in lista:
        newLista.append(int(pessoa[1]))
    return newLista
```