Corrigé feuille d'exercices n°0 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>
```

Exercice n°5

Voici les principales fonctions sur les nombres complexes :

```
type complexe = {re : float;im : float};;

let zero = {re = 0.; im = 0.};;   (* Définition de quelques constantes *)
let un = {re = 1.; im = 0.};;
let Pi = 4. *. atan 1.;;

let somme (x : complexe) (y : complexe) = (* somme de deux complexes *)
    {re = x.re  +. y.re;
     im = x.im +. y.im};;

let difference (x : complexe) (y : complexe) = (* différence de deux complexes *)
```

```
12      {re = x.re  -. y.re;
13      im = x.im -. y.im};;
14
15  let minus_compl (z : complexe) =  (* opposée d'un complexe *)
16       {re = -. z.re ; im = -. z.im};;
17
18  let produit (x : complexe) (y : complexe) = (* produit de deux complexes *)
19      {re = (x.re  *. y.re) -. (x.im *. y.im);
20      im = (x.re *. y.im) +. (x.im *. y.re)};;
21
22  let rec puissance z n = (* complexe élevé à la puisance n *)
23      match n with
24        0 -> un
25      | n -> produit z (puissance z (n-1)) ;;
26
27  let module (x : complexe) =  (* module d'un complexe *)
28    sqrt (x.re *. x.re +. x.im *. x.im);;
29
30  let module1 (x : complexe) = (* autre fonction "module" plus précise *)
31     if abs_float(x.re) >. abs_float(x.im)
32         then
33              let r = x.im /. x.re in
34              abs_float(x.re) *. sqrt( 1. +. r *. r)
35        else
36              if x.im =. 0.0
37                   then 0.0
38                   else
39                        let r = x.re /. x.im in
40                        abs_float(x.im) *. sqrt( 1. +. r *. r);;
41
42  let Re (x : complexe) = x.re;; (* partie réelle d'un complexe *)
43  let Im (x : complexe) = x.im;; (* partie imaginaire d'un complexe *)
44
45  let print_compl z =  (* affichage d'un complexe *)
46        print_float (Re z); print_string "+I*"; print_float (Im z);
47        print_newline ();;
48
49  let racine (z:complexe) = (* racine carrée d'un complexe *)
50     if Im(z) >. 0. then
51           {re = sqrt((Re(z) +. module(z)) /. 2.) ; im = sqrt((module(z) -. Re(z)) /. 2.)}
52       else
53          {re = sqrt((Re(z) +. module(z)) /. 2.) ; im = -. sqrt((module(z) -. Re(z)) /. 2.)};;

54  #open "graphics";;
55  open_graph "";;
56
57  let WinXmin = ref 0.;; let WinXmax = ref 0.;;
58  let WinYmin = ref 0.;; let WinYmax = ref 0.;;
59  let WinXcor = ref 0.;; let WinYcor = ref 0.;;
60
61  (* Pour définir la fenêtre du plan complexe correspondant à la fenêtre graphique *)
62  let window x1 y1 x2 y2 =
63        WinXmin := x1;  WinXmax := x2;
64        WinYmin := y1;  WinYmax := y2;
65        WinXcor := float_of_int (size_x ()) /. (x2 -. x1);
66        WinYcor := float_of_int (size_y ()) /. (y2 -. y1);;
```

```
67
68  let couleurs = [|black;red;blue;yellow;cyan;green;magenta|];;
69  let nbcouleurs = vect_length couleurs;;
70
71  (* pointe le point de l'écran correspondant au complexe z avec la couleur n *)
72  let affiche z n =
73         set_color couleurs.(n mod nbcouleurs);
74         plot  (int_of_float(( Re(z) -. !WinXmin) *. !WinXcor))
75                       (int_of_float(( Im(z) -. !WinYmin) *. !WinYcor));;
76
77  (* Dessine l'ensemble de Julia ayant le paramètre complexe c avec k itérations *)
78  let Julia c k =
79       let rec suivant z n =
80             affiche z n;
81             if n < k then
82                 let z1 = racine(difference z c) in
83                      let z2 = minus_compl z1 in
84                            suivant z1 (n+1); suivant z2 (n+1)
85       in
86            suivant zero 0;;
87
88  (* définition de la fenêtre de visualisation *)
89  window (-. 12. /. 7.) (-. 1.) (12. /. 7.) 1.;;
90  Julia (minus_compl un) 10;; (* tracé de l'ensemble de Julia pour c=-1 *)
```

Exercice n°6

Tracé de l'ensemble des complexes de la forme $a_0 + a_1 * c + ... + a_n * c^n$ où $(a_0, a_1, ..., a_n)$ appartiennent à $\{0, 1\}$.

```
1   #let Ensemble c k =
2      let rec suivant z n =
3      if n<k then
4          let z2 = somme z (puissance c n) in
5                affiche z2 n;
6                suivant z (n+1); suivant z2 (n+1)
7          in
8              affiche zero 0;
9              suivant zero 0;
10             suivant un 0;;
11
12  Ensemble : complexe -> int -> unit = <fun>
```