

# Disciplina de Programação Funcional Engenharia de Computação

Linguagem Haskell

**Mauro Hemerly Gazzani**  
[mauro.hemerly@gmail.com](mailto:mauro.hemerly@gmail.com)

Universidade Estadual de Minas Gerais  
Câmpus de Ituiutaba, 2º semestre de 2018

<https://bit.ly/2A0eUI8>  
<https://github.com/mauro-hemerly/UEMG-2018-2>

# Linguagem Haskell

- Algumas características de **Haskell**

- ▶ Programas são **concisos**
- ▶ **Tipagem estática**
- ▶ **Sistema de tipos** poderoso
- ▶ Tipos e funções **recursivas**
- ▶ **Funções de ordem superior**
- ▶ Linguagem **pura** (declarativa)
- ▶ Avaliação **lazy**

- **Em 1987:** Um comitê internacional de pesquisadores inicia o desenvolvimento de **Haskell**, uma **linguagem funcional lazy** padrão.
- **Em 2003:** O comitê publica o relatório **Haskell 98**, a definição de uma versão estável da linguagem **Haskell**.
- **Em 2010:** O comitê publica o relatório **Haskell 2010**, uma revisão da definição da linguagem **Haskell**.

# Linguagem Haskell

- Algumas características de **Haskell**

- ▶ Programas são **concisos**
- ▶ **Tipagem** estática
- ▶ **Sistema de tipos** poderoso
- ▶ Tipos e funções **recursivas**
- ▶ **Funções de ordem superior**
- ▶ Linguagem **pura** (declarativa)
- ▶ Avaliação **lazy**

- **Em 1987:** Um comitê internacional de pesquisadores inicia o desenvolvimento de **Haskell**, uma **linguagem funcional lazy** padrão.
- **Em 2003:** O comitê publica o relatório **Haskell 98**, a definição de uma versão estável da linguagem **Haskell**.
- **Em 2010:** O comitê publica o relatório **Haskell 2010**, uma revisão da definição da linguagem **Haskell**.

# Linguagem Haskell

- Algumas características de **Haskell**
  - ▶ Programas são **concisos**
  - ▶ **Tipagem estática**
  - ▶ **Sistema de tipos** poderoso
  - ▶ Tipos e funções **recursivas**
  - ▶ **Funções de ordem superior**
  - ▶ Linguagem **pura** (declarativa)
  - ▶ Avaliação **lazy**
- **Em 1987:** Um comitê internacional de pesquisadores inicia o desenvolvimento de Haskell, uma **linguagem funcional lazy** padrão.
- **Em 2003:** O comitê publica o relatório **Haskell 98**, a definição de uma versão estável da linguagem Haskell.
- **Em 2010:** O comitê publica o relatório **Haskell 2010**, uma revisão da definição da linguagem Haskell.

# Linguagem Haskell

- Algumas características de **Haskell**

- ▶ Programas são **concisos**
- ▶ **Tipagem estática**
- ▶ **Sistema de tipos poderoso**
- ▶ Tipos e funções **recursivas**
- ▶ **Funções de ordem superior**
- ▶ Linguagem **pura** (declarativa)
- ▶ Avaliação **lazy**

- **Em 1987:** Um comitê internacional de pesquisadores inicia o desenvolvimento de Haskell, uma **linguagem funcional lazy** padrão.
- **Em 2003:** O comitê publica o relatório **Haskell 98**, a definição de uma versão estável da linguagem Haskell.
- **Em 2010:** O comitê publica o relatório **Haskell 2010**, uma revisão da definição da linguagem Haskell.

# Linguagem Haskell

- Algumas características de **Haskell**

- ▶ Programas são **concisos**
- ▶ **Tipagem estática**
- ▶ **Sistema de tipos** poderoso
- ▶ **Tipos e funções recursivas**
- ▶ **Funções de ordem superior**
- ▶ Linguagem **pura** (declarativa)
- ▶ Avaliação **lazy**

- **Em 1987:** Um comitê internacional de pesquisadores inicia o desenvolvimento de Haskell, uma **linguagem funcional lazy** padrão.
- **Em 2003:** O comitê publica o relatório **Haskell 98**, a definição de uma versão estável da linguagem Haskell.
- **Em 2010:** O comitê publica o relatório **Haskell 2010**, uma revisão da definição da linguagem Haskell.

# Linguagem Haskell

- Algumas características de **Haskell**

- ▶ Programas são **concisos**
- ▶ **Tipagem estática**
- ▶ **Sistema de tipos** poderoso
- ▶ Tipos e funções **recursivas**
- ▶ **Funções de ordem superior**
- ▶ Linguagem **pura** (declarativa)
- ▶ Avaliação **lazy**

- Em 1987: Um comitê internacional de pesquisadores inicia o desenvolvimento de Haskell, uma **linguagem funcional lazy** padrão.
- Em 2003: O comitê publica o relatório **Haskell 98**, a definição de uma versão estável da linguagem Haskell.
- Em 2010: O comitê publica o relatório **Haskell 2010**, uma revisão da definição da linguagem Haskell.

