

TP 6 - Fonctions d'ordre supérieur -

Les fonctions d'ordre supérieur sont des fonctions qui prennent une ou plusieurs fonctions en entrée (comme les lambdas vues en Java) et/ou renvoient une fonction. Le but du TP est d'utiliser et d'écrire de telles fonctions.

Exercice 1. Premières fonctionnelles et curryfication

Dans cet exercice, on utilisera les liaisons suivantes :

```
# let sqrt x = x * x;;  
# let my_list = [3 ; 12 ; 3 ; 40 ; 6 ; 4 ; 6 ; 0];;
```

1. Quel est le type de la fonction `f_sum` qui prend en paramètre une fonction f et un couple d'entiers a et b et calcule $f(a) + f(b)$?
2. Écrire la fonction `f_sum`.

```
# f_sum sqrt (2, 3);;  
- : int = 13  
f_sum sqrt 3;;  
Error: This expression has type int but an expression was expected of type int * int
```

3. Curryfier (explicitement) la fonction `f_sum`.

```
# f_sum sqrt 3;;  
- : int -> int = <fun>
```

4. Donner des expressions ayant pour type :

```
val f : int -> int -> int  
val g : (int -> int) -> int = <fun>  
val h : (int -> int) -> int -> int = <fun>  
val i : (int -> int) -> (int -> int) = <fun>  
val j : ((int -> int) -> int) -> int = <fun>
```

5. Sans définir de nouvelle fonction, créer la liste des carrés des éléments de `my_list`.

```
- : int list = [6400; 1444; 4; 1444; 100; 2916; 49; 2704; 8100; 625]
```

6. Sans définir de nouvelle fonction, créer la liste des doubles des éléments de `my_list`.

```
- : int list = [6; 24; 6; 80; 12; 8; 12; 0]
```

7. Quel est le type de la fonction `make_list` qui prend en paramètre une taille n et une fonction `make` (sans argument) et qui renvoie une liste de n éléments fabriqués avec la fonction `make` ?

8. Écrire la fonction `make_list`.

```
# let f () = 0;;  
val f : unit -> int = <fun>  
# make_list f 10;;  
- : int list = [0; 0; 0; 0; 0; 0; 0; 0; 0; 0]
```

9. En utilisant `make_list`, créer une liste de 50 booléens aléatoires.

10. En utilisant `make_list` et une fonction anonyme, créer une liste de 20 entiers aléatoires.

Exercice 2. One liners

En une ligne (en utilisant les fonctions d'ordre supérieur sur les listes), écrire les fonctions suivante :

1. (`sum l`) qui renvoie la somme des éléments d'une liste.
2. (`size l`) qui renvoie la taille d'une liste.
3. (`last l`) qui renvoie le dernier élément d'une liste non vide.
4. (`nb_occ e l`) qui compte le nombre d'occurrences de e dans ℓ .
5. (`max_list l`) qui renvoie le maximum d'une liste non vide.
6. (`average l`) qui renvoie la moyenne (en flottant) des nombres entiers de ℓ .

Pour les exemples, vous pouvez vous reporter au sujet du TP 3.

Exercice 3. Prédicats sur les listes

Un prédicat est une fonction qui prend une expression en paramètre et renvoie un booléen. Par exemple :

```
# let is_even = fun x -> x mod 2 == 0;;
val is_even : int -> bool = <fun>
# is_even 0, is_even 1, is_even 2;;
- : bool * bool * bool = (true, false, true)
```

1. Écrire la fonction récursive `my_for_all` qui prend en paramètre un prédicat p et une liste et renvoie `true` si tous les éléments de la liste respectent le prédicat.
2. Écrire la fonction `my_for_all2` qui fait la même chose en utilisant `fold_left`.
3. Si ce n'est pas déjà fait, ré-écrire la fonction `my_for_all2` sans utiliser `map`.
4. Écrire la fonction `my_for_all3` qui fait la même chose en utilisant `fold_right`.
5. Écrire la fonction `my_exists` qui prend en paramètre un prédicat p et une liste et renvoie `true` s'il existe un élément de la liste qui respecte le prédicat.
6. Écrire la fonction récursive `none` qui prend en paramètre un prédicat p et une liste et renvoie `true` si aucun des éléments de liste ne respecte le prédicat.
7. Écrire la fonction `do_not_exists` qui prend en paramètre un prédicat p et une liste et renvoie `true` s'il n'existe pas d'élément dans la liste qui respecte le prédicat.
8. Écrire la fonction `ordered` qui prend en paramètre un prédicat binaire p et une liste $[a_1; a_2; a_3; \dots; a_n]$ et renvoie `true` si $(p\ a_1\ a_2) = (p\ a_2\ a_3) = \dots = (p\ a_{n-1}\ a_n) = true$.

```
# ordered (<) [1;2;3];;
- : bool = true
# ordered (<) [1;4;3];;
- : bool = false
```

