Par convention, les identificateurs des variables mathématiques sont en police mathématique n, les identificateurs des variables informatiques sont en police bureautique `n`,
Etant entendu que le même identificateur, peu importe sa police, fait référence à la même variable, la police change en fonction de son contexte d'utilisation.

1. Programmation de certaines primitives sur les listes.

   L'idée de ce paragraphe est de mettre en place les instructions classiques sur les listes à partir des commandes de bases que sont :
   — la liste vide `[]`,
   — l'ajout-en-tête d'une liste `::` ,
   — la tête d'une liste `List.hd`,
   — la queue d'une liste `List.tl`.

   Par exemple, mettons en place une fonction qui calcule la longueur d'une liste :

```
let rec longueur l = match l with
    | [ ] -> 0
    | t::q -> 1 + longueur q
;;
```

   Dans la suite, les commandes sur les listes se limitent aux commandes de bases.

   (a) Ecrire en Caml, une fonction qui détermine l'appartenance d'un élément à une liste :
   `let rec appartient x l`

   (b) Ecrire en Caml, une fonction qui détermine le nombre d'apparitions d'un élément dans une liste :
   `let rec occurence x l`

   (c) Ecrire en Caml, une fonction qui extrait le dernier élément d'une liste :
   `let rec dernier l`

   (d) Ecrire en Caml, une fonction qui ajoute un élément en queue de liste :
   `let rec ajout_fin x l`

   (e) Ecrire en Caml, une fonction qui extrait le i^{e} élément d'une liste :
   `let rec i_eme i l`

   (f) Ecrire en Caml, une fonction qui supprime le i^{e} élément d'une liste :
   `let rec supprime i l`

   (g) Ecrire en Caml, une fonction qui supprime toutes les occurences d'un élément dans une liste :
   `let rec supprime_tout x l`

   (h) Ecrire en Caml, une fonction qui insère un élément dans une liste en i^{e} position :
   `let rec insere x i l`

   (i) Ecrire en Caml, une fonction qui concatène deux listes :
   `let rec concatene l1 l2`

   (j) Ecrire en Caml, une fonction qui effectue la concaténation inverse de deux listes :
   `let rec concatene_inverse l1 l2`
   Par exemple, `concatene_inverse [2;4;7] [1;3]` retourne `[7;4;2;1;3]` .

En déduire une fonction en Caml qui calcule le miroir d'une liste :

```
let miroir l
```

Par exemple, `miroir [7;4;2;1;3]` retourne `[3;1;2;4;7]` .

2. Opérations « algébriques » sur les listes.

   (a) Ecrire en Caml une fonction d'addition de deux listes :

   ```
   let rec addition l1 l2
   ```

   Par exemple, `addition [1;3;2;6;7] [2;2;4]` retourne

   `[3(=1+2);5(=3+2);6(=2+4);6;7]` .

   (b) Ecrire en Caml une fonction de fusion de deux listes triées par ordre croissant

   ```
   let rec fusion l1 l2
   ```

   Par exemple, `fusion [1;3;4;7;8] [2;4;5;9]` retourne

   `[1;2;3;4;4;5;7;8;9]` .

3. Algorithmes classiques sur les listes.

   (a) Ecrire en Caml, une fonction qui détermine le maximum des éléments d'une liste, les éléments de la liste appartenant à un ensemble ordonné :

   ```
   let rec maximum l
   ```

   (b) Réfléchir à une fonction récursive qui détermine le second maximum, s'il existe, des éléments d'une liste, les éléments de la liste appartenant à un ensemble ordonné.

   ```
   let rec second_maximum l
   ```

   (c) Ecrire en Caml, une fonction qui calcule la somme des éléments d'une liste, les éléments de la liste appartenant à un ensemble muni d'une addition :

   ```
   let rec somme l
   ```

4. Algorithme de Hörner.

   Les coefficients d'un polynôme $P = \sum_{k\in\mathbb{N}} a_k X^k$ de degré inférieur strictement à un entier naturel non nul n sont stockés dans une liste $l = [a_0; a_1; \ldots; a_{n-1}]$.

   Ecrire en Caml une fonction d'évaluation du polynôme P en une valeur entière x :

   ```
   let rec valeur_en_un_point P x
   ```

5. Coefficients binômiaux.

   (a) Ecrire en Caml une fonction de calcul du coeffcient binômial $\binom{n}{p}$, $(n,p) \in \mathbb{N}^2$:

   ```
   let rec binomial n p
   ```

   (b) Ecrire en Caml une fonction qui, étant une ligne du triangle de Pascal représentée sous forme d'une liste, calcule la ligne suivante :

   ```
   let rec ligne_suivante l
   ```

   Par exemple, `ligne_suivante [1;3;3;1]` retrourne `[1;4;6;4;1]` .

   (c) Ecrire en Caml une fonction qui pour un entier naturel n , renvoie la n^{e} ligne du triangle de Pascal.

   ```
   let rec pascal n
   ```

   Per exemple, `pascal 4` retourne `[1;4;6;4;1]` .