# Linguagem Haskell

- Algumas características de **Haskell**

- ▶ Programas são **concisos**
- ▶ **Tipagem estática**
- ▶ **Sistema de tipos** poderoso
- ▶ Tipos e funções **recursivas**
- ▶ **Funções de ordem superior**
- ▶ **Linguagem pura** (declarativa)
- ▶ Avaliação **lazy**

- Em 1987: Um comitê internacional de pesquisadores inicia o desenvolvimento de Haskell, uma **linguagem funcional lazy** padrão.
- Em 2003: O comitê publica o relatório **Haskell 98**, a definição de uma versão estável da linguagem Haskell.
- Em 2010: O comitê publica o relatório **Haskell 2010**, uma revisão da definição da linguagem Haskell.



# Linguagem Haskell

- Algumas características de **Haskell**

- ▶ Programas são **concisos**
- ▶ **Tipagem estática**
- ▶ **Sistema de tipos** poderoso
- ▶ Tipos e funções **recursivas**
- ▶ **Funções de ordem superior**
- ▶ Linguagem **pura** (declarativa)
- ▶ **Avaliação lazy**

- Em 1987: Um comitê internacional de pesquisadores inicia o desenvolvimento de Haskell, uma **linguagem funcional lazy** padrão.
- Em 2003: O comitê publica o relatório **Haskell 98**, a definição de uma versão estável da linguagem Haskell.
- Em 2010: O comitê publica o relatório **Haskell 2010**, uma revisão da definição da linguagem Haskell.

# Linguagem Haskell

- Algumas características de **Haskell**
  - ▶ Programas são **concisos**
  - ▶ **Tipagem estática**
  - ▶ **Sistema de tipos** poderoso
  - ▶ Tipos e funções **recursivas**
  - ▶ **Funções de ordem superior**
  - ▶ Linguagem **pura** (declarativa)
  - ▶ Avaliação **lazy**
- **Em 1987:** Um comitê internacional de pesquisadores inicia o desenvolvimento de **Haskell**, uma **linguagem funcional lazy** padrão.
- **Em 2003:** O comitê publica o relatório **Haskell 98**, a definição de uma versão estável da linguagem **Haskell**.
- **Em 2010:** O comitê publica o relatório **Haskell 2010**, uma revisão da definição da linguagem **Haskell**.

# Linguagem Haskell

- Algumas características de **Haskell**
  - ▶ Programas são **concisos**
  - ▶ **Tipagem estática**
  - ▶ **Sistema de tipos** poderoso
  - ▶ Tipos e funções **recursivas**
  - ▶ **Funções de ordem superior**
  - ▶ Linguagem **pura** (declarativa)
  - ▶ Avaliação **lazy**
- **Em 1987:** Um comitê internacional de pesquisadores inicia o desenvolvimento de **Haskell**, uma **linguagem funcional lazy** padrão.
- **Em 2003:** O comitê publica o relatório **Haskell 98**, a definição de uma versão estável da linguagem **Haskell**.
- **Em 2010:** O comitê publica o relatório **Haskell 2010**, uma revisão da definição da linguagem **Haskell**.

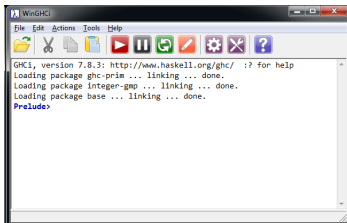
# Linguagem Haskell

- Algumas características de **Haskell**
  - ▶ Programas são **concisos**
  - ▶ **Tipagem estática**
  - ▶ **Sistema de tipos** poderoso
  - ▶ Tipos e funções **recursivas**
  - ▶ **Funções de ordem superior**
  - ▶ Linguagem **pura** (declarativa)
  - ▶ Avaliação **lazy**
- **Em 1987:** Um comitê internacional de pesquisadores inicia o desenvolvimento de **Haskell**, uma **linguagem funcional lazy** padrão.
- **Em 2003:** O comitê publica o relatório **Haskell 98**, a definição de uma versão estável da linguagem **Haskell**.
- **Em 2010:** O comitê publica o relatório **Haskell 2010**, uma revisão da definição da linguagem **Haskell**.

# Algumas Empresas que Usam Haskell

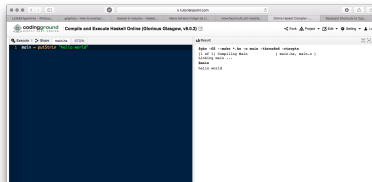
- Exemplos de empresas que usam **Haskell**:
  - ▶ **ABN AMRO** análise de riscos financeiros
  - ▶ **AT&T** automatização de processamento de formulários
  - ▶ **Bank of America Merrill Lynch** transformação de dados
  - ▶ **Bump** servidores baseados em Haskell
  - ▶ **Facebook** manipulação da base de código PHP
  - ▶ **Google** infra-estrutura interna de TI
  - ▶ **MITRE** análise de protocolos de criptografia
  - ▶ **NVIDIA** ferramentas usadas internamente
  - ▶ **Qualcomm, Inc** geração de interfaces de programação para Lua
  - ▶ **The New York Times** processamento de imagens
- Para mais detalhes visite a página Haskell na indústria em [https://wiki.haskell.org/Haskell\\_in\\_industry](https://wiki.haskell.org/Haskell_in_industry).

