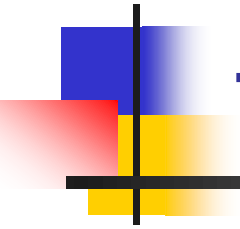
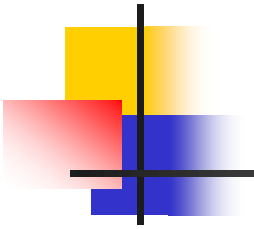


Tipos, Condicionais e Recursão

Profª. Rachel Reis

rachel@inf.ufpr.br



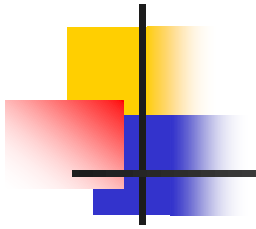
Tipos

- Um tipo é uma coleção de valores relacionados.
- Em Haskell nomes de tipos devem começar com letra maiúscula.
- Exemplo:
  - Bool contém os valores lógicos True e False.

Tipos Numéricos

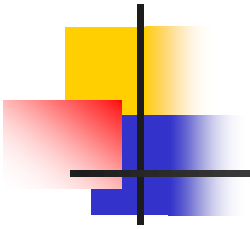
- **`Int`**:
  - Valores inteiros de precisão fixa.
  - Limitado, representa os valores numéricos no intervalo de -2^{63} até $2^{63} - 1$.
  - Exemplo: **`750, 2023`**

- **`Integer`**:
  - Valores inteiros de precisão arbitrária.
  - Ilimitado, representam valores inteiros de qualquer precisão.
  - Exemplo: **`17, 75467898723456056 78`**



Tipos Numéricos

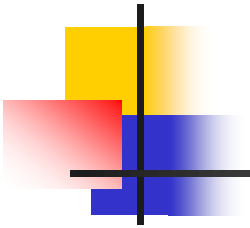
- **`Float`**:
  - Valores em ponto-flutuante de precisão simples (32 bits).
  - Em média, representa números com até 7 dígitos.
  - Exemplo: **`4.56, 0.205`**
- **`Double`**:
  - Valores em ponto-flutuante de precisão dupla (64 bits).
  - Em média, representa números com quase 16 dígitos.
  - Exemplos: **`78937.5, 987.3201E-60`**



Tipos Lógico e Caractere

- **`Bool`**:
  - Contém os valores lógicos: verdadeiro e false.
  - Expressões booleanas podem ser executadas com os operadores && (e), II (ou) e not.
  - Exemplos: **`True`**, **`False`**

- **`Char`**:
  - Contém todos os caracteres do sistema Unicode.
  - Exemplos: **`'B', '!', '\n'`**



Tipos Lista

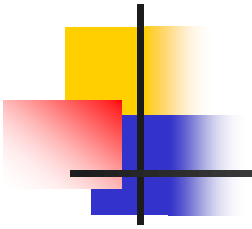
- **`[t]`**
  - Sequência de valores do mesmo tipo
  - Exemplo: **`['O', 'L', 'A']`** **`[Char]`**
    `[1, 2, 3, 4]` **`[Int]`**

- **`String`**
  - Sequência de caracteres delimitados por aspas duplas
  - Sinônimo para **`[Char]`**
  - Exemplo: **`"UFPR"`** **`['U', 'F', 'P', 'R']`**

Tipos Tupla

- $(t_1 \ldots t_2)$
  - Sequência de valores possivelmente de tipos diferentes.
  - Não existe tupla de um único componente.
  - Exemplo:

```
('O', 'I')          (Char, Char)
("Joel", 'M', 22)   (String, Char, Int)
```

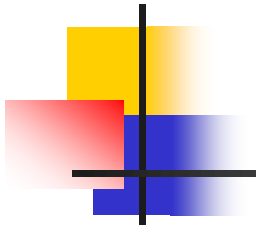


Assinaturas de Tipo

- Qualquer expressão pode ter o seu tipo anotado.
- Se exp é uma expressão e t é um tipo, então

  $exp :: t$

  lê-se: "exp é do tipo t"
- `::` tem precedência menor do que todos os operadores de Haskell.



