TP : Listes

En Caml, les types ont souvent des définitions récursives. Par exemple, le type des entiers `int` est défini comme soit 0, soit quelque chose de la forme Sn, où n est de type `int`. Une manière de l'écrire serait la suivante :

```
type int =
  |0
  |S of int
```

Les listes sont définies de la même manière : une liste est soit la liste vide, soit un élément suivi d'une liste.

On dit que le type a deux *constructeurs* : la liste vide, et la fonction qui a un élément et une liste associe une liste. La liste vide sera notée `[]`, et la liste formée à partir de l'élément `a` et de la liste `l` sera notée `a::l`.

Pour définir la liste des entiers de 1 à 5, on pourra donc écrire `1::2::3::4::5::[]`. Plus simplement, on pourra écrire `[1,2,3,4,5]`.

Pour manipuler les listes, on a aussi besoin des *destructeurs* : on a des fonctions `List.hd` et `List.tl` qui vérifient pour tout élément `a` et toute liste `l` :

`List.hd(a::l) = a` et `List.tl(a::l) = l`.

Le meilleur moyen d'utiliser les listes sera d'utiliser le pattern-matching : une liste est soit `[]`, soit `a::l`.

Exercice 1. Proposer une définition des fonctions `List.hd` et `List.tl`.

On pourra utiliser la fonction `failwith "..."` *qui permet de renvoyer un message d'erreur.*

Exercice 2. Proposer des définitions des fonctions calculant

- la longueur d'une liste
- le dernier élément d'une liste
- le nombre d'occurences d'un élément dans une liste
- la concaténation de deux listes

Exercice 3. Définir une fonction `applique : ('a -> 'b) -> 'a list -> 'b list` qui applique une fonction à chaque élément d'une liste.

Exercice 4. Définir une fonction calculant le miroir d'une liste. En proposer aussi une version récursive terminale.

Exercice 5. Écrire une fonction qui supprime de k-ième élément d'une liste.

Exercice 6. Écrire une fonction `partition : ('a -> bool) -> 'a list -> ('a list *'a list)` qui partitionne une liste suivant un prédicat.