1 FONCTIONS ET VARIABLES

Haskell n'a pas de notion de variable au sens qu'a ce terme en programmation procédurale. Il est possible d'assigner une expression ou une valeur à un nom, avec la syntaxe `nom = expression`, mais `nom` est immuable, et est donc plus proche d'une constante (c'est une variable au sens mathématique du terme).

En combinant ceci avec les principe de transparence référentielle ⁇, d'évaluation paresseuse ⁇ et d'application partielle ❷, on voit facilement qu'il n'existe aucune différence stricte entre une fonction et une variable, donc qu'il n'existe pas de variables. Par exemple :

```
a = 3 * 2
times3 x = 3 * x
b = times3 2
c = 6
```

Ici, `times3` est une fonction, `a`, `b` et `c` des variables. Dans la mesure où la valeur d'aucune n'est évaluée tant qu'elle n'est pas utilisée, la variable `a` a strictement la même valeur que `b`, qui n'est pas 6, mais le calcul différé (le *thunk*) `3 * 2`.

> `times3 x = 3 * x` peut s'écrire plus simplement `times3 = (*) 3` ❷.

Signature de type

la signature a la forme `f :: a -> b`, ce qui signifie que la fonction prend un paramètre de type `a` et renvoie une valeur de type `b`. (Le type d'une « variable » est simplement `nom :: Type`)

Les fonctions d'ordre supérieur utilisent les parenthèses pour indiquer qu'elles prennent une autre fonction en paramètre. Par exemple, le type `map :: (a -> b) -> [a] -> [b]` se lit : `map` prend comme premier paramètre une fonction quelconque `x :: a -> b`.

```
f :: [Int] -> Integer
```

f prend un tableau d'entiers et renvoie un entier.

```
g :: Int -> Integer -> Integer
```

g prend deux entiers et renvoie un entier (plus précisément, g prend un entier et renvoie une fonction qui prend un entier et renvoie un entier ❷).

```
h :: [a] -> Int
```

h prend un tableau d'un type quelconque a ⁇ et renvoie un entier

```
i  :: [a] -> a
```

i prend un tableau d'un type quelconque a et renvoie un a

```
j  :: (Eq a) => [a] -> a
```

j prend un tableau d'un type quelconque a instance de la classe de type ?? `Eq` et renvoie une valeur de type `a`

Portée des variables et variables locales

La

Fonctions préfixes et infixes

Fonctions pures et impures

2 APPLICATION PARTIELLE ET *CURRYING*

Une fonction, quel que soit le nombre de paramètres avec lequel elle a été déclarée, ne prend qu'un seul paramètre et renvoie une autre fonction. Le type de +, par exemple, est : `Num a => Num a -> Num a -> Num a`, ce qui signifie que + prend un premier paramètre d'un type de type `Num`