TP de Logique. Caml

Exercice 1.— Ecrire une fonction `sum` qui calcule la somme de deux entiers

Exercice 2.— Ecrire une fonction `incr` qui incrémente de 1 un entier en utilisant la fonction `sum`. Comparer avec `let g = sum 1`.

Exercice 3.— Ecrire la fonction `fact`.

Exercice 4.— Ecrire une fonction `comp` telle que `comp f g` soit la composée des fonctions `f` et `g` notée habituellement $f \circ g$. Observer le type donné.

Exercice 5.— Ecrire la fonction `exp` qui calcule l'exponentielle d'une fonction. On devra avoir `exp f n x` vaut `f(f ... (f x)...)` (n fois).

Exercice 6.— Ecrire une fonction `length` qui calcule la longueur d'une liste.

Exercice 7.— Ecrire une fonction `gen_inverse` telle que `gen_inverse n = [n;n-1;...;2;1]`.
Ecrire une fonction `gen` telle que `gen n = [1;2;...;n]`. Indication : on peut utiliser une fonction auxiliaire `gen_aux k n`.

Exercice 8.— Ecrire la fonction `map` telle que `map f [1;2;3;4] = [f 1; f 2; f 3; f 4]`. Essayer `map length`, `map gen (gen 5)`, `map length (gen 5)`, `comp (map gen) gen`.

Exercice 9.— Ecrire une fonction `split` qui sépare une liste en deux listes de longueurs égales (à 1 près). On ne tiendra pas compte de l'ordre des éléments.

Exercice 10.— Les deux fonctions suivantes `rev_app` et `rev_accu` renversent une liste.

```
#let rec rev_app  = function
          [] -> []
      | x::l ->  (rev_app l) @ [x];;
rev_app : 'a list -> 'a list = <fun>

#let rec rev l accu =
    match l with
        [] -> accu
    | x::reste -> rev reste (x::accu);;
rev : 'a list -> 'a list -> 'a list = <fun>
#let rev_accu l = rev l [];;
rev_accu : 'a list -> 'a list = <fun>
```

Fabriquer une grosse liste avec `comp gen fact`. Comparer le temps d'exécution des deux algorithmes.

Exercice 11.— Ecrire une fonction `merge` qui fusionne deux listes triées.

Exercice 12.— Ecrire une fonction `tri` qui trie une liste par le tri fusion.