# Ambientes de Desenvolvimento Haskell



WinGHC

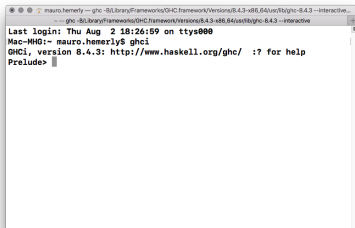
```
File Edit Actions Tools Help
[Icons]
GHCi, version 7.8.3: http://www.haskell.org/ghc/ :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
Prelude>
```



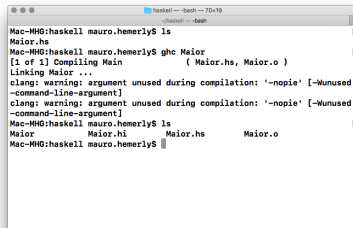
codingground

Compile and Execute Haskell Online (Glasgow, v8.0.2)

```
ghc -c main.hs -o main -x86_64-linux-glibc2.17
[1 of 1] Compiling Main
Linking main ...
Main
hello world
```



```
mauro.hemerly ~ ghc -B/Library/Frameworks/GHC.framework/Versions/8.4.3-x86_64/usr/lib/ghc-8.4.3 --interactive...
Last login: Thu Aug 2 18:26:59 on ttys000
Mac-MHG:~ mauro.hemerly$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/ :? for help
Prelude>
```



```
Mac-MHG:haskell mauro.hemerly$ ls
Maior.hs
Mac-MHG:haskell mauro.hemerly$ ghc Maior
[1 of 1] Compiling Main
Linking Maior ...
clang: warning: argument unused during compilation: '-nopie' [-Wunused-command-line-argument]
clang: warning: argument unused during compilation: '-nopie' [-Wunused-command-line-argument]
Mac-MHG:haskell mauro.hemerly$ ls
Maior      Maior.hi      Maior.hs      Maior.o
Mac-MHG:haskell mauro.hemerly$
```

# Linguagem Haskell

## O que é um Programa Funcional

- Um programa funcional é uma **expressão**.

- Exemplo:

$(2 * 3) + 4$

- Executar um programa funcional significa **avaliar** a expressão:

$(2 * 3) + 4$  avalia para 10.

# Linguagem Haskell

## O que é um Programa Funcional

- A Programação Funcional é um estilo de programação em que o método básico de computação é a **expressão**.
- Haskell deve seu nome ao matemático **Haskell B. Curry**, conhecido por seu trabalho em lógica combinatória e pioneiro no desenvolvimento do Cálculo Lambda (Cálculo- $\lambda$ ), inspiração aos projetistas da maioria das linguagens funcionais.



# Java vs Haskell

- Para somar os números inteiros de 1 a 10 podemos escrever em linguagem **Java**:

```
total = 0;
for (i = 1; i <= 10; i++) {
    total = total + i;
}
```

**O método da computação é baseado em atribuição de valores à variáveis.**

- A soma dos números inteiros de 1 a 10 pode ser escrita em **Haskell** como:

```
sum [1..10]
```

**O método da computação é baseado em aplicação de funções à argumentos.**

# Java vs Haskell

- Para somar os números inteiros de 1 a 10 podemos escrever em linguagem **Java**:

```
total = 0;
for (i = 1; i <= 10; i++) {
    total = total + i;
}
```

**O método da computação é baseado em atribuição de valores à variáveis.**

- A soma dos números inteiros de 1 a 10 pode ser escrita em **Haskell** como:

```
sum [1..10]
```

**O método da computação é baseado em aplicação de funções à argumentos.**

# Java vs Haskell

- Para somar os números inteiros de 1 a 10 podemos escrever em linguagem **Java**:

```
total = 0;
for (i = 1; i <= 10; i++) {
    total = total + i;
}
```

**O método da computação é baseado em atribuição de valores à variáveis.**

- A soma dos números inteiros de 1 a 10 pode ser escrita em **Haskell** como:

```
sum [1..10]
```

**O método da computação é baseado em aplicação de funções à argumentos.**

# Como Definir uma Função

- Funções em Haskell são normalmente definidas pelo uso de equações. Por exemplo, a função **soma** pode ser escrita:

```
soma x y = x + y  
Prelude> soma 12 34  
46
```



- A função **incrementar** pode ser escrita e testada:

```
incrementar x = x + 1  
Prelude> incrementar 99  
100
```

# Como Definir uma Função

- Funções em Haskell são normalmente definidas pelo uso de equações. Por exemplo, a função **soma** pode ser escrita:

```
soma x y = x + y  
Prelude> soma 12 34  
46
```



- A função **incrementar** pode ser escrita e testada:

```
incrementar x = x + 1  
Prelude> incrementar 99  
100
```

# Como Definir uma Função

## Assinatura e Parâmetro Formal

- Retomando a função **incrementar** do slide anterior.

```
incrementar :: Int -> Int
```

```
incrementar x = x + 1
```

- Detalhes da definição da função:

- A linha 1 é a **assinatura** da função, isto é, ela declara como uma função recebendo um inteiro como entrada e dando ao usuário um inteiro como saída.
- A linha 2 é o **corpo** da função, isto é, ela determina um inteiro  $x + 1$  a partir de um inteiro  $x$  recebido.
- A linha 3 é o **tipo** da função, isto é, ela declara que a função recebe um inteiro e retorna um inteiro.

