

Exercice 1 : Écrire des fonctions répondant aux spécifications suivantes :

1. `premierElement` : `'a list -> 'a` renvoyant le premier élément d'une liste s'il existe.
2. `dernierElement` : `'a list -> 'a` renvoyant le dernier élément d'une liste s'il existe.
3. `avantDernier` : `'a list -> 'a` renvoyant l'avant dernier élément d'une liste s'il existe.
4. `element` : `'a list -> int -> 'a` renvoyant le k -ème élément d'une liste s'il existe.
5. `indice` : `'a list -> 'a -> int` renvoyant l'indice d'un élément x dans une liste.
6. `ajouterEnQueue` : `'a list -> 'a -> 'a list` rajoutant un élément à la fin d'une liste.
7. `miroir` : `'a list -> 'a list` renvoyant le miroir d'une liste.
8. `maMap` : `('a -> 'b) -> 'a list -> 'b list` telle que l'instruction `maMap f [x1; x2; ...; xn]` renvoie la liste `[f(x1); f(x2); ...; f(xn)]`.
9. `somme` : `int list -> int` renvoyant la somme des éléments d'une liste.

Exercice 2 : On considère la fonction OCaml suivante :

```
let rec mystere f l = match l with
| [] -> []
| t::q -> (f t)::mystere f q
```

1. Quel est son type ?
2. Déterminer son rôle au moyen d'une phrase.
3. Faire le lien avec une fonction vue en cours.

Exercice 3 : On considère la fonction OCaml suivante :

```
let rec mystere' f = function
| [] -> []
| t::q when f t > 1 -> t::(mystere' f q)
| t::q -> mystere' f q
```

1. Quel est le type de cette fonction ?
2. Quel est son rôle ?

3. Écrire une fonction `filtrer` : `('a list -> bool) -> 'a list -> 'a list` renvoyant la liste des éléments d'une liste vérifiant $f(t) = \text{true}$ où f est une fonction à valeurs booléennes.

Exercice 4 : Écrire une fonction `concat` : `'a list -> 'a list -> 'a list` prenant en arguments deux listes de même type et renvoyant leur concaténation (on n'utilisera pas l'opérateur `@`).

Exercice 5 : Écrire une fonction `listIt f l b` : `('a -> 'b -> 'b) -> 'a list -> 'b -> 'b` prenant en arguments une fonction f , une liste $l = [a_0, \dots, a_{n-1}]$ et un objet b et qui renvoie la valeur :

$$f(a_0, f(a_1, f(a_2, \dots f(a_{n-1}, b))))$$

Exercice 6 : Il existe une fonction `List.exists` : `('a -> bool) -> 'a list -> bool` qui détermine si, dans une liste prise en argument, il existe un élément vérifiant une certaine proposition (c'est à dire un élément x tel que $f(x)$ est vrai où f est une fonction à valeurs booléenne).

1. Donner une implémentation `existe` pour cette fonction en utilisant le filtrage par motif.
2. Écrire une fonction `pourTout` : `('a -> bool) -> 'a list -> bool` qui détermine si tous les éléments d'une liste vérifient une certaine propriété.

Exercice 7 : On souhaite écrire une fonction `purge` : `'a list -> 'a list` qui, appliquée à une liste, retourne une liste dans laquelle les doublons ont été éliminés.

1. Écrire une fonction `appartient` : `'a -> 'a list -> bool` telle que `appartient x l` renvoie `true` si $x \in l$ (et `false` sinon).
2. Écrire une première version de `purge` dans laquelle seule la dernière occurrence de chaque doublon sera conservée. Par exemple, `purge [1; 2; 3; 1; 4; 3; 1]` renverra `[2; 4; 3; 1]`.
3. Écrire une seconde version de `purge` dans laquelle seule la première occurrence de chaque doublon sera conservée. Cette fois, `purge [1; 2; 3; 1; 4; 3; 1]` renverra `[1; 2; 3; 4]`.

Exercice 8 : Dans cet exercice, on représente les ensembles au moyen de listes. Proposer des fonctions permettant de :

1. Déterminer si un élément x appartient à un ensemble (type `'a -> 'a list -> bool`).
2. Renvoyer l'union des deux ensembles sans tenir compte des doublons.
3. Renvoyer l'intersection des deux ensembles.
4. Renvoyer l'union des deux ensembles en tenant compte des doublons.
5. Analyser la complexité des fonctions proposées.

Exercice 9 : Le **tri par insertion** vise à trier dans l'ordre croissant une liste d'entiers. Il repose sur le principe suivant :

- On dispose d'une fonction `insere x l : int -> int list -> int list` qui insère l'élément x à sa place dans la liste l supposée déjà triée.
- Ensuite on crée une fonction de tri ainsi :
 - si la liste est vide : on ne fait rien,
 - si la liste est de la forme $t :: q$ on trie récursivement q et on insère t à sa place dans la liste triée.

1. Implémenter le tri par insertion sur des listes OCaml.
2. En ne comptant comme opérations élémentaires que les opérations `::` et les comparaisons, déterminer une relation de récurrence vérifiée dans le pire des cas par le tri par insertion.
3. Résoudre cette relation de récurrence.
4. Si on suppose que la liste est déjà triée, quelle est la complexité du tri par insertion ?

Exercice 10 : Le **tri par sélection** vise à trier dans l'ordre croissant une liste d'entiers. Il repose sur le principe suivant :

- On cherche le minimum de la liste à trier,
- On place ce minimum en tête de liste,
- On trie récursivement la liste privée de son premier élément.

1. Implémenter cette méthode de tri.
2. En ne comptant comme opérations élémentaires que les opérations `::` et les comparaisons, déterminer une relation de récurrence vérifiée dans le pire des cas par le tri par sélection.
3. Résoudre la récurrence.

Exercice 11 : Le **tri à bulles** vise à trier dans l'ordre croissant une liste d'entiers. Il repose sur le principe suivant :

- On parcourt la liste de gauche à droite, en comparant les éléments consécutifs : s'ils ne sont pas dans le bon ordre, on les échange.
- À la fin du parcours :
 - si on a effectué au moins un échange durant ce parcours, on recommence ;
 - sinon, la liste est triée.

1. Implémenter cette méthode de tri.
2. En ne comptant comme opérations élémentaires que les opérations `::` et les comparaisons, déterminer une relation de récurrence vérifiée dans le pire des cas par le tri à bulles.
3. Résoudre la récurrence.