Programação Funcional — BCC222

Aula 1

Funções

Lucília Camarão de Figueiredo
Departamento de Ciência da Computação
Universidade Federal de Ouro Preto

Parte I

Introdução

Porque aprender Haskell?

- É importante aprender diversas linguagens ao longo da carreira
- Aprender a pensar de maneira diferente sobre programas
- Aprender a operar sobre estruturas de dados de maneira geral
- Mesmo que você não se apaixone por Haskell, nem venha a usar liguagens funcionais ao longo de sua carreira profissional, aprender os conceitos de programação funcional irá contribuir para que você desenvolva programas mais claros, concisos, modulares e reusáveis

O que é Haskell?

Uma linguagem de programação funcional

- Pura
- Funções de ordem superior
- *Lazy*
- Fortemente tipada
- Usa *type classes*
- Para uso educacional, em pesquisa e profissionalmente
- Projetada por um comitê
- Recentemente celewbrou 20 anos

"A History of Haskell: being lazy with class", Paul Hudak (Yale University), John Hughes (Chalmers University), Simon Peyton Jones (Microsoft Research), Philip Wadler (Edinburgh University), The Third ACM SIGPLAN History of Programming Languages Conference (HOPL-III), San Diego, California, June 9–10, 2007

Famílias de Linguagens de Programação

- Funcional
  Erlang, F#, Haskell, Hope, Javascript, Miranda, O'Caml, Scala, Scheme, SML
  - Mais expressivas
  - Programas mais compactos e modulares
- Orientada a Objetos
  C++, F#, Java, Javascript, O'Caml, Perl, Python, Ruby, Scala
  - Mais largamente usadas
  - Maior conjunto de blibliotecas

Programação funcional é o futuro

Características de linguagens funcionais têm sido incorporadas em outras linguagens

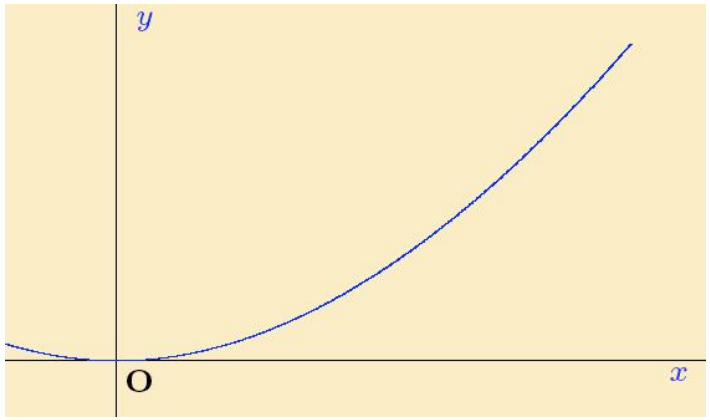
- Garbage collection (Java, C#, Python, Perl, Ruby, Javascript)
- Higher-order functions (Java, C#, Python, Perl, Ruby, Javascript)
- Generics (Java, C#)
- List comprehensions (C#, Python, Perl 6, Javascript)
- Type classes (C++ “concepts”)

Parte II

Funções

O que é uma função?

- Uma receita para gerar uma saída a partir de dados de entrada:
  "Multiplicar um número por ele próprio"
- Um conjunto de pares (entrada, saída):
  (1,1) (2,4) (3,9) (4,16) (5,25) ...
- Um gráfico relacionando entradas e saídas (apenas para números):

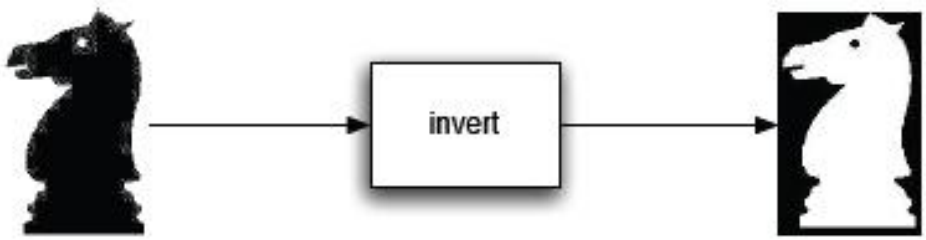


Tipos de dados

- Integers: 42, -69
- Floats: 3.14
- Characters: ’h’
- Strings: "hello"
- Pictures: 

