Paradigmas de Programação

Fabrício Olivetti de França

05 de Junho de 2018

Paradigmas de Programação

~~Paradigmas de Programação~~

Haskell 😀

Haskell

- Surgiu em 1990 com o objetivo de ser a primeira linguagem puramente funcional.
- Por muito tempo considerada uma linguagem acadêmica.
- Atualmente é utilizada em diversas empresas (totalmente ou em parte de projetos).

Haskell: Características

Por ter sido criada por um comitê de estudiosos de linguagem de programação funcional e com a mentalidade de mantê-la útil para o ensino e pesquisa de linguagem de programação, assim como uso em empresas, a linguagem adquiriu diversas características distintas e interessantes não observadas em outras linguagens.

Haskell: Características

- **Códigos concisos e declarativos:** o programador *declara* o que ele quer ao invés de escrever um passo-a-passo. Programas em Haskell chegam a ser dezenas de vezes menores que em outras linguagens.

```
take 100 [x | x <- N, primo x]
```

Haskell: Características

- **Sistema de tipagem forte:** ao contrário de linguagens como *Java* e *C*, as declarações de tipo no Haskell são simplificadas (e muitas vezes podem ser ignoradas), porém, seu sistema rigoroso permite que muitos erros comuns sejam detectados em tempo de **compilação**.

```
int x    = 10;
double y = 5.1;
System.out.println("Resultado:  " + (x*y));
```

OK!

Haskell: Características

- **Sistema de tipagem forte:** ao contrário de linguagens como *Java* e *C*, as declarações de tipo no Haskell são simplificadas (e muitas vezes podem ser ignoradas), porém, seu sistema rigoroso permite que muitos erros comuns sejam detectados em tempo de **compilação**.

```
x = 10  :: Int
y = 5.1 :: Double
print ("Resultado: " + (x*y) )
```

ERRO!

Haskell: Características

- **Compreensão de listas:** listas são frequentemente utilizadas para a solução de diversos problemas. O Haskell utiliza listas como um de seus conceitos básicos permitindo uma notação muito parecida com a notação de conjuntos na matemática.

$$xs = \{x \mid x \in \mathbb{N}, x \text{ impar}\}$$

```
xs = [x | x <- N, impar x]
```

Haskell: Características

- **Imutabilidade:** não existe um conceito de variável, apenas nomes e declarações. Uma vez que um nome é declarado com um valor, ele não pode sofrer alterações.

```
x = 1.0
x = 2.0
```

ERRO!

Haskell: Características

- **Funções Recursivas:** com a imutabilidade, o conceito de laços de repetição também não existe em linguagens funcionais. Eles são implementados através de funções recursivas.

```
int x = 1;
for (int i = 1; i <= 10; i++) {
    x = x*2;
}
```

Haskell: Características

- **Funções Recursivas:** com a imutabilidade, o conceito de laços de repetição também não existe em linguagens funcionais. Eles são implementados através de funções recursivas.

```
f 0 = 1
f n = 2 * f (n-1)

print (f 1 10)
```

Haskell: Características

- **Funções de alta ordem:** funções podem receber funções como parâmetros. Isso permite definir funções genéricas, compor duas ou mais funções e definir linguagens de domínio específicos (ex.: *parsing*).

```
print (aplique dobro [1,2,3,4])
> [2,4,6,8]
```

Haskell: Características

- **Tipos polimórficos:** permite definir funções genéricas que funcionam para classes de tipos. Por exemplo, o operador de soma + pode ser utilizado para qualquer tipo numérico.

```
1 + 2           -- 3
1.0 + 3.0       -- 4.0
(2%3) + (3%6)   -- (7%6)
```

Haskell: Características

- **Avaliação preguiçosa:** ao aplicar uma função, o resultado será computado apenas quando requisitado. Isso permite evitar computações desnecessárias, estimula uma programação modular e permite estruturas de dados infinitos.

```
listaInf = [1..] -- 1, 2, 3, ...
print (take 10 listaInf)
```

Haskell: Características

- **Raciocínio equacional:** podemos usar expressões algébricas para otimizar nosso programa ou provar sua corretude.

muito cedo para dar um exemplo...

Ambiente de Programação

GHC e GHCi

Glasgow Haskell Compiler: compilador de código aberto para a linguagem Haskell.

Possui um modo interativo **ghci** (similar ao **iPython**).

Glasgow Haskell Compiler

Uso recomendado de:

Git - controle de revisão

Stack - gerenciamento de projeto e dependências

Haddock - documentação

Instalação

`https://www.haskell.org/downloads#platform`

Para o Linux escolha a distribuição *Generic*, mesmo que tenha pacote para sua distribuição.

