

PROGRAMMATION FONCTIONNELLE 3

VERS LE FILTRAGE PAR MOTIFS

Mise en jambes

Exercice 1. Que fait la fonction suivante ?

```
let third = function (x,y,z) -> z;;
```

Que fait alors l'instruction suivante ?

```
# third (1,2,3,4);;
```

Pourquoi ?

Exercice 2. Que fait l'expression suivante ?

```
( < ) 0
```

Manipulation de listes

Avec OCaml, la structure de donnée évoluée la plus fréquente est la liste.

On note `[]` la liste vide, `[1;2;3;4]` la liste composée des éléments 1, 2, 3, 4. On note aussi `h::t` la liste dont le premier élément est `h` (la *tête*) et dont la suite est `t` (la *queue*), ce qui permet de noter aussi `1::2::3::4::[]` la liste précédente, ou encore `1::2::3::[4]`, ou encore `1::2::[3;4]`, ou encore `1::[2;3;4]`.

Une liste d'éléments de type `w` est une `w list`. Si elle n'est pas vide, sa tête est donc un élément de type `w` et sa queue est encore une `w list`.

Exercice 3. Que fait l'instruction suivante ?

```
# let rec facto n =
  if n = 0 then
    (1,[1])
  else
    let precedent,toutes = facto (n-1) in
    let resultat = n*precedent in
    (resultat, resultat::toutes);;
```

Notez que les parenthèses autour de `1,[1]` et celles autour de `resultat, resultat::1` ne sont pas nécessaires. Si vous avez des difficultés à saisir ce que fait cette fonction, vous pouvez utiliser l'instruction `#trace facto` (pour une fois, le caractère `#` fait partie de l'instruction), qui demande à OCaml de vous donner, à l'avenir, plus de détails sur `facto` à chaque fois que vous appellerez.

Exercice 4. À partir du résultat de l'exercice précédent, déduisez ce que fait l'instruction suivante :

```
# let factorielle n =
  let rec aux n =
    if n = 0 then
      1,[1]
    else
      let precedent,toutes = aux (n-1) in
      let resultat = n*precedent in
      resultat, resultat::toutes
  in
```

```
snd (aux n);;
```

Exercice 5. Écrivez une fonction qui construisse la liste de tous les entiers entre p et q .

Exemple :

```
# segment 5 10;;  
- : int list = [5;6;7;8;9;10]
```

Exercice 6. Déterminez ce que fait la fonction `List.fold_left`.

Exercice 7. À l'aide de la fonction `List.fold_left`, écrivez une fonction qui calcule la somme des éléments d'une liste d'entiers.

Exemple :

```
# somme_liste [1;2;3;4;5];;  
- : int = 15  
# somme_liste [];;  
- : int = 0
```

Exercice 8. * Écrivez une fonction qui extrait d'une liste la sous-liste des éléments vérifiant une propriété donnée.

Exemple :

```
# extrait [1;2;0;-5;7;8] (( > ) 0);;  
- : int list = [-5]  
# extrait [1;2;0;-5;7;8] (( < ) 0);;  
- : int list = [1;2;7;8]  
# extrait [1;2;0;-5;7;8] (( = ) 3);;  
- : int list = []
```

Exercice 9. * Écrivez une fonction qui cherche le plus petit élément d'une liste – et renvoie `None` si la liste est vide. En fait, vous aurez probablement besoin de définir une fonction auxiliaire.

Exemple :

```
# minimum [1;2;0;-5;7;8];;  
- : int option = Some (-5)
```