Exercice 1 [Typage]

Que répond l'interpréteur

Exercice 2 [Démonstration de propriétés]

1. Définir la fonction de composition de fonctions

```
val compose : ('a → 'b) → ('c → 'a) → ('c → 'b) = <fun>
let compose f g = (fun x → f (g x));;
```

2. Démontrer que pour toutes fonctions `f: 'a → 'b`, `g: 'c → 'a` et toute liste `l: 'a list`, on a :

```
map f (map g l) = map (compose f g) l
```

Cas de base : `l = []`

```
map f (map g []) = map f [] = []
map (compose f g) [] = []
```

Donc le cas de base est **vérifié ✓**.

Cas inductif : `l = t::q` et on suppose que `map f (map g q) = map (compose f g) q`

```
map f (map g l) = map f (map g t::q)
                = map f ((g t)::(map g q))
                = (f g t)::(map f (map g q))
                = ((compose f g) t)::(map (compose f g) q)
                = map (compose f g) t::q = map (compose f g) l
```

Donc le cas inductif est **vérifié ✓**.

Exercice 4

1. Nous souhaitons inventer la notion d'associations. Une association sera une structure permettant d'associer des valeurs à des données en entrée. Afin de simplifier le problème, il existera un type générique des associations du type `int` vers un type `'a`. Nous souhaitons avoir une structure permettant les opérations suivantes :

- **association vide (aucune valeur associée)**
- **test de vacuité de la structure**
- **ajout d'une association de la clef `k` à la valeur `v` dans une structure `m`**
- **modification de la valeur associée à une clef**
- **recherche de la valeur associée à une clef (si elle existe)**
- **suppression d'une clef**
- **`fold` sur l'association entière prenant en compte les clefs et les valeurs**

Proposer une interface pour un module prenant en compte les clefs et les valeurs.

```
module type I_assoc = sig
      type 'a assoc
      exception Already_in
      exception Not_found
      val empty: 'a assoc
      val is_empty: 'a assoc → bool
      val add: 'a assoc → int → 'a → 'a assoc (* raises Already_in *)
      val edit: 'a assoc → int → 'a → 'a assoc (* raises Not_found *)
      val del: 'a assoc → int → ('a assoc * 'a)
      val fold: (int → 'a → 'b → 'b) → 'a assoc → 'b
      val find: 'a assoc → int → 'a
end;;
```

2. À l'aide de vos connaissances, proposer une ou deux solutions rapides (pas besoin de donner le code) de cette structure.

```
type 'a assoc = int list * 'a list;;
type 'a assoc = (int * 'a) list;;
type 'a assoc = (int * 'a) avl;;
```