Programação Funcional

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

Aprenderemos nesta aula...

- Conceito de Listas em Haskell;
- Manipulação de listas;

Listas

Listas são estruturas homogêneas em Haskell (sempre possuem elementos do mesmo tipo);

Ex: *numerosAleatorios* = [1,9,5,3]

Listas

Strings são listas de caracteres:

ghci> ['h','a','s','k','e','l','l'] == "haskell"
ghci> TRUE

Listas

Concatenação de listas:

ghci> “Hello” ++“ ”++ ['h','a','s','k','e','l','l']
ghci> “Hello Haskell”

ghci> 1:[2,3,4]
ghci> [1,2,3,4]

Listas

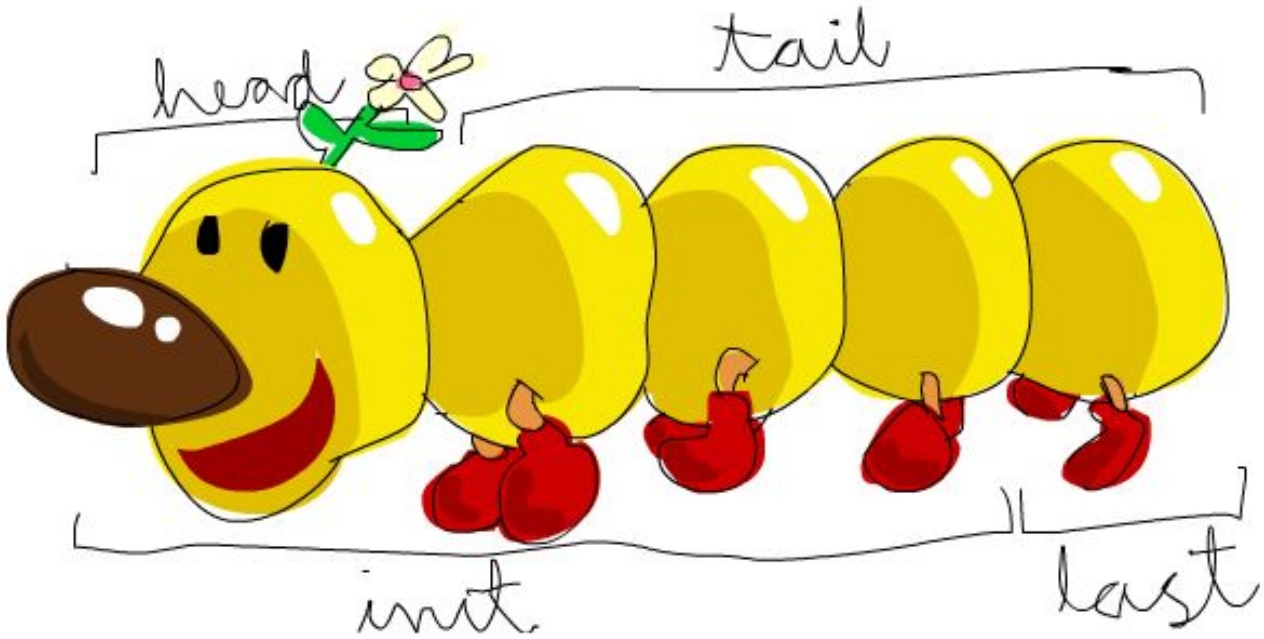
Comparação de listas:

ghci> [3,2,1] > [2,1,0]
True

ghci> [3,2,1] > [2,10,100]
True

ghci> [3,4,2] == [3,4,2]
True

Listas



Listas

Funções básicas com listas:

ghci> head [5,4,3,2,1]
5

ghci> tail [5,4,3,2,1]
[4,3,2,1]

ghci> init [5,4,3,2,1]
[5,4,3,2]

ghci> last [5,4,3,2,1]
1

Listas

Outras funções com listas:

length [10,20,30] *=> 3*
null [] *=> True*
null [1,2,3] *=> False*
reverse [1,2,3] *=> [3,2,1]*
take 3 [10,20,30,40] *=> [10,20,30]*
drop 2 [10,20,30,40] *=> [30,40]*

