Créer un module en OCaml

→ Un module permet de regrouper plusieurs définitions de valeur, de type, de fonction ou d'exception se rapportant à un même objet. Par exemple `List` et `String` sont des modules.

→ Le nom d'un module commence toujours par une majuscule.

→ Un module est défini selon la syntaxe suivante.

```
module Nom_module =
  struct
   définition de type
   définition de fonctions
   définition d'exceptions
  end
```

→ En dehors du corps de la définition du module `Mod`, on accède à un élément (une valeur, un type simple, une fonction ou une exception) `obj` de ce module par `Mod.obj`. De plus on accède à un type paramétré `'a t` défini dans `Mod` par `'a Mod.t` à l'extérieur.

On donne ci-après un exemple de définition module pour les fractions (en supposant que la fonction `pgcd` est déjà définie), puis quelques exemples d'utilisation dans un interpréteur.

```
module Fraction =
  struct
    type t = int*int
    let un_tiers : t = (1,3)
    exception FractionMalDefinie

    let simplifie ((num,den):t) : t =
      if den = 0 then raise FractionMalDefinie
      else let d = (pgcd num den) in (num/d, den/d)
  end
```

```
# let f1:Fraction.t = 2,6;;
val f1 : Fraction.t = (2, 6)
# Fraction.simplifie f1;;
- : Fraction.t = (1, 3)
# Fraction.simplifie Fraction.un_tiers;;
- : Fraction.t = (1, 3)
# let f0 = 1,0 in Fraction.simplifie f0;;
Exception: Fraction.FractionMalDefinie.
```

Remarque : Bien qu'on l'évite pour améliorer la lisibilité et éviter des erreurs, il est possible d'utiliser la commande `open Nom_module` pour éviter d'avoir à préfixer tous les éléments du module par le nom du module. De plus, il est aussi possible de factoriser le préfixage par un nom de module, ce qui revient à réaliser une ouverture locale du module. Exemple : `fun l -> List.(length (rev l));;`