TP n° 4 : Listes

Dans ce TP, nous vous proposons de travailler avec la structure naturellement récursive de liste chaînée en CAML.

On rappelle qu'une liste, de type `'a list` :

- peut être vide, et est alors notée `[]` ;
- a tous ses éléments de même type : `'a` ;
- possède, lorsqu'elle n'est pas vide une tête (premier élément) et une queue (liste des autres éléments), auxquelles on accède en temps constant par filtrage :

OCAML
```
match liste with
  | [] ->
  | tete :: queue ->
```

- `[1; 2; 3; 4]` est une écriture raccourcie pour `1 :: 2 :: 3 :: 4 :: []`.

1 Fonctions de base sur les listes

EXERCICE 1 *Longueur d'une liste*

Écrire une fonction `list_length : 'a list -> int` qui renvoie la longueur d'une liste.

EXERCICE 2 *Concaténation*

Écrire une fonction `concat : 'a list -> 'a list -> 'a list` qui renvoie la liste concaténée à partir de deux listes.

Par exemple, `concat [1; 2] [3; 4; 5]` doit renvoyer `[1; 2; 3; 4; 5]`.

2 D'autres fonctions

EXERCICE 3 *Maximum*

Écrire une fonction `maxi : 'a list -> 'a` qui renvoie l'élément maximal d'une liste (d'éléments comparables).

EXERCICE 4 *n^e élément*

Écrire une fonction `element_numero : int -> 'a list -> 'a` qui, à partir d'un entier n et d'une liste, renvoie le n^e élément de la liste.

EXERCICE 5 *Liste arithmétique*

Écrire une fonction `liste_arithm : int -> int -> int -> int list` qui prend en entrée des entiers n, a et b et qui renvoie une liste de longueur n dont les éléments sont les n premiers termes de la suite arithmétique de premier terme a et de raison b.

Exercice 6 *Fonction mystère*

Quel est le type de la fonction suivante, et que fait-elle ?

OCaml
```
let rec mystere f liste =
  match liste with
  | [] -> []
  | tete :: queue -> (f tete) :: mystere f queue
;;
```

Exercice 7 *Découplage*

Écrire une fonction `decouple : ('a * 'b) list -> 'a list * 'b list` qui à partir d'une liste de couple renvoie un couple de listes

Exercice 8 *for_all*

Écrire une fonction `for_all : ('a -> bool) -> 'a list -> bool` prenant en argument un prédicat qui a un élément de type `'a` associe un booléen et une liste, et renvoyant `true` si le prédicat est vérifié pour tous les éléments de la liste, faux sinon.

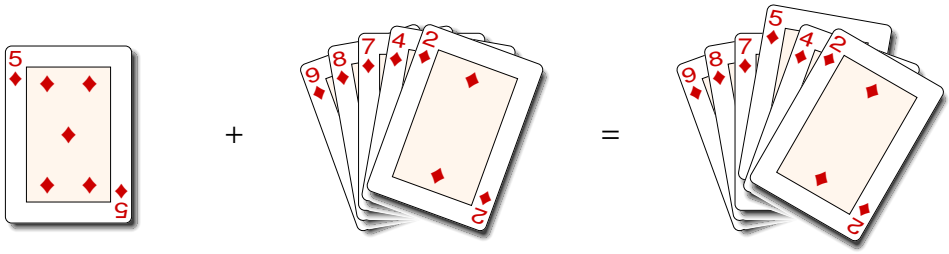
Par exemple,

- `for_all (function x -> x > 2) [1; 2; 3]` renvoie `false`
- `for_all (function x -> x > 2) [4; 7; 9]` renvoie `true`.

3 Un tri

Exercice 9

On se propose de programmer le tri insertion (*insertion sort*) sur les listes. Le principe de ce tri est celui que vous utilisez pour trier un jeu de cartes en main : au fur et à mesure du tri, vous insérez un nouvel élément parmi ceux déjà triés.



1. Écrire une fonction `insere : 'a -> 'a list -> 'a list` prenant en argument un objet et une liste triée par ordre croissant et renvoyant la liste obtenue en insérant l'objet dans la liste en conservant son caractère trié.

   Par exemple, `insere 5 [2; 4; 7; 8; 9]` renvoie `[2; 4; 5; 7; 8; 9]`.

2. Écrire une fonction `tri_insertion : 'a list -> 'a list` triant la liste : lorsque c'est encore possible, il suffit d'insérer la tête dans la queue qui aura été triée récursivement.