Filtrage direct avec le mot-clé `function`

```
let pair_cmp pair =
  match pair with
    P(x, y) when x > y -> 1
  | P(x, y) when x < y -> 1
  | _ -> 0

let pair_cmp = function
    P(x, y) when x > y -> 1
  | P(x, y) when x < y -> 1
  | _ -> 0
```

Comparaison pratique

- utiliser `Sys.time` (appel à la command Unix `time`)
- faire la différence entre les temps de fin et de début de l'excécution

```
utop[30]> Sys.time();;
- : float = 6.536402
utop[31]> let start = Sys.time();;
val start : float = 6.540147
utop[32]> Sys.time() -. start;;
- : float = 0.00526800000000005
```

Exemple de comparaison pratique

```
let time f =
  let start = Sys.time() in
  let _ = f () in
  Sys.time() -. start
```

permet de mesurer le temps d'exécution de la fonction `f` sans argument passée en paramètre.

```
utop[36]> time;;
- : (unit -> 'a) -> float = <fun>
utop[37]> time (fun () -> List.mem 100 (iota 100));;
- : float = 8.00000000111822374e-06
```

Exemples avec trois versions de la fonction qui renverse une liste.
(voir `test-efficacite.ml`)

Fonction `List.fold_left`

```
utop[6]> List.fold_left;;
- : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a = <fun>
```

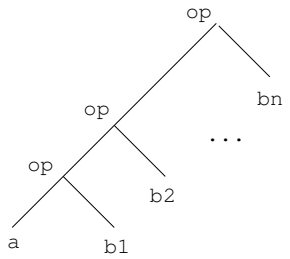
`List.fold_left op a l`

`op` est un opérateur binaire `'a -> 'b -> 'a`

`a` est une valeur de type `'a`

`l` une liste d'éléments de type `'b` : `[b1; b2; ...; bn]`

calcule `op (op ... (op (op a b1) b2) ... ) bn`



Fonction `List.fold_right`

```
utop[7]> List.fold_right;;
- : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b = <fun>
```

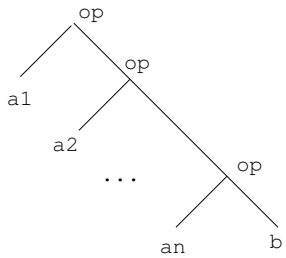
`List.fold_right op l b`

`op` est un opérateur binaire `'a -> 'b -> 'b`

`b` est une valeur de type `'b`

`l` une liste d'éléments de type `'a` : `[a1; a2; ...; an]`

calcule `op a1 (op a2 (... (op an b))...)`



`List.fold_right` vs `List.fold_left`