Listas

Outras funções com listas:

maximum [4,1,9,7] *=> 9*
minimum [4,1,9,7] *=> 1*
sum [1,2,3,4] *=> 10*
product [2,3,4] *=> 24*
elem 3 [1,2,3,4] *=> True*
elem 5 [1,2,3,4] *=> False*
zip [1,2,3] ['a','b','c'] => [(1,'a'),(2,'b'),(3,'c')]

Intervalos

Intervalos são uma forma de fazer listas que são sequências aritméticas de elementos que podem ser enumerados;

ghci> [1..20]
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]

ghci> ['a'..'z']
"abcdefghijklmnopqrstuvwxyz"

Intervalos

Qual serão os elementos do intervalo abaixo?

ghci> [2,9..38]
ghci>???

Intervalos

Qual serão os elementos do intervalo abaixo?

ghci> [2,9..38]
ghci> ***[2,9,16,23,30,37]***

Listas infinitas

É possível usar intervalos para fazer listas infinitas, apenas não especificando o limite. Isso é possível pois Haskell é *lazy* (preguiçosa) (será explorado mais adiante). Verifique o resultado abaixo:

ghci> take 10 [2,4..]
ghci>???

Listas infinitas

Há funções que podem criar **listas infinitas**:

ghci> take 10 (cycle [1,2,3])
[1,2,3,1,2,3,1,2,3,1]

ghci> take 12 (cycle "LOL ")
"LOL LOL LOL "

OBS: Tente utilizar o *cycle* sem *take*.
dica: Ctrl +c vai te salvar 😉

Hora de praticar!

Minha idade: Crie uma função minhaldade, que dado uma lista com o ano do seu nascimento até 2025, retorna sua idade;

Sequência Misteriosa: Crie uma lista com os números de 1 a 50 e depois pegue apenas os 10 elementos centrais (ou seja, remova 20 do início e 20 do fim);

Palavra espelhada:
Crie uma função que receba uma palavra e devolva a palavra seguida de sua versão invertida

Hora de praticar!

Sistema acadêmico: Crie uma lista com 5 notas (de 0 a 10), calcule a média e mostre se o aluno foi aprovado (>= 6) ou reprovado

Seleção brasileira: Crie duas listas, uma com nomes de jogadores e outra com seus números de camisa.*Use zip para associar cada jogador ao número correspondente*

Hora de praticar: gabarito

- **Minha idade**:
  minhaldade = length [2005..2025] - 1
- **Sequência Misteriosa**:
  drop 20 (take 30 [1..50])
- **Palavra espelhada:**
  espelhada s = s ++ reverse s
- **Ranking de pontuações:**
  verificaAprovacao notas = if (sum notas / fromIntegral (length notas)) >= 6 then "aprovado" else "reprovado"
- **Seleção brasileira**
  jogadores = ["Ronaldo", "Neymar", "Vinicius Jr", "Casemiro", "Alisson"]
  numeros = [9, 10, 7, 5, 1]
  jogadoresComNumeros = zip jogadores numeros

Desafio!

Dobre cada outro: Crie uma função chamada: *dobreCadaOutro*. A função deve dobrar todos os números alternados, começando da direita para a esquerda. Ou seja, o penúltimo, o antepenúltimo e assim por diante devem ser dobrados.
Ex:

dobreCadaOutro [8,7,6,5] -> [16,7,12,5]
dobreCadaOutro [1,2,3] -> [1,4,3]

Dicas: Use Reverse, zipWith, cycle

Gabarito Desafio!

Dobre cada outro:
dobreCadaOutro lista = reverse (zipWith (*) (cycle [1,2]) (reverse lista))

Explicação:
O comando *zipWith (*) (cycle [1,2]) (reverse lista)* multiplica cada elemento alternadamente por 1 e 2, da direita pra esquerda. Depois o *reverse* final devolve à ordem original.

Dica: procure entender o que cada função faz separadamente!

Programação Funcional