TD 2 - Les fonctions

1 Les incontournables

Exercice 1 Évaluer les expressions suivantes :

```
let f=function x -> function y ->  x+y;;
(f 1);;
((f 1) 2);;
(f 1 2);;
f (1,2);;
f(1 2);;
let f=function x -> (f x);;
(f 1);;
(f 1 2);;
let h=function x -> (h x);;
```

Exercice 2 Reprogrammer à l'aide d'un filtrage la fonction logique `non`, puis la fonction `et` sous forme curryfiées.

Exercice 3 Quel est le type des expressions suivantes :

1. `let f x y = x*y in f 4;;`
2. `let x y z = (z-3*y) = y;;`

Exercice 4 Écrire une fonction `puiss` qui pour deux entiers naturels a et b non simultanément nuls, renvoie a^b.

Exercice 5 On considère les suites (u_n) et (v_n) définies sur $\mathbb{N}$ par

$$\begin{cases} u_0 = 2 \\ v_0 = -1 \end{cases} \quad \text{et } \forall n \in \mathbb{N}, \begin{cases} u_{n+1} = 2u_n - v_n \\ v_{n+1} = u_n + v_n \end{cases}$$

1. Écrire deux fonctions `u` et `v` récursives prenant chacune en entrée un paramètre `n` et revoyant respectivement $u_{\texttt{n}}$ et $v_{\texttt{n}}$.
2. Réaliser la même fonction `uv` de type `int -> (int * int)` qui renvoie directement le couple (u_n, v_n).
3. Que dire de ces 2 solutions ?

2 Pour s'entrainer

Exercice 6 [**sur feuille - exercice du DS 1 de 2022**]

```
let rec truc a = function
        | 1 -> a
        | b when b mod 2 = 0 -> 2 * truc a (b/2)
        | b -> a + truc a (b-1)
;;
```

1. Quel est le type (la signature) de la fonction `truc` ?
2. Que renvoie `truc 5 4` et `truc 10 15` ? (Détailler les étapes)
3. Que peut-on conjecturer pour `truc a b` ?
4. Démontrer ce résultat en effectuant une récurrence sur `b`.
5. Que se passe-t-il si on tape `truc 5 0` ? Corriger ce problème.

Exercice 7 Reprogrammer à l'aide d'un filtrage les fonctions logiques `ou` et `xou` sous forme curryfiées.

Exercice 8 Écrire une fonction `factorielle` qui, en entrée attend un nombre entier naturel n et renvoie l'entier égal à $n!$

Correction 2 1. Pour le `non` :

```
let non = function
        true -> false
   | _   -> true
;;
```

2. Solution pour le `et` :
   - Première version :

```
let et a b = match (a,b) with
         (true, true) -> true
        | _ -> false
;;
```

   - Une seconde :

```
let et a = function
          true -> a
        | _ -> false
;;
```

Correction 3 Tapez pour avoir la réponse, et posez des questions si vous ne comprenez pas :

```
# let x y z = (z-3*y) = y;;
val x : int -> int -> bool = <fun>
# let f x y = x*y in f 4;;
- : int -> int = <fun>
```

Correction 4

```
let rec puiss a = function
          0 -> 1
        | b -> a*(puiss a (b-1))
;;
```

Correction 5 1. Solution utilisant la récursivité croisée :

```
let rec u = function
          0 -> 2
        | n -> 2*(u (n-1))-(v (n-1))
and v = function
          0 -> -1
        | n -> (u (n-1)) + (v (n-1))
;;
```

2. Solution de type `int -> (int * int)`

```
        let rec uv = function
          0 -> (2,-1)
        | n -> let u,v = uv (n-1) in (2*u-v, u+v)
;;
```

3. Le première méthode est bien plus longue (calculer `u 30` puis `uv 30`). Voyez-vous pourquoi ? Posez la question si besoin.

Correction 6 1. `truc : int -> int -> int`

2. `truc 5 4` renvoie 20, en effet : `truc` $5\ 4 = 2 \times$ `truc` $5\ 2 = 2 \times 2 \times$ `truc` $5\ 1 = 2 \times 2 \times 5 = 20$
   `truc 15 10` renvoie 150, en effet :
   `truc` $10\ 15 = 10 +$ `truc` $10\ 14 = 10 + 2 \times$ `truc` $10\ 7 = 10 + 2 \times \big(10 +$ `truc` $10\ 6\big)$
   $= 10 + 2 \times \big(10 + 2 \times$ `truc` $10\ 3\big) = 10 + 2 \times \big(10 + 2 \times (10 +$ `truc` $10\ 2)\big)$
   $= 10 + 2 \times \big(10 + 2 \times (10 + 2 \times$ `truc` $10\ 1)\big) = 10 + 2 \times \big(10 + 2 \times (10 + 2 \times 10)\big)$
   $= 10 + 2 \times \big(10 + 2 \times 30\big) = 10 + 2 \times 70 = 150$

3. On peut donc conjecturer que `truc a b` renvoie `a` $\times$ `b`

4. Pour $n \in \mathbb{N}$, notons $\mathcal{P}_n$: « pour tout entier `a`, `truc a` n vaut `a`$\times n$ »
   - pour tout entier `a`, `truc a 1` renvoie `a`, donc $\mathcal{P}_1$ est vraie.
   - Supposons pour $n \geqslant 1$ fixé, $\mathcal{P}_1, ..., \mathcal{P}_n$ soient vraies.
     - ◇ Si $n+1$ est pair, posons $n+1 = 2p$, alors `truc a (n+1)`$=2\times$`truc a` p. Or $p < n+1 \leqslant n$, donc $\mathcal{P}_p$ est vraie. Ainsi : `truc a (n+1)`$=2\times$`truc a` p=`a`$\times p) = a \times (2p) = a \times (n+1)$ et $\mathcal{P}_{n+1}$ est vraie.
     - ◇ Si $n+1$ est impair, alors `truc a (n+1)`=`a` + `truc a` n= `a` + `a`$\times n$=`a`$\times(n+1)$ et $\mathcal{P}_{n+1}$ est vraie aussi.

   Dans tous les cas $\mathcal{P}_{n+1}$ est vraie. Et par le principe de récurrence forte, $\forall n \in \mathbb{N}^*,\ \mathcal{P}_n$ est vraie.

5. Si $b = 0$, on entre dans une boucle infinie, on peut modifier ainsi :

```
let rec truc a = function
        | 0 -> 0
        | 1 -> a
        | b when b mod 2 = 0 -> 2 * truc a (b/2)
        | b -> a + truc a (b-1)
;;
```

   En fait, le filtrage dans le cas où le second paramètre vaut 1 n'est pas utile :

```
let rec truc a = function
        | 0 -> 0
        | b when b mod 2 = 0 -> 2 * truc a (b/2)
        | b -> a + truc a (b-1)
;;
```

Correction 7 1. Même idée pour le `ou` :

```
let ou a = function
         false -> a
        | _ -> true
;;
```

2. Pour le `xou`, cette expression convient :

```
let xou a = function
     true -> not a
        | _ -> a
;;
```

   mais on peut faire plus court sans filtre (voir TD 1)

Correction 8

```
let rec factorielle = function
          0 -> 1
        | n -> n*(factorielle (n-1))
;;
```

On remarque que 21! dépasse les limites sur les entiers finalement un tableau de valeurs est aussi efficace ! :

```
let factorielle n =
   let f = [|1;1;2;6;24;120;720;5040;40320;362880;3628800;39916800;
   479001600;6227020800;87178291200;1307674368000;20922789888000;
   355687428096000;6402373705728000;121645100408832000;
   2432902008176640000|] in f.(n);;
```