info401 : Programmation fonctionnelle
Contrôle des connaissances – 1
CORRECTION

Pierre Hyvernat
Laboratoire de mathématiques de l'université de Savoie
bâtiment Chablais, bureau 22, poste : 94 22
email : Pierre.Hyvernat@univ-savoie.fr
www : http://www.lama.univ-savoie.fr/~hyvernat/
wiki : http://www.lama.univ-savoie.fr/wiki

Tout comme pour les TP, n'oubliez pas de commentez votre code pour préciser les points importants.

Un point est réservé pour la présentation.

Partie 1 : questions de cours

(4) *Question 1.* Pour chacune des expressions suivantes, dites :
- si l'expression est bien typée,
- si oui, donnez sa valeur et son type.

```
let exp1 = 1::[2+3]
let exp2 = 1::(2,3)
let exp3 = (1,2)::[(2,3); (3,4,5)]
let exp4 = [ (true,42) ; (-42,false)]
let exp5 = (fun x->x+1)::[]
let exp6 f = fun a b -> f a
```

Correction :
- `exp1` de type `int list`, sa valeur est `[1 ; 5]`,
- `exp2` mal typé,
- `exp3` mal typé,
- `exp4` mal typé,
- `exp5` de type `(int->int) list`, sa valeur est `[fun x->x+1]`,
- `exp6` de type `('a->'b) -> 'a -> 'c -> 'b` et sa valeur est `fun f a b -> f a`.

(2) *Question 2.* Dans le morceau de code suivant, donnez la liste des variables de l'environnement aux quantre endroits `(*1*)`, `(*2*)`,`(*3*)` et `(*4*)`.

```
(*0 on suppose que l'environnement est vide *)
let z = 0 (*1*) + 0
let rec f n l = (*2*) match l with
      []   ->  n
    | a::l -> let m = n+1 in
              (*3*) f m l
(*4*)
```

Correction :
- `(*1*)` l'environnement est toujours vide,
- `(*2*)` l'environnement contient `z`, `f`, `n` et `l`,
- `(*3*)` l'environnement contient `z`, `f`, `n`, `a`, `l` et `m`,
- `(*4*)` l'environnement contient `z` et `f`.

(2) *Question 3.* Quelle est la valeur de

```
let x =
  (fun x y a ->
     if a<0
     then (fun i -> i::x)
     else (fun i -> y::[i]))  []
                              3.141
                              ((fun a b c -> a+c) (-12) [(-42,12)] 42)
                              2.718
```

Détaillez votre raisonnement.

Correction : On obtient `[3.141 ; 2.718]` :

`((fun a b c -> a+c) (-12) [(-42,12)] 42)` donne `30`,

On obtient donc

```
let x =
  (fun x y a ->
     if a<0
     then (fun i -> i::x)
     else (fun i -> y::[i]))  []
                              3.141
                              30
                              2.718
```

Le premier argument (`x`) est remplacé par `[]`, le second (`y`) par `3.141` et le troisième (`a`) par `30`. L'expression devient :

```
let x =
     if 30<0
     then (fun i -> i::[])
     else (fun i -> 3.141::[i]))
                              2.718
```

Comme 30 est plus grand que 0,

```
let x = (fun i -> 3.141::[i])) 2.718
```

Le résultat final est donc bien `[3.141 ; 2.718]`.

Partie 2 : récursion

(2) *Question 1.* Programmez une fonction `iter` qui étant donnés f et $n \in \mathbf{N}$ permet de calculer la fonction

$$x \quad \mapsto \quad f^n(x) = \underbrace{f(f(...f}_{n}(x)))$$

Correction :

```
let rec iter f n x =
  if (n=0)
  then x
  else f (iter f (n-1) x)
```

ou

```
let rec iter f n x =
  if (n=0)
  then x
  else (iter f (n-1) (f x))
```

(1) *Question 2.* Quel est le type de votre fonction ?

Correction : Dans les deux cas, le type est `('a->'a) -> int -> 'a -> 'a`.

(2) *Question 3.* Programmez une fonction `dernier` qui cherche le dernier élément d'une liste.

Quel est le type de votre fonction ?

Est-ce que votre programme fonctionne tout le temps ? Commentez.

Correction :

```
let rec dernier = function
    [a]  ->  a
  | _::l ->  dernier l
```

Le type de cette fonction est `'a list -> 'a`.

Cette fonction provoque une erreur si on l'applique sur la liste vide. C'est normal, car la liste vide n'a pas de dernier élément.

(2) *Question 4.* Programmez une fonction `range` de type `int -> int -> int list` qui fait la chose suivante : `range a b` devra renvoyer la liste des entiers entre `a` (compris) et `b` (non compris).

Par exemple :

```
range 1 10   -->  [1; 2; 3; 4; 5; 6; 7; 8; 9]
range 10 11  -->  [10]
range 11 10  -->  []
```

Correction : Par exemple :

```
let rec range a b =
  if a<b
  then a::(range (a+1) b)
```

Une meilleure solution serait :

```
let range a b =
  let rec range_aux a b acc =
    if a<b
    then range_aux a (b-1) (b-1::acc)
    else acc
  in range_aux a b []
```

(On verra en cours pourquoi cette solution est meilleure...)

(1) *Question 5. (bonus)* Modifiez votre fonction pour que quand `a` est plus grand que `b`, on obtienne les entiers consécutifs entre `a` et `b` dans l'ordre décroissant :

```
range 1 10   -->  [1; 2; 3; 4; 5; 6; 7; 8; 9]
range 10 11  -->  [10]
range 11 10  -->  [11]
range 13 8   -->  [13; 12; 11; 10; 9]
```

Correction : Il y a de nombreuses solutions, en voici une : (on peut faire plus simple, mais ça sera un peu moins efficace)

```
let range a b =
  let rec range_aux_incr a b acc =
    if a<b
    then range_aux_incr a (b-1) (b-1::acc)
    else acc
  in
  let rec range_aux_decr a b acc =
    if a<b
    then range_aux_decr (a+1) b (a::acc)
    else acc
  in
  if a<b
  then range_aux_incr a b []
```

```
    else range_aux_decr (b+1) (a+1) []
```

Partie 3 : types avec constructeurs

Rappel : le type `'a option` est défini comme suit :

```
type 'a option  =  None  |  Some of 'a
```

(2) *Question 1.* On utilise le type `int option` pour représenter les nombres entiers avec une valeur *indéfinie* `undef` (`None`).

Programmez une fonction d'addition, de type `int option -> int option -> int option` qui respecte la règle :

$$n + \texttt{undef} \quad = \quad \texttt{undef} + n \quad = \quad \texttt{undef}$$

Correction :

```
let addition a b =
  match a with
      None   -> None
    | Some a -> match b with
                  None   -> None
                | Some b -> Some (a+b)
```

On peut faire un peu plus court avec :

```
let addition a b =
  match a,b with
      None,_ | _,None   -> None
    | Some a,Some b     -> Some (a+b)
```

(2) *Question 2.* Définissez un type `inf` pour représenter les nombres entiers avec en plus :

- $+\infty$,
- $-\infty$.

Programmez une fonction `inferieur : inf -> inf -> bool` pour tester si un tels nombre est inférieur *ou égal* à un autre.

Correction :

```
type inf =  MInf | Nb of int | PInf
let inferieur a b =
  match a,b with
      MInf,_ | _,PInf -> true
    | _,MInf | PInf,_ -> false
    | Nb a,Nb b -> a <= b
```