Programação Funcional - Compreensão de Listas

```
universidade = "Universidade Federal de Alfenas"
professor = "Romário da Silva Borges"
```

Aprenderemos nesta aula...

- Entender a sintaxe e a semântica de **compreensões de listas**;
- Aprender a gerar, filtrar e transformar listas de forma concisa.

Nas aulas anteriores...

- Lista em Haskell = [], (x:xs), literais [1,2,3].
  - 1:[] = [1];

- Intervalos: [1,2..5] = [1,2,3,4,5].
- Listas infinitas: [1,2..]
- Operações:, ++, take, drop, sum, length, zip, zipWith...

Compreensão de Listas

- Compreensão de Listas é uma sintaxe concisa, inspirada na **Notação de Construtor de Conjuntos da matemática** *(e.g., $\{ x^2 \mid x \in Z, x > 0 \}$)*, para gerar novas listas a partir de listas existente;

Compreensão de Listas

- É uma forma **declarativa** e expressiva de realizar operações de **mapeamento** (cria novos valores com base em uma lista) e **filtragem** (filtra valores com base em uma lista).
- Em outras linguagens exigiriam laços *for* ou *while* para realizar essas operações.

Compreensão de Listas

Estrutura: **[** *expressão* **|** *gerador(es)*, *filtro(s)* **]**

- expressão: expressão de saída por elemento.
- Geradores: x <- lista (percorrem listas).
- Filtros: condições booleanas (ex: x > 0).

Exemplo 0

Estrutura: **[** *expressão* **|** *gerador(es)*, *filtro(s)* **]**

Veja a execução da função abaixo:

testeInicial lista = [1 | x <- lista]
(ou apenas [1 | _ <- lista])

Exemplo 1

Estrutura: **[** *expressão* **|** *gerador(es)*, *filtro(s)* **]**

Crie uma função que retorne o quadrado de cada elemento de uma lista.

```
quadrados = [ x*x | x <- [1..5] ]
```

saída: [1,4,9,16,25]

(Ocorre o **mapeamento** para uma nova lista)

Exemplo 2

Estrutura: **[** *expressão* **|** *gerador(es)*, *filtro(s)* **]**

Utilizando compreensão de listas, crie uma função, que tem como entrada uma lista e retorna apenas os números pares.
Ex: pares [1, 2, 3, 4, 5, 6] -> [2, 4, 6] .

```
pares :: [Int] -> [Int]
pares lista = [ x | x <- lista, mod x 2 == 0 ]
```

(Ocorre o **filtro** de uma lista)

Sua vez!

Estrutura: **[** *expressão* **|** *gerador(es)*, *filtro(s)* **]**

Crie um função que retorne o quadrado dos elementos pares de uma lista.
Ex: paresAoQuadrado -> [4,16,36,64,100].

paresAoQuadrado :: [Int]

Sua vez! (gabarito)

Estrutura: **[** *expressão* **|** *gerador(es)*, *filtro(s)* **]**

Crie um função que retorne o quadrado dos elementos pares de uma lista.

Ex: paresAoQuadrado -> [4,16,36,64,100] .

```
paresAoQuadrado :: [Int]
paresAoQuadrado = [ x*x | x <- [1..10], mod x 2 == 0 ]
```

(Ocorre o **mapeamento** da lista, e em sequência o **filtro** da lista)

sua vez!

Estrutura: **[** *expressão* **|** *gerador(es)*, *filtro(s)* **]**

Crie uma função que, dada uma lista de tuplas, retorne uma lista com os valores quadrados de cada elemento da tupla;

EX: tuplasAoQuadrado [(3,3),(2,2)]

Sua vez! (gabarito)

Estrutura: **[** *expressão* **|** *gerador(es)*, *filtro(s)* **]**

Crie uma função que, dada uma lista de tuplas, retorne uma lista com os valores quadrados de cada elemento da tupla;
EX: tuplasAoQuadrado [(3,3),(2,2)]

```
tuplasAoQuadrado listaTuplas = [(x*x,y*y) | (x,y) <- listaTuplas]
```

Exemplo 4 - múltiplos filtros

Estrutura: **[** *expressão* **|** *gerador(es)*, *filtro(s)* **]**

Crie uma função que, dada uma lista de entrada, retorne números maiores que 10 e ímpares;

```
filtrar lista = [x | x <- lista, x > 10, odd x]
```

