Feuille d'exercices

Exercice 1.

1. Écrire une fonction non récursive terminale qui prend deux entiers x et y et calcule x^y sans utiliser $**$ ou les floats.
2. Réécrire cette fonction de manière récursive terminale, toujours sans utiliser $**$ ni les floats.

Exercice 2. Écrire une fonction qui calcule le reste et le quotient de la division entière de deux entiers (sans utiliser la division et l'opération modulo).

Exercice 3. Soit la fonction `enum` telle que `enum n` retourne la liste des entiers `[0; 1; ...; n]`

```
let enum = fun n ->
  let rec aux = fun i ->
    if i <= n then i :: aux (i+1) else []
in if n < 0 then [] else aux 0 ;;
```

Modifier la fonction pour la rendre récursive terminale.

Exercice 4. On propose la fonction `renverse'` pour renverser une liste.

Dérouler l'appel `renverse [1; 2; 3; 4]` et l'appel `renverse' [1; 2; 3; 4]`. Pour quelles raisons la fonction `renverse'` est-elle plus efficace que la fonction `renverse` ?

```
let renverse' = fun l ->
    let rec aux = fun u l ->
    match l with
      | [ ] -> u
      | h :: t -> aux (h :: u) t
in aux [ ] l ;;
```

```
let rec renverse = fun l ->
    match l with
      | [ ] -> []
      | h :: t -> (renverse t) @ [h] ;;
```

Exercice 5. Écrire des fonctions récursives terminales qui rendent une liste dans laquelle

1. une occurrence de l'élément donné `e` est supprimée
2. toutes les occurrences de `e` sont supprimées.

À chaque fois, on renverra la liste initiale si l'élément à supprimer n'apparaît pas dans la liste.

Exercice 6. On représente l'écriture binaire d'un entier par une liste de 0 et de 1. Écrire une fonction récursive terminale qui convertit

1. un entier en sa représentation binaire,
2. une écriture binaire en l'entier correspondant.

Exercice 7. Soit la fonction `reduction` définie par :

```
let rec reduction = fun operator neutral l ->
  match l with
    | [] -> neutral
    | h::t -> operator (reduction operator neutral t) h ;;
```

Réécrire `reduction` pour qu'elle soit récursive terminale.