

Programação Funcional - Recursos sintáticos da Linguagem Haskell

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

Na aula de hoje aprenderemos...

Haskell possui recursos sintáticos poderosos que facilitam a **definição clara e expressiva de funções**, dentre vários, vamos falar desses:

- Casamento de padrões;
- Guardas
- Cláusula Where

Casamento de Padrões

Casamento de padrões (Pattern matching)



Casamento de padrão é uma operação envolvendo um **padrão** e uma **expressão** que **faz a correspondência** (casamento) entre o padrão e o valor da expressão. Um casamento de padrão pode suceder ou falhar, dependendo da forma do padrão e da expressão envolvidos.

- Em Haskell um padrão é uma construção da linguagem de programação que permite analisar a estrutura de um valor e associar variáveis aos seus componentes.

Padrão variável

O **padrão variável** é simplesmente um identificador de variável de valor (e como tal deve começar com letra minúscula). O casamento sucede sempre. A **variável** é associada ao valor.

```
saudar :: String -> String
```

```
saudar a = "Olá, " ++ a ++ "!"
```

Padrão constante

O padrão constante é simplesmente uma constante. O casamento sucede se e somente se o padrão for idêntico ao valor. Nenhuma associação de variável é produzida;

padrão	valor	casamento
10	10	✓
10	28	×
10	'P'	<i>erro de tipo</i>
'P'	'P'	✓
'P'	'q'	×
'P'	True	<i>erro de tipo</i>

Padrão constante

Ao chamar *avaliarNota* 7, por exemplo, ela deve retornar “Boa”.

```
avaliarNota :: Int -> String
```

```
avaliarNota 10 = "Excelente!"
```

```
avaliarNota 7  = "Boa!"
```

```
avaliarNota 5  = "Regular."
```

```
avaliarNota x  = "Insuficiente."
```

Padrão constante

O que aconteceria se mudarmos a posição das declarações?

```
avaliarNota2 :: Int -> String
avaliarNota2 x = "Insuficiente."
avaliarNota2 10 = "Excelente!"
avaliarNota2 7  = "Boa!"
avaliarNota2 5  = "Regular."
```

Qual seria o resultado de
informássemos o valor
10?

Padrão Curinga

O **padrão curinga** é escrito como um sublinhado (`_`). O **casamento sucede sempre**. *Nenhuma* associação de variável é produzida. `_` é também chamado de **variável anônima**, pois, assim como a variável, **casa com qualquer valor**.

padrão	valor	casamento
<code>_</code>	10	✓
<code>_</code>	28	✓
<code>_</code>	'P'	✓
<code>_</code>	()	✓
<code>_</code>	(18,3,2012)	✓
<code>_</code>	"Ana Maria"	✓
<code>_</code>	[5.6,7.1,9.0]	✓

Padrão Curinga

No exemplo abaixo, não importa quais são os dois primeiros valores, o que importa é pegar o terceiro elemento.

```
pegarTerceiro :: (a, b, c) -> c
```

```
pegarTerceiro (_, _, x) = x
```

```
retornaPrimeiro :: a -> b -> a
```

```
retornaPrimeiro x _ = x
```

Padrão Tupla

O casamento sucede se e somente se cada um dos padrões casar com o componente correspondente do valor. Se os tamanhos do padrão tupla e do valor tupla forem diferentes, então ocorre um erro de tipo;

padrão	valor	casamento
(18, True)	(18, True)	✓
(97, True)	(18, True)	×
(18, False)	(18, True)	×
(18, 'M')	(18, True)	<i>erro de tipo</i>
(18, True , 'M')	(18, True)	<i>erro de tipo</i>

Padrão Lista

Uma lista pode ser vazia ou não, logo:

padrão	valor	casamento
$[]$	$[]$	✓
$[]$	$[1, 2, 3]$	✗

padrão	valor	casamento
$x:y:_$	$[]$	✗
$x:y:_$	$["ana"]$	✗
$x:y:_$	$[1, 2]$	✓ $x \mapsto 1, y \mapsto 2$
$x:y:_$	$[1, 2, 3, 4]$	✓ $x \mapsto 1, y \mapsto 2$

padrão	valor	casamento
$x:xs$	$[]$	✗
$x:xs$	$[1, 2, 3, 4]$	✓ $x \mapsto 1, xs \mapsto [2, 3, 4]$
$x:xs$	$['A']$	✓ $x \mapsto 'A', xs \mapsto []$

Exemplo

Como retirar o primeiro elemento de uma lista sem usar a função *head*:

```
retirarPrimeiro :: [Int] -> Int
```

```
retirarPrimeiro (cabeca:cauda) = cabeca
```

Outros recursos sintáticos

Guardas



Linguagens funcionais modernas usam casamento de padrão para selecionar componentes de estruturas de dados e também para **selecionar alternativas em expressões**. Podemos usar, por exemplo, **guardas**.

