

# Programmation fonctionnelle

## Fiche de TP 3

L3 Informatique 2020-2021

*Listes, fonctions sur les listes*

Rappel : une liste est un objet récursif défini comme étant

- soit vide,
- soit un élément suivi d'une liste.

En Caml, la liste vide est notée []. Le premier élément d'une liste est obtenu grâce à la fonction `List.hd` et la fin de la liste est donnée par `List.tl`. La fonction qui ajoute un élément en tête de liste est l'opérateur infixe `::`. On peut également concaténer des listes avec l'opérateur `@`.

`List` est un module Caml. Il est possible d'utiliser ses fonctions en se passant du préfixe `List.` en chargeant le module au préalable.

```
1# open List;;
2# let l = [3; 5; 1; 3];;
3val l : int list = [3; 5; 1; 3]
4
5# hd l, tl l;;
6- : int * int list = (3, [5; 1; 3])
7
8# 1::l;;
9- : int list = [1; 3; 5; 1; 3]
10
11# l @ l;;
12- : int list = [3; 5; 1; 3; 3; 5; 1; 3]
```

### Exercice 1. (Une simple liste d'entiers)

On veut fabriquer une liste contenant les entiers de 0 à  $n$ .

1. On écrit la fonction récursive non terminale suivante pour fabriquer une telle liste. Quel est problème?

```
1 let rec integers_1 n = if n < 0 then [] else n :: integers_1 (n - 1) ;;
```

2. Ré-essayer en utilisant @ et nommer la fonction obtenue integers\_2. Quel est problème (tester la fonction avec  $n = 100\,000$ ) ?
3. Ré-essayer en utilisant List.rev et nommer la fonction obtenue integers\_3.
4. Que se passe-t-il si l'on transforme la fonction integers\_1 en une fonction récursive terminale ?

```

1# #use "tp3.ml";;
2
3...
4val integers_1 : int -> int list = <fun>
5
6...
7val integers_2: int -> int list = <fun>
8
9# let l = integers_1 10;;
10val l : int list = [10; 9; 8; 7; 6; 5; 4; 3; 2; 1; 0]
11
12# integers_2 10;;
13- : int list = [0; 1; 2; 3; 4; 5; 6; 7; 8; 9; 10]
14
15# List.rev l;;
16- : int list = [0; 1; 2; 3; 4; 5; 6; 7; 8; 9; 10]

```

## Exercice 2. (Traiter des listes de nombres)

Écrire les fonctions suivantes (et observer leur type). **Dans la mesure du possible, il ne faut parcourir la liste qu'une seule fois.**

Certaines fonctions demandent aux préalable des conditions précises sur la liste (comme sa différence par rapport à la liste vide). Pour écrire ces fonctions, on définira une **exception** par

```
1exception Empty_list
```

et on la **lèvera** si besoin par

```
1raise Empty_list
```

(Conseil : demander à l'interpréteur le type de la fonction raise.)

1. (three\_or\_more l) qui teste si la liste l a au moins trois éléments (utiliser la fonction List.length n'est pas une bonne idée!).
2. (size l) qui renvoie la taille de la liste l (toujours sans utiliser List.length).
3. (last l) qui renvoie le dernier élément de la liste l **non vide**.
4. (is\_increasing l) qui teste si la liste l est croissante.

