Fiche de TD-TP no. 1

En TD

Exercice 1. Une liste s'écrit entre crochets, avec les éléments de la liste séparés par des virgules.

1. Évaluez les expressions des listes suivantes :
   (a) `1:[2]`,
   (b) `[3,4]++[1,2]`,
   (c) `[3..10]`,
   (d) `tail [1..4] ++ 5:[]` .
   (e) `head [1..4] : [5]`.
   (f) `reverse [1..4] ++ [5]` .
2. Expliquez ce que font les les opérateurs `[],:,++,..` et les fonctions `head`, `tail`, `reverse`.

Exercice 2. Rappelez ce que font les fonctions `head`, `tail`, `reverse`, `length`, `drop`, `take`, `!!`, définie dans `prelude.hs` et vue en cours.

1. La fonction `last`, définie dans `prelude.hs`, sélectionne le dernier élément d'une liste. Montrez comment cette fonction peut se définir en utilisant les fonctions sur le listes présentées dans le cours.
2. Pouvez vous penser à d'autres définitions de `last` ?
3. De façon semblable, montrez comment la fonction `init`—du `prelude.hs`, qui enlève le dernier élément d'une liste—-peut se définir de plusieurs façons.

Exercice 3. Une chaîne de caractères est une liste de caractères. On souhaite définir une fonction qui répond à la question si une chaîne de caractères est un palindrome.

1. Quel est le type de cette fonction : explicitez son domaine et son codomaine.
2. Proposez une définition pour cette fonction.

Exercice 4.

1. Réécrivez, en notation mathématique, les expression suivantes de Haskell :
   (a) `f 3 + g 9 *8`,
   (b) `f 22 33 * (g 44 h 55)`,
   (c) `f 22 33 * g 44 (h 55)`.
   Étés vous capables de deviner le type de f, g, et h, dans les trois expressions Haskell ci-dessus ?
2. Réécrivez, dans le langage Haskell, les expressions mathématique suivantes :
   (a) $f(g(y,z),33)$,
   (b) $8g(44,23)$,
   (c) $f(8\pi(33+21))$.

Exercice 5. Expliquez pourquoi, dans l'exercice 1, l'expression `reverse [1..4] ++ [5]` s'évalue à `[5,4,3,2,1]`.

Exercice 6. Quel est le type des valeurs suivants ?

```
['a','b','c']
('a','b','c')
[(False,'0'),(True,'1')]
([False,True],['0','1'])
[tail,init,reverse]
```

Exercice 7. Quel est le type des fonctions suivantes :

```
second xs = head (tail xs)
swap (x,y) = (y,x)
pair x y = (x,y)
mult x y = x * y
double = mult 2
palindrome xs = reverse xs == xs
twice f x = f (f x)
```

Pour chaque fonction qui contient (au moins) deux variables dans ses arguments, dites s'il s'agit d'une fonction currifiée.

Exercice 8. (Tuplets, éventuellement en TP) On peut représenter un nombre complexe comme un couple de nombres réels, la partie réelle et la partie imaginaire. Écrivez un script où les opérations élémentaires sur les nombres complexes : `somme`, `produit`, `division`, `module`, `angle`.
Dans ce script, la déclaration du type de la fonction apparaîtra avant sa définition.

Exercice 9. (Éventuellement en TP) Une image est une suite de lignes, chaque ligne étant une suite de `0` et `1`. Ainsi, nous allons représenter une image comme un valeur de type `[[Int]]`. Expliquez ceux qui sont les transformations sur les images définies dans le script Haskell suivant :

```
inv :: Int -> Int
inv 0 = 1
inv x = 0

inverser :: [[Int]] -> [[Int]]
inverser image = map (map inv) image

refleter :: [[Int]] -> [[Int]]
refleter image = reverse image

eln :: Int -> [a] -> a
eln n xs = xs !! n

colonne :: [[Int]] -> Int -> [Int]
colonne image n = map (eln n) image

transposer :: [[Int]] -> [[Int]]
transposer image = map (colonne image) [0..(length (head image) -1)]
```

1. Lisez le code et remarquez toutes les constructions que vous ne connaissez pas. (La fonction `map` applique f à toues les elements d'une liste, i.e.

   $$\texttt{map f } [\texttt{x}_1, \ldots, \texttt{x}_n] = [\texttt{f x}_1, \ldots, \texttt{f x}_n].$$

   Par exemple, `map s [1,2,3] = [2,3,4]`, ou `s x = 1 + x`.
2. Expliquez ce qui est achevé par ces définitions, en terme de transformations d'images.
3. Comment définir une fonctions qui fait une rotation à gauche de l'image ? Et une rotation à droite ?

En TP

Exercice 10. Parcourez les transparents du cours, avec l'interprète (hugs ou ghci) à la main.

Exercice 11. Le script ci-dessous contient deux erreurs de syntaxe :

```
N = a 'div' length xs
    where
      a = 10
     xs = [1,2,3,4,5]
```

En rappelant ce qui a été dit en cours,

1. trouvez ces erreurs,
2. rappelez quelle est la règle générale qui n'est pas respectée,
3. corrigez les.

Si en TP, aidez vous avec l'interprète Haskell.

Exercice 12. Completez le script de l'exercice 8 et testez-le avec l'interprète.

Exercice 13. Completez le script de l'exercice 9 et testez-le avec l'interprète.

Exercice 14. Considerez les script `pair-impair.hs` suivant :

```
pair 0 = True
pair n = impair (n -1)
impair 0 = False
impair n = pair (n-1)
```

Ouvrez ce script avec `ghci`, et tapez les commandes suivantes :

```
> :break pair
> :break impair
> :set stop :list
> :step pair 10
...
> :step
```

Que ce passe t'il ?