# Como Definir uma Função

## Assinatura e Parâmetro Formal

- Retomando a função **incrementar** do slide anterior.

```
incrementar :: Int -> Int
```

```
incrementar x = x + 1
```

- Detalhes da definição da função:
  - linha 1: a **assinatura** declara **incrementar** como uma função esperando um **inteiro na entrada** e tendo na **saída um inteiro**.
  - linha 2: **define** que a função **incrementar** determina um inteiro  $x + 1$  a partir de um inteiro  $x$  qualquer.
  - o símbolo  $x$  é um **parâmetro formal**.

# Como Definir uma Função

## Assinatura e Parâmetro Formal

- Retomando a função **incrementar** do slide anterior.

**incrementar** :: Int -> Int

**incrementar** x = x + 1

- Detalhes da definição da função:
  - ▶ linha 1: a **assinatura** declara **incrementar** como uma função esperando um **inteiro na entrada** e tendo na **saída um inteiro**.
  - ▶ linha 2: **define** que a função **incrementar** determina um inteiro  $x + 1$  a partir de um inteiro  $x$  qualquer.
  - ▶ o símbolo  $x$  é um **parâmetro formal**.



# Como Definir uma Função

## Assinatura e Parâmetro Formal

- Retomando a função **incrementar** do slide anterior.

```
incrementar :: Int -> Int
```

```
incrementar x = x + 1
```

- Detalhes da definição da função:
  - ▶ linha 1: a **assinatura** declara **incrementar** como uma função esperando um **inteiro na entrada** e tendo na **saída um inteiro**.
  - ▶ linha 2: **define** que a função **incrementar** determina um inteiro  $x + 1$  a partir de um inteiro  $x$  qualquer.
  - ▶ o símbolo  $x$  é um **parâmetro formal**.

# Como Definir uma Função

## Assinatura e Parâmetro Formal

- Retomando a função **incrementar** do slide anterior.

```
incrementar :: Int -> Int
```

```
incrementar x = x + 1
```

- Detalhes da definição da função:
  - ▶ linha 1: a **assinatura** declara **incrementar** como uma função esperando um **inteiro na entrada** e tendo na **saída um inteiro**.
  - ▶ linha 2: **define** que a função **incrementar** determina um inteiro  $x + 1$  a partir de um inteiro  $x$  qualquer.
  - ▶ o símbolo  $x$  é um **parâmetro formal**.

# Como Definir uma Função

## Convenções de Nomeação Obrigatórias

- Exemplo:

```
incrementar :: Int -> Int
```

```
incrementar x = x + 1
```

- Convenções:

• Funções e parâmetros sempre iniciam com letra minúscula.

• Variáveis e constantes sempre iniciam com letra maiúscula.

• Constantes sempre iniciam com uma letra maiúscula.

• Funções sempre iniciam com uma letra minúscula.

• Variáveis sempre iniciam com uma letra minúscula.

• Constantes sempre iniciam com uma letra maiúscula.

# Como Definir uma Função

## Convenções de Nomeação Obrigatórias

- Exemplo:

```
incrementar :: Int -> Int
```

```
incrementar x = x + 1
```

- Convenções:

- ▶ Funções e parâmetros formais iniciam com letra **minúscula**.
- ▶ Tipos iniciam com letra **maiúscula**.

# Como Definir uma Função

## Convenções de Nomeação Obrigatórias

- Exemplo:

```
incrementar :: Int -> Int
```

```
incrementar x = x + 1
```

- Convenções:

- ▶ **Funções e parâmetros formais** iniciam com letra **minúscula**.
- ▶ **Tipos** iniciam com letra **maiúscula**.

# Como Definir uma Função

## Convenções de Nomeação Obrigatórias

- Exemplo:

```
incrementar :: Int -> Int
```

```
incrementar x = x + 1
```

- Convenções:

- ▶ **Funções** e **parâmetros formais** iniciam com letra **minúscula**.
- ▶ **Tipos** iniciam com letra **maiúscula**.

# Como Definir uma Função

## Definição vs Atribuição

- A definição

`incrementar x = x + 1`

Não deve se confundida com a atribuição

`x = x + 1`

em linguagens imperativas.

