TP 2 Programmation fonctionnelle

1. Écrire la fonction `produit l` qui renvoie le produit des élément de la liste l. Optimisez votre fonction pour qu'elle renvoie 0 dès qu'un élément égal à 0 est vu dans la liste.

   Solution.

```
let rec produit l =
  match l with
  | [] -> 1
  | 0::tl -> 0
  | hd::tl -> hd * produit tl
```

   □

2. Écrire la fonction `count l a` comptant le nombre d'apparitions de a dans la liste l

   `count [1;2;3;4;3;5;6] 3 -->  2`

   Solution.

```
let rec count l a =
  match l with
  | [] -> 0
  | hd::tl -> (if hd = a then 1 else 0) + count tl a
```

   □

3. Écrire la fonction `is_equal l` qui est vraie si tous les éléments de la liste sont les mêmes.

   `is_equal [1;1;1;1;1] --> true`

   Solution.

```
let rec is_equal l =
  match l with
  | a::b::tl -> a=b && is_equal (b::tl)
  | _ -> true
```

   □

4. Écrire la fonction `append` qui concatene deux listes

   Solution.

```
let rec append ls1 ls2 =
  match ls1 with
  | [] -> ls2
  | hd::tl -> hd :: (append tl ls2)
```

   □

5. Écrire la fonction `invert l` qui inverse une liste invert

   `[10;20;30] -> [30;20;10]`

Solution.

```
let rec invert l =
  match l with
  | [] -> []
  | hd::tl -> append (invert tl) [hd]
```

Variant plus efficace :

```
let rec invertacc l acc =
  match l with
  | [] -> acc
  | hd::tl -> invertacc tl (hd::acc)

let invert2 l = invertacc l []
```

□

6. Écrire la fonction `clean l` qui supprime les éléments en double.

   `clean [1;1;4;5;5;4] --> [1;4;5]`

Solution.

```
let rec contain l n =
  match l with
  | [] -> false
  | hd::tl -> hd=n || contain tl n

let rec cleanacc l acc=
  match l with
  | [] -> []
  | hd::tl ->
      if contain acc hd then clean tl acc
      else hd::(clean tl (hd::acc))

let clean l = cleanacc l []
```

□

7. Écrire la fonction `slice : 'a list -> int -> int -> 'a list` telle que `slice l a b` renvoie la sous-liste de l commençant à l'indice a inclus et terminant à l'indice b exclu. On assume $0 \leq a \leq b$.

   `slice [0;1;2;3;4;5] 2 4 --> [2;3]`

Solution.

```
let rec slice l a b =
  match l,a,b with
  | [],_,_ -> []
  | hd::tl,0,0 -> []
  | hd::tl,0,y -> hd::(slice tl 0 (y-1))
  | hd::tl,x,y -> slice tl (x-1) (y-1)
```

□