Programmation fonctionnelle

Haskell est un langage fortement typé statiquement :

Type = Un ensemble de valeurs.

Fortement = Toujours là

Statiquement : à la compilation

Inférence de types

En majuscule on a les noms de modules et les noms de types et les constructeurs.

En terminal :

:l ou :load file.hs permet de charger un fichier haskell

:r ou reload permet de recharger le dernier fichier chargé

module Types where

// Les booléens :

a :: Bool

a = True

// Les caracteres

c :: Char

c = ‘Y’

// les Entiers

// Integer

n :: Int

n = 12

borne :: Int

borne = maxBound ou minBound

n’ :: Integer

n = 9223372036…. + 1

f :: Float

f : 1.2

d :: Double

d = 34.4

Synonyme de type String = [Char]

Exemple :

s’ :: [Char]

s’ = [‘a’,’b’,’c’]

Liste de caleur de type t [t]

Pour avoir tous les caractere unicode :

tousChars = [‘\NUL’..]

t1 -> t2 , types des fonctions qui ont un argument de types t1 et calculer une valeur de type t2

add :: (Int, Int) -> Int

add (x,y) = x + y

zeroTo :: Int -> [Int]

zeroTo n = [0..n]

Version Curryfiée de l’addition

Haskell Brooks Curry

add’ :: Int -> Int -> Int

add’ x y = x + y

Une application partielles : seulement une partie des arguments

Fonction polymorphe :

Le type avec une variable de types (minuscule)

F :: [2] -> [2]

Fonction applicable à toute liste [quel que soit le type des valeurs dans la liste ]

Polymorphe Parametré

Definitions de fonctions

x y -> x C y -> xACBy