TD 3 : Types

1 Types → Expressions

Pour chacun des types suivants, donnez une expression de Haskell qui lui correspond.

1. `[ (Int, Float) ]`
2. `[Int -> Int]`
3. `a -> (a,a)`
4. `(a,a) -> a`
5. `(Int->Int)->(Int->Int)->Int->Int`
6. `a->[a]->[a]`
7. `a->[b]`
8. `a -> b`

NB : Vous allez constater que vérifier vos solutions n'est pas forcément simple. Par exemple, si vous avez pensé à l'expression `[ (2,3.4) ]` pour la première question, vous pouvez essayer de faire `:t [(1,3.4)]` dans ghci. Or, cela retourne, `[(1,3.4)] :: (Num t1, Fractional t) => [(t1, t)]`, car ghci donne le type **le plus générique** qui correspond à l'expression. Cela n'empêche que l'expression corresponde aussi à un type plus spécifique. Donc, pour vérifier avec ghci que votre expression correspond bien au type demandé vous pouvez préciser le type explicitement avec l'opérateur `::`. Par exemple, on écrit

```
*Main>  [(1,3.4)] :: [(Int,Float)]
[(1,3.4)]
*Main>  [(1,3.4)] :: [(Int,Integer)]
<interactive>:48:6: error:...
```

2 Expressions → Types

Pour les expressions suivantes (`y1,..,y7`), quel est le type inféré par ghci (si les expressions sont bien typées) ?

1. `y1 = 5.`
2. `y2 x = x+5`
3. `y3 f g = f (head g) : tail g`
4. `y4 x = [x]`
5. `y5 f 0 x = x`
   `y5 f n x = f (y5 f (n-1) x)`
6. `y6 f 0 = []`
   `y6 f 1 = [f]`
   `y6 f n = (f):(y6 f (n-1))`
7. `y7 x y z = if x then y else z`
8. `y6 y2 3`
9. `y3 (y6 y2 3)`
10. `y5 y4 2 "a"`

3 Répétitions

Écrire une fonction qui, étant donné une fonction f et deux entiers n, x retourne le résultat d'appliquer la fonction f sur x, n fois. (Donc, $f(f(f(f \ldots f(x) \ldots)$.

Utiliser cette fonction pour donner une fonction qui implémente la multiplication (en utilisant l'opération `(+)` en répétition) et une fonction qui calcule la puissance x^y, pour x, y entier, en utilisant votre implémentation de la multiplication.

4 Count Frequency

Écrire une fonction qui étant donné une liste `xs` et un élément `x` retourne le nombre de fois que `x` paraît dans `xs`. **NB** votre fonction doit être polymorphique !

Ex :

```
*Main> countFreq 'a' "Barbara"
3
*Main> countFreq 2 [1,2,3,1,2,3,1,2,4,2,3]
4
```