- As **guardas** permitem testar condições lógicas dentro da definição de uma função. Guardas tornam o código mais limpo do que usar múltiplos IFs

Guardas

Exemplo:

```
classifica :: Int -> String
```

```
classifica x
```

```
  | x < 0 = "Negativo"
```

```
  | x == 0 = "Zero"
```

```
  | otherwise = "Positivo"
```

Claúsula *Where*

Uma função pode ser definida localmente ou globalmente. A cláusula *Where* permite definir **uma função localmente**:

```
saudacao :: String -> String
```

```
saudacao nome
```

```
| nome == "Padwan" = saudacaoLegal ++ " " ++ nome
```

```
| nome == "Darth Vader" = saudacaoInfeliz ++ " " ++ nome
```

```
| otherwise = saudacaoLegal ++ " " ++ nome
```

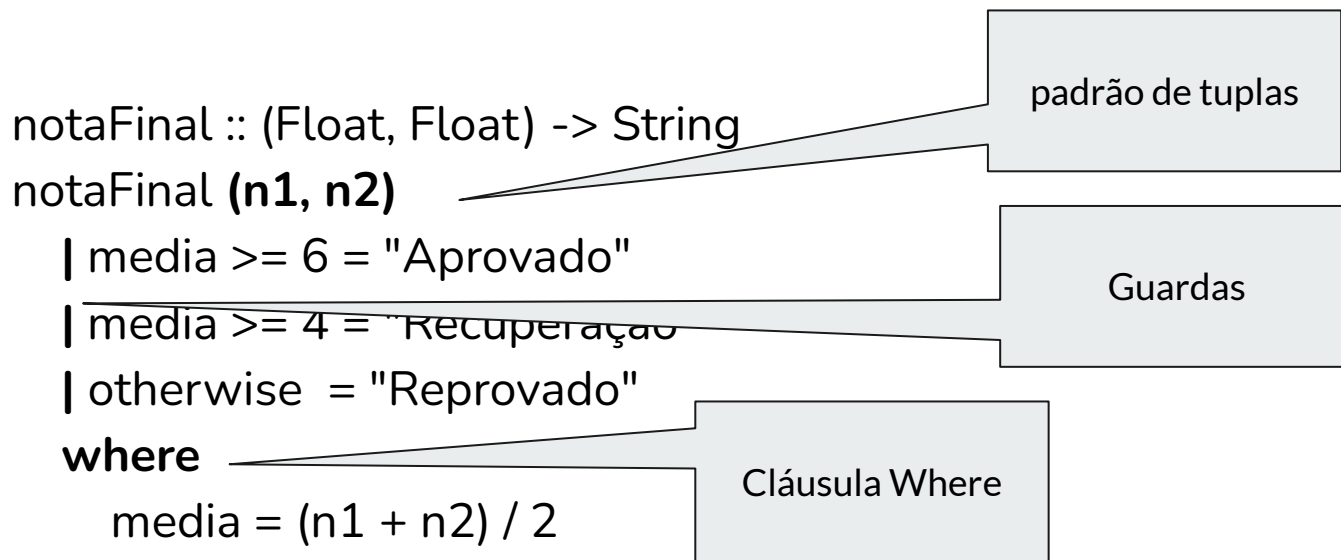
```
where
```

```
    saudacaoLegal = "Olah! Que bom encontrar voce, "
```

```
    saudacaoInfeliz = "Afff! Pfft. Eh voce, "
```

Combinação de diversos recursos

É possível a utilização de diversos recursos juntamente. No exemplo abaixo, são utilizados Guardas, casamento de padrões e a cláusula Where:



Hora de praticar!

- Implemente uma função *cabeca* que devolve o primeiro elemento de uma lista, e uma função *cauda* que devolve todos os elementos, menos o primeiro.
- Crie *somaDoisPrimeiros* que recebe uma lista e soma os dois primeiros elementos. Dica: use o seguinte casamento de padrão: (x:_).
- Implemente *distancia*, que recebe duas coordenadas (x1, y1) e (x2, y2) e retorna a distância entre os pontos.
fórmula: $(d=(x2-x1)^2+(y2-y1)^2)$

Referências

- Malaquias, J. R. (2018). *Programação Funcional em Haskell*. Universidade Federal de Ouro Preto, Departamento de Computação.
- DU BOIS, André Rauber. *Programação Funcional com a Linguagem Haskell*. Porto Alegre: Bookman, 2015.

Programação Funcional