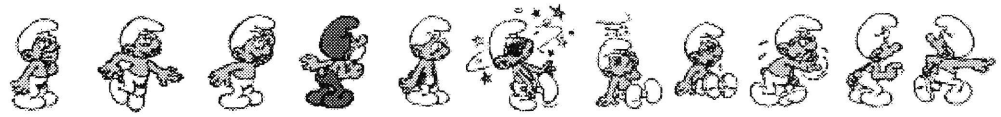
Les listes



1 Parcours simples

Exercice 1.1 (Somme)

Écrire une fonction qui calcule la somme de tous les éléments d'une liste d'entiers.

Exercice 1.2 (Compte)

1. Écrire une fonction qui compte le nombre d'occurrences d'une valeur donnée dans une liste quelconque.
2. Écrire une fonction qui compte le nombre d'occurrences d'une valeur donnée dans une liste triée en ordre croissant.

Exercice 1.3 (Recherche)

Écrire une fonction qui recherche si un élément est présent dans une liste triée (en ordre croissant).

Exercice 1.4 (i^{ème})

Écrire une fonction qui donne la valeur du $i^{ème}$ élément d'une liste. La fonction devra déclencher une exception `Invalid_argument` si i est négatif ou nul, ou une exception `Failure` si la liste est trop courte.

Exercice 1.5 (Croissance ?)

Écrire une fonction qui teste si une liste est triée en ordre croissant.

Exercice 1.6 (Maximum)

Écrire une fonction qui retourne la valeur maximum d'une liste.

---

2 Le résultat est une liste

Exercice 2.1 (Liste arithmétique)

Écrire la fonction `arith_list` n a_1 r qui construit la liste des n premiers termes de la suite arithmétique de premier terme a_1 et de raison r.

Exemples d'application :

```
# arith_list 12 2 1;;
 - : int list = [2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12; 13]
# arith_list 11 0 3;;
 - : int list = [0; 3; 6; 9; 12; 15; 18; 21; 24; 27; 30]
```

Exercice 2.2 (Concaténation)

Écrire une fonction qui concatène deux listes (l'équivalent de l'opérateur `@`).

Exemple d'application :

```
# append [1;2;3;4] [5;6];;
- : int list = [1; 2; 3; 4; 5; 6]
```

Exercice 2.3 (Suppression)

Écrire une fonction qui supprime d'une liste l triée (en ordre croissant) la première occurrence d'un élément x (s'il est présent).

Exercice 2.4 (Insertion)

Écrire une fonction qui ajoute un élément à sa place dans une liste triée en ordre croissant.

Exercice 2.5 (Inverse)

Écrire une fonction qui inverse une liste :

1. en utilisant l'opérateur `@` ;
2. sans utiliser l'opérateur `@`.

Que pensez-vous de la complexité de ces deux fonctions ?

---

3 Deux listes

Exercice 3.1 (Égalité)

Écrire une fonction qui teste si deux listes sont identiques.

Exercice 3.2 (Communs)

Soient deux listes d'éléments strictement croissantes. Écrire une fonction qui construit la liste des éléments communs aux deux listes.

Exemple d'application :

```
# shared [1; 3; 6; 6; 8; 9] [2; 5; 6; 6; 8; 9] ;;
- : int list = [6; 6; 8; 9]
```

Exercice 3.3 (Fusion)

Écrire une fonction qui fusionne deux listes triées en une seule.
Comment éliminer les éléments communs aux deux listes ?

Exemple d'application :

```
# merge [1; 5; 6; 8; 9; 15] [2; 3; 4; 5; 7; 9; 10; 11];;
- : int list = [1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 15]
```

4 Petits problèmes

Exercice 4.1 (Représentation des polynômes par listes – *Machine Exam (2016)*)

◦ Un polynôme $a_n.x^n + ... + a_1.x + a_0$ peut être représenté par la liste $[a_0; a_1; ...; a_n]$ de ses coefficients [1].
La représentation proposée ci-dessus a l'inconvénient d'avoir une taille proportionnelle à l'ordre du polynôme et non pas au nombre de monômes non nuls.

Par exemple, le polynôme $4x^5 + x^2 - 3x + 1$ serait représenté par la liste :

```
[1; -3; 1; 0; 0; 4]
```

◦ Une autre solution (utilisée ici) consiste donc à représenter un polynôme par la liste des monômes. Un monôme $c.X^d$ est représenté par le couple d'entiers (c, d). Les monômes sont triés par degrés décroissants et chaque monôme a un coefficient non nul ($c \neq 0$).

La liste vide représentera le polynôme nul.

Par exemple, le polynôme $4x^5 + x^2 - 3x + 1$ sera représenté par la liste :

```
# let poly = [(4,5); (1,2); (-3,1); (1,0)] ;;
 val poly : (int * int) list = [(4, 5); (1, 2); (-3, 1); (1, 0)]
```

Et $2x^5 + 2x^4 - x^2 + x$:

```
# let poly2 = [(2,5) ; (2,4) ; (-1,2) ; (1,1)] ;;
 val poly2 : (int * int) list = [(2, 5); (2, 4); (-1, 2); (1, 1)]
```

Les polynômes résultats devront être bien formés, c'est à dire :
- il n'y a qu'un seul monôme par degré ;
- il n'y a aucun coefficient nul dans la liste ;
- les monômes sont triés par degrés décroissants.

