

Programação Funcional

Classes em Linguagem Haskell

Classes de Tipos

- Usadas para permitir diferentes tipos para uma função ou operação.
- Operação de igualdade (==)
 - diferentes significados para diferentes tipos
 - nota-se que ela tem tipo:

`(==) :: a -> a -> Bool`

Funções polimórficas

```
Hugs> :type (==)  
(==) :: Eq a => a -> a -> Bool
```

```
Hugs> :type (>)  
(>) :: Ord a => a -> a -> Bool
```

```
Hugs> :type show  
show :: Show a => a -> String
```

Funções polimórficas

```
Hugs> :type show
show :: Show a => a -> String
Hugs> show 1
"1"
Hugs> show 45.4
"45.4"
Hugs> show [1,2,3]
"[1,2,3]"
Hugs> show ((+) 1 2)
"3"
Hugs> show ((+) 1 2) ++ show ((div) 4 2)
"32"
Hugs> show ((+) 1 2) ++ show ((/) 4 2)
"32.0"
```

Função monomórfica

- É definida apenas para tipos específicos.

```
sucessor :: Int -> Int  
sucessor x = x+1
```

```
Main> sucessor 5  
6
```

```
capitalize :: Char -> Char  
capitalize ch = chr (ord ch + offset)  
    where offset = ord 'A' - ord 'a'
```

```
Main> capitalize 'f'  
'F'
```

```
-- chr(ord 'f' - 32)  
-- chr (102 - 32)  
-- chr (70) = 'F'
```

Função polimórfica

- Uma única definição pode ser usada para diversos tipos de dados

```
concatena :: [a] -> [a] -> [a]
```

```
concatena [] y = y
```

```
concatena (x:xs) y = x: concatena xs y
```

```
Main> concatena [1,2,3] [4,5]  
[1,2,3,4,5]
```

```
Main> concatena ["aba", "ba"] ["va", "vav"]  
["aba", "ba", "va", "vav"]
```

```
Main> concatena [('a',1), ('b',2)] [('c',3)]  
[('a',1), ('b',2), ('c',3)]
```

Sobrecarga

- A função pode ser usada para vários (alguns) tipos de dados
 - diferentes definições para cada tipo
- Classe: coleção de tipos para os quais uma função está definida
- O conjunto de tipos para os quais ($=$) está definida é a *classe igualdade*, Eq

Classe Eq (igualdade)

- Define-se o que é necessário para um tipo t ser da classe
 - deve possuir uma função $(==)$ definida sobre t , do tipo $t \rightarrow t \rightarrow \text{Bool}$

```
class Eq t where
    (==) :: t -> t -> Bool
```


Instâncias

- Tipos membros de uma classe são chamadas instâncias
- São instâncias de Eq os tipos primitivos e as listas e tuplas de instâncias de Eq
 - `Int, Float, Char, Bool, [Int],
(Int, Bool), [[Char]], [(Int, [Bool])]`

Funções que usam igualdade

```
iguaisInt :: Int -> Int -> Int -> Bool  
iguaisInt n m p = (n == m) && (m == p)
```

```
iguais :: Eq t => t -> t -> t -> Bool  
iguais n m p = (n == m) && (m == p)
```

```
Main> iguaisInt 1 2 3
```

```
False
```

```
Main> iguais 'c' 'c' 'c'
```

```
True
```

Exercício

```
f :: Num a => a -> [b] -> [b] -> (a, [b])  
f x y z = (x + 1, y ++ z)
```

```
Main> [1] ++ [2]
```

```
[1,2]
```

```
Main> [1,2] ++ [3,4]
```

```
[1,2,3,4]
```

```
Main> :type (++)
```

```
(++) :: [a] -> [a] -> [a]
```

```
Main> f 1 [2] [3]
```

```
Main> f 1 2 3
```

```
Main> f 1 ['a'] ['b']
```

Definições Padrão

```
class Eq t where
  (==), (/=) :: t -> t -> Bool
  a /= b = not (a==b)
```

podem ser substituídas (*sobrecarregadas*)

Classes derivadas

```
class Eq t => Ord t where
  (<) , (<=) , (>) , (>=) :: t -> t -> Bool
  max, min :: t -> t -> t
  a <= b = (a < b || a == b)
  a > b  = b < a
  a < b  = b > a
  a >= b = (a > b || a == b)
  iSort :: Ord t => [t] -> [t]
```

A classe Eq é superclasse de Ord

Tipos Numéricos

Int, Integer

Float, Double

Rational

Complex

Classes: Num, Real, Fractional, Floating

```
rep 0 ch = []
```

```
rep n ch = ch : rep (n-1) ch
```

```
Main> :type rep
```

```
rep :: Num a => a -> b -> [b]
```

```
Main> rep 4 'd'      (??)
```

```
Main> rep 4 2         (??)
```

