Faculdade de Computação
Programação Funcional (BCC/BSI) - 1° Período
**Aula Prática: Listas por Compreensão em Haskell**

---

## Listas x Conjuntos

Na comparação entre duas listas, a ordem dos elementos é avaliada:

```
[1,2,3] /= [3,2,1]
"maria" /= "airam"
```

O número de elementos (repetição) também importa:

```
[True,True] /= [True]
```

## Definição de Lista por Compreensão

Uma lista pode ser especificada pela definição de seus elementos. Por exemplo:

$$A = \{x^2+x+1 \mid x \in \mathbb{N} \wedge x \text{ é par}\}$$

```
[x^2+x+1 | x<-[1..], even x]  ->   lista A definida acima
[x | x<-[1..10]]              ->   números de 1 a 10
[x | x<-[1,22..200]]          ->   números de 1 a 200, passo 21
[x^2 | x<-[10..20], odd x]    ->   quadrados dos impares de 10 a 20
[x | x<-[0..50], x^2>500 ]    ->   números de 0 a 50 com quadrado
                                   maior que 500
```

## Função para unir os elementos de uma lista de listas:

A função ***concat*** (do módulo Prelude padrão) reúne numa única lista os elementos de uma lista de listas:

```
> concat [[1,2,3],[4,5],[6,7]]
[1,2,3,4,5,6,7]
```

Podemos definir a função ***concat*** recursivamente usando o operador (++):

```
concat1::[[a]]->[a]
concat1 [] = []
concat1 (a:x) = a ++ concat1 x
```

Podemos definir uma nova versão da função ***concat*** usando lista por compreensão:

```
concat2 :: [[a]] -> [a]
concat2 listaComListas = [x | lista<-listaComListas, x<-lista]
```

## Exercícios:

**1)** Defina as seguintes listas por compreensão:

```
a) [0,3,6,9,12,15]
b) Os múltiplos de 2 e 3 entre 0 e 20.
c) [[1],[2],[3],[4],[5]]
d) [[1], [1,1], [1,1,1], [1,1,1,1], [1,1,1,1,1]]
e) [(1,3),(1,2),(1,1),(2,3),(2,2),(2,1),(3,3),(3,2),(3,1)]
```

---

**2)** Escreva uma função ***insere*** que receba um elemento e uma lista (tipagem genérica) e insere o elemento se somente se ele ainda não pertence à lista. Em seguida faça os seguintes testes:

```
> insere 2 [1,2,3]
> insere 7 [x+4|x<-[10,8..1]]
```

Como modificar a função para que possamos inserir ordenadamente elementos em listas previamente ordenadas? Defina esta função.

---

**3)** Defina uma função recursiva que dada uma lista de inteiros, retorna uma nova lista contendo os elementos de valor superior a um número *n* qualquer.

```
> retornaSup 4 [3,2,5,6]
[5,6]
```

Reescreva a função usando lista por compreensão.

---

**4)** Analise a função abaixo e descreva sua funcionalidade. Em seguida teste a função no interpretador WinHugs.

```
misterio :: String -> String
misterio p = [c | c<-p, c>='a' && c<='z']
```

---

**5)** A função ***zip*** definida no módulo Prelude padrão tem a finalidade de criar pares a partir dos elementos de duas listas iniciais. Veja a tipagem e o exemplo a seguir:

```
zip :: [a] -> [b] -> [(a,b)]
zip [] ys = []
zip (x:xs) [] = []
zip (x:xs) (y:ys) = (x,y) : zip xs ys
```

Exemplos:

```
> zip ['a','b','c'] [1,2,3,4]
[('a',1), ('b',2), ('c',3)]
```

```
> zip [0..] "maria"
[(0,'m'),(1,'a'),(2,'r'),(3,'i'),(4,'a')]
```

Usando a função ***zip***, defina uma nova função ***paresCons*** que receba uma lista, e retorne outra lista com os pares consecutivos da lista de origem:

```
> paresCons [1..10]
[(1,2),(2,3),(3,4),(4,5),(5,6),(6,7),(7,8),(8,9),(9,10)]

> pares "paris"
[('p','a'),('a','r'),('r','i'),('i','s')]
```

A partir da função ***paresCons***, pense numa forma de verificar se uma lista está ordenada.

---

**6)** A função divisores abaixo retorna todos os divisores de um determinado número inteiro positivo. Se a lista de divisores de um número contiver apenas 1 e o próprio número, então dizemos que o número é primo. Por exemplo, 5 é um número primo.

```
divisores :: Int -> [Int]
divisores n = [x | x<-[1..n], n`mod`x==0]

> divisores 5 == [1,5]
True
```

Retorne, a partir da função divisores, todos os números primos de um intervalo. Para realizar esta tarefa, faça duas versões da função: uma recursiva e outra baseada em lista por compreensão.

```
> primos [5..90]
[5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89]
```

---

*Bom trabalho!*