# Como Definir uma Função

## Funções com Mais de um Parâmetro Formal

- Exemplo de uma função definida com dois parâmetros:

```
dobraSoma :: Int -> Int -> Int
dobraSoma x y = 2 * (x + y)
```

- A combinação das funções **incrementa** e **dobraSoma**:

```
f :: Int -> Int -> Int
f x y = incrementa(dobraSoma x y)
```



# Como Definir uma Função

## Funções com Mais de um Parâmetro Formal

- Exemplo de uma função definida com dois parâmetros:

```
dobraSoma :: Int -> Int -> Int
dobraSoma x y = 2 * (x + y)
```

- A combinação das funções **incrementa** e **dobraSoma**:

```
f :: Int -> Int -> Int
f x y = incrementa(dobraSoma x y)
```

# Como Definir uma Função

## Matemática vs Haskell

Maths	Haskell
$\max(x, y)$	<code>max x y</code>
$3(x^2 + y^2) + 5$	<code>3*(x^2+y^2)+5</code>
$\text{ar} : \text{Float} \rightarrow \text{Float}$	<code>ar :: Float -&gt; Float</code>

# Linguagem Haskell

## Outros Exemplos de Funções

- Algumas funções para calcular a área de figuras podem ser definidas:

```
areaTriangulo b h = (b * h) / 2
```

```
areaRetangulo b h = b * h
```

```
areaCirculo r = pi * r * r
```

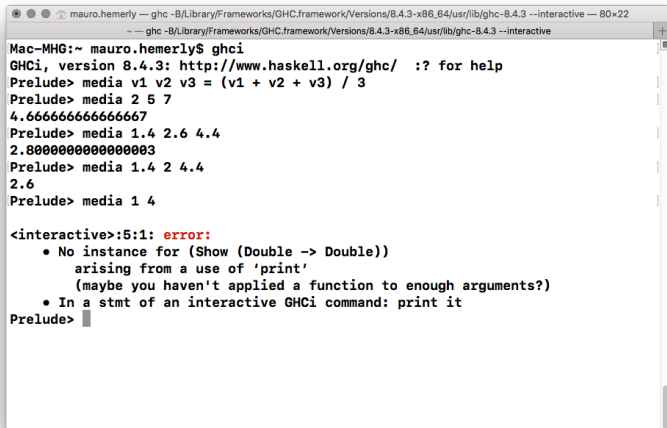
```
areaTrapezio a b h = (a + b) * h / 2
```



# Linguagem Haskell

## Outros Exemplos de Funções

- Função para calcular a média entre três números:



```
mauro.hemerly — ghc -B/Library/Frameworks/GHC.framework/Versions/8.4.3-x86_64/usr/lib/ghc-8.4.3 --interactive — 80x22
~ — ghc -B/Library/Frameworks/GHC.framework/Versions/8.4.3-x86_64/usr/lib/ghc-8.4.3 --interactive
Mac-MHG:~ mauro.hemerly$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Prelude> media v1 v2 v3 = (v1 + v2 + v3) / 3
Prelude> media 2 5 7
4.666666666666667
Prelude> media 1.4 2.6 4.4
2.8000000000000003
Prelude> media 1.4 2 4.4
2.6
Prelude> media 1 4
<interactive>:5:1: error:
• No instance for (Show (Double -> Double))
  arising from a use of 'print'
  (maybe you haven't applied a function to enough arguments?)
• In a stmt of an interactive GHCi command: print it
Prelude> 
```

# Exercícios em Haskell

- 1 Escreva uma função para calcular o **dobro de um número**.
- 2 Escreva uma função para **quadruplicar um número**, usando a função definida no exercício anterior.
- 3 Defina uma função para calcular a **distância entre dois pontos** (num plano).
- 4 Dadas as medidas dos catetos de um triângulo retângulo, calcular o **valor de sua hipotenusa**.
- 5 Considere as seguintes definições de funções:

**inc**  $x = x + 1$

**dobro**  $x = x + x$

**quadrado**  $x = x * x$

**media**  $x\ y = (x + y)/2$

Efetuando **reduções passo-a-passo**, calcule os valores das expressões seguintes:

(a) **inc** (**quadrado** 5)

(b) **quadrado** (**inc** 5)

(c) **media** (**dobro** 3) (**inc** 5)

# Linguagem Haskell

## Tipos Básicos

tipo	características	exemplos de valores
<b>Int</b>	<ul style="list-style-type: none"><li>– inteiros de precisão fixa</li><li>– limitado (tem um valor mínimo e um valor máximo)</li><li>– faixa de valores determinada pelo tamanho da palavra da plataforma</li></ul>	876 2012
<b>Integer</b>	<ul style="list-style-type: none"><li>– inteiros de precisão arbitrária</li><li>– ilimitado (qualquer número inteiro pode ser representado desde que haja memória suficiente)</li><li>– menos eficiente que <b>Int</b></li></ul>	10 7547387487840030454523342092382
<b>Float</b>	<ul style="list-style-type: none"><li>– aproximação de números reais em ponto flutuante</li><li>– precisão simples</li></ul>	4.56 0.201E10
<b>Double</b>	<ul style="list-style-type: none"><li>– aproximação de números reais em ponto flutuante</li><li>– precisão dupla</li></ul>	78643 987.3201E-60

# Linguagem Haskell

## Tipos Básicos

<b>Rational</b>	<ul style="list-style-type: none"><li>– números racionais</li><li>– precisão arbitrária</li><li>– representados como uma razão de dois valores do tipo <b>Integer</b></li><li>– os valores podem ser construídos usando o operador % do módulo <b>Data.Ratio</b> (precedência 7 e associatividade à esquerda) <b>import Data.Ratio</b></li></ul>	<pre>3 % 4 8 % 2 5 % (-10)</pre>
<b>Bool</b>	<ul style="list-style-type: none"><li>– valores lógicos</li></ul>	<pre>False True</pre>

