


# Haskell


Et la programmation purement fonctionnelle



# Programmation fonctionnelle

- Immutabilité
  - Fonction pure
  - Fonction d'ordre supérieur
  - Abstraction
  - Composabilité
- 


# Famille Lisp

- Scheme
  - Racket
  - Common Lisp
  - Clojure
- 


# Famille Lisp

- Homoiconicité
- Métaprogrammation
- Macro

# Famille ML

- Standard ML
  - Ocaml
  - Haskell
  - Haskell-like (Agda, Idris, Elm, Purescript)
  - Scala (vue sous un certain angle)
- 

# Famille ML

- Système de type puissant
  - Base mathématique
  - Inférence de type
- 

# Haskell

- Est un descendant de ML
- Purement fonctionnelle
- Évaluation paresseuse

# Type produit

data Point = Point Float Float

data Personne = Personne Nom Prenom

# Type synonyme

Type Nom = String

type Prenom = String



# Type Somme

```
data Couleur = Bleu | Rouge | Vert
```

```
data JValue = JString String
```

```
    | JNumber Double
```

```
    | JBool Bool
```

```
    | JNull
```

```
    | JObject [(String, JValue)]
```

```
    | JArray [JValue]
```

# ADT: Algebraic Data Type

Combinaison de produit et de somme

```
data List a = Empty | Cons a (List a)
```

```
data Tree a = Leaf a | Tree (Tree a) (Tree a)
```

# Filtrage par motif (Pattern matching)

$$\text{fib } 0 = 0$$

$$\text{fib } 1 = 1$$

$$\text{fib } n = \text{fib } (n - 1) + \text{fib } (n - 2)$$

# Currification

`insert :: Key -> a -> IntMap a -> IntMap a`

`delete :: Key -> IntMap a -> IntMap a`

# Currification

`insert :: (Key -> (a -> (IntMap a -> IntMap a)))`

`delete :: (Key -> (IntMap a -> IntMap a))`

# Currification

- Deux petite fonction simple

$(\$)\ ::\ (a \rightarrow b) \rightarrow a \rightarrow b$

$(.)\ ::\ (b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow a \rightarrow c$

# Currfication

`insert :: Key -> a -> IntMap a -> IntMap a`

`delete :: Key -> IntMap a -> IntMap a`

`insert 7 "Piano" . delete 2 . insert 42 "Hi friend" . insert 2 "Hello" $ myIMap`

# Purement fonctionnel

- Pas d'effet de bord? (Faux)



# Purement fonctionnel

- Effet de bord isolé via les types
- Fonction au sens mathématique
- Transparence référentiel

Ex:  $\text{id} :: a \rightarrow a$

Combien d'implémentation possible?

# Purement fonctionnel

- Effet de bord isolé via les types
- Fonction au sens mathématique
- Transparence référentiel

Ex:  $\text{id} :: a \rightarrow a$

Combien d'implémentation possible?

Une seule!

$\text{id } x = x$

# Purement fonctionnel

Type IO

```
putStr :: String -> IO ()
```

```
getLine :: IO String
```

```
main :: IO ()
```

```
main = putStrLn "hello world"
```

# TypeClass

- Permet de mettre des contraintes polymorphique
- Proche des interfaces en POO (ex Java)
- On va aller voir dans la Doc quelque exemple
- Eq, Ord, Monoid, Foldable, Functor

# Paresseux

Code	Stricte	Paresseux
A:=3 + 4	A==7	A== (3 + 4)
B:=1 + 2	A==7 B==3	A== (3 + 4) B== (1 + 2)
C:=A + B	A==7 B==3 C==10	A== (3 + 4) B== (1 + 2) C== (A + B)
print(C)	10	

# Paresseux

Code	Stricte	Paresseux
A:=3 + 4	A==7	A== (3 + 4)
B:=1 + 2	A==7 B==3	A== (3 + 4) B== (1 + 2)
C:=A + B	A==7 B==3 C==10	A== (3 + 4) B== (1 + 2) C== (A + B)
print(C)	10	


- 1.C==(A + B)
- 2.C==((3 + 4) + B)
- 3.C==(7 + B)
- 4.C==(7 + (1 + 2))
- 5.C==(7 + 3)
- 6.C==10

# Paresseux: Structure infini

```
data [a] = [] | a : [a]
```

```
fibs = 0 : 1 : zipWith (+) fibs (tail fibs)
```

# Haskell: Outils

- Compilateur: GHC
  - Build system/Gestion des dépendences : cabal-install et/ou stack
  - Éditeur de code: Vim, Emacs, Atom, Sublime Text, VS Code
  - Ghc-mod (multi-editeur)
  - Intero (Emacs, multi-editeur)
- 



# Exemple de Code: Jeux d'échec

# Livre

- Haskell Programming from First Principles (60\$, ~1200 pages)
- Learn you a Haskell for great good (Gratuit, Existe en français)
- Real World Haskell (Gratuit, Plusieurs partit obsolète)

# Échelle de connaissance

<https://pbs.twimg.com/media/CydL5EYUsAAI-61.jpg:large>

# Questions?