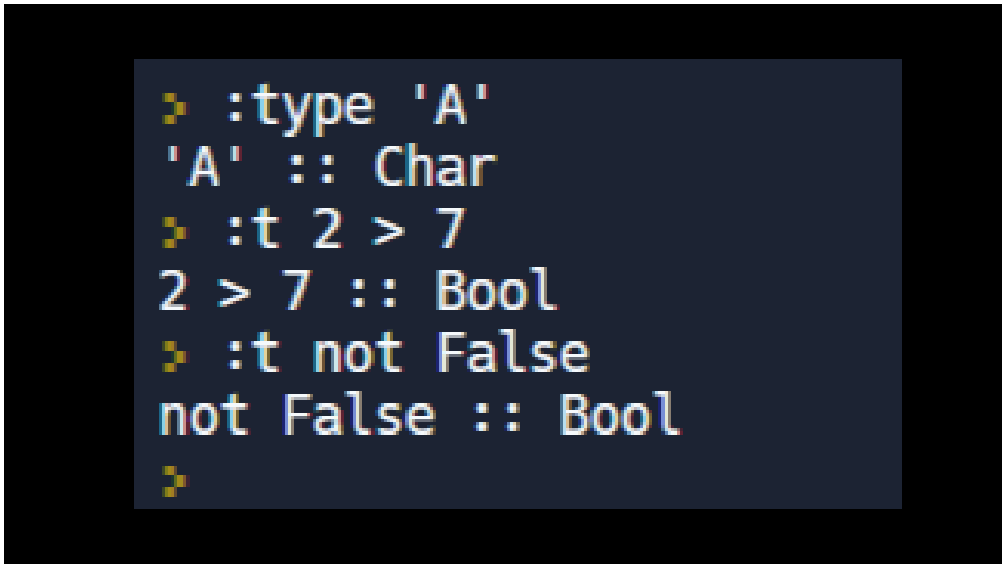
Assinaturas de Tipo

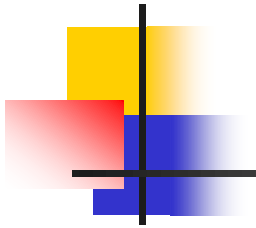
- Exemplos

`'a'`	`::`	Char
`"joao da silva"`	`::`	String
`45`	`::`	Int
`2 > 7`	`::`	Bool

Consulta de Tipo no GHCi

- No GHCi, o comando :type (ou de forma abreviada :t) exibe o tipo de uma expressão.





Tipos e Funções

```
x :: Int
x = 3
```

- O sinal de igual **não** representa atribuição, e sim definição
- Alguns autores consideram a definição acima como função: "*x é uma função que não recebe parâmetros e retorna um inteiro constante*"

Tipos e Funções

- Ao definir uma função, o seu tipo pode ser anotado (boa prática de programação).

```
x :: Int -> Float -> Bool -> Int
```

  - “x” é o nome da função
  - o último tipo especificado identifica o tipo de dado a ser retornado.
  - os três tipos do **meio** são os tipos dos argumentos da função

Tipos e Funções

- Definição da função multiplica com seu tipo anotado.

```
module Operacoes where

multiplica :: Int -> Int -> Int
multiplica  x y = x * y
```

Tipos e Funções

- Definição da função soma com seu tipo anotado.

```
module Operacoes where

multiplica :: Int -> Int -> Int
multiplica  x y = x * y

soma :: Int -> Int -> Int
soma x y = x + y
```

Tipos e Funções

- Função multiplica com três parâmetros.

```
module Operacoes where

multiplica :: Int -> Int -> Int -> Int
multiplica  x y z = x * y * z
```

Consulta de Tipo no GHCi

- Se quiser verificar a assinatura de uma função no GHCi, basta digita: :t ou :type <nome da função>

```
> :t multiplica
multiplica :: Int -> Int -> Int
> :t soma
soma :: Int -> Int -> Int
> :t True
True :: Bool
```

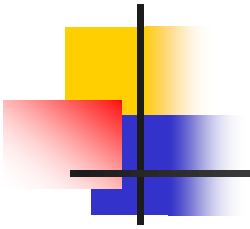
- Algumas funções podem operar sobre vários tipos de dados.
- Exemplo: a função *head* recebe uma lista e retorna o primeiro elemento, não importa o tipo dos elementos.

```
head ['B', 'O', 'L', 'A']
> B

head ["Pedro", "Laura", "Marcos"]
> "Pedro"
```

Qual o tipo de head?

```
head :: [Char]   -> Char
head :: [String] -> String
```

*** head pode ter vários tipos ***



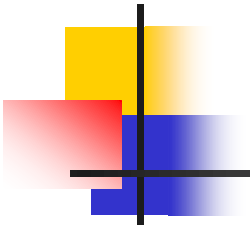
Variáveis de Tipo

- Quando um tipo pode ser qualquer tipo da linguagem, ele é representado por uma variável de tipo.
- No exemplo da função *head*, *a* representa o tipo dos elementos da lista passados como argumento

```
head :: [a] -> a
```

a é uma variável de tipo que pode ser substituída por qualquer tipo.

- Variáveis de tipo devem começar com letra minúscula e são geralmente denominadas a, b, c, etc.