Ex: filtrar [5, 11, 12, 13] => [11, 13]

Exemplo 5- múltiplos geradores

Estrutura: **[** *expressão* **|** *gerador(es)*, *filtro(s)* **]**

Gere todas combinações possíveis entre os jogadores e o número das camisas;

jogadoresCamisas = [(x, y) | x <- [10, 11], y <- [“Romário”, 'Rivaldo']]

Ex: jogadoresCamisas => [(10,"Romario"),(10,"Rivaldo"),(11,"Romario"),(11,"Rivaldo")]

Exemplo 5- múltiplos geradores

Repare que usamos múltiplos geradores para criar o **Produto Cartesiano das listas**. A nova lista será gerada por todas as combinações possíveis.

Ex: jogadoresCamisas =>
[(10,"Romario"),(10,"Rivaldo"),(11,"Romario"),(11,"Rivaldo")]

Reflita: Como essa mesma operação seria feita em uma linguagem imperativa?

Hora de praticar!

Notação de conjuntos: Como seria reproduzido em Haskell a seguinte notação de conjunto: $\{ 2x \mid x \in N, x<5 \}$

Estrutura: **[** *expressão* **|** *gerador(es)*, *filtro(s)* **]**

Hora de praticar! (Gabarito)

Notação de conjuntos: Como seria reproduzido em Haskell a seguinte notação de conjunto: **$\{ 2x \mid x \in N, x<5 \}$**

```
[ 2*x | x <- [1..4]]
```

Estrutura: **[** *expressão* **|** *gerador(es)*, *filtro(s)* **]**

Hora de praticar!

Triângulos equilátero: Crie uma função *triangulosEquilateros*, que dado 3 listas com intervalo [1..10], retorna todos triangulos equilateros;
Ex: *triangulosEquilatero -> [(1,1,1),(2,2,2)...(10,10,10)]*

Triângulos retângulo: Crie uma função *trianguloRetangulo*, que dado 3 listas com intervalo [1..10], retorna todos os triângulos retângulos;
Ex: *trianguloRetangulo-> [(3,4,5),(4,3,5), (6, 8, 10), (8, 6, 10)]*
lembre: Teorema de Pitágoras -> $a^2 + b^2 = c^2$

Estrutura: **[** *expressão* **|** *gerador(es)*, *filtro(s)* **]**

Hora de praticar!

Como seria realizado em C a questão anterior com múltiplos geradores?

```c
for (int c = 1; c <= 10; c++) {
    for (int b = 1; b <= 10; b++) {
        for (int a = 1; a <= 10; a++) {
            if (a*a + b*b == c*c)
                printf("(%d, %d, %d)\n", a, b, c);
        }
    }
}
```

Hora de praticar! (Gabarito)

Triângulos equilátero:

```
triangulosEquilateros = [(a,b,c) | a <- [1..10], b <- [1..10], c <- [1..10], a == b && b == c ]
```

Triângulos retângulo:

```
triangulosRetangulos = [(a,b,c) | a <- [1..10], b <- [1..10], c <- [1..10], a*a + b*b == c*c ]
```

Desafio!

Triângulos retângulo sem repetição: Crie uma função *trianguloRetangulo*, que dado 3 listas com intervalo [1..10], retorna todos os triângulos retângulos, mas sem valores repetidos;
Ex: *trianguloRetangulo-> [(3,4,5), (6, 8, 10)]*

Desafio! (Gabarito)

Triângulos retângulo sem repetição:

```
triangulosRetangulosSemRepeticao = [(a,b,c) | c <- [1..10], b <- [1..c], a <- [1..b], a*a + b*b == c*c]
```

Obs: a lista de geração se inicia pelo gerador mais a esquerda.

Desafio! (Gabarito)

Resolução do desafio anterior em C:

```c
for (int c = 1; c <= 10; c++) {
    for (int b = 1; b <= c; b++) {
        for (int a = 1; a <= b; a++) {
            if (a*a + b*b == c*c)
                printf("(%d, %d, %d)\n", a, b, c);
        }
    }
}
```

Programação Funcional