5. (even\_odd l) qui teste si la liste l est telle que ses 1<sup>er</sup>, 3<sup>e</sup>, 5<sup>e</sup>, etc. éléments sont impairs et les autres pairs.
6. (find e l) qui teste si l'élément e est dans la liste l.
7. (member e l) qui renvoie la portion de la liste l commençant à la première occurrence de l'élément e.
8. (member\_last e l) qui renvoie la portion de la liste l commençant à la dernière occurrence de l'élément e.
9. (nb\_occ e l) qui compte le nombre d'occurrences de l'élément e dans la liste l.
10. (nth n l) qui renvoie le n<sup>e</sup>élément de la liste l.
11. (max\_list l) qui renvoie le maximum de la liste l **non vide**.
12. ★ (nb\_max l) qui renvoie le nombre de maximums de la liste l. Essayer de réaliser ce calcul en un seul parcours.
13. (average l) qui renvoie la moyenne des nombres de la liste de flottants l.
14. ★ (size\_in\_range a b l) qui teste si la longueur de la liste l est dans l'intervalle [a, b] (ou [b, a]).
15. (find\_pattern p l) qui teste si la liste p est un motif de la liste l.

```

1# three_or_more [], three_or_more [1; 1; 1; 1; 1];;
2- : bool * bool = (false, true)
3
4# size [], size [3; 1; 4; 5; 2];;
5- : int * int = (0, 5)
6
7# last [1], last [3; 1; 4; 5; 2];;
8- : int * int = (1, 2)
9
10# is_increasing [], is_increasing [3; 1; 4; 5; 2], is_increasing [1; 3; 5; 5; 7];;
11- : bool * bool * bool = (true, false, true)
12
13# even_odd [], even_odd [1; 4; 3; 6; 9; 2], even_odd [2; 3; 3];;
14- : bool * bool * bool = (true, true, false)
15
16# find 3 [], find 3 [1; 2; 3], find 3 [2; 4; 6];;
17- : bool * bool * bool = (false, true, false)
18
19# member 3 [], member 3 [1; 2; 3; 4; 3; 5], member 3 [2; 4; 6];;
20- : int list * int list * int list = ([], [3; 4; 3; 5], [])
21
22# member_last 3 [1; 2; 3; 4; 3; 5], member_last 3 [2; 4; 6];;
23- : int list * int list = ([3; 5], [])
24
25# nb_occ 3 [], nb_occ 3 [1; 2; 3; 4; 3; 5], nb_occ 3 [2; 4; 6];;
26- : int * int * int = (0, 2, 0)

```

```

27
28# nth 3 [1; 2; 3; 4; 3; 5], nth 3 [2; 4; 6];;
29- : int * int = (3, 6)
30
31# max_list [1; 2; 3; 0; 3; 0], max_list [2; 4; 6];;
32- : int * int = (3, 6)
33
34# nb_max [1; 2; 3; 0; 3; 0], nb_max [2; 4; 6];;
35- : int * int = (2, 1)
36
37# average [5.; 8.5; 11.5; 15.];;
38- : float = 10.
39
40# size_in_range 0 0 [], size_in_range 1 3 [0; 0], size_in_range 1 3 [0; 0; 0; 0];;
41- : bool * bool * bool = (true, true, false)
42
43# find_pattern [] [1; 2], find_pattern [1; 1] [1; 2; 1], find_pattern [1; 1] [1; 2; 1; 1];;
44- : bool * bool * bool = (true, false, true)

```

### Exercice 3. (Créer des listes de nombres)

Écrire les fonctions suivantes (et observer leur type).

1. (list\_copy l) qui renvoie la copie de la liste l.
2. (random\_list n max) qui renvoie une liste de n entiers aléatoires strictement inférieurs à max.
3. (reverse l) qui renvoie l'image miroir de la liste l.
4. (flatten\_list l) qui aplatit la liste de listes l.
5. (fibonacci n) qui crée la liste des n premiers nombres de Fibonacci sans utiliser la fonction fib du TP précédent. ★ Essayer de le faire sans avoir besoin de renverser la liste.
6. (without\_duplicates l) qui supprime les doublons dans la liste triée l.
7. ★ (records l) qui calcule la liste des records de la liste l. Un record dans une liste est une valeur strictement plus grande que toutes les précédentes.
8. ★ (look\_and\_say n) qui calcule les n premiers termes de la suite <https://oeis.org/A005150> qui commence par

1, 11, 21, 1211, 111221, 312211, 13112221.

9. (frequences l) qui calcule le nombre d'occurrences de chaque élément de la liste l.

```

1# list_copy [1; 2; 3];;
2- : int list = [1; 2; 3]
3
4# let l = random_list 10 2;;

