Série TP n°5
Les listes

Exercice 1
Ecrire la fonction **nbElem** qui permet de calculer le nombre d'éléments d'une liste notée t.

Exercice 2
Ecrire la fonction **somElem** qui permet de calculer la somme des éléments d'une liste notée t.

Exercice 3
Ecrire la fonction **moy** qui permet de calculer la moyenne des éléments d'une liste, notée t, en utilisant les fonctions **nbElem** et **somElem** (développées ci-dessus, exercices 1 et 2).

Exercice 4
Ecrire la fonction **ins_deb** qui permet d'insérer un élément x au début d'une liste notée t.

Exercice 5
Ecrire la fonction **ins_fin** qui permet d'insérer un élément x à la fin d'une liste notée t.

Exercice 6
Ecrire la fonction **sup_deb** qui permet de supprimer un élément au début d'une liste, notée t, supposée non vide.

Exercice 7
Ecrire la fonction **sup_fin** qui permet de supprimer un élément à la fin d'une liste, notée t, supposée non vide.
Ind : Utiliser la fonction **nbElem** (développée ci-dessus, exercice 1).

Exercice 8
Ecrire la fonction **inv** qui permet d'inverser les éléments d'une liste notée t.

Exercice 9

Ecrire la fonction **proj** qui permet de projeter le ième (i>0) élément d'une liste, notée t, supposée non vide.

Exercice 10

Ecrire la fonction e**xist** permet de vérifier si un élément x existe dans une liste notée t. On donnera une réponse logique vrai ou faux.

Exercice 11

Ecrire la fonction **occur** qui permet de calculer le nombre les occurrences (les répétitions) d'un élément x dans une liste notée t.

Exercice 12

1- Ecrire la fonction **map** qui distribue la fonction f sur les éléments d'une liste, notée t, comme suit :

$$map : [a_1; \ldots ; a_n] \rightarrow [(f(a_1); \ldots ; f(a_n)]$$

2- Appliquer la fonction **map** pour élever au carrée les éléments d'une liste d'entiers.

Ind : utiliser la fonction **carre** (voir TP 2 / Ex 3).

3- Appliquer la fonction **map** pour inverser les mots dans une liste de mots.

Ind : utiliser la fonction **inv** (voir TP 4 / Ex 3).

Exercice 13

Ecrire les fonctions permettant de trier une liste, notée t, par ordre croissant en utilisant :

1- La méthode de tri par sélection (la fonction est notée tri_sel);

2- La méthode de tri par insertion (la fonction est notée tri_ins).

Corrigé de la série TP n°5
Les listes

Mr. AMANI Ferhat / Email : amani.f1963@yahoo.fr
Lundi 07 Juin 2019 (revu le Mardi 01 juin 2021)
(Nous vous serions gré de nous signaler les erreurs éventuelles dans le corrigé)

Exercice 1

Ecrire la fonction **nbElem** qui permet de calculer le nombre d'éléments d'une liste notée t.

Solution :

(* TP 5 : Les listes / Ex 1 *)

```
let  rec  nbElem(t) = if (t=[]) then 0 else 1 + nbElem(tl(t)) ;;
```

Après l'exécution de cette fonction, le langage Caml retourne son type :

nbElem : 'a list → int = <fun>

où,

1- La liste t a le type " **'a list** " qui désigne le type liste dont les éléments ont un type quelconque " **'a** " (qui sera fixé lors de leur création) ;
2- La fonction **nbElem** a le type "**int**" (car elle retourne la valeur entière 0).

Application :

On peut tester la fonction nbElem avec les listes suivantes t1 et t2 :

```
1- let t1 = [];;   (* t1 = liste vide *)
2- let nb1 = nbElem(t1);;
3- let t2 = ["Alger"; "Tunis"; "Berlin"];;
4- let nb2 = nbElem(t2);;
```

Les réponses du langage Caml sont :

```
1- t1 : 'a list = []
2- nb1 : int = 0
3- t2 : string list = ["Alger"; "Tunis"; "Berlin"]
4- nb2 : int = 3
```

Exercice 2

Ecrire la fonction **somElem** qui permet de calculer la somme des éléments d'une liste notée t.