Função Polimórfica

- Uma função é chamada polimórfica se o seu tipo contém uma ou mais variáveis de tipo.

- Exemplo 1

```
head :: [a] -> a
```

Leitura: para qualquer tipo a, *head* recebe uma lista de valores do tipo *a* e retorna um valor do tipo *a*

Função Polimórfica

- Exemplo 2

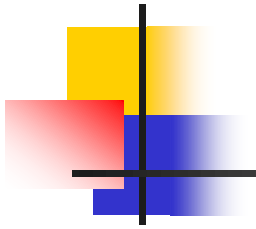
```
length :: [a] -> Int
```

Leitura: para qualquer tipo a, *length* recebe uma lista de valores do tipo *a* e retorna um inteiro

- Exemplo 3

```
fst:: (a, b) -> a
```

Leitura: para quaisquer tipos a e b, *fst* recebe um par do tipo *(a, b)* e retorna um valor do tipo *a*.

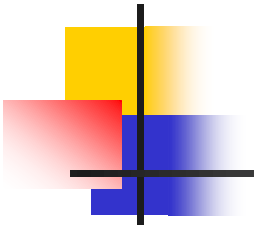


Função Polimórfica

- Muitas funções definidas no módulo Prelude são polimórficas.

- Exemplos

```
head :: [a] -> a                -- seleciona o 1º item de uma lista
fst  :: (a, b) -> a             -- seleciona o 1º item de um par
snd  :: (a, b) -> b             -- seleciona o 2º item de um par
take :: Int -> [a] -> [a]       -- seleciona os 1os itens de uma lista
```



Erros de Tipo

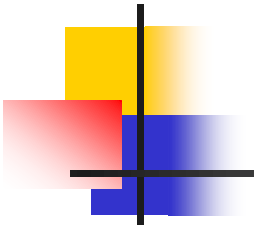
- Toda expressão sintaticamente correta tem seu tipo calculado em tempo de compilação.
- Se não for possível determinar o tipo de uma expressão ocorre um erro de tipo.
- A aplicação de uma função a um ou mais argumentos de tipo inadequado constitui um erro de tipo.

Erros de Tipo

- Exemplo

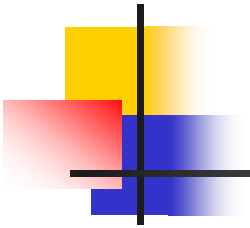
```
> not 'A'

<interactive>:1:5: error:
    • Couldn't match expected type ‘Bool’ with actual type ‘Char’
    • In the first argument of ‘not’, namely ‘'A'’
      In the expression: not 'A'
      In an equation for ‘it’: it = not 'A'
>
```

- Explicação: a função *not* requer um valor Bool como argumento, porém, foi passado *‘A’* que é do tipo Char.



Checagem de Tipos

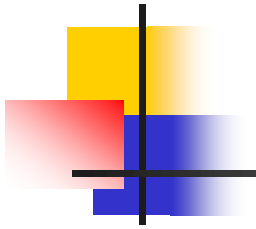
- Haskell é uma linguagem fortemente tipada, com um sistema de tipos muito avançado.
- Todos os possíveis erros de tipos são encontrados em tempo de compilação (tipagem estática).
- Vantagem: programas mais seguros e rápidos, eliminando a necessidade de verificação em tempo de execução.



Condicionais em Haskell

- Uma função em Haskell pode incluir estruturas condicionais para desviar o fluxo do programa.

- Isso pode ser feito de duas formas:
  1) Usando a estrutura **if-then-else** (comum na programação imperativa).
  2) Usando **guardas**, representado no código por uma barra vertical ‘|’



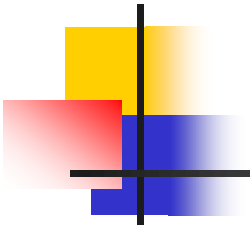
Exemplo 1 - if-then-else

- Escreva uma função que receba dois inteiros e retorne o maior. Use a estrutura if-then-else.

Exemplo 1 - if-then-else

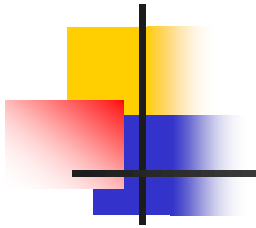
```
-- Usando if then else
maior :: Int -> Int -> Int
maior a b = if a >= b
      then a
      else b
```

- O *if-then-else* pode ser escrito em uma única linha.
- A cláusula *else* não é opcional, omiti-la é um erro.
- O uso dos parênteses na condição é opcional.



Condicionais com guardas

- Guardas são equações condicionais que especificam cada uma das circunstâncias nas quais a definição da função pode ser aplicada.
- Pode ou não conter a palavra *otherwise* (de outra maneira) como a última condição em uma expressão condicional.
- Com guardas, a primeira expressão avaliada como verdadeira determina o valor da função.



