Rappel : une liste « `li: 'a list` » est soit la liste vide notée `[]`, soit une liste non vide de la forme « `e :: q` » où « `e: 'a` » est la tête de `li` et « `q: 'a list` » est sa queue.

En général, lorsqu'on écrit une fonction prenant en entrée une liste, on utilise un filtrage comme dans le programme ci-contre.

```
let rec f li = match li with
  | []   -> ...
  | e::q -> ...
```

Exercice 1. Fonctions élémentaires sur les listes

Le but de cet exercice est de réécrire les fonctions `List.hd`, `List.tl`, `List.length`, `List.nth` et `List.append` d'OCaml. Dans toutes les questions, vous devez définir une fonction récursive sans passer par une fonction intermédiaire et ne pas utiliser les fonctions du module `List`.

1. Écrire une fonction « `hd: 'a list -> 'a` » qui renvoie le premier élément de la liste.
2. Écrire une fonction « `tl: 'a list -> 'a list` » qui renvoie la queue de la liste.
3. Écrire une fonction « `length: 'a list -> int` » qui renvoie la taille de la liste.
4. Écrire une fonction « `nth: 'a list -> int -> 'a` » qui renvoie l'élément d'indice n de la liste.
5. Écrire une fonction « `append: 'a list -> 'a list -> 'a list` » qui concatène deux listes.

Exercice 2. Suites de Skolem

Dans son laboratoire, le professeur Skolem réalise une expérience avec $n \in \mathbb{N}^*$ machines numérotées de 1 à n. Initialement, toutes les machines sont éteintes. Pour que son expérience soit un succès :

- ⋆ Chaque machine doit être allumée une et une seule fois, puis éteinte une et une seule fois.
- ⋆ Pour chaque $k \in [\![1, n]\!]$, la machine numéro k doit fonctionner k minutes.
- ⋆ Au début de chaque minute, le professeur peut allumer ou bien éteindre l'une des machines. Le temps de l'expérience étant limité, il doit obligatoirement faire une action par minute.

Par exemple, avec $n = 4$, voici une suite d'actions pour laquelle l'expérience est un succès :

Minute	1	2	3	4	5	6	7	8
Action	Allumer machine 2	Allumer machine 3	Éteindre machine 2	Allumer machine 4	Éteindre machine 3	Allumer machine 1	Éteindre machine 1	Éteindre machine 4

Pour tout $n \in \mathbb{N}^*$, on appelle ***nombre de Skolem*** et on note S_n le nombre de suites d'actions pour lesquelles l'expérience est un succès. En d'autres termes, S_n est le nombre de tableaux (comme celui ci-dessus) qui respectent les contraintes de l'énoncé.

1. À la main, calculer S_1, S_2, S_3 et S_4.
2. Écrire une fonction qui prend en entrée n et renvoie S_n. On pourra vérifier qu'on obtient bien le nombre de "suites de Skolem" sur le site `https://oeis.org/`.

Exercice 3. Conversion entre listes et tableaux

Dans cet exercice, on souhaite réécrire les fonctions `Array.to_list` et `Array.of_list`.

1. À l'aide d'une fonction récursive, écrire une fonction « `list_of_array: 'a array -> 'a list` » qui convertit un tableau en une liste.
2. À l'aide d'une fonction récursive, écrire une fonction « `array_of_list: 'a list -> 'a array` » qui convertit une liste en un tableau.

Exercice 4. Diagrammes de Ferrers et tableaux de Young

Soit `li` = $[p_0; \ldots; p_{k-1}]$ une liste d'entiers de taille `k` $\in \mathbb{N}$ triée par ordre décroissant :

$$p_0 \geqslant p_1 \geqslant \cdots \geqslant p_{k-1}.$$

Le ***diagramme de Ferrers*** associé à `li` est composé de `k` lignes où la ligne d'indice `i` $\in [\![0; k-1]\!]$ comporte p_i points. Par exemple, le diagramme de Ferrers de `[8; 4; 4; 3; 1]` est :

×	×	×	×	×	×	×	×
×	×	×	×				
×	×	×	×				
×	×	×					
×							

À partir d'un diagramme de Ferrers, on obtient une autre liste notée $[q_0; \ldots; q_{h-1}]$ appelée ***liste duale*** de `li` :

- La taille `h` de cette liste est le nombre de colonnes dans le diagramme de Ferrers.
- Pour chaque `j` $\in [\![0; h-1]\!]$, q_j est le nombre de points dans la colonne d'indice `j`.

Par exemple, en utilisant le diagramme ci-dessus, la liste duale de `[8; 4; 4; 3; 1]` est `[5; 4; 4; 3; 1; 1; 1; 1]`, elle est de taille `h = 8`.

1. Écrire une fonction « `duale: int list -> int list` » qui renvoie la liste duale de son argument.

Soit `n` = $p_0 + p_1 + \ldots + p_{k-1}$ la somme des éléments de `li`. Remplaçons chaque point du diagramme de Ferrers par un nombre de $[\![0; n-1]\!]$ en respectant les trois contraintes suivantes :

- Chaque nombre de $[\![0; n-1]\!]$ doit apparaître une et une seule fois.
- Chaque ligne doit être triée par ordre croissant.
- Chaque colonne doit être triée par ordre croissant.

L'objet obtenu s'appelle un ***tableau de Young*** pour `n`. Par exemple, voici trois tableaux de Young pour `n = 20` :

0	1	2	3	4	5	6	7
8	9	10	11				
12	13	14	15				
16	17	18					
19							

0	2	7	12	13	16	18	19
1	3	8	14				
4	6	11	15				
5	9	17					
10							

0	2	3	10	16
1	4	9	11	
5	6	12	17	
7	8	18	19	
13	15			
14				

2. Écrire une fonction « `nbTabYoung: int -> int` » qui prend en entrée un entier `n` $\in \mathbb{N}$ et renvoie le nombre de tableaux de Young associés à `n`.

