Travaux Pratiques sur Caml Light n°1

Cours d'Informatique de l'École Centrale d'Électronique

—Ingénieur Deuxième Année—

L'objet de ce TP est de vérifier que les concepts de base vus en cours ont bien été assimilés :

- notion de type ;
- types de base ;
- fonctions ;
- variables locales et globales.

▸ **Exercice 1.** *Donnez des exemples de fonctions dont les types synthétisés par Caml Light correspondent aux affichages suivants :*

1. `int -> int -> int` ;
2. `(int -> int) -> int` ;
3. `int -> int -> int -> int` ;
4. `int -> (int -> int) -> int` ;
5. `(int -> int -> int) -> int` ;
6. `(int -> int) -> int -> int`.

▸ **Exercice 2.** *Quel est le type de la fonction suivante ?*

```
#let h (f,g) = function x ->  f (g x);;
```

▸ **Exercice 3.** *Quel est le type de la fonction suivante ? Expliquez.*

```
#function f -> fun g h -> function x -> h f ((g x + 1),true);;
```

▸ **Exercice 4.** *Complétez les lignes manquantes et expliquez les résultats :*

```
#let x = 1 and z = 2;;
...................
...................
#let f = function y -> let z = function x -> x + 1 in
                         y + x + (z 2);;
```

```
...................
#f 3;;
...................
#let x = 2;;
...................
#f 3;;
...................
```

▸**Exercice 5.** *La suite (F_i) des nombres de Fibonacci est définie comme suit : $Fib(0) := 0$, $Fib(1) := 1$ et $Fib(n) := Fib(n-1) + Fib(n-2)$ si $n > 1$. Dans cet exercice, on cherche à exprimer cette suite comme somme de deux suites, la première ayant toujours un argument pair et la seconde un argument impair. Plus précisément, $Fib(n) = Fib_1(n-1) + Fib_2(n-2)$, si n est impair, et $Fib(n) = Fib_2(n-1) + Fib_1(n-2)$ sinon (on utilisera une définition locale de ces deux suites nécessairement mutuellement récursives).*

▸**Exercice 6.** *On définit une fonction* `loop` *qui, appliquée à un prédicat p, une fonction f et une variable x, rend la valeur $f^n(x)$ où n est le plus petit entier qui vérifie $p(f^n(x)) = Vrai$:*

```
#let rec loop p f x = if p x then x else loop p f (f x);;
```

1. *donnez et expliquez le type de la fonction* `loop`*;*
2. *utilisez* `loop` *pour définir une fonction* `find_power` *qui, étant donnés un entier* `x` *et un entier* `n`*, retourne la première puissance de* `x` *supérieure ou égale à* `n`*.*

▸**Exercice 7. (pour les plus rapides)** *Dans cet exercice, on cherche à calculer la racine d'une fonction par la méthode de Newton. Cette méthode est telle que si y est une approximation de la racine d'une fonction f, alors :*

$$y - \frac{f(y)}{f'(y)} \quad ,$$

où $f'(y)$ désigne la dérivée de f au point y, est une meilleure approximation. On définit tout d'abord une fonction `derive` *qui approxime la dérivée d'une fonction* `f` *en un point donné :*

```
#let derive f x dx = (f (x +. dx) -. f x) /. dx;;
```

1. *quel est le type de la fonction* `derive` *? Expliquez ;*
2. *donnez une fonction* `abs` *retournant la valeur absolue de son argument de type* `float`*;*
3. *donnez une fonction* `newton` *prenant en argument une fonction* `f`*, un réel* `epsilon` *et une approximation* `y` *de la racine de* `f`*, itérant la méthode de Newton jusqu'à ce que* `abs (f y) <. epsilon`*;*
4. *appliquez la fonction* `newton` *au calcul de $\sqrt{2}$ à 10^{-5} près.*