```

```

5 val l : int list = [0; 0; 1; 1; 0; 1; 1; 0; 0; 0]
6
7 # reverse l;;
8 - : int list = [0; 0; 0; 1; 1; 0; 1; 1; 0; 0]
9
10 # flatten_list [[1; 2]; []; [3; 4; 5]; [6]];;
11 - : int list = [1; 2; 3; 4; 5; 6]
12
13 # fibo 10;;
14 - : int list = [0; 1; 2; 3; 5; 8; 13; 21; 34; 55]
15
16 # without_duplicates [0; 0; 1; 2; 3; 3; 3; 3; 4; 5; 5; 6; 8; 8];;
17 - : int list = [0; 1; 2; 3; 4; 5; 6; 8]
18
19 # records [0; 2; 3; 2; 6; 3; 2; 7; 4; 8; 4];;
20 - : int list = [0; 2; 3; 6; 7; 8]
21
22 # look_and_say 6;;
23 - : int list list = [[3; 1; 2; 2; 1; 1]; [1; 1; 1; 2; 2; 1]; [1; 2; 1; 1]; [2; 1]; [1; 1]; [1]]
24
25 # frequences l;;
26 - : (int * int) list = [(0, 6); (1, 4)]
27
28 # frequences (random_list 10000 5);;
29 - : (int * int) list = [(3, 1977); (1, 2027); (0, 2044); (4, 1980); (2, 1972)]

```

#### Exercice 4. (Tris)

On souhaite trier une liste d'entiers en utilisant la méthode « diviser pour régner ». L'idée est de diviser la liste en deux, de trier chacune des listes obtenues et de les recombinaison.

##### 1. Le tri fusion.

- (a) Écrire une fonction `f_split` qui sépare une liste quelconque en deux listes de tailles à peu près égales.
- (b) Écrire une fonction `f_merge` qui fabrique une liste triée à partir de deux listes triées quelconques.
- (c) Écrire une fonction `fusion_sort` qui trie une liste récursivement en utilisant les fonctions précédentes.

##### 2. Le quick-sort.

- (a) Écrire une fonction `q_split` qui sépare une liste quelconque `l` en trois listes en fonction d'un *pivot* (on peut choisir le premier élément de `l`) :
  - les éléments strictement plus petits que le pivot,
  - les éléments égaux au pivot,
  - les éléments plus grands que le pivot.

- (b) Écrire une fonction `q_merge` qui fabrique une liste triée à partir de trois listes triées obtenues comme ci-dessus.
- (c) Écrire une fonction `quick_sort` qui trie une liste récursivement en utilisant les fonctions précédentes.

```
1# let l = random_list 20 100;;
2val l : int list = [58; 37; 58; 72; 19; 58; 18; 41; 58; 86; 94; 59; 92; 35; 40; 47; 92; 6; 42; 95]
3
4# let l1, l2 = f_split l;;
5val l1 : int list = [58; 58; 19; 18; 58; 94; 92; 40; 92; 42]
6val l2 : int list = [37; 72; 58; 41; 86; 59; 35; 47; 6; 95]
7
8# let l1, l2 = (List.sort compare l1), (List.sort compare l2);;
9val l1 : int list = [18; 19; 40; 42; 58; 58; 58; 92; 92; 94]
10val l2 : int list = [6; 35; 37; 41; 47; 58; 59; 72; 86; 95]
11
12# f_merge l1 l2;;
13- : int list = [6; 18; 19; 35; 37; 40; 41; 42; 47; 58; 58; 58; 58; 59; 72; 86; 92; 92; 94; 95]
14
15# fusion_sort l;;
16- : int list = [6; 18; 19; 35; 37; 40; 41; 42; 47; 58; 58; 58; 58; 59; 72; 86; 92; 92; 94; 95]
17
18# let l1, l2, l3 = q_split l;;
19val l1 : int list = [37; 19; 18; 41; 35; 40; 47; 6; 42]
20val l2 : int list = [58; 58; 58; 58]
21val l3 : int list = [72; 86; 94; 59; 92; 92; 95]
22
23# let l1, l2, l3 = (List.sort compare l1), l2, (List.sort compare l3);;
24val l1 : int list = [6; 18; 19; 35; 37; 40; 41; 42; 47]
25val l2 : int list = [58; 58; 58; 58]
26val l3 : int list = [59; 72; 86; 92; 92; 94; 95]
27
28# q_merge l1 l2 l3;;
29- : int list = [6; 18; 19; 35; 37; 40; 41; 42; 47; 58; 58; 58; 58; 59; 72; 86; 92; 92; 94; 95]
30
31# quick_sort l;;
32- : int list = [6; 18; 19; 35; 37; 40; 41; 42; 47; 58; 58; 58; 58; 59; 72; 86; 92; 92; 94; 95]
```