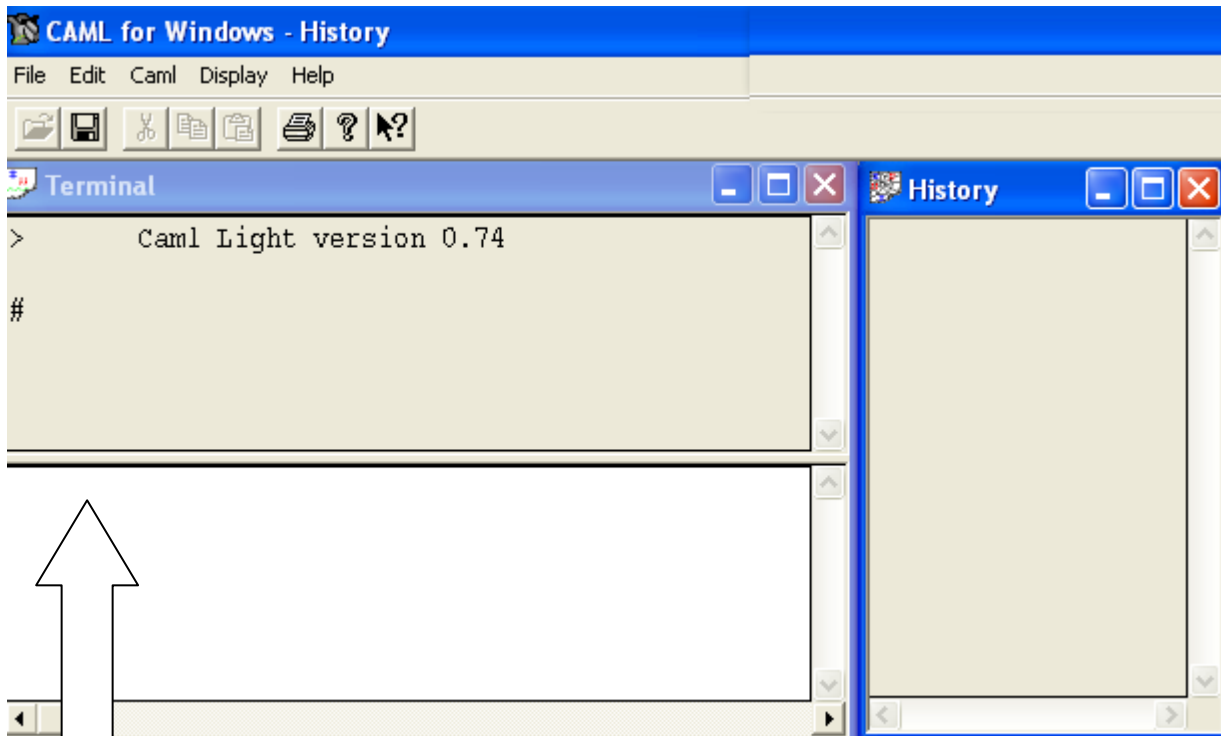


## TP 1

### 1. Lancement de CaML light

Ceci a pour effet de déclencher l'exécution de la boucle de **Lecture/Evaluation/Impression** qui se manifeste par les 2 fenêtres : Terminal (Editeur et Interprétation) et History (trace des expressions).



EDITEUR

En mode interactif, CaML donne systématiquement une réponse qui contient :

- Le nom de la variable déclarée s'il y en a.
- Le type trouvé pendant le typage.
- La valeur calculée après exécution.

## TP CaML Light

### 2. Evaluation d'expressions

Pour vous entrainer à l'utilisation de CAML, demandez à l'interprète d'évaluer les expressions suivantes (et chercher à chaque fois à donner un sens à la réponse de l'interprète) :

#### Expressions sur les entiers :

|                     |                       |
|---------------------|-----------------------|
| 1098 ;;             | 5 * 99 ;;             |
| 10 / 5 ;;           | 10 / 6 ;;             |
| 276 mod 23 ;;       | (3 * 5) + (10 - 6) ;; |
| 21 + 35 + 12 + 7 ;; | 2.7 + 10 ;;           |
| int_of_float 1.;;   | int_of_float 1.1;;    |

#### Expressions sur les réels (flottants) :

|                      |                |
|----------------------|----------------|
| 3.45;;               | 10.0 /. 6.0 ;; |
| 1.1 +. 2.2 ;;        | ceil 3.14 ;;   |
| floor 3.14 ;;        | exp 1.0 ;;     |
| log 2.71828182846 ;; | 3. ** 2. ;;    |
| 3.45e10;;            | sqrt 25. ;;    |

#### Expressions sur les booléens :

|                   |                        |
|-------------------|------------------------|
| true ;;           | not true ;;            |
| true && false ;;  | true or false ;;       |
| 1<2;;             | 1<=2;;                 |
| 0<1 & 3<9;;       | float_of_int 76;;      |
| “ un“ < “ deux“;; | (not false) or true ;; |