Exemplo 1 - guardas

- Escreva uma função que receba dois inteiros e retorne o maior. Use guardas.

Exemplo 1 - guardas

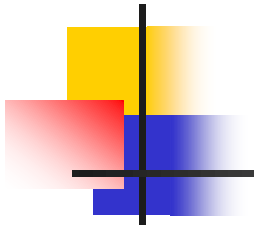
```
-- Usando guardas
maiorG :: Int -> Int -> Int
maiorG a b
      | a >= b = a
      | b > a = b
```

```
-- Versão alternativa com otherwise
maiorG :: Int -> Int -> Int
maiorG a b
      | a >= b = a
      | otherwise = b
```

Exemplo 2

- Altere o exemplo abaixo para que a função retorne zero quando os valores a e b forem iguais.

```
-- Versão alternativa com otherwise
maiorG :: Int -> Int -> Int
maiorG a b
      | a >= b = a
      | otherwise = b
```



Exemplo 2

- Solução

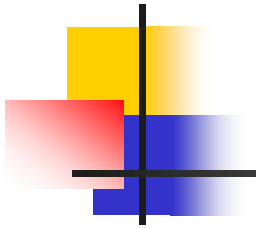
```
-- Versão alternativa com otherwise
maiorG :: Int -> Int -> Int
maiorG a b
      | a > b = a
      | b > a = b
      | otherwise = 0
```

Comparando if-then-else e guardas

```
-- Usando if then else
maior :: Int -> Int -> Int
maior a b = if a >= b
      then a
      else b
```

```
-- Usando guardas
maiorG :: Int -> Int -> Int
maiorG a b
      | a >= b = a
      |otherwise = 0
```

- Atenção para o sinal de igual.
- Atenção para indentação. Linhas de código no mesmo nível de indentação pertencem a um mesmo bloco.

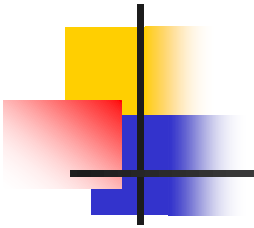


Exemplo 3

- Escreva uma função que informe se um dado número é par usando if-then-else e guardas.

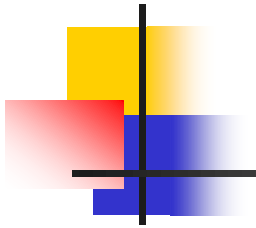
```haskell
-- Solução usando if-then-else
ehPar :: Int -> Bool
ehPar x = if mod x 2 == 0
      then True
      else False
```

```haskell
-- Solução usando guardas
ehParG :: Int -> Bool
ehParG x
      | (mod x 2 == 0) = True
      | otherwise = False
```



Função recursiva

- Em Haskell, como não é possível controlar o estado do programa ou de variáveis de controle, não existe estruturas de repetição.
- Toda repetição deve ser efetuada por meio de recursão.
- Uma função recursiva é formada por duas partes:
  - Caso base
  - Passo recursivo

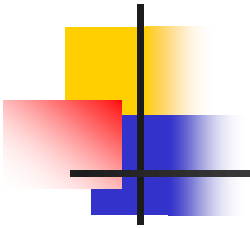


Exemplo 4

- Escreva uma função para calcular o **fatorial** de um número com guardas e sem guardas.

```haskell
-- Função recursiva usando guardas
fatorialG :: Int -> Int
fatorialG n
        | n == 0 = 1
        | n > 0 = n * fatorial(n-1)
```

```haskell
-- Função recursiva sem guardas
fatorial :: Int -> Int
fatorial 0 = 1
fatorial n = n * fatorial (n - 1)
```

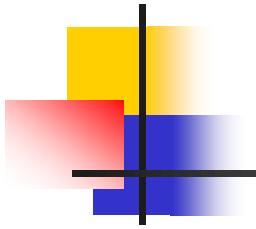


Exemplo 5

- Escreva uma função recursiva em Haskell para calcular a potência de x^n, sendo x > 0 e n ≥ 0. Implemente a função com guardas e sem guardas.

```haskell
-- Função recursiva usando guardas
potenciaG :: Int -> Int -> Int
potenciaG x n
      | n == 0 = 1
      | n > 0 = x * potencia x (n-1)
```

```haskell
-- Função recursiva sem guardas
potencia :: Int -> Int -> Int
potencia x 0 = 1
potencia x n = x * potencia x (n-1)
```



Para praticar...

- Implemente uma função recursiva que calcule o somatório em um intervalo [x, y], sendo x e y números inteiros, e x < y.