Solution :

(* TP 5 : Les listes / Ex 2 *)

let rec somElem(t) = if (t=[]) then 0 else hd(t) + somElem(tl(t)) ;;

Après l'exécution de cette fonction, le langage Caml retourne son type :

somElem : int list → int = <fun>

où,

1- La liste t a le type **"int list"** qui désigne le type liste dont les éléments ont le type **"int"** (car l'opération utilisée **"+"** est de type **"int"**, donc forcément les opérandes (dont hd(t)=élément de début de liste) doivent être de type **"int"**) ;
2- La fonction somElem a le type **"int"** (car elle retourne la valeur entière 0).

Rem : On aurait pu écrire la fonction somElem avec un résultat réel comme suit :

let rec somElem(t) = if (t=[]) then 0.0 else hd(t) +. somElem(tl(t)) ;;

Après l'exécution de cette fonction, le langage Caml retourne son type :

somElem : float list → float = <fun>

Application :

On peut tester la fonction somElem avec les listes suivantes t1 et t2 :

1- let t1 = [];; (* t1 = liste vide *)
2- let s1 = somElem(t1);;
3- let t2 = [2; 3; 6; 7];;
4- let s2 = somElem(t2);;

Les réponses du langage Caml sont :

1- t1 : 'a list = []
2- s1 : int = 0
3- t2 : int list = [2; 3; 6; 7]
4- s2 : int = 18

Exercice 3

Ecrire la fonction **moy** qui permet de calculer la moyenne des éléments d'une liste, notée t, en utilisant les fonctions **nbElem** et **somElem** (développées ci-dessus, exercices 1 et 2).

Solution :

```
(* TP 5 : Les listes / Ex 3 *)

let moy(t) = if (nbElem(t)=0)
                then (-1.0)
                 else float_of_int(somElem(t)) /. float_of_int(nbElem(t)) ;;
```

Rem 1 : la valeur (-1.0) représente un message d'erreur (qui permet aussi d'éviter la division par 0).

Rem 2 : float_of_int(a) est une fonction prédéfinie de conversion de type. Elle convertit le type "**int**" de la variable a vers le type "**float**". Autrement, on aura un résultat erroné de la moyenne.

Rem 3 : "**/.**" **d**ésigne l'opération de division de type réel.

Après l'exécution de cette fonction, le langage Caml retourne son type :

moy : int list → float = <fun>

où,

1- La liste t a le type " **int list** " qui désigne le type liste dont les éléments ont le type "**int**" (car la fonction somElem(t) utilise la liste t qui est de type "**int**", voir exercice 2 ci-dessus) ;
2- La fonction moy a le type "**float**" (car la fonction retourne la valeur réelle (-1.0)).

Application :

On peut tester la fonction moy avec les listes suivantes t1 et t2 :

```
1- let t1 = [];;   (* t1 = liste vide *)
2- let moy1 = moy(t1);;
3- let t2 = [2; 3; 6; 7];;
4- let moy2 = moy(t2);;
```

Les réponses du langage Caml sont :

```
1- t1 : 'a list = []
2- moy1 : float = -1.0
3- t2 : int list = [2; 3; 6; 7]
4- moy2 : float = 4.50
```

Exercice 4

Ecrire la fonction **ins_deb** qui permet d'insérer un élément x au début d'une liste notée t.

Solution :

(* TP 5 : Les listes / Ex 4 *)

```
let  ins_deb(t, x) = x :: t ;;
        (* :: = opérateur d'insertion en début de liste *)

(* ou  solution 2 : let  ins_deb(t, x) = [x] @ t ;; *)
        (* @ = opérateur de concaténation de 2 listes *)
```

Rem : Si on utilise l'opérateur de concaténation de 2 listes **"@",** alors on doit inclure la variable x dans une liste en l'écrivant entre les 2 crochets **"["** et **"]"**.

Après l'exécution de cette fonction, le langage Caml retourne son type :

ins_deb : 'a list * 'a → 'a list = <fun>

où,

1- La liste t a le type **" 'a list "** qui désigne le type liste dont les éléments ont un type quelconque **" 'a "** (qui sera fixé lors de leur création) ;
2- La variable x a le type **" 'a "** qui désigne un type quelconque (qui sera fixé lors de sa création) ;
3- La fonction ins_deb a le type **" 'a list "** qui désigne le type liste dont les éléments ont un type quelconque **" 'a "** (qui sera fixé lors de leur création).