# Linguagem Haskell

## Tipos Básicos

<b>Char</b>	<ul style="list-style-type: none"><li>– enumeração cujos valores representam caracteres unicode</li><li>– estende o conjunto de caracteres ISO 8859-1 (latin-1), que é uma extensão do conjunto de caracteres ASCII</li></ul>	<code>'B'</code> <code>'!'</code> <code>'\n'</code> nova linha <code>'\LF'</code> nova linha <code>'^J'</code> nova linha <code>'\10'</code> nova linha <code>'\''</code> aspas simples <code>'\\'</code> barra invertida
<b>String</b>	<ul style="list-style-type: none"><li>– sequências de caracteres</li></ul>	<code>"Brasil"</code> <code>""</code> <code>"bom\ndia"</code> <code>"altura:\10\&amp;199.4"</code> <code>"primeiro/ /segundo"</code>



# Linguagem Haskell

## Tipos Básicos: observações

- Alguns literais são **sobrecarregados**. Isto significa que um mesmo literal pode ter mais de um tipo, dependendo do contexto em que é usado. O tipo correto do literal é escolhido pela análise desse contexto.
- Em particular:
  - ▶ os **literais inteiros** podem ser de qualquer tipo numérico, como **Int**, **Integer**, **Float**, **Double** ou **Rational**, e
  - ▶ os **literais fracionários** podem ser de qualquer tipo numérico fracionário, como **Float**, **Double** ou **Rational**.
- Por exemplo:
  - ▶ o literal inteiro **2018** pode ser de **qualquer tipo numérico** (como **Int**, **Integer**, **Float**, **Double** ou **Rational**)
  - ▶ o literal **5.61** pode ser de **qualquer tipo fracionário** (como **Float**, **Double** ou **Rational**).

# Linguagem Haskell

## Valores Booleanos: **Bool**

- Valores **True** e **False**

- Funções pré-definidas:

<code>not :: Bool -&gt; Bool</code>	negação
<code>(&amp;&amp;) :: Bool -&gt; Bool -&gt; Bool</code>	conjunção (infixa)
<code>(  ) :: Bool -&gt; Bool -&gt; Bool</code>	disjunção inclusiva (infixa)

- Exemplos:

`True && False ~> False`      (`~>` significa avalia como ...)

`True || False ~> True`

`True || True ~> True`

# Linguagem Haskell

## Valores Booleanos: **Bool**

- Valores **True** e **False**
- Funções pré-definidas:

`not :: Bool -> Bool`

negação

`(&&) :: Bool -> Bool -> Bool`

conjunção (infixa)

`(||) :: Bool -> Bool -> Bool`

disjunção inclusiva (infixa)

- Exemplos:

`True && False ~> False`

(~> significa avalia como ...)

`True || False ~> True`

`True || True ~> True`

# Linguagem Haskell

## Valores Booleanos: **Bool**

- Valores **True** e **False**
- Funções pré-definidas:

`not :: Bool -> Bool`

negação

`(&&) :: Bool -> Bool -> Bool`

conjunção (infixa)

`(||) :: Bool -> Bool -> Bool`

disjunção inclusiva (infixa)

- Exemplos:

`True && False ~> False`

(~> significa avalia como ...)

`True || False ~> True`

`True || True ~> True`

# Linguagem Haskell

## Valores Booleanos: **Bool**

- Exemplo: disjunção exclusiva

```
exOr :: Bool -> Bool -> Bool
```

```
exOr x y = (x || y) && (not (x && y))
```

```
exOr True True ~> False
```

```
exOr True False ~> True
```

```
exOr False True ~> True
```

```
exOr False False ~> False
```

# Linguagem Haskell

## Exemplos de Funções

```
media2 :: Double -> Double -> Double  
media2 x y = (x + y)/2
```

```
notaFinal :: Double  
notaFinal = media2 4.5 7.2
```

```
discriminante :: Double -> Double -> Double -> Double  
discriminante a b c = b^2 - 4*a*c
```

# Linguagem Haskell

## Exemplos de Funções

```
ladosTriangulo :: Float -> Float -> Float -> Bool
ladosTriangulo a b c =
    a > 0 &&
    b > 0 &&
    c > 0 &&
    a < b + c && b < a + c && c < a + b
```

```
ladosTriangulo2 :: Float -> Float -> Float -> Bool
ladosTriangulo2 a b c =
    a > 0 && b > 0 && c > 0 && a < b + c && b < a + c && c < a + b
```

# Linguagem Haskell

- Operadores do tipo **Int**

+, *	Soma e multiplicação de inteiros
^	Potência: $2^4$ é 16
-	Serve para mudar o sinal de um inteiro ou para fazer a subtração