1. Écrire la fonction `add_poly` calculant la somme de deux polynômes.
   Exemple : $(4x^5 + x^2 - 3x + 1) + (2x^5 + 2x^4 - x^2 + x) = 6x^5 + 2x^4 - 2x + 1$

```
val add_poly : (int * 'a) list -> (int * 'a) list -> (int * 'a) list = <fun>

# add_poly (poly1, poly2) ;;
 - : (int * int) list = [(6, 5); (2, 4); (-2, 1); (1, 0)]
```

2. Écrire la fonction `mult_poly poly mono` qui multiplie un polynôme `poly` par un monôme `mono`.
   Exemple : $(4x^5 + x^2 - 3x + 1) \times (2x) = 8x^6 + 2x^3 - 6x^2 + 2x$

```
val mult_poly : (int * int) list -> (int * int) -> (int * int) list = <fun>

# mult_poly poly1 (2, 1) ;;
- : (int * int) list = [(8, 6); (2, 3); (-6, 2); (2, 1)]
```

3. Écrire la fonction `product_poly` calculant le produit de deux polynômes.
   Exemple : $(4x^5 + x^2 - 3x + 1) \times (2x^5 + 2x^4 - x^2 + x) = 8x^{10} + 8x^9 - 2x^7 - 4x^5 + x^4 + 4x^3 - 4x^2 + x$

```
val product_poly : (int * int) list -> (int * int) list -> (int * int) list = <fun>

# product_poly poly1 poly2 ;;
 - : (int * int) list = [(8, 10); (8, 9); (-2, 7); (-4, 5); (1, 4); (4, 3); (-4, 2); (1, 1)]
```

1. Il peut sembler plus logique d'inverser l'ordre des coefficients, mais la manipulation sera beaucoup trop compliquée !

Exercice 4.2 (Crible d'Ératosthène)

Le *crible d'Ératosthène* est un algorithme simple qui permet de trouver tous les nombres premiers inférieurs à un certain entier naturel donné N. L'algorithme procède par éliminations :
- Au départ, tout nombre à partir de 2 est supposé premier.
- Ensuite, pour chaque entier qui est premier, ses multiples ne sont pas premiers et sont donc éliminés.

Écrire la fonction `eratosthenes` qui appliquée à un entier n strictement supérieur à 1 donne la liste de tous les entiers premiers jusqu'à n.
Exemple d'application :

```
# eratosthenes 30 ;;
- : int list = [2; 3; 5; 7; 11; 13; 17; 19; 23; 29]
```

---

5 Des bonus

Exercice 5.1 (Codage RLE simplifié)

Le but de cet exercice est de compresser une liste d'éléments grâce à une version simplifiée de l'algorithme RLE (Run Length Encoding).
La compression RLE standard permet de compresser des éléments par factorisation, mais uniquement lorsque ceci permet de gagner de la place. Cette compression est essentiellement utilisée dans le format d'images pcx, elle est non destructive, et très performante sur des images comportant de longues suites de pixels de couleur constante. Votre algorithme RLE simplifié effectuera cette factorisation dans tous les cas, y compris quand cela prend plus de place que l'objet non compressé.

Le flux que l'on encode est une liste d'éléments ; le flux encodé est une liste de couples, chaque couple étant composé du nombre d'éléments consécutifs identiques, puis de cet élément.

1. Écrire une fonction `encode_rle` de type `'a list -> (int * 'a) list` qui permet de compresser une liste en utilisant l'algorithme RLE.

```
# encode_rle ['b';'b';'b';'c';'a';'a';'e';'e';'e';'e';'d';'d'] ;;
- : (int * char) list = [(3, 'b'); (1, 'c'); (2, 'a'); (4, 'e'); (2, 'd')]
```

2. Écrire une fonction `decode_rle` qui permet de décompresser une liste compressée en RLE.

```
# decode_rle [(6, "grr")] ;;
- : string list = ["grr"; "grr"; "grr"; "grr"; "grr"; "grr"]
```

Exercice 5.2 (Liste de couples)

1. Écrire la fonction `combine` ayant les spécifications suivantes :

```
val combine : 'a list -> 'b list -> ('a * 'b) list
```

Transform a pair of lists into a list of pairs : `combine [a1;...;an] [b1;...; bn]` is `[(a1,b1);...; (an,bn)]`. Raise `Invalid_argument` if the two lists have different lengths. Not tail-recursive.

Exemple :

```
# combine [1; 2; 3] ['a'; 'b'; 'c'];;
- : (int * char) list = [(1, 'a'); (2, 'b'); (3, 'c')]
```

2. Écrire la fonction `split` ayant les spécifications suivantes :

```
val split : ('a * 'b) list -> 'a list * 'b list
```

Transform a list of pairs into a pair of lists : split [(a1,b1) ; ... ; (an,bn)] is ([a1 ; ... ; an], [b1 ; ... ; bn]). Not tail-recursive.

Exemple :

```
# split [(1,'a'); (2,'b'); (3,'c')];;
- : int list * char list = ([1; 2; 3], ['a'; 'b'; 'c'])
```