Application :

On peut tester la fonction ins_deb avec les listes suivantes t1, t2, t3 et t4 :

```
1- let t1 = [];;   (* t1 = liste vide *)
2- let t2 = ins_deb(t1, 2);;
3- let t3 = [3; 5; 7];;
4- let t4 = ins_deb(t3, 2);;
```

Les réponses du langage Caml sont :

```
1- t1 : 'a list = []
2- t2 : int list = [2]
3- t3 : int list = [3; 5; 7]
4- t4 : int list = [2; 3; 5; 7]
```

Exercice 5

Ecrire la fonction **ins_fin** qui permet d'insérer un élément x à la fin d'une liste notée t.

Solution :

(* TP 5 : Les listes / Ex 5 *)

let ins_fin(t, x) = t @ [x] ;;

Après l'exécution de cette fonction, le langage Caml retourne son type :

ins_fin : 'a list * 'a → 'a list = <fun>

où,

1- La liste t a le type " **'a list** " qui désigne le type liste dont les éléments ont un type quelconque " **'a** " (qui sera fixé lors de leur création) ;
2- La variable x a le type " **'a** " qui désigne un type quelconque (qui sera fixé lors de sa création) ;
3- La fonction ins_fin a le type " **'a list** " qui désigne le type liste dont les éléments ont un type quelconque " **'a** " (qui sera fixé lors de leur création).

Application :

On peut tester la fonction ins_fin avec les listes suivantes t1, t2, t3 et t4 :

```
1- let t1 = [];;   (* t1 est une liste vide *)
2- let t2 = ins_fin(t1, 2);;
3- let t3 = [3; 5; 7];;
4- let t4 = ins_fin(t3, 2);;
```

Les réponses du langage Caml sont :

```
1- t1 : 'a list = []
2- t2 : int list = [2]
3- t3 : int list = [3; 5; 7]
4- t4 : int list = [3; 5; 7; 2]
```

Exercice 6

Ecrire la fonction **sup_deb** qui permet de supprimer un élément au début d'une liste, notée t, supposée non vide.

Solution :

(* TP 5 : Les listes / Ex 6 *)
(* On a l'hypothèse : la liste est non vide *)

let sup_deb(t) = tl(t) ;;

Après l'exécution de cette fonction, le langage Caml retourne son type :

sup_deb : 'a list → 'a list = <fun>

où,

1- La liste t a le type " **'a list** " qui désigne le type liste dont les éléments ont un type quelconque " **'a** " (qui sera fixé lors de leur création) ;
2- La fonction sup_deb a le type " **'a list** " qui désigne le type liste dont les éléments ont un type quelconque " **'a** " (qui sera fixé lors de leur création).

Application :

On peut tester la fonction sup_deb avec les listes suivantes t1, t2, t3 et t4 :

1- let t1 = [3];;
2- let t2 = sup_deb(t1);;
3- let t3 = [3; 5; 7];;
4- let t4 = sup_deb(t3);;

Les réponses du langage Caml sont :

1- t1 : int list = [3]
2- l2 : int list = []
3- t3 : int list = [3; 5; 7]
4- t4 : int list = [5; 7]

Exercice 7

Ecrire la fonction **sup_fin** qui permet de supprimer un élément à la fin d'une liste, notée t, supposée non vide.

Ind : Utiliser la fonction **nbElem** (développée ci-dessus, exercice 1).

Solution :

(* TP 5 : Les listes / Ex 7 *)

(* On a l'hypothèse : la liste est non vide *)

```
let  rec  sup_fin(t) = if (nbElem(t)=1) then [] else [hd(t)] @ sup_fin(tl(t)) ;;
```

Après l'exécution de cette fonction, le langage Caml retourne son type :

sup_fin : 'a list → 'a list = <fun>

où,

1- La liste t a le type " **'a list** " qui désigne le type liste dont les éléments ont un type quelconque " **'a** " (qui sera fixé lors de leur création) ;
2- La fonction sup_fin a le type " **'a list** " qui désigne le type liste dont les éléments ont un type quelconque " **'a** " (qui sera fixé lors de leur création).

Application :

On peut tester la fonction sup_fin avec les listes suivantes t1, t2, t3 et t4 :

```
1- let t1 = [3];;
2- let t2 = sup_fin(t1);;
3- let t3 = [3; 5; 7];;
4- let t4 = sup_fin(t3);;
```

Les réponses du langage Caml sont :

```
1- t1 : int list = [3]
2- t2 : int list = []
3- t3 : int list = [3; 5; 7]
4- t4 : int list = [3; 5]
```

Exercice 8

Ecrire la fonction **inv** qui permet d'inverser les éléments d'une liste notée t.