9. Écrire la fonction `filter2` qui prend en paramètre un prédicat binaire p et deux listes $[a_1; a_2; a_3; \dots; a_n]$ et $[b_1; b_2; b_3; \dots; b_n]$ et renvoie la liste des couples (a_i, b_i) tels que $(p\ a_i\ b_i)$ est vrai.

```
# filter2 (<) [2;2;3] [1;4;5];;
- : (int * int) list = [(2, 4); (3, 5)]
```

Exercice 4. Génération exhaustive de permutations ★

Écrire une fonction `perm l` qui fabrique la liste de toutes les permutations de l .

```
# perm [1;2];;
- : int list list = [[1; 2]; [2; 1]]
# perm [1;2;3];;
- : int list list =
[[1; 2; 3]; [2; 1; 3]; [2; 3; 1]; [1; 3; 2]; [3; 1; 2]; [3; 2; 1]]
# perm [1;2;3;4];;
- : int list list =
[[1; 2; 3; 4]; [2; 1; 3; 4]; [2; 3; 1; 4]; [2; 3; 4; 1]; [1; 3; 2; 4];
 [3; 1; 2; 4]; [3; 2; 1; 4]; [3; 2; 4; 1]; [1; 3; 4; 2]; [3; 1; 4; 2];
 [3; 4; 1; 2]; [3; 4; 2; 1]; [1; 2; 4; 3]; [2; 1; 4; 3]; [2; 4; 1; 3];
 [2; 4; 3; 1]; [1; 4; 2; 3]; [4; 1; 2; 3]; [4; 2; 1; 3]; [4; 2; 3; 1];
 [1; 4; 3; 2]; [4; 1; 3; 2]; [4; 3; 1; 2]; [4; 3; 2; 1]]
```

Exercice 5. Et sur les arbres ?

Dans cet exercice, nous réutiliserons les définitions du TP 4 :

```
type bintree = Empty | Node of int * bintree * bintree;;

let example_tree =
  Node(1, Node(2, Node(4, Empty, Empty),
    Node(5, Node(7, Empty, Empty), Node(8, Empty, Empty))),
    Node(3, Empty, Node(6, Node(9, Empty, Empty), Empty)));;
```

1. Écrire la fonction `map_tree` équivalente à `map`, mais pour des arbres binaires.

```
# tree_map sqrt example_tree;;
- : bintree =
Node (1,
  Node (4, Node (16, Empty, Empty),
    Node (25, Node (49, Empty, Empty), Node (64, Empty, Empty))),
```

2. Écrire la fonction `fold_tree` équivalente à `fold_right` pour des arbres binaires. Sur l'arbre d'exemple, cela donne :

```
(f 1 (f 2 (f 4 x x) (f 5 (f 7 x x) (f 8 x x))) (f 3 x (f 6 (f 9 x x) x)))
```

3. Utiliser `fold_tree` pour écrire les fonctions suivantes :

- (a) `tree_size` qui calcule la taille d'un arbre;
- (b) `tree_leaves` qui calcule la liste des feuilles d'un arbre;
- (c) `tree_nodes` qui calcule la liste des nœuds d'un arbre. Dans quel ordre sont les nœuds ? Comment obtenir les 2 autres ordres ?

```
# tree_size example_tree ;;
- : int = 9
# tree_leaves example_tree ;;
- : int list = [4; 7; 8; 9]
# tree_nodes example_tree ;;
- : int list = [1; 2; 4; 5; 7; 8; 3; 6; 9]
```