Corrigé de la feuille d'exercices n°2.

Exercice n°1 :

```
#(1.5,("3",(4,5)));;
- : float * (string * (int * int)) = 1.5, ("3", (4, 5))
#((1,2),(3,5));;
- : (int * int) * (int * int) = (1, 2), (3, 5)
#[|2,3.5;4,5.2;6,7.5|];;
- : (int * float) vect = [|2, 3.5; 4, 5.2; 6, 7.5|]
#([|'a';'b'|],[|[||];[|1,2,3|]|]);;
- : char vect * (int * int * int) vect vect = [|'a'; 'b'|], [|[||]; [|1, 2, 3|]|]
```

Exercice n°2 :

```
#(5,3.4,"toto");;
- : int * float * string = 5, 3.4, "toto"
#(4,"caml"),(2.7,"ballon");;
- : (int * string) * (float * string) = (4, "caml"), (2.7, "ballon")
#(-1,(2.71827,"exp1")),456;;
- : (int * (float * string)) * int = (-1, (2.71827, "exp1")), 456
#('t',5),("bidule",45.12456);;
- : (char * int) * (string * float) = ('t', 5), ("bidule", 45.12456)
```

Exercice n°3 :

```
#let min (x,y,z) =
if x < y then if x < z then x
                       else z
         else if y < z then y
                       else z;;
#min : 'a * 'a * 'a -> 'a = <fun>
```

Exercice n°4 :
Dans la première version du la fonction **somfct** les fonctions f et g sont des applications à valeurs dans $\mathbb{R}$ sans précision sur l'ensemble de départ :

```
#let somfct f g = function
x -> f(x) +. g(x);;
somfct : ('a -> float) -> ('a -> float) -> 'a -> float = <fun>
```

Dans la deuxième version, on force le type de x pour faire agir notre fonction **somfct** seulement sur les applications de $\mathbb{R}$ dans $\mathbb{R}$.

```
#let somfct f g = function
(x:float) -> f(x) +. g(x);;
somfct : (float -> float) -> (float -> float) -> float -> float = <fun>
```

```
#let prodfct f g = function
(x:float) -> f(x) *. g(x);;
prodfct : (float -> float) -> (float -> float) -> float -> float = <fun>
#let prodext a f = function
(x:float) -> a *. f(x);;
prodext : float -> (float -> float) -> float -> float = <fun>
#let derive f epsilon = function
x -> (f(x +. epsilon) -. f(x -. epsilon)) /. (2. *. epsilon);;
derive : (float -> float) -> float -> float -> float = <fun>
#let compose f g = function
x -> f(g(x));;
compose : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b = <fun>
```

Exercice n°5 :
Version récursive :

```
#let rec somf f a = function
  b when b<a -> 0.
| b  -> f(float_of_int a) +. somf f (a+1) b;;
somf : (float -> float) -> int -> int -> float = <fun>
```

Version itérative :

```
#let somf f a b =
let res = ref 0. in
for i = a to b do
  res := !res +. f(float_of_int a) done;
!res;;
somf : (float -> float) -> int -> int -> float = <fun>
```

Exercice n°6 :

```
#let integre f a b n =
    let pas = (b -. a) /. (float_of_int n)  and s = ref 0.0 in
        for k = 1 to n do
            s := !s +. f(a +. (float_of_int k) *. pas)
            done;
    pas *. !s;;
integre : (float -> float) -> float -> float -> int -> float = <fun>
#integre (fun x -> x +. 1.) 0.0 1.0 1000;;
- : float = 1.5005
#integre (fun x -> x *. x) 0.0 1.0 1000;;
- : float = 0.3338335
```

Exercice n°7 :

```
#(*  fonction d'affichage des entiers longs *)
let affiche = function
  0,n2 ->  print_int (n2); print_newline();
 | n1,0 -> print_int(n1);print_string("0000");print_newline();
 | n1,n2 when n2 < 10 -> print_int(n1);print_string("000"); print_int(n2);
 | n1,n2 when n2 <100 -> print_int (n1); print_string("00"); print_int(n2); print_newline();
```

```
 | n1,n2 when n2 < 1000 -> print_int(n1); print_string("0");print_int(n2);print_newline();
 | n1,n2 -> print_int(n1); print_int(n2); print_newline();;
affiche : int * int -> unit = <fun>

#let entier_long_de_int n = (n /10000,n mod 10000);;
entier_long_de_int : int -> int * int = <fun>

#entier_long_de_int 32654;;
- : int * int = 3, 2654

#(* fonction qui convertit un entier long en un entier *)
let int_de_entier_long n = fst(n)*10000 + snd(n);;
int_de_entier_long : int * int -> int = <fun>

#int_de_entier_long (entier_long_de_int 32654);;
- : int = 32654

#(* fonction qui additionne deux entiers longs *)
let somme n1 n2 =
    let q = snd(n1) + snd(n2) in
        match (fst(n1) + fst(n2) + (q / 10000)) with
          p when p <= 10000 -> (p,(q mod 10000))
        | _ -> failwith("somme : débordement");;
somme : int * int -> int * int -> int * int = <fun>

#affiche (somme (entier_long_de_int 1465400) (entier_long_de_int 5868456));;
7333856
- : unit = ()

#(* fonction qui multiplie deux entiers longs *)
let produit n1 n2 =
    let q = snd(n1) * snd(n2) in
        match (fst(n1) * fst(n2) * 10000 + fst(n1) * snd(n2) + snd(n1) * fst(n2) + (q / 10000
          p when p <= 10000 -> (p,(q mod 10000))
        | _ -> failwith("produit : débordement");;
produit : int * int -> int * int -> int * int = <fun>

#affiche (produit (entier_long_de_int 14650) (entier_long_de_int 5456));;
79930400
- : unit = ()

#(* fonction qui compare deux entiers longs *)
let compare  = fun
   (a,b) (c,d) -> (a>c) || (a=c && b >= d);;
compare : 'a * 'b -> 'a * 'b -> bool = <fun>
```