# Exercice 1 : Suppression du $n$-ième élément d'une liste

**Q. 1** Définir une fonction prenant en argument un entier $n$ et une liste $l$ et calculant la liste $l$ privée de son $n$-ième élément.

# Exercice 2 : Mélange

Dans cet exercice, on s'autorise à utiliser la fonction `List.rev : 'a list -> 'a list` calculant le miroir d'une liste.

**Q. 1** Définir une fonction `melange: 'a list -> 'a list` prenant en paramètre une liste `[`$x_0$`; `$x_1$`; `$x_2$`; …; `$x_{n-2}$`; `$x_{n-1}$`]` et retournant la liste `[`$x_0$`; `$x_{n-1}$`; `$x_1$`; `$x_{n-2}$`; `$x_2$`; …]`

**Q. 2** Si la réponse à la première question est de complexité quadratique en la taille de la liste initiale, même question mais avec une complexité linéaire.

# Exercice 3 : Parmi

**Q. 1** Donner une fonction `k_parmi_n : 'a list -> int -> 'a list list` prenant en paramètre une liste $l$ et un entier $k$ et calculant l'ensemble de toutes les sous-listes de $k$ éléments de la liste $l$ (on devra garder les éléments dans l'ordre d'apparition dans `l`). Ainsi `k_parmi_n [1; 2; 3; 4] 2` devra produire `[[1; 2]; [1; 3]; [1; 4]; [2; 3]; [2; 4]; [3; 4]]`.

## Exercice 4 : Un sur deux

**Q. 1** Donner une fonction prenant en argument une liste $l$ et calculant la liste obtenue en prenant un élément sur deux de $l$.

## Exercice 5 : Partition listes

**Q. 1** Implémenter la fonction `partition: ('a -> bool) -> 'a list -> 'a list * 'a list` prenant en argument un prédicat et une liste et calculant la paire de la liste des éléments satisfaisant le prédicat et ceux ne le satisfaisant pas.

On définit le type `type 'a option = None | Some of 'a`

**Q. 2** Implémenter une fonction `filter_map : ('a -> 'b option) -> 'a list -> 'b list` telle que `filter_map f l` applique `f` à chaque élément de `l`, ôte de la liste les éléments pour lesquels l'appel produit `None` et calcule la liste des éléments se trouvant dans le `Some` sinon.

## Exercice 6 : Énumération des listes de booléens

**Q. 1** Donner une fonction `generate : int -> bool list list` prenant en argument un entier $n$ et générant la liste de toutes les listes de booléens de taille $n$.

**Q. 2** Donner une fonction `enumerate : int -> (int -> bool list)` prenant en argument un entier $n$ et calculant une fonction $f$ réalisant une bijection de $[\![0, 2^n - 1]\!]$ dans l'ensemble des listes de booléens de longueur $n$.

## Exercice 7 : Suppression des occurrences consécutives

**Q. 1** Écrire une fonction `compresse : 'a list -> 'a list` qui élimine les éléments consécutifs identiques d'une liste. Par exemple :

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

## Exercice 8 : Liste 2

**Q. 1** Implanter une fonction `combine : 'a list -> 'b list -> ('a * 'b) list` prenant en argument deux listes de même longueur et les combinant : `combine [a1; ...; an] [b1; ...; bn] = [(a1, b1); ...; (an, bn)]`.

## Exercice 9 : Permutation

**Q. 1** Donner une fonction `permutations : 'a list -> 'a list list` prenant en argument une liste et retournant l'ensemble de toutes les permutations de cette liste. L'ordre des permutations dans la liste résultat n'a pas d'importance.