Corrigé de la feuille d'exercices n°5.

1.

```
#let f (x,y) = sin(x) *. cos(y);;
f : float * float -> float = <fun>
```

Dans cette première version, f est une fonction de $\mathbb{R}^2$ dans $\mathbb{R}$.

```
#let f x y = sin(x) *. cos(y);;
f : float -> float -> float = <fun>
```

dans cette version, f est une fonction de $\mathbb{R}$ dans $\mathcal{F}(\mathbb{R},\mathbb{R})$. Pour s'en convaincre :

```
#let g = f(2.5);;
g : float -> float = <fun>
#g(6.4);;
- : float = 0.594393507821
f 2.5 6.4;;
#- : float = 0.594393507821
```

```
#let h (x, y, z) = (x+y) * z;;
h : int * int * int -> int = <fun>
```

Ici, h est une fonction de $\mathbb{N}^3$ dans $\mathbb{N}$.

```
#let h x y z = (x+y) * z;;
h : int -> int -> int -> int = <fun>
```

Ici h est une fonction de $\mathbb{N}$ dans $\mathcal{F}(\mathbb{N},\mathcal{F}(\mathbb{N},\mathbb{N}))$

2.

```
#let abs_int = function
  n when n < 0 -> - n
| n            -> n;;
abs_int : int -> int = <fun>

#let abs_float = function
  n when n < 0. -> -. n
| n             -> n;;
abs_float : float -> float = <fun>
```

3.

```
#let sgn_int = function
  n when n < 0 -> -1
| n when n = 0 -> 0
| n            -> 1;;
sgn_int : int -> int = <fun>

#let sgn_float = function
  x when x < 0. -> -1.
| x when x = 0. -> 0.
| x             -> 1.;;
sgn_float : float -> float = <fun>
```

4.

```
#let rec somme = function
  n when n <= 0 -> 0
| n             -> n + somme (n-1);;
somme : int -> int = <fun>
#somme 10;;
- : int = 55
```

5.

```
#let rec somcarre = function
 n when n < 0 -> 0
| n           -> n*n + somcarre (n-1);;
somcarre : int -> int = <fun>
```

6.

```
#let rec binomial n = function
  p when n < 0 or p < 0 or p > n -> failwith "Erreur"
| p when n=0 or p=0 or p = n -> 1
| p -> binomial (n-1) (p-1) + binomial (n-1) p;;
binomial : int -> int -> int = <fun>
#binomial 10 4;;
- : int = 210
```

7.

```
#let rec carre = function
 0 -> 0
| n when n < 0 -> carre (-n)
| n -> carre (n-1) + 2*n - 1;;
carre : int -> int = <fun>
```

8.

```
#let compose f n = function
  x when (n <= 0) ->
       failwith "Erreur : exposant négatif ou nul dans compose"
| x when (n = 1)  -> f(x)
| x  -> f(compose f (n-1) x);;
compose : ('a -> 'a) -> int -> 'a -> 'a = <fun>
#compose (function x -> x*.x) 2 5.;;
- : float = 625.0
```

9. On balaye le tableau en repérant les plus petit et plus grand éléments du tableau. Il suffit alors de calculer la différence entre les deux pour trouver l'amplitude.

```
#let amplitude t =
let n = vect_length t  and min = ref t.(0) and max = ref t.(0) in
for i = 0 to n-1 do
   if t.(i) < !min then min := t.(i);
   if t.(i) > !max then max := t.(i);
done;
!max - ! min;;

amplitude : int vect -> int = <fun>
```

10. Dans la variable `imin` on stocke l'indice courant où le minimum est atteint et dans la variable `croiss` le taux de croissance courant le plus élevé. Quand on fait varier i, le taux de croissance le plus élevé se fera toujours à partir du point le plus bas entre les indices 0 et i.

```
#let croissance_max t =
let n = vect_length t and croiss = ref 0
and imin = ref 0 in
 for i = 1 to n-1 do
    if t.(i) - t.(!imin) > !croiss then croiss := t.(i) - t.(!imin);
    if t.(i) < t.(!imin) then imin := i;
done;
!croiss;;
croissance_max : int vect -> int = <fun>
```

11.

```
let premier n =
let t = make_vect (n+1) 1 in
let m = int_of_float(sqrt(float_of_int n)) in
let nb_prem = ref 0 in
for i = 2 to m do
   if t.(i) = 1 then
      for j = i to (n/i) do
         t.(i*j) <- 0
      done;
done;
(* On compte le nombre de nombres premiers *)
for i = 2 to n do
   if t.(i) = 1 then incr nb_prem
done;
(* On crée le tableau des nombres premiers *)
let prem = make_vect (!nb_prem) 0 and j = ref 0 in
for i = 2 to n do
  if t.(i) = 1 then (prem.(!j) <- i; incr j)
done;
prem;; (* on retourne le tableau des nombres premiers *)
```

12.

```
let ecart_type a =
  let moyenne a =
   let s = ref 0. and n = vect_length a in
     for i = 0 to n-1 do s := !s +. a.(i) done;
   !s /. (float_of_int n) in
 let m = moyenne a and n = vect_length a in
 let e = ref 0. in
```

```
   for i = 0 to n-1 do e := !e +. (a.(i) -. m)*.(a.(i) -. m) done;
 sqrt(!e /. (float_of_int n));;
```

13. Ici, on est confronté au problème de débordement dans les calculs, le nombre cherché étant trop grand pour le type entier standard de `caml`. Nous utilisons donc ici la bibliothèque `num` de fonctions `caml` permettant le calcul avec des entiers arbitrairement grands.

```
##open "num";;

let savantfou () =
let n = ref (num_of_int 101)
and dix = num_of_int 10
and zero = num_of_int 0
and un = num_of_int 1
and d = num_of_int 89 in
while (mod_num !n d) <> zero do n := ((!n -/ un) */ dix) +/ un done;
print_num !n;;
savantfou : unit -> unit = <fun>
#savantfou();;
1000000000000000000001- : unit = ()
```

14.

```
#let policiers () =
let nb = ref 0 in
 for a = 0 to 8 do
    for b = 0 to (8 - a) do
        for c = 0 to (8 - (a+b)) do
           for  d = 0 to (8- (a+b+c)) do
              nb := !nb + (9- (a+b+c+d))
           done;
        done;
    done;
 done;
 !nb;;
policiers : unit -> int = <fun>
#policiers();;
- : int = 1287
```