Listes

Les listes peuvent être définies comme un type variant polymorphe récursif :

```
type 'a liste = Nil | Cons of 'a * 'a liste
```

Les noms de ces constructeurs sont traditionnels dans la communauté des langages fonctionnels. Le premier, `Nil`, dénote la liste vide ; le second, `Cons`, dénote la liste non vide. Une liste non vide est alors modélisée par une paire dont la première projection est un élément de la liste (de type `'a`) et la seconde la sous-liste restante (donc de type `'a liste`). Par exemple :

```
let liste_vide= Nil
let liste_singleton = Cons ('a', Nil)
let liste_singleton_bis = Cons (7, Nil)
let liste_longue = Cons (1, Cons (2, Cons (3, Cons (4, Nil))))
```

Les listes prédéfinies et la biliothèque List

Par défaut, le système prédéfinit un type `'a list`, dont le constructeur de liste vide est `[]`, et celui de liste non vide est `::` (utilisé en position infixe). La fonction de concaténation de deux listes est notée `@` (**Ce n'est pas un constructeur.**).

Exemples de formes équivalentes :

```
let l = 1 :: (2 :: 3 :: (4 :: []))
let l = 1 :: 2 :: 3 :: 4 :: []
let l = [1;2;3;4]
```

La biliothèque List fournit des fonctions sur les listes.

Les listes prédéfinies

Exemple Une fonction qui retourne une liste.

```
let rec reverse = function
  [] -> []
| h::l -> (reverse l) @ [h]
```

ou, plus efficacement :

```
let reverse l =
  let rec reverse_aux acc = function
    [] -> acc
  | h::t -> reverse_aux (h::acc) t
in reverse_aux [] l
```