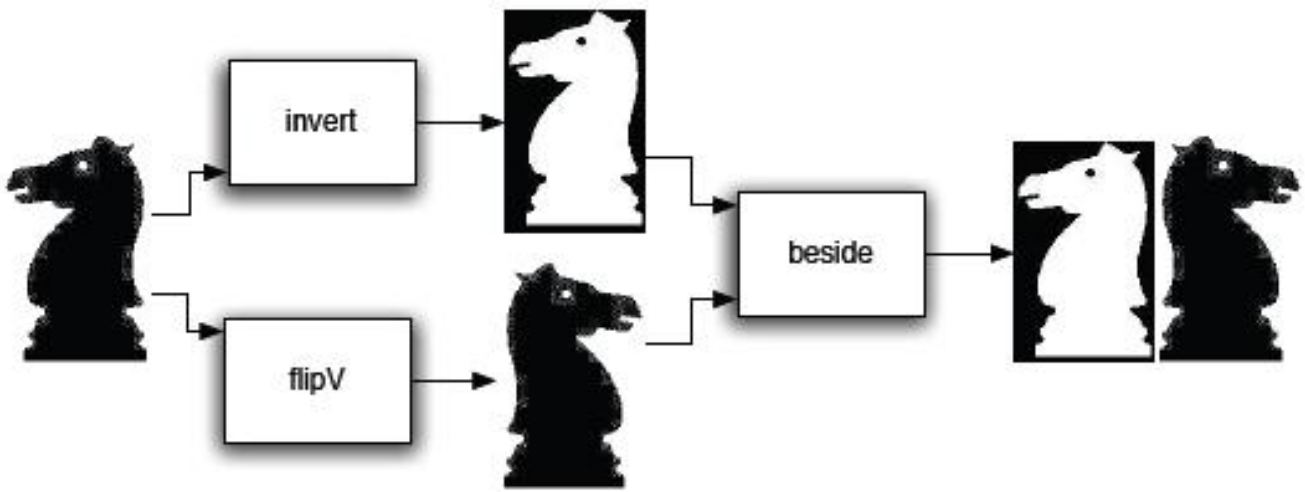
Aplicando uma função

```
invert ::  Picture -> Picture
knight ::  Picture

invert knight
```



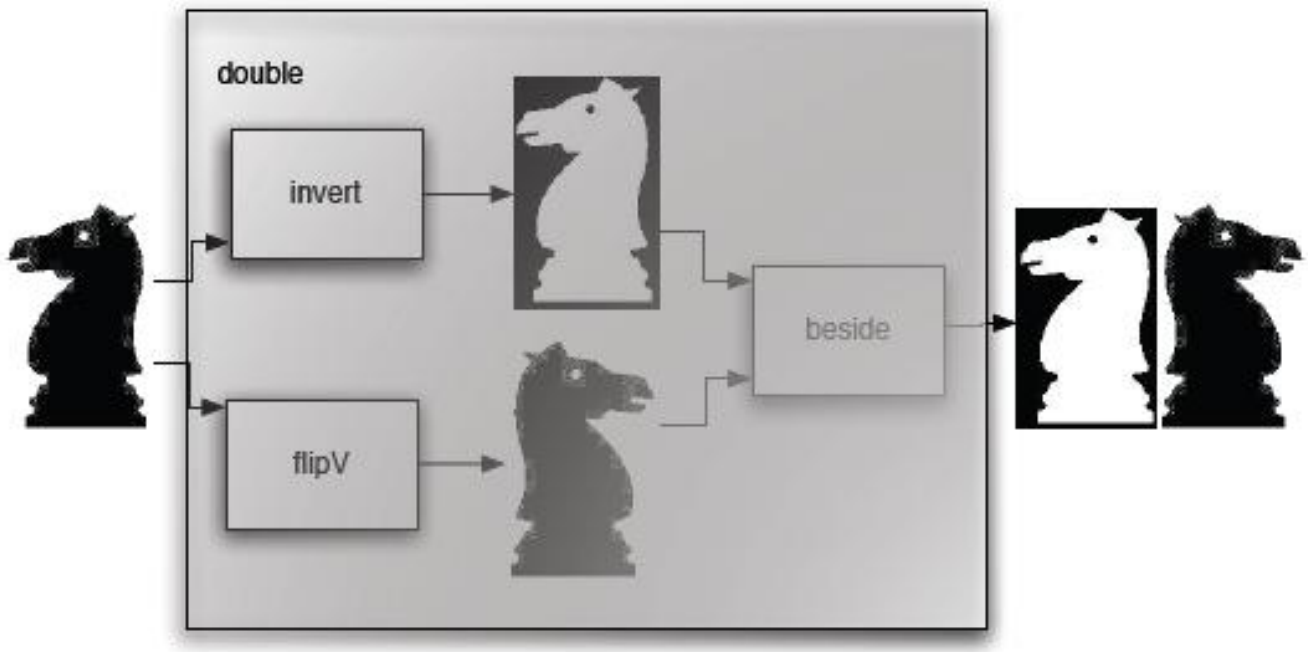
Compondo funções

```
beside ::  Picture -> Picture -> Picture
flipV ::  Picture -> Picture
invert ::  Picture -> Picture
knight ::  Picture

beside (invert knight) (flipV knight)
```



Definindo uma nova função

```
double ::  Picture -> Picture
double p = beside (invert p) (flipV p)

double knight
```



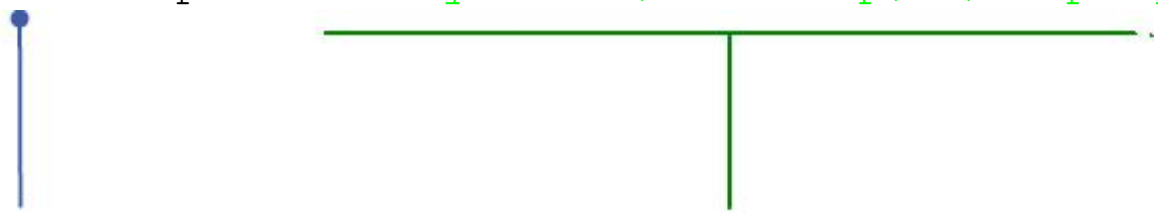
Terminologia

Declaração de tipo (assinatura)

```
makePicture ::  Picture -> Picture
```

Declaração de função

```
makePicture p = sideBySide (invert p) (flipV p)
```



nome da função

corpo da função

Terminologia