

# # Cheat Sheet Haskell


## # Syntaxe

- Indentation comme en python (offside rule)
- Séparateur implicite  $\rightarrow$  ; (transformé auto)
- 19 mots clés réservés
- Nomage: ◦ Identique au C + apostrophe (pour pouvoir écrire des nom proche des maths:  $f'$ )
- Premières lettres Capitalisées pr livres

## # Expressions

- Expressions littérales: s'évaluent si elle n'
- Expr. combinées: ◦ Distinct de  $\lambda$  mais avec point
  - fnc  $\rightarrow$  notat° préfixe mais selon syntaxe peut avoir une notat° infix préfixe
    - $(+)$  3 4  $\rightsquigarrow$  3 + 4
    - div 3 4  $\rightsquigarrow$  3 'div' 4
  - associativité droite pr fct° + plus haute précedence
    - $\rightarrow$  f 1 + 2  $\rightsquigarrow$  (f 1) + 2
    - f (1 + 2)  $\rightsquigarrow$  (f (1 + 2))
  - permet de créer new op. ordre infixe
- Expr abstraites: abs. syntaxique

# # Typage

- typage statique: - variable types  
- verif type à la compil
- forte<sup>tr</sup> typée mais est capable de guess un type si  
pas précisé:  pas parce que type pas précisé que typage  
dynamique → guess le type à la compil

# # Nomage et assignation

- pas de variable en Haskell car langage fonctionnel pur → que des cste
- créat° de constante:

`foo :: Float` ← optionnel car peut guess le type  
`foo = 13.5`

↳ une fois assigné → on changeable

- créat° de fct:

`bez :: Float → Float → Float`

↳ aussi optionnel  
`bez a b = sqrt (3 * (a + 7) - b)`

↳ équivalent en C:

```
{  
    Float bez (Float a, Float b) {  
        return sqrt (3 * (a + 7) - b)  
    }  
}
```

# # polymorphisme

- ° pour pas avoir à réécrire la même fct° pour chaque type → polymorphisme

≠ typage dynamique

ex:  $\text{length} :: [a] \rightarrow \text{length}$   
 $\text{length} [] = 0$   
 $\text{length} (x : xs) = 1 + \text{length} xs$

→ prend une liste de type  $a$  et renvoie sa longueur

type  $a$  désigne n'importe quel type  
→ marche avec integer, string, ...

# # Expressions conditionnelles

° booléen:

- type Bool
- valeurs True et False
- opérateurs booléens &&, ||, not
- opérateurs booléens w/ "variadicité"

and et or :  $[Bool] \rightarrow Bool$

• Construct logique:

```
if cond 1 && cond 2 then ____  
else if cond 3  
  then ____ else ____
```

• pattern matching  $\rightarrow$  trois dev.

avec guards:

```
func :: Int  $\rightarrow$  Int  
func a  
  | a == 1 = 3  
  | a == 2 = 4  
  | otherwise = 5
```

sans guards

```
func 1 = 3  
func 2 = 4  
func a = 5
```

équivalent

$\Rightarrow$  possibilité de pattern matching des liste en le décomposant

```
func [] = 0
```

```
func (x:xs) = x
```

$\rightarrow$  renvoie 0 si liste vide, le tête  
sinon

# # Arithmétique

- opérateurs "de base" infixes:  $1+2$   
grâce sucre syntaxique
- Surcharge des littéraux:  $2 \rightarrow \text{int}$  et un  $\text{float}$
- Surcharge des opérateurs: redéfini par chaque type
- $\neq$  de transypage auto:  $\text{Int} \neq \text{float}$

# # Liste

- Syntaxe:  $[1, 2, 3, \dots]$  ou  $[]$  « liste vide »  
↳ pas avec parenthèse !
- Homogène:  $[1, \text{"string"}] \rightarrow$  interdit
- Consneur:  $(::) :: a \rightarrow [a] \rightarrow [a]$
- constructeur:  $\text{replicate} :: \text{Int} \rightarrow a \rightarrow [a]$
- énumérateur:  $[1..3] = [1, 2, 3]$   
 $[1, 4..10] = [1, 4, 7, 10]$   
↳ sucre syntaxique
- Accesseur:  $\text{Head}, \text{tail}$ 
  - pattern matching  $(x:xs)$
  - $(!!) :: [a] \rightarrow \text{Int} \rightarrow a$   
↳ renvoie l'elt à l'indice  $\text{Int}$

• generer une liste selon des criteres :

$$[f(n) \mid n \in \text{list}, \text{cond} 1, \dots]$$

ex:  $[n+2 \mid n \leftarrow [2, 3, 4], n \neq 3]$   
 $= [6]$  parce q ensemble  $a' \in$

$\Delta \neq : [n \neq 3 \mid n \leftarrow [4, 6, 8]]$   
 $= [\text{false}, \text{false}, \text{false}]$

→ on peut maitenir matcher dedans

$$[m+n \mid [m, n] \leftarrow [[2, 3], [4, 5], [5, 6], m > 2]]$$
$$= [9, 11]$$

→ peut être utilisé comme filtre avec  
cond → filter

→ peut permettre des heuristiques pour  
reconstruire l'écriture d'une fct.

⚠ Var sont locales au generateur  
func a = [a | a ← [1, 2]]  
↑                    ↑  
les 2 a sont ≠

# # Chaîne de caractères

- Caractère: 'c'
  - ↳ type char
  - ↳ int char: ord et chr
- String: "string"
  - ↳ type [char] ⇒ liste de char (le type string existe aussi mais juste à l'écrit)
- liste ⇒ on peut faire des trucs marrant comme
  - $[c / c \leftarrow \text{"abcde", is\_vowel } c] = \text{"ae"}$

# # Compositor et fonction anonyme

- $(f \circ g) x \rightarrow f(g(x))$  mathématique
  - ↳ ce ressemble au  $\circ$  mathématique ( $f \circ g(x)$ )
- ⚠ précedence:
  - $f \circ g x = f \circ (g x)$
- $\text{func} :: \text{Int} \rightarrow (\text{Int} \rightarrow \text{Int})$ 
  - $\text{func } a = \lambda x \rightarrow x + a$
  - ↳ renvoie fonction anonyme qui additionne  $a$  à  $x$ ,  $\lambda \rightarrow$  fonction anonyme ce ressemble