Solution :

(* TP 5 : Les listes / Ex 8 *)

```
let  rec  inv(t) = if (t=[]) then [] else inv(tl(t)) @ [hd(t)] ;;
```

Après l'exécution de cette fonction, le langage Caml retourne son type :

inv : 'a list → 'a list = <fun>

où,

1- La liste t a le type " **'a list** " qui désigne le type liste dont les éléments ont un type quelconque " **'a** " (qui sera fixé lors de leur création) ;
2- La fonction inv a le type " **'a list** " qui désigne le type liste dont les éléments ont un type quelconque " **'a** " (qui sera fixé lors de leur création).

Application :

On peut tester la fonction inv avec les listes suivantes t1, t2, t3, t4, t5 et t6 :

```
1- let t1 = [];;   (* t1 est une liste vide *)
2- let t2 = inv(t1);;
3- let t3 = [3];;
4- let t4 = inv(t3);;
5- let t5 = [3; 5; 7; 9];;
6- let t6 = inv(t5);;
```

Les réponses du langage Caml sont :

```
1- t1 : 'a list = []
2- t2 : 'a list = []
3- t3 : int list = [3]
4- t4 : int list = [3]
5- t5 : int list = [3; 5; 7; 9]
6- t6 : int list = [9; 7; 5; 3]
```

Exercice 9

Ecrire la fonction **proj** qui permet de projeter le ième (i>0) élément d'une liste, notée t, supposée non vide.

Solution :

(* TP 5 : Les listes / Ex 9 *)
(* On a l'hypothèse : la liste est non vide *)

let rec proj(t, i) = if (i=1) then hd(t) else proj(tl(t), (i-1)) ;;

Après l'exécution de cette fonction, le langage Caml retourne son type :

proj : 'a list * int → 'a = <fun>

où,

1- La liste t a le type " **'a list** " qui désigne le type liste dont les éléments ont un type quelconque " **'a** " (qui sera fixé lors de leur création) ;
2- La variable i a le type " **int** "(car i est comparé avec la valeur entière 1) :
3- La fonction ins_fin a le type " **'a** " qui désigne le type liste dont les éléments ont un type quelconque " **'a** " (qui sera fixé lors de leur création).

Application :

On peut tester la fonction proj avec les listes suivantes t1et t2 :

1- let t1 = [3];;
2- let x1 = proj(t1, 1);;
3- let t2 = [3; 5; 7];;
4- let x2 = proj(t2, 3);;

Les réponses du langage Caml sont :

1- t1 : int list = [3]
2- x1 : int = 3
3- t2 : int list = [3; 5; 7]
4- x2 : int = 7

Exercice 10

Ecrire la fonction **exist** permet de vérifier si un élément x existe dans une liste notée t. On donnera une réponse logique vrai ou faux.

Solution :

(* TP 5 : Les listes / Ex 10 *)

```
let rec exist(t, x) = if (t=[]) then false
                      else if (x=hd(t)) then true else exist((tl(t), x)) ;;
```

Après l'exécution de cette fonction, le langage Caml retourne son type :

exist : 'a list * 'a → bool = <fun>

où,

1- La liste t a le type " **'a list** " qui désigne le type liste dont les éléments ont un type quelconque " **'a** " (qui sera fixé lors de leur création) ;
2- La variable x a le type " **'a** " qui désigne un type quelconque (qui sera fixé lors de sa création) ;
3- La fonction exist a le type " **bool** " qui désigne le type booléen (car elle retourne la valeur logique false).

Application :

On peut tester la fonction exist avec les listes suivantes t1, t2, et t3 :

```
1- let t1 = [];;   (* t1 est une liste vide *)
2- let x1 = exist(t1, 2);;
3- let t2 = [3; 5; 7];;
4- let x2 = exist(t2, 2);;
5- let t3 = [3; 5; 7];;
6- let x3 = exist(t3, 7);;
```

Les réponses du langage Caml sont :

```
1- t1 : 'a list = []
2- x1 : bool =false
3- t2 : int list = [3; 5; 7]
4- x2 : bool =false
5- t3 : int list = [3; 5; 7]
6- x3 : bool =true
```

Exercice 11

Ecrire la fonction **occur** qui permet de calculer le nombre les occurrences (les répétitions) d'un élément x dans une liste notée t.

