Corrigé feuille d'exercices n°9 de l'option d'informatique.

Du bon usage du graphisme en Caml

Exercices n°1

La fonction `limite` retourne le quadruplet $xmin, xmax, ymin, ymax$ d'un nuage de points donné sous forme de liste

```
#let rec limite = function
    []   -> failwith "limite : erreur (liste vide)"
 | [x,y] -> (x,x,y,y)
 | (x,y)::l -> let (xm,xM,ym,yM) = limite l in
                   ((min x xm),(max x xM),(min y ym),(max y yM));;
limite : ('a * 'b) list -> 'a * 'a * 'b * 'b = <fun>
```

Exercices n°2

La fonction `trace` donne une représentation graphique d'un nuage de points donné sous forme d'une liste.

```
#let trace ll =
   let (xm,xM,ym,yM) = limite ll in
   let l = (xM -. xm) and h = (yM -. ym) and
   L = float_of_int (size_x ()) and H = float_of_int (size_y ()) in
   let rec dessine = function
       [] -> ()
     | (x,y)::queue -> dessine queue;
       plot (int_of_float (L *. (x -. xm) /. l))
                        (int_of_float (H *. (y -. ym) /. h))
     in
   dessine ll;;

trace : (float * float) list -> unit = <fun>
```

Exercice n° 3

La fonction `suite` retourne les termes d'indices compris entre n et m de la suite récurrente définie par u_0 et $u_{k+1} = f(u_k)$.

```
#let suite f u0 n m =
  let l = ref [] and u = ref u0 in
     for k = 0 to n-1 do
         u := f !u done;
     for k = n to m do
         l := (float_of_int k,!u)::!l;
         u := f !u;
         done;
     rev !l;;

suite : ('a -> 'a) -> 'a -> int -> int -> (float * 'a) list = <fun>

suite (fun x -> 2.85 *. x *. (1.-. x)) 0.5 0 10;;
```

Exercice n° 4

La fonction `trace_ligne` est identique à la fonction `trace` précédente sauf que deux points consécutifs de la liste sont reliés par un segment.

```
#let trace_ligne ll =
   let (xm,xM,ym,yM) = limite ll and x = moveto 0 0 in
   let l = (xM -. xm) and h = (yM -. ym) and
   L = float_of_int (size_x ()) and H = float_of_int (size_y ()) in
   let r = min (L/.l)  (H/.h) in
   let rec dessine = function
       [] -> ()
     | (x,y)::queue ->
          lineto (int_of_float (r *. (x -. xm)))
                                    (int_of_float (r *. (y -. ym)));
         dessine queue;
     in
   dessine ll;;
trace_ligne : (float * float) list -> unit = <fun>
#
```

La fonction `etape` calcule les points A, B, C, D à partir des points A et E (voir l'énoncé) .

```
let etape (x0,y0) (x1,y1) =
  let x = (x1 -. x0) /. 3. and y = (y1 -. y0) /. 3. and r = sqrt(3.) /. 2.in
    [(x0,y0);(x0+.x,y0+.y);
          (-.r*.y+.1.5*.x+.x0,1.5*.y+.r*.x+.y0);
          (2.*.x+.x0,2.*.y+.y0)];;

etape : float * float -> float * float -> (float * float) list = <fun>
```

La fonction `Von_Koch` est une fonction récursive pour le tracé de la courbe de Von Koch.

```
#let rec Von_Koch n =
let rec successeur = function
    [] -> failwith "successeur : erreur (liste vide)"
  | [x] -> [x]
  | x::y::ys -> (etape x y)@(successeur (y::ys))
   in
   match n with
      0 -> [(0.0,0.0);(1.0,0.0)]
    | n -> let l = Von_Koch (n-1) in successeur l;;

Von_Koch : int -> (float * float) list = <fun>
```