

Introdução

O que é programação funcional

Haskell



Cronograma do Minicurso

As aulas serão todas nesta sala (334 do CT)

Para ganhar o certificado:

- 50% de presença (quatro dias)

- 70% das atividades

A maioria das aulas vão ter uma pequena atividade por Google Forms

- Vocês não precisam acertar, apenas tentar fazer

Material (e atividades) em github.com/Fleivio/Minicurso_Haskell

STOP DOING HASKELL

- CODING LANGUAGES WERE NOT SUPPOSED TO BE PURE
- YEARS OF CODING yet NO REAL-WORLD USE FOUND for IMMUTABLE DATA
- Wanted to ^{make data immutable} anyway for a laugh? We had a tool for that: It was called **CONST**
- "Yes please ^{make iteration unviable}. Please force me to use currying." - Statements dreamed up by the utterly Deranged

LOOK at what Mathematicians have been demanding your Respect for all this time, with the pure coding environment we built for them

(This is REAL Haskell syntax ^{done by} tortured computer scientists):

```
concatEvenOdd :: [String] -> (String, String)
concatEvenOdd xs = foldl accumulation ("","") (zip [0,1..] xs)
  where
    accumulation :: (String, String) -> (Int, String) -> (String, String)
    accumulation (evenStr, oddStr) (i, str)
      | even i = (evenStr ++ str, oddStr)
      | otherwise = (evenStr, oddStr ++ str)
```

```
<#> :: Ord a => [a] -> [a] -> Int
[] <#> _ = 0
_ <#> [] = 0
(x:xs) <#> (y:ys)
  | x < y = 1 + xs <#> ys
  | otherwise = xs <#> ys
```

```
listSearch :: Eq a => [a] -> [a] -> Maybe Int
listSearch _ [] = Nothing
listSearch xs ys = parseList xs ys 0
  where
    parseList _ [] = Nothing
    parseList xs ys n
      | xs `isPrefixOf` ys = Just n
      | otherwise = parseList xs (tail ys) (n + 1)
```

?????

???????

????????????????

"Hello I would like $x \leftarrow [1,2..]$ apples please"

They have played us for absolute fools

Programação Funcional

**Funções de Alta
Ordem**

Imutabilidade

Puridade

Programação Funcional

Funções
Lambda

Tipos de
Dados
Algébricos

Laziness

**Funções de Alta
Ordem**

Imutabilidade

Puridade

Closures

Pattern
Matching

Aplicação
Parcial

Sem
efeitos
colaterais

Primeiro, um pouco de sintaxe

Aplicando funções

Haskell não usa parênteses:

`func x y ~ func(x, y)`

Primeiro, um pouco de sintaxe

```
mult2 :: Int -> Int
```

```
mult2 x = 2 * x
```

```
concat' :: String -> String -> String
```

```
concat' x y = x ++ y
```

Funções de Alta Ordem

Funções são cidadãos de primeira classe:

Funções recebem, modificam e retornam outras funções, como qualquer outro tipo de dado.

```
applyTwice :: (Int -> Int) -> Int -> Int
```

```
applyTwice f x = f (f x)
```

```
mult4 :: Int -> Int
```

```
mult4 x = applyTwice mult2 x
```

```
compose :: (Int -> Int) -> (Int -> Int) ->
```

```
Int -> Int
```

```
compose f g x = f (g x)
```


Cálculo Lambda

Programa é uma
composição de funções
e não como comandos
ordenados

Todas as funções são
anônimas

Variáveis:

x, y, z

Abstrações:

$(\lambda x \rightarrow x), (\lambda x y \rightarrow y x)$

Aplicações:

$(\lambda x \rightarrow x) a$

Redução β :

$(\lambda x \rightarrow x) a = a$

$(\lambda x y \rightarrow y x) a b = b a$

$(\lambda x y \rightarrow y x) a = (\lambda y \rightarrow y a)$

Aplicação parcial

Redução η :

$(\lambda x \rightarrow f x) = f$

Uma função é pura se

Não possui side-effects:

- Não printa ou acessa arquivos
- Não altera variáveis globais
- Sem variáveis estáticas locais
- Não altera argumentos passados por referência

É transparente de referencial:

Quando chamada com os mesmos argumentos, quantas vezes se queira, sempre retorna o mesmo resultado

Questão (10pts). Sendo $g(x)$ uma função, tal que

$$g(3) = 4$$

Quanto vale $g(3)$?

```
int numCalls = 0;
```

```
int impureInc(int x) {  
    numCalls++;  
    return x + numCalls;  
}
```

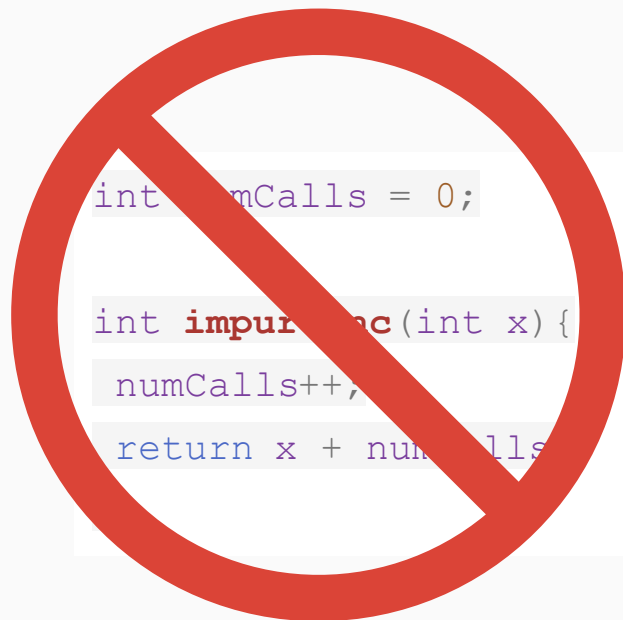
```
int pureInc(int x, int numCalls) {  
    numCalls++;  
    return x + numCalls;  
}
```

Não existem variáveis

Reforça a pureza

Uma vez que $x=3$,
 $x=3$ para sempre

Uma vez que $f(3) = 4$,
 $f(3) = 4$ para sempre



Consequência: Laziness

Se toda função é pura

Se o estado global não
importa/não existe



A ordem com que as funções
são avaliadas não importa

A linguagem pode avaliar
apenas o que for necessário

A linguagem só vai avaliar o necessário

Listas infinitas

Recursões infinitas

Resultados podem ser
reaproveitados/memoizados

Puridade dá margem para
otimizações

GHCup

Instalador

GHC

Compilador

HLS

Language
Server

Cabal

Gerenciador
de pacotes

Hoogle

Busca por
funções e
pacotes

Hackage

Repositório de
pacotes da
comunidade