#### Expressions sur les caractères et chaînes de caractères :

|                          |                                               |
|--------------------------|-----------------------------------------------|
| `&` ;;                   | `a` ;;                                        |
| int_of_char `a`;;        | char_of_int 97;;                              |
| “a“;;                    | “Ceci est une ch“ ^ “aine de caractères !“ ;; |
| “toutou“.[5] ;;          | “kArim“.[1] ;;                                |
| string_length “KADER“ ;; | sub_string “bonjour“ 0 3 ;;                   |

#### Expressions sur les Uplets

|                         |                           |
|-------------------------|---------------------------|
| (1,2) ;;                | (“coucou“,3.1,(‘A’,2)) ;; |
| 1,2,3 ;;                | fst (12, “octobre“) ;;    |
| (1,(2,3)) ;;            | snd (12, “octobre“) ;;    |
| (12, “octobre“) ;;      |                           |
| (12, “octobre“,true) ;; |                           |

## Définition :

Une définition est une déclaration qui associe un nom à une valeur. On distingue les déclarations globales des déclarations locales.

## Expressions de déclarations globales :

Syntaxe : **let** *nom* = *expr* ;;

où *nom* représente l'identificateur et *expr* l'expression qui lui est associée.

|              |                     |                                             |
|--------------|---------------------|---------------------------------------------|
| let x = 5 ;; | let taille = 2.0 ;; | let pi = 3.14159 ;;                         |
| x ;;         | taille ;;           | let rayon = 10.0 ;;                         |
| x + 2 ;;     | 5 * taille ;;       | (pi *. rayon *. rayon) ;;                   |
|              |                     | let circonference = (2.0 *. Pi *. Rayon) ;; |

## Expressions de déclarations globales simultanées :

Syntaxe : **let** *nom*<sub>1</sub> = *expr*<sub>1</sub>

**and** *nom*<sub>2</sub> = *expr*<sub>2</sub>

.

.

**and** *nom*<sub>n</sub> = *expr*<sub>n</sub> ;;

|                        |  |
|------------------------|--|
| let a = 3 and b = 2 ;; |  |
| a + b ;;               |  |

## Expressions de déclarations locales :

Syntaxe : **let** *nom* = *expr*<sub>1</sub> **in** *expr*<sub>2</sub>;;

la valeur *expr*<sub>1</sub> du nom *nom*, n'est connue que pour le calcul de *expr*<sub>2</sub> .

|                                                                 |                                    |
|-----------------------------------------------------------------|------------------------------------|
| let x = 2 in x * x ;;                                           | let a = 1 and b = 2 in 2*a+b ;;    |
| x ;;                                                            | let a = 1 and b = 2 * a in b + a;; |
| let y = 3 in x + y ;;                                           | let a=1 in let b=2*a in b+a ;;     |
| y;;                                                             |                                    |
| let y = (let x = 3 in x + 3) ;;                                 |                                    |
| let x =3 in y = x + 3 ;;                                        |                                    |
| let x=(1 , "coucou") and y=("hello", 2.1) in (snd x , fst y) ;; |                                    |

## Expressions de conditions :

Syntaxe : **if** *expr*<sub>Bool</sub> **then** *expr*<sub>1</sub>**else** *expr*<sub>2</sub> ;;

La valeur de cette expression est la valeur de *expr*<sub>1</sub> si l'expression booléenne *expr*<sub>Bool</sub> s'évalue à **true** et la valeur de *expr*<sub>2</sub> sinon (à **false**).

|                                |                                       |
|--------------------------------|---------------------------------------|
| if true then 1 else 0 ;;       | let x=3 in if x=0 then 0 else 15/x ;; |
| (if 3=5 then 8 else 10) + 5 ;; |                                       |

## TP 2 : Les Fonctions

### 1. Expressions fonctionnelles (fonctions anonymes).

Définition : Une expression fonctionnelle est constituée d'un (ou plusieurs) *paramètre* et d'un *corps*. Elle est introduite par les mots réservés **function** (pour les fonctions à un seul argument) ou **fun** (pour les fonctions à plusieurs arguments).

#### a/ Fonction anonyme à un seul argument :

Syntaxe :  $\text{function } p \rightarrow \text{expr} ;;$

Exemple :

la fonction qui élève au carré son argument s'écrit en CAML :

```
function x -> x*x ;;  
- : int -> int = <fun>
```

L'application d'une fonction à un argument s'écrit comme la fonction suivie par l'argument.

```
(function x -> x * x) 5 ;;  
- : int = 25
```

#### b/ Fonction anonyme à plusieurs arguments :

Syntaxe :  $\text{fun } p_1 \dots p_n \rightarrow \text{expr} ;;$

Exemple :

```
fun x y -> 3*x + y ;;  
-: int -> int -> int = <fun>
```

L'application d'une fonction à plusieurs arguments s'écrit comme la fonction suivie par les arguments.

```
(fun x y -> 3*x + y) 4 5 ;;  
-: int = 17
```

### 2. Définitions de fonctions

Comme dans le cas des valeurs simples, il est possible d'assigner un nom à une fonction.

Syntaxe :  $\text{let } \text{nom} = \text{expr\_fonc} ;;$

Où *nom* est le nom de la fonction et *expr\_fonc* est une expression fonctionnelle (*function* ou *fun*)

## TP CaML Light

---

Comme le montrent les exemples suivants (définitions et applications