Exercice 3 : **Typage de fonctions**

```
1 let f1 (x, y) = 2 * x - 3 * y
2 let f2 x = sin x +. x
  let f3 (x, y) = if x -. 1. > y then x else y
4 let f4 = fun (x, y, z) -> (x, (y, z))
```

(a) `f1` a un type de la forme `'a * 'b -> 'c` car c'est une fonction qui prend en argument un couple. `'a` est le type de `x` et `'b` est le type de `'y`.
Or l'opérateur $*$ a deux opérandes entières. Donc `x` et `y` sont des entiers, donc `'a = int` et `'b = int`.
On vérifie aisément que les autres types conviennent. Le résultat de l'opérateur `-` est entier, donc `'c = int`.
On a donc `f1 : int * int -> int`.

(b) `f2` a un type de la forme `'a -> 'b` car c'est une fonction.
Or `sin : float -> float`. Donc `x` est un flottant, donc `'a = float`.
L'opérateur `-.` a alors bien deux opérandes de type `float`. Son résultat est aussi de type `float`, donc `'b = float`.
On a donc `f2 : float -> float`.

(c) `f3` a un type de la forme `'a * 'b -> 'c` car c'est une fonction qui prend en argument un couple. `'a` est le type de `x` et `'b` est le type de `'y`.
Or l'opérateur `-.` a deux opérandes flottantes. Donc `'a = float`, et `x -. 1.` est de type `float`.
Or `>` a deux opérandes de même type. Donc `'b = float`.
L'opérateur `if .. then .. else ..` a bien son premier opérande de type `bool` et les deux autres de même type (ici `float`). Le résultat de cet opérateur est donc un flottant, donc `'c = float`.
On a donc `f3 : float * float -> float`.

(d) `f4` a un type de la forme `'a * 'b * 'c -> 'd`.
Or `(y, z)` est de type `'b * 'c`, donc `(x, (y, z))` est de type `'a * ('b *'c)`.
On a donc `f4 : 'a * 'b * 'c -> 'a * ('b * 'c)`.

```
1 let g1 (x, y) = x y
2 let g2 = fun x -> fun y -> fun z -> x
  let g3 x = fun (y, z) -> x (y z)
4 let g4 (x, y) = x (x y)
```

2. (a) `g1` a un type de la forme `'a * 'b -> 'c`.
Or `x` est appliquée à `y`, donc `'a = 'b -> 'd`.
L'expression `x y` est donc de type `'d`, donc `'c = 'd`.
On a donc `g1 : ('b -> 'c) * 'b -> 'c`

(b) `g2` a un type de la forme `'a *-> 'b -> 'c -> 'd`.
Or `x` est par définition de type `'a`.
Donc on a `g2 : 'a -> 'b -> 'c -> 'a`

(c) `g3` a un type de la forme `'a -> 'b * 'c -> 'd`.
Or `y` est appliquée à `z`, donc `'b = 'c -> 'e`.
L'expression `y z` est donc de type `'e`, et `x` est appliquée à cette expression, donc `'a = 'e -> 'f`.
'expression `x (y z)` est donc de type `'f`, donc `'f = 'd`.
On a donc `g3 : ('e -> 'd) -> ('c -> 'e) * c -> 'd`

(d) `g4` a un type de la forme `'a * 'b -> 'c`.
Or `x` est appliquée à `y`, donc `'a = 'b -> 'd`.
L'expression `x y` est donc de type `'d`, et `x` est appliquée à cette expression, donc `'b -> 'd = 'd -> 'e`.
Donc `'b = 'd = 'e`.
On a donc `g4 : ('a -> 'a) * 'a -> 'a`