Correction TD de programmation fonctionnelle n°5

Licence d'informatique

Les listes en Caml

Nous allons voir dans ce TD comment manipuler très simplement des listes en Caml grâce à la définition récursive suivante : une liste est soit vide (`[]`) soit un élément suivi d'une liste (`element::liste`).

▸ Exercice 1. Opérations de base

1. Écrire une fonction qui teste si une liste est vide.
2. Écrire une fonction qui renvoie le nombre d'éléments d'une liste.
3. Écrire une fonction insérant un entier à la fin d'une liste.
4. Écrire une fonction qui teste si un entier appartient à une liste.
5. Écrire une fonction prenant 2 listes et qui renvoie leur concaténation.

✂ -

```
# let est_vide = function
    [] -> true
  | _ -> false;;
val est_vide : 'a list -> bool = <fun>
# (* ou tout simplement *)
# let est_vide2 = (=) [];;
# liste_vide [];;
- : bool = true
# liste_vide [1;3;2;2;2];;
- : bool = false

# let rec nb_elements = function
    [] -> 0
  | _::t -> 1+nb_elements(t);;
val nb_elements : 'a list -> int = <fun>
# (* en version récursive terminale *)
# let nb_elements2 =
    let rec aux n = function
        [] -> 0
      | _::t -> aux (n+1) t in
    aux 0;;

# let rec insere_fin l a = match l with
    [] -> [a]
  | h::t -> h::(insere_fin t a);;
```

```
val insere_fin : 'a list -> 'a -> 'a list = <fun>
# (* en version récursive terminale *)
# let rev =
    let rec aux m = function
        [] -> m
      | h::t -> aux (h::m) t in
    aux [];;
# let insere_fin2 l a = rev (a::(rev l));;


# let rec appartient l a = match l with
    [] -> false
  | h::t -> a=h || (appartient t a);;
val appartient : 'a list -> 'a -> bool = <fun>
# (* à noter qu'on ne fait pas l'appel récursif si a=h *)


# let rec concat l m = match l with
    [] -> m
  | h::t -> h::(concat t m);;
val concat : 'a list -> 'a list -> 'a list = <fun>
# (* en version récursive terminale *)
# let rec revcat l m = match l with
    [] -> m
  | h::t -> revcat t (h::m);;
# let concat2 l = revcat (rev l);;

# (* Il y a un opérateur direct: @ *)
# [1;2;3]@[4;5;6];;
- : int list = [1; 2; 3; 4; 5; 6]
```

▸ Exercice 2. Opérations sur les éléments

1. Écrire des fonctions qui calculent la somme et le produit d'une liste d'entiers.
2. Écrire une fonction qui calcule le minimum d'une liste d'entiers positifs.
3. Écrire une fonction qui retourne la concaténation de toutes les chaînes de caractères d'une liste.
4. Quelles ressemblance voit-on apparaître dans la structure de ces fonctions ? Nous allons voir comment écrire une fonction qui fait le travail en une fois.

```
# let rec somme l = match l with
    [] -> 0
  | h::t -> h+(somme t)
val sum : int list -> int = <fun>
```

```
# let rec produit l = match l with
    [] -> 1
  | h::t -> h*(produit t)
val prod : int list -> int = <fun>
```

```
# let rec minimum l = match l with
    [] -> failwith "Liste vide!"
  | [h] -> h
  | h::t -> let tmp = (min2 t) in
  (* variable locale pour éviter les deux appels récursifs *)
      if h<tmp then h else tmp;;
val min : 'a list -> 'a = <fun>
```

```
# let rec strcat l = match l with
    [] -> ""
  | h::t -> h^(strcat t);;
val strcat : string list -> string = <fun>
```

\- ✂

Toutes ces fonctions retournent le résultat de l'application d'une fonction à une liste.

▸ Exercice 3. Fonctionnelles sur les listes

1. Écrire une fonction `do_list` qui appelle une fonction `f : ('a -> unit)` pour chacun des éléments d'une liste.
2. Écrire la fonction `map` qui applique une fonction `f` à tous les éléments `x` d'une liste, et renvoie la liste de tous les résultats `f(x)`.
3. Écrire une fonction `iterateur` qui permette d'unifier les écritures des fonctions de l'exercice précédent. Cette fonction prendra en paramètres une liste `l` et une fonction `f : 'a -> 'a -> 'a`, et retournera un résultat dont de type `'a`.
   Récrire les fonctions de l'exercice 2 en utilisant `iterateur`.

✂ -

```
# let rec do_list f = function
   [] -> ()
 | h::t -> f(h); (do_list f t);;
val do_list : ('a -> 'b) -> 'a list -> 'b = <fun>
```

```
# let rec map f = function
   [] -> []
 | h::t -> f(h)::(map f t);;
val map : ('a -> 'b) -> 'a list -> 'b list = <fun>
```

```
# let rec iterateur f = function
   [] -> failwith "Liste vide!"
 | [h] -> h
 | h::t -> (f h (iterateur f t));;
val iterateur : ('a -> 'a -> 'a) -> 'a list -> 'a = <fun>

# let rec iterateur2 f i = function
   [] -> i
 | h::t -> (f h (iterateur2 f i t));;
val iterateur2 : ('a -> 'b -> 'b) -> 'b -> 'a list -> 'b = <fun>
# let somme2 = iterateur2 (+) 0;;
val somme2 : int list -> int = <fun>

# let produit2 = iterateur2 ( * ) 1;;
val produit2 : int list -> int = <fun>

# let minimum2 = iterateur (fun a b -> if a<b then a else b);;
val minimum2 : '_a list -> '_a = <fun>

# let strcat2 = iterateur2 (^) "";;
val strcat2 : string list -> string = <fun>
```