TP2 *Filtrage et Listes*

Exercice 1 : Fonctions de base sur les listes

Dans cet exercice vous devez réécrire des fonctions de base sur les listes, *sans utiliser celles déjà existantes*.

1. **longueur**, qui calcule la longueur d'une liste.
2. **concat**, qui concatène deux listes en une seule.
3. **nieme**, qui renvoie le n^{ième} élément d'une liste.

Exercice 2 : Fonctions intermédiaires

Définir les fonctions suivantes :

1. **npremiers**, qui extrait les n premiers éléments d'une liste (et les renvoie dans une liste).
2. **met_a_plat**, qui transforme une liste de listes en une liste simple.
3. **paire_vers_liste**, qui transforme un couple de listes en une liste de couples.

Ex : paire_vers_liste ([1 ; 2 ; 3] , ['a' ; 'b' ; 'c']) = [(1,'a') ; (2,'b') ; (3,'c')]

4. **liste_vers_paire**, qui transforme une liste de couples en un couple de listes.
5. **supprime1**, qui supprime une occurrence d'un élément ***x*** d'une liste ***li.***
6. **supprime2**, qui supprime toutes les occurrences de l'élément ***x*** de la liste ***li.***
7. **min_liste**, qui retourne le plus petit élément d'une liste.
8. **doublon**, qui conserve une seule occurrence de chaque élément d'une liste.

Exercice 3 : Fonctions avancées

Écrire les fonctions suivantes :

1. **parties**, qui retourne une liste correspondant à l'ensemble des parties d'une liste.

Ex : parties [1; 2; 3; 4];;
– : int list list = [[1; 2; 3; 4]; [1; 2; 3]; [1; 2; 4]; [1; 2]; [1; 3; 4]; [1; 3]; [1; 4]; [1]; [2; 3; 4]; [2; 3]; [2; 4]; [2]; [3; 4]; [3]; [4]; []]

Indication : si les parties de [2;3] sont [[]; [2]; [3]; [2;3]], alors les parties de [1;2;3] vont être les parties de [2;3] ainsi que ces mêmes listes auxquelles on ajoute 1 en tête : [[1]; [1;2]; [1;3]; [1;2;3]].
Conseil : écrire une fonction (inserer_tete x ll) qui ajoute l'élément x en tête de chacune des listes de ll (qui est une liste de listes).

2. **sous_listes**, qui retourne toutes les listes de *n* éléments extraites d'une liste donnée, au sens des ensembles, c'est-à-dire que deux listes contenant les mêmes éléments placés dans un ordre différent sont identiques.

Ex : sous_listes 2 [1; 2; 3; 4];;
- : int list list = `[[1; 2]; [1; 3]; [1; 4]; [2; 3]; [2; 4]; [3; 4]]`

Exercice 4 : Appliquer : la fonction map

La fonction List.map permet d'appliquer une fonction sur tous les éléments d'une liste :

Exemple, List.map `(function x->x+1) [1; 2; 3]` donne `[2; 3; 4]`

a. Réécrire en utilisant la fonction map, la fonction (**inserer_tete** `x ll`) de l'exercice 3.
b. Réécrire la fonction **parties** de l'exercice 3 sans utiliser inserer_tete (mais en utilisant map).

Exercice 5 : Appliquer itérativement : les fonctions fold_left et fold_right

Les fonctions `List.fold_left` et `List.fold_right` consistent à appliquer successivement une fonction f à un élément a et au premier (resp. dernier) élément d'une liste, puis au résultat obtenu et au deuxième (resp. avant-dernier) élément de la liste, etc. La fonction `List.fold_right` prend les éléments de la liste du dernier au premier, et `List.fold_left` les prend du premier au dernier.

```
       List.fold_left f a [b1; b2;...;bn] = (f...(f(f a b1) b2)... bn),
 et    List.fold_right f [a1; a2;...; an] b = (f a1 (f a2 ...(f an b))...)
```

Redéfinir les fonctions suivantes à l'aide de `List.fold_left` et/ou `List.fold_right` : **longueur**, **concat**, **met_a_plat**, **supprime2**, **doublon**, **map**.