Nom	
Prénom	
Groupe	
Prof TD	

Note	18 /20

Algorithmique (Caml)
Examen B1 #1

INFO-SUP S1

EPITA

30 Oct. 2023

6,5/6,5
5,5/5,5
6/8

Remarques (à lire !) :

- ☐ Vous devez répondre directement **sur ce sujet.**
  - — Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées.**
  - — Aucune réponse au crayon de papier ou au stylo rouge ne sera corrigée.
- ☐ **CAML :**
  - — Tout code CAML non indenté ne sera pas corrigé.
  - — En l'absence d'indication dans l'énoncé, les seules fonctions que vous pouvez utiliser sont `failwith` et `invalid_arg` (aucune autre fonction prédéfinie de CAML).
  - — Une seule version doit être présentée pour chaque fonction à écrire.
  - — **Tout code CAML doit être suivi du résultat de son évaluation (fait partie de la note) : la réponse de CAML .**
- ☐ La présentation est notée.

Exercice 1 (Insertion multiple – *6,5 points*)

Écrire la fonction `insert_mult n x lst` qui prend en paramètres :

- un entier `n`
- un élément `x`
- une liste `lst`

et qui insère l'élément `x` dans la liste `lst` après chaque groupe de `n` éléments.
La fonction devra déclencher une exception `Invalid_argument` si le paramètre `n` est négatif ou nul.

cas 1
```
# insert_mult 0 1 [1; 2; 3; 4; 5];;
Exception: Invalid_argument "insert_mult: n must be > 0".
```
cas 2
```
# insert_mult 4 42 [10; 20; 30; 40];;
- : int list = [10; 20; 30; 40; 42]
```
cas 3
```
# insert_mult 2 0 [1; 2; 3; 4; 5];;
- : int list = [1; 2; 0; 3; 4; 0; 5]
```
cas 4
```
# insert_mult 3 'x' ['a'; 'b'; 'c'; 'd'; 'e'; 'f'; 'g'];;
- : char list = ['a'; 'b'; 'c'; 'x'; 'd'; 'e'; 'f'; 'x'; 'g']
```
cas 5
```
# insert_mult 1 42 [];;
- : int list = []
```
cas 6
```
# insert_mult 1 42 [10];;
- : int list = [10; 42]
```

Fonction CAML :

```
let insert_mult n x lst =
  if n <= 0 then invalid_arg "insert_mult: n must be > 0"
  else let rec aux count = function
    | list when count = 0 -> x :: aux n list
    | [] -> []
    | h::t -> h :: aux (count-1) t
  in aux n lst ;;
// val insert_mult : int -> 'a -> 'a list -> 'a list = <fun>
```

Exercice 2 (Suppression doublons – *5,5 points*)

1. Écrire la fonction `remove_x x lst` qui prend en paramètres : 2pts
   — un élément `x`
   — une liste `lst`
   et qui supprime toutes les occurrences de l'élément `x` dans la liste `lst`.

cas 1
cas 2
cas 3
cas 4
cas 5

```
# remove_x 1 [];;
- : int list = []
# remove_x "apple" ["banana"; "apple"; "cherry"; "date"; "apple"];;
- : string list = ["banana"; "cherry"; "date"]
# remove_x 5 [5; 5; 2; 3; 5; 1];;
- : int list = [2; 3; 1]
# remove_x true [true; false; true; false; true];;
- : bool list = [false; false]
# remove_x 'a' ['a'; 'b'; 'a'; 'c'; 'd'];;
- : char list = ['b'; 'c'; 'd']
```

Fonction CAML :

```
let rec remove_x x lst = match lst with
  | [] -> []
  | h::t when h = x -> remove_x x t
  | h::t -> h :: remove_x x t;;
(* val remove_x : 'a -> 'a list -> 'a list = <fun> *)
```

2. Écrire la fonction `remove_duplicates lst` qui prend en paramètre : 3pts
   — une liste `lst`
   et qui supprime tous les doublons de la liste `lst`.
   La fonction **doit** utiliser la fonction précédente `remove_x x lst`.

```
1  # remove_duplicates [3; 1; 2; 2; 3; 1; 4; 5; 5];;
   - : int list = [3; 1; 2; 4; 5]
2  # remove_duplicates ["cherry"; "banana"; "apple"; "apple"; "banana"; "cherry"];;
   - : string list = ["cherry"; "banana"; "apple"]
3  # remove_duplicates [42; 13; 42; 7; 7; 13; 42; 42];;
   - : int list = [42; 13; 7]
4  # remove_duplicates ["dog"; "cat"; "dog"; "rat"; "rat"; "bat"];;
   - : string list = ["dog"; "cat"; "rat"; "bat"]
5. # remove_duplicates [true; true; false; true; false; true; false];;
   - : bool list = [true; false]
```

Fonction CAML :

```
let rec remove_duplicates lst = match lst with
  | [] -> []
  | h::[] -> h::[]
  | h::t -> h:: remove_duplicates (remove_x h t);;
//val remove_duplicates : 'a list -> 'a list = <fun>
```

Exercice 3 (Split – *8 points*)

Écrire la fonction `split sep lst` qui prend en paramètres :

- un prédicat à un paramètre `sep`
- une liste `lst`

et qui coupe la liste `lst` en sous-listes dès que le prédicat `sep` est vérifié sur un élément de `lst`. Les éléments de `lst` qui vérifient le prédicat sep ne font partie d'aucune sous-liste.

Si les sous-listes ne sont pas dans l'ordre initial, l'exercice sera noté sur un peu moins. Parfois il vaut mieux moins de points que pas de points.

```
# split (function x -> x = ',') [];;
- : char list list = []
# split (function x -> x = 3) [1; 2; 3; 4; 5];;
- : int list list = [[1; 2]; [4; 5]]
# split (function x -> x = 20) [10; 20; 30; 40; 50];;
- : int list list = [[10]; [30; 40; 50]]
# split (function x -> x = 0) [0; 1; 0; 2; 0; 3; 0; 4];;
- : int list list = [[]; [1]; [2]; [3]; [4]]
```

Fonction CAML :

```
let split sep lst =
  if (lst = []) then []
  else let rec rev_list final = function
         | [] -> final
         | e::l -> rev_list e::final l
  in let rec aux lst temp_lst final_lst = match lst with
       | [] -> (rev_list [] temp_lst) :: final_lst
       | h::t when sep h -> aux t [] (rev_list [] temp_lst)::final_lst
       | h::t -> aux t h:: temp_lst final_lst
  in
     aux lst [] [];;
//val split : ('a->bool)-> 'a list -> 'a list list = <fun>
```