Solution :

(* TP 5 : Les listes / Ex 11 *)

```
let rec occur(t, x) = if (t=[]) then 0
                      else if (x=hd(t)) then (1 + occur(tl(t), x))
                           else (0 + occur(tl(t), x)) ;;
```

//$0 + a = a \ \forall a$ (ajouter 0 ne change rien)

Après l'exécution de cette fonction, le langage Caml retourne son type :

occur : 'a list * 'a → int = <fun>

où,

1- La liste t a le type " **'a list** " qui désigne le type liste dont les éléments ont un type quelconque " **'a** " (qui sera fixé lors de leur création) ;
2- La variable x a le type " **'a** " qui désigne un type quelconque (qui sera fixé lors de sa création) ;
3- La fonction occur a le type " **int** " (car elle retourne la valeur entière 0).

Application :

On peut tester la fonction occur avec les listes suivantes t1, t2, et t3 :

```
1- let t1 = [];;   (* t1 est une liste vide *)
2- let x1 = occur(t1, 2);;
3- let t2 = [3; 5; 7];;
4- let x2 = occur(t2, 2);;
5- let t3 = [3; 5; 7; 7];;
6- let x3 = occur(t3, 7);;
```

Les réponses du langage Caml sont :

```
1- t1 : 'a list = []
2- nb : int =0
3- t2 : int list = [3; 5; 7]
4- nb2 : int =0
5- t3 : int list = [3; 5; 7 ; 7]
6- nb3 : int =2
```

Exercice 12

1- Ecrire la fonction **map** qui distribue la fonction f sur les éléments d'une liste, notée t, comme suit :

$$map : [a_1; \ldots ; a_n] \rightarrow [(f(a_1); \ldots ; f(a_n)]$$

Solution :

(* TP5 : Les listes / Ex 12 / Q1 *)

let rec map(t, f) = if (t=[]) then [] else f(hd(t)) :: map(tl(t), f) ;;

Après l'exécution de cette fonction, le langage Caml retourne son type :

map : 'a list * ('a → 'b) → 'b list = <fun>

où,

1- La liste t a le type " **'a list** " qui désigne le type liste dont les éléments ont un type quelconque " **'a** " (qui sera fixé lors de leur création) ;
2- L'ensemble de départ de la fonction f a le type " **'a** " qui désigne un type quelconque (qui sera fixé lors de sa création) et la fonction f a le type " **'b** " qui désigne un type quelconque (qui sera fixé lors de sa création) ;
3- La fonction map a le type " **'b list** " qui désigne le type liste dont les éléments ont un type quelconque " **'b** " (qui est celui de la fonction f).

2- Appliquer la fonction **map** pour élever au carrée les éléments d'une liste d'entiers.

Ind : utiliser la fonction **carre** (voir TP2/Q7).

Solution :

(* TP5 : Les listes / Ex 12 / Q2 *)

La fonction carre est comme suit (voir TP2/Q7 *) :

let carre(x) = x * x ;;

On teste l'application de la fonction map pour la fonction carre avec les listes suivantes t1, t2, t3 et t4 :

1- let t1 = [];; (* t1 est une liste vide *)
2- let t2 = map(t1, carre);;
3- let t3 = [3; 5];;
4- let t4 = map(t3, carre);;

Les réponses du langage Caml sont :

1- t1 : 'a list = []
2- t2 : int list = []
3- t3 : int list = [3; 5]

4- t4 : int list = [9; 25]

3- Appliquer la fonction **map** pour inverser les mots dans une liste de mots.
Ind : utiliser la fonction **inv_mot** (voir TP4/Ex3/Q1).

Solution :

(* TP5 : Les listes / Ex 12 / Q3 *)
La fonction inv_mot est comme suit (voir TP4/Ex3/Q1 *) :

```
let long(ch) = string_length(ch) ;;    (* long(ch) = longueur  de la chaîne ch *)
let rec invmot(ch) = if ch=”” 
    then ""
    else (sub_string ch (long(ch) - 1) 1) ^
                                (invmot( (sub_string  ch 0 (long(ch) - 1)))) ;;
```

On teste l'application de la fonction map pour la fonction invmot avec les listes suivantes t1, t2, t3 et t4 :

```
1- let t1 = [];;   (* t1 est une liste vide *)
2- let t2 = map(t1, invmot);;
3- let t3 = ["ALGER"; "ORAN"];;
4- let t4 = map(t3, invmot);;
```

Les réponses du langage Caml sont :

```
1- t1 : 'a list = []
2- t2 : string  list = []
3- t3 : string list = ["ALGER"; "ORAN"]
4- t4 : string list = ["REGLA"; "NARO"]
```