Editor recomendado

Atom

com os pacotes:

- haskell-grammar
- language-haskell

Interpretador GHCi

```
$ ghci
> 2+3*4
14

> (2+3)*4
20

> sqrt (3^2 + 4^2)
5.0
```

Interpretador GHCi

O operador de exponenciação (^) tem prioridade maior do que o de multiplicação e divisão (*,/) que por sua vez tem prioridade maior que a soma e subtração (+,-).

```
$ ghci
> 2+3*4^5 == 2 + (3 * (4^5))
```

Informação de operadores

Para saber a prioridade de um operador basta digitar:

```
> :i (+)
class Num a where
  (+) :: a -> a -> a
  ...
    -- Defined in ‘GHC.Num’
infixl 6 +
```

A informação indica que + é um operador que pode ser utilizado para qualquer tipo numérico, tem precedência nível 6 (quanto maior o número maior sua prioridade) e é associativo a esquerda. Ou seja: $1 + 2 + 3$ vai ser computado na ordem $(1 + 2) + 3$.

Funções

Na matemática a aplicação de funções em seus argumentos é definido pelo nome da função e os parâmetros entre parênteses. A expressão $f(a, b) + c * d$ representa a aplicação de f nos parâmetros a, b e, em seguida, a soma do resultado com o resultado do produto entre c, d.

No Haskell, a aplicação de função é definida como o nome da função seguido dos parâmetros separados por espaço com a maior prioridade na aplicação da função. O exemplo anterior ficaria:

```
f a b + c*d
```

Funções

A tabela abaixo contém alguns contrastes entre a notação matemática e o Haskell:

Matemática	Haskell
$f(x)$	f x
$f(x, y)$	f x y
$f(g(x))$	f (g x)
$f(x, g(y))$	f x (g y)
$f(x)g(y)$	f x * g y

Hello World

Criem um arquivo *teste.hs*, abram no editor e no mesmo diretório iniciem o GHCi. No arquivo digitem:

```
dobra x = x + x

quadruplica x = dobra (dobra x)
```

Hello World

No GHCi:

```
> :l teste.hs
> quadruplica 10
40
```

O comando *:l* carrega as definições contidas em um arquivo fonte.

Hello World

Acrescentem a seguinte linha no arquivo fonte:

```
fatorial n = product [1..n]
```

e no GHCi:

```
> :reload
> fatorial 5
120
```

Outros comandos do GHCi

O comando *:t* mostra o tipo da função enquanto o comando *:q* sai do ghci.

```
> :t dobra
dobra :: Num a => a -> a

> :q
$
```

Convenções

Requisitos dos nomes

Os nomes das funções e seus argumentos devem começar com uma letra minúscula e seguida por 0 ou mais letras, maiúsculas ou minúsculas, dígitos, *underscore*, e aspas simples:

funcao, ordenaLista, soma1, x'

Nomes reservados

Os únicos nomes que não podem ser utilizados são:

case, class, data, default, deriving do, else, foreign, if, import, in infix, infixl, infixr, instance, let module, newtype, of, then, type, where

Convenção para listas

As listas são nomeadas acrescentando o caractere ‘s’ ao nome do que ela representa.

Uma lista de números n é nomeada ns, uma lista de variáveis x se torna xs. Uma lista de listas de caracteres tem o nome css.

Regra de layout

O layout dos códigos em Haskell é similar ao do Python, em que os blocos lógicos são definidos pela indentação.

```
f x = a*x + b
      where
        a = 1
        b = 3
z = f 2 + 3
```

A palavra-chave *where* faz parte da definição de *f*, da mesma forma, as definições de *a, b* fazem parte da cláusula *where*. A definição de *z* não faz parte de *f*.

Tabs vs Espaço

A definição de tabulação varia de editor para editor. Como o espaço é importante no Haskell, usem espaço ao invés de tab.

Comentários

Comentários em uma linha são demarcados pela sequência **--**, comentários em múltiplas linhas são demarcados por **{-** e **-}**:

```
-- função que dobra o valor de x
dobra x = x + x

{-
dobra recebe uma variável numérica
e retorna seu valor em dobro.
-}
```

Primeiro Projeto

Primeiro projeto compilável

Para criar projetos, utilizaremos a ferramenta **stack**. Essa ferramenta cria um ambiente isolado

```
$ stack new primeiro-projeto simple
$ cd primeiro-projeto
$ stack setup
$ stack build
$ stack exec primeiro-projeto
```

Os dois últimos comandos são referentes a compilação do projeto e execução.

Stack

O stack cria a seguinte estrutura de diretório:

- **LICENSE:** informação sobre a licença de uso do software.
- **README.md:** informações sobre o projeto em formato Markdown.
- **Setup.hs:** retrocompatibilidade com o sistema cabal.
- **primeiro-projeto.cabal:** informações das dependências do projeto. Atualizado automaticamente pelo stack.

Stack

- **stack.yaml:** parâmetros do projeto
- **package.yaml:** configurações de compilação e dependências de bibliotecas externas.
- **src/Main.hs:** arquivo principal do projeto.

Main.hs

```haskell
module Main where    -- indica que é o módulo principal

main :: IO ()
main = do                          -- início da função principal
  putStrLn "hello world"    -- imprime hello world
```

Atividade

Modifique o código *Main.hs* do *primeiro-projeto* criando uma função triplo que multiplica um valor x por 3.

Modifique a função *main* da seguinte forma para testar:

```
main :: IO ()
main = do
  print (triplo 2)
```