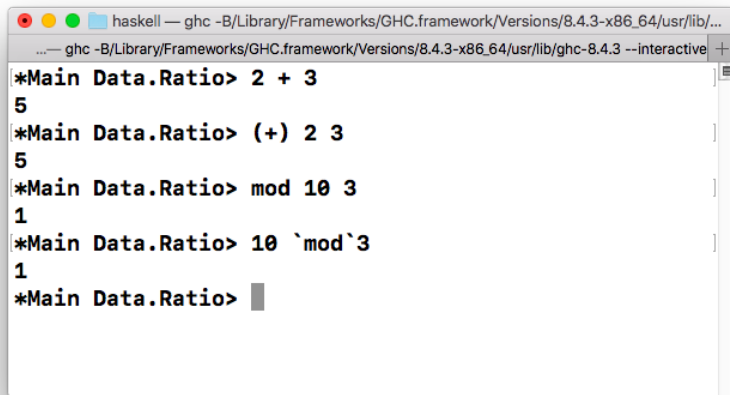
- Funções do tipo **Int**

div	Divisão de números inteiros; <code>div 10 3</code> é 3
mod	O resto de uma divisão de inteiros; <code>mod 10 3</code> é 1
abs	Valor absoluto de um inteiro (remove o sinal).
negate	Muda o sinal de um inteiro.



# Linguagem Haskell

- Qualquer **operador** pode ser usado como **função**, e qualquer **função** pode ser usada como um **operador**, basta incluir o **operador entre parênteses**, e a **função entre crases**.

A screenshot of a Haskell GHC interactive session window. The window title is "haskell — ghc -B/Library/Frameworks/GHC.framework/Versions/8.4.3-x86\_64/usr/lib/...". The prompt is "\*Main Data.Ratio>". The user has entered four expressions: "2 + 3", "(+) 2 3", "mod 10 3", and "10 `mod` 3". The corresponding outputs are "5", "5", "1", and "1". The cursor is at the end of the prompt line.

```
haskell — ghc -B/Library/Frameworks/GHC.framework/Versions/8.4.3-x86_64/usr/lib/...
...— ghc -B/Library/Frameworks/GHC.framework/Versions/8.4.3-x86_64/usr/lib/ghc-8.4.3 --interactive +
*Main Data.Ratio> 2 + 3
5
*Main Data.Ratio> (+) 2 3
5
*Main Data.Ratio> mod 10 3
1
*Main Data.Ratio> 10 `mod` 3
1
*Main Data.Ratio> █
```

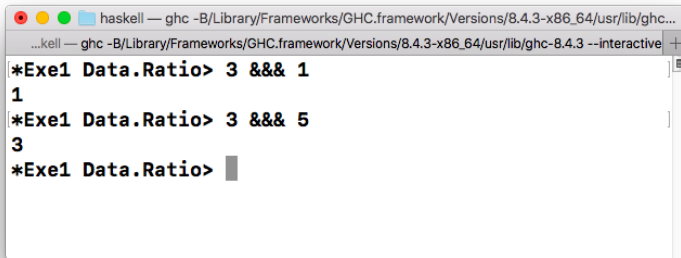
# Linguagem Haskell

- Você pode definir os seus próprios operadores

## Definindo operador &&&

```
(&&&) :: Int -> Int -> Int
```

```
a &&& b = if a < b then a else b
```



```
haskell — ghc -B/Library/Frameworks/GHC.framework/Versions/8.4.3-x86_64/usr/lib/ghc-...
...kell — ghc -B/Library/Frameworks/GHC.framework/Versions/8.4.3-x86_64/usr/lib/ghc-8.4.3 --interactive +
*Exe1 Data.Ratio> 3 &&& 1
1
*Exe1 Data.Ratio> 3 &&& 5
3
*Exe1 Data.Ratio> █
```

# Linguagem Haskell

## Operadores de Comparação

>	Maior que
>=	Maior ou igual
==	Igual
/=	Diferente
<=	Menor ou igual
<	Menor

# Linguagem Haskell

## Caracteres e Strings

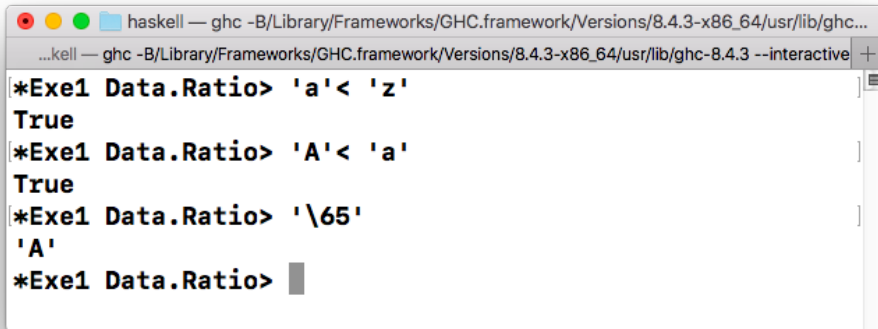
- O tipo **Char** é o tipo composto de caracteres, dígitos e caracteres especiais, como nova linha, tabulação, etc. Caracteres individuais são escritos entre aspas simples: **'a'** é o caracter **a** e **'7'** é o caracter **sete**.
- Alguns **caracteres especiais** são representados da seguinte maneira:

<code>'\t'</code>	Tabulação
<code>'\n'</code>	Nova linha
<code>'\''</code>	Aspas simples (')
<code>'\"'</code>	Aspas duplas (")
<code>'\\'</code>	Barra (\)

