Corrigé du contrôle : les listes Caml

Question 1.

```
let rec premiers n = function
  | _ when n = 0 -> []
  | []           -> []
  | t::q         -> t::(premiers (n-1) q) ;;
```

Question 2. La fonction **flatten** est de type *'a list list -> 'a list* et réalise la concaténation des listes présentes dans la liste passée en paramètre. Par exemple, **flatten** `[[1; 2]; [3]; []; [4; 5]]` renvoie la liste `[1; 2; 3; 4; 5]`. La version avec itérateur est la suivante :

```
let flatten lst = list_it (prefix @) lst [] ;;
```

Question 3. Deux solutions sont possibles :

```
let rec itere n x f = match n with
  | 0 -> [x]
  | _ -> x::(itere (n-1) (f x) f) ;;
```

```
let rec itere n x f = match n with
  | 0 -> [x]
  | _ -> x::(map f (itere (n-1) x f)) ;;
```

Question 4.

a) On définit la fonction :

```
let rec compose = function
  | []   -> (function x -> x)
  | f::q -> (function x -> f (compose q x)) ;;
```

ou si on préfère un itérateur :

```
let compose2 lst = list_it (fun a b x -> a (b x)) lst (function x -> x) ;;
```

b) Même chose ici :

```
let rec compose_gauche = function
  | []   -> (function x -> x)
  | f::q -> (function x -> compose_gauche q (f x)) ;;
```

ou avec itérateur :

```
let compose_gauche2 lst = it_list (fun a b x -> b (a x)) (function x -> x) lst ;;
```

On acceptera aussi la version « petit joueur » suivante (honte à vous si vous l'avez choisie) :

```
let compose_gauche lst = compose (rev lst) ;;
```

Question 5. Si $[y_1;\cdots;y_p]$ est une sous-liste de $[x_2;\cdots;x_n]$, les sous-listes de $[x_1;\cdots;x_n]$ sont de la forme $[x_1;y_1;\cdots;y_p]$ ou $[y_1;\cdots;y_p]$ suivant qu'ils contiennent ou pas x_1. On en déduit la fonction :

```
let rec parts = function
  | []   -> [[]]
  | t::q -> let s = parts q in (map (function l -> t::l) s) @ s ;;
```

Question 6.

```
let rec somme n = function
  | []   -> n = 0
  | t::q -> somme (n - t) q || somme n q ;;
```