Corrigé feuille d'exercices n°7 de l'option d'informatique.

a)
```
#let verif rect =
let (x1,y1) = rect.(0) and (x2,y2) = rect.(1) in
  (x1 <= x2) && (y1 <= y2) && (vect_length rect = 2);;
verif : ('a * 'b) vect -> bool = <fun>
```

b)
```
#let EstIntérieur P rect =
let (x1,y1) = rect.(0) and (x2,y2) = rect.(1)
and (x,y) = P in
  (x1 <= x) && (x <= x2) && (y1 <= y) && (y <= y2);;
EstIntérieur : 'a * 'b -> ('a * 'b) vect -> bool = <fun>
```

c)
```
#let rec Int_list P = function
 x::xs when EstIntérieur P x -> x::(Int_list P xs)
| x::xs -> Int_list P xs
| [] -> [];;
Int_list : 'a * 'b -> ('a * 'b) vect list -> ('a * 'b) vect list = <fun>
```

d)
```
#let aire rect =
let (x1,y1) = rect.(0) and (x2,y2) = rect.(1) in
  (x2 - x1) * (y2 - y1);;
aire : (int * int) vect -> int = <fun>

#let SommeAires liste_rect =
let s = ref 0 and l = ref liste_rect in
while !l <> [] do
  s := !s + (aire (hd(!l))); l := tl(!l)
done;
!s;;
SommeAires : (int * int) vect list -> int = <fun>
```

e) Pour voir si deux rectangles sont disjoints, on regarde si le premier est à gauche du deuxième ou à droite ou au dessus ou en dessous

```
#let disjoint rect1 rect2 =
let (x1,y1) = rect1.(0) and (x2,y2) = rect2.(1) in
let (X1,Y1) = rect2.(0) and (X2,Y2) = rect2.(1) in
   (x1 > X2) || (x2 < X1) || (y1 > Y2) || (y2 < Y1);;
disjoint : ('a * 'b) vect -> ('a * 'b) vect -> bool = <fun>
```

f) Le pire des cas correspond à une liste de rectangles disjoints. Dans ce cas il faut tester toutes les intersections possibles ce qui correspond au nombre de sous-ensemble de cardinal 2 dans un ensemble de cardinal n. Il y aura donc $C_n^2 = \frac{n(n-1)}{2}$ appels à la fonction `disjoint`. Dans le meilleur des cas, les deux premiers rectangles de la liste ne sont pas disjoints, lors de l'exécution du programme, il n'y aura qu'un seul appel à la fonction `disjoint`. En effet, il faut savoir que dans une expression du type `A && B`, si l'évaluation de `A` donne `false`, l'expression `B` n'est pas évaluée. De même, dans une expression du type `A || B`, si l'évaluation de `A` donne `true`, l'expression `B` n'est pas évaluée.

```
let disjoints lrect =
 let rec ldisj rect = function
     []  -> true
 | x::xs -> (disjoint rect x)&&(ldisj rect xs)
in
 let rec Disjoints = function
     []   -> true
  | x::xs -> (ldisj x xs) && (Disjoints xs)
in Disjoints lrect;;
```

g)
```
#let intersect rect1 rect2 =
match disjoint rect1 rect2 with
  true -> failwith "intersection vide"
| false -> let (x1,y1) = rect1.(0) and (x2,y2) = rect1.(1)
           and (x3,y3) = rect2.(0) and (x4,y4) = rect2.(1)  in
             [|(max x1 x3), (max y1 y3); (min x2 x4),(min y2 y4)|];;
(* intersect : ('a * 'b) vect -> ('a * 'b) vect -> ('a * 'b) vect = <fun> *)

let dessine rect =
let (x1,y1) = rect.(0) and (x2,y2) = rect.(1) in
fill_rect x1 y1 (x2-x1) (y2-y1);;

let trace rect1 rect2 =
set_color red; dessine rect1;
set_color blue; dessine rect2;
set_color green; dessine(intersect rect1 rect2);;

clear_graph();;trace [|100,100;300,250|] [|30,120;260,260|];;
```