# Linguagem Haskell

## Caracteres e Strings

- Os caracteres são **ordenados** internamente pela tabela **ASCII**.



```
haskell — ghc -B/Library/Frameworks/GHC.framework/Versions/8.4.3-x86_64/usr/lib/ghc...
...kell — ghc -B/Library/Frameworks/GHC.framework/Versions/8.4.3-x86_64/usr/lib/ghc-8.4.3 --interactive +
[*Exe1 Data.Ratio> 'a' < 'z'
True
[*Exe1 Data.Ratio> 'A' < 'a'
True
[*Exe1 Data.Ratio> '\65'
'A'
[*Exe1 Data.Ratio> ]
```

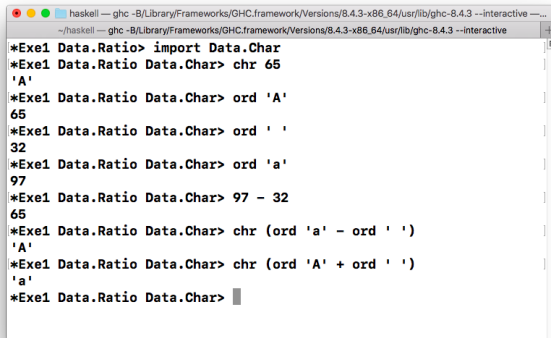
# Linguagem Haskell

## Caracteres e Strings

- Existem funções que transformam um **número em caracter**, e um **caracter em número inteiro**, baseando-se na tabela ASCII. Respectivamente:

`chr :: Int -> Char`

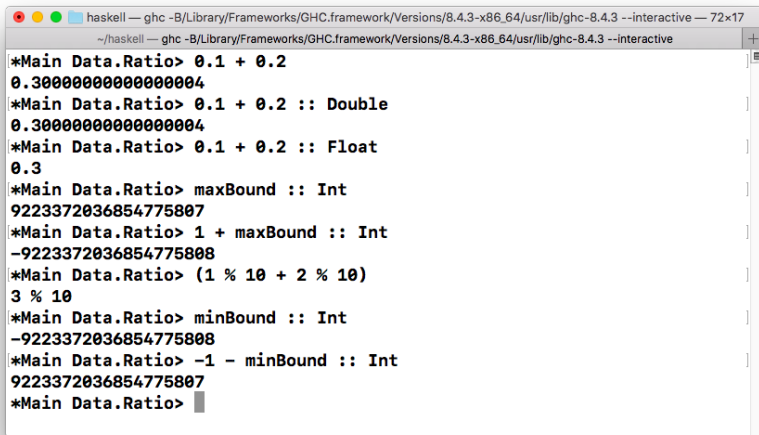
`ord :: Char -> Int`



```
haskell — ghc -B/Library/Frameworks/GHC.framework/Versions/8.4.3-x86_64/usr/lib/ghc-8.4.3 --interactive --...
~/haskell — ghc -B/Library/Frameworks/GHC.framework/Versions/8.4.3-x86_64/usr/lib/ghc-8.4.3 --interactive
*Exe1 Data.Ratio> import Data.Char
*Exe1 Data.Ratio Data.Char> chr 65
'A'
*Exe1 Data.Ratio Data.Char> ord 'A'
65
*Exe1 Data.Ratio Data.Char> ord ' '
32
*Exe1 Data.Ratio Data.Char> ord 'a'
97
*Exe1 Data.Ratio Data.Char> 97 - 32
65
*Exe1 Data.Ratio Data.Char> chr (ord 'a' - ord ' ')
'A'
*Exe1 Data.Ratio Data.Char> chr (ord 'A' + ord ' ')
'a'
*Exe1 Data.Ratio Data.Char> 
```

# Linguagem Haskell

## Algumas Curiosidades

A screenshot of a Haskell GHC interactive session window. The window title is "haskell — ghc -B/Library/Frameworks/GHC.framework/Versions/8.4.3-x86\_64/usr/lib/ghc-8.4.3 --interactive — 72x17". The terminal shows several commands and their outputs, demonstrating type coercion and arithmetic operations.

```
haskell — ghc -B/Library/Frameworks/GHC.framework/Versions/8.4.3-x86_64/usr/lib/ghc-8.4.3 --interactive — 72x17
~/haskell — ghc -B/Library/Frameworks/GHC.framework/Versions/8.4.3-x86_64/usr/lib/ghc-8.4.3 --interactive
*Main Data.Ratio> 0.1 + 0.2
0.30000000000000004
*Main Data.Ratio> 0.1 + 0.2 :: Double
0.30000000000000004
*Main Data.Ratio> 0.1 + 0.2 :: Float
0.3
*Main Data.Ratio> maxBound :: Int
9223372036854775807
*Main Data.Ratio> 1 + maxBound :: Int
-9223372036854775808
*Main Data.Ratio> (1 % 10 + 2 % 10)
3 % 10
*Main Data.Ratio> minBound :: Int
-9223372036854775808
*Main Data.Ratio> -1 - minBound :: Int
9223372036854775807
*Main Data.Ratio> 
```