AFP6b

December 10, 2024

1 (Rappel) Dataflow model

[2]:
```
[3;7;2;1] |> List.filter ((=) 1) |> List.length |> ((=) 1) |> printfn "%b"
```

```
true
```

[3]:
```
let contains x xs = xs |> List.filter ((=) x) |> List.length |> ((=) 1)

contains 1 [3;7;2;1] |> printfn "%b"
```

```
true
```

[5]:
```
[3;7;2;1] |> List.filter (fun x-> contains x [7;5;3;9]) |> printfn "%A"
```

```
[3; 7]
```

[6]:
```
let intersec xs ys = xs |> List.filter (fun x-> contains x ys)

intersec [3;7;2;1] [7;5;3;9] |> printfn "%A"
```

```
[3; 7]
```

Exercise. Comment définir la différence de 2 ensembles puis l'union ?

2 Notation en Compréhension

[9]:
```
[3;7;2;1] |> List.filter (fun x -> (x%2)=1) |> printfn "%A"

[3;7;2;1] |> List.map (fun x -> if (x%2)=1 then [x] else []) |> List.concat |>␣
 ↪printfn "%A"
```

```
[3; 7; 1]
[3; 7; 1]
```

[15]:
```
let ret x = [x]
let (>>=) xs f = List.concat (List.map f xs)

let cross =
  [1..5] >>= fun x ->
  [1..5] >>= fun y ->
```

```
  ret (x,y)

printfn "%A" cross
```

```
[(1, 1); (1, 2); (1, 3); (1, 4); (1, 5); (2, 1); (2, 2); (2, 3); (2, 4); (2, 5);
 (3, 1); (3, 2); (3, 3); (3, 4); (3, 5); (4, 1); (4, 2); (4, 3); (4, 4); (4, 5);
 (5, 1); (5, 2); (5, 3); (5, 4); (5, 5)]
```

[17]:
```
let fail    = []
let guard c = if c then ret () else fail

let upper =
  [1..5] >>= fun x ->
  [1..5] >>= fun y ->
  guard (x<y) >>= fun _ ->
  ret (x,y)

printfn "%A" upper
```

```
[(1, 2); (1, 3); (1, 4); (1, 5); (2, 3); (2, 4); (2, 5); (3, 4); (3, 5); (4, 5)]
```

Exercise. Comment trouver 5 triplets de Pythagore (a,b,c) tel que $a^2 + b^2 = c^2$?

3 Mini-Application: Tableur

Ligne <-> Colonne

[20]:
```
List.transpose [[12;15;13;14]] |> printfn "%A"
```

```
[[12]; [15]; [13]; [14]]
```

Fusion

[21]:
```
let l1 = List.transpose [[12;15;13;14]]
let l2 = List.transpose [[18;17; 8;16]]

List.map2 (@) l1 l2 |> printfn "%A"
```

```
[[12; 18]; [15; 17]; [13; 8]; [14; 16]]
```

Traitement/Calcul

[25]:
```
let csv = List.map2 (@) l1 l2

#nowarn "40"
let db =
  csv >>= fun [n1;n2] ->
  ret [n1;n2;(n1+n2)/2]

printfn "%A" db
```

```
[[12; 18; 15]; [15; 17; 16]; [13; 8; 10]; [14; 16; 15]]
```

Exercise. Comment indexer les données puis extraires celles contenant une valeur inférieure à 8 ?

Jointure

[26]:
```
List.zip [1..4] [12;15;13;14] |> printfn "%A"
```

```
[(1, 12); (2, 15); (3, 13); (4, 14)]
```

[27]:
```
let l3 = List.zip [1..4] [12;15;13;14]
let l4 = List.zip [1..4] [18;17; 8;16]

let join =
  l3 >>= fun (k1,v1) ->
  l4 >>= fun (k2,v2) ->
  guard (k1=k2) >>= fun _ ->
  ret [v1;v2]

join |> printfn "%A"
```

```
[[12; 18]; [15; 17]; [13; 8]; [14; 16]]
```

Sérialisation

[39]:
```
let content =
  join >>= fun [n1;n2] ->
  ret ((string n1)+","+(string n2)+"\n")

open System.IO
content |> List.reduce (+) |> fun c -> File.WriteAllText ("out.csv",c)

File.ReadAllText "out.csv" |> printfn "%s"
```

```
12,18
15,17
13,8
14,16
```

3.1 (Chap. suivant) De-sérialisation

Grammaine

```
cvs    = lines*
line   = column(','line)?'\n'
column = digit+
```

Base

[42]:
```
let car = function
  | "" -> []
```

```
  | s  -> [(s[0],s[1..])]

(car "12,18") |> printfn "%A"
```

```
[('1', "2,18")]
```

Composition séquentielle (monade)

[48]:
```
let (>>) p f s =
  match p s with
  | [(r,s2)] -> f r s2
  | []       -> []

let retn v s = [(v,s)]
let failed s = []

let digit =
  car >> fun c ->
  if (List.contains c ['0'..'9']) then retn c else failed

(digit "12,18") |> printfn "%A"
```

```
[('1', "2,18")]
```

Alternatives et répétitions

[60]:
```
let (<|>) p1 p2 s =
  match p1 s with
  | [] -> p2 s
  | r  -> r

let rec star p =
  (p      >> fun x ->
   star p >> fun xs ->
   rtn (x::xs)
  ) <|> rtn []

let plus p =
  p      >> fun x ->
  star p >> fun xs ->
  rtn (x::xs)

let option p = (p >> fun r -> rtn [r]) <|> rtn []

let sym c =
  car >> fun c2 ->
  if (c=c2) then retn c else failed
```

Application

[96]:
```
let value xs = xs |> List.map (fun n->(int n)-48) |> List.reduce (fun n m->␣
 ↪10*n+m)

let column =
  plus digit >> fun v ->
  rtn (value v)

let rec line =
  column                              >> fun c ->
  option (sym ',' >> fun _ -> line) >> fun o ->
  match o with
  | []  -> sym '\n' >> fun _ -> rtn [c]
  | [r] -> rtn (c::r)

let csv = star line

File.ReadAllText "out.csv" |> csv |> fun [r,_] -> r |> printfn "%A"
```

```
[[12; 18]; [15; 17]; [13; 8]; [14; 16]]
```