Exercice 5. Fonctionnelles itératives

1. Écrire une fonction « `map: ('a -> 'b) -> 'a list -> 'b list` » qui prend en argument une fonction `f` ainsi qu'une liste `[a0; a1; ...; an]`, et renvoie la liste `[f a0; f a1; ...; f an]`. En d'autres termes, `map` doit avoir le même comportement que la fonction `List.map`.
2. Écrire une fonction `iter` qui prend en argument une fonction « `f: 'a -> unit` » ainsi qu'une liste « `li: 'a list` », et applique successivement `f` à chaque élément de `li`. En d'autres termes, `iter` doit avoir le même comportement que la fonction `List.iter`. Utiliser votre fonction pour afficher tous les éléments d'une liste d'entiers.
3. Écrire une fonction `fold_left` qui prend en argument une fonction « `f: 'a -> 'b -> 'a` » ainsi qu'un argument « `a: 'a` » et une liste `[b1; b2; ...; bn]: 'b list`, et renvoie « `f (...f (f a b0) b1) ... bn` ». En d'autres termes, `fold_left` doit avoir le même comportement que la fonction `List.fold_left`. Par exemple :

   « `fold_left (fun a b -> a +. float_of_int b) 1. [2; 3; 4]` » vaut `10.`

   « `fold_left (fun a b -> not a || b) false [false; true]` » vaut `true`.

4. Écrire une fonction « `for_all: ('a -> bool) -> 'a list -> bool` » qui prend en argument un prédicat « `p: 'a -> bool` » ainsi qu'une liste, et renvoie `true` si et seulement si « `p a` » vaut `true` pour tous les éléments de la liste. En d'autres termes, « `for_all p [a1; a2; ...; an]` » vaut `(p a1) && (p a2) && ... && (p an)`.
5. Écrire une fonction « `exists: ('a -> bool) -> 'a list -> bool` » qui prend en argument un prédicat « `p: 'a -> bool` » ainsi qu'une liste, et renvoie `true` si et seulement s'il existe un élément de la liste `a` tel que « `p a` » vaut `true`.
6. Si vous n'avez pas utilisé la question 3 dans les questions 4 et 5, refaire ces questions en utilisant la fonction `fold_left`.
7. À l'aide de la fonction `fold_left`, écrire une fonction « `length: 'a list -> int` » qui renvoie la taille de la liste passée en argument. En d'autres termes, `length` doit avoir le même comportement que la fonction `List.length`.
8. À l'aide de la fonction `fold_left`, écrire une fonction « `iter_bis: ('a -> unit) -> 'a list -> unit` » comme dans la question 2.

Exercices à rendre au plus tard le 14/03/2024 à 20h

Rappel : les fonctions doivent être testées et vos tests doivent apparaître dans le fichier que vous rendez. Un seul test n'est pas suffisant et vos tests doivent être pertinents. Si vous ne respectez pas ces consignes, vous perdez des points.

Exercice 6.

À partir de maintenant, merci d'utiliser exclusivement le raccourci clavier `Ctrl + Maj + Entrée` pour exécuter vos programmes sous VS-code. Les messages d'erreurs posent parfois problème avec le raccourci `Ctrl + Entrée`.

Si ce n'est pas encore fait, vous devez donc mettre en place ce raccourci en suivant les instructions décrites dans le pdf disponible en haut de la page du cours :

`https://informatique-lhp.fr/opt-mpsi.html`

Exercice 7. Fonctions sur les listes d'entiers

1. Sans utiliser de fonction intermédiaire, écrire une fonction récursive « `maxi: int list -> int` » qui renvoie le maximum des éléments de la liste donnée en entrée.
2. Sans utiliser de fonction intermédiaire, écrire une fonction récursive « `prod: int list -> int` » qui renvoie le produit des éléments de la liste donnée en entrée.
3. Sans utiliser de fonction intermédiaire, écrire une fonction récursive « `add: int list -> int list -> int list` » qui additionne terme à terme les éléments des deux listes en entrée. Dans le cas où les deux listes n'ont pas la même longueur, on s'arrêtera à la fin de la liste la plus courte. Par exemple :

li1	[2; 6; 9]	[1; 4; 7; 2]	[1; 4]	[1; 4; 7; 2]
li2	[]	[-2; 5; -2; 5]	[-2; 5; -2; 5]	[-2; 5]
add li1 li2	[]	[-1; 9; 5; 7]	[-1; 9]	[-1; 9]

4. Écrire une fonction récursive qui prend en entrée une liste d'entiers $[a_1; \ldots; a_n]$ et renvoie :

$$\prod_{i=1}^{n} \prod_{j=i+1}^{n} (a_i - a_j)$$

Exercice 8. Tableaux sans double montée (facultatif)

Soit $n \in \mathbb{N}$ un entier et `tab` un tableau de taille n contenant une et une seule fois chaque élément de $[\![1, n]\!]$. On appelle ***double montée*** un indice $i \in [\![0, n-3]\!]$ tel que $tab.(i) \leqslant tab.(i+1) \leqslant tab.(i+2)$.

1. Écrire une fonction « `sans_DM: int -> int` » qui prend en entrée n et renvoie le nombre de tableaux sans double montée. Par exemple :

n	0	1	2	3	4	10
sans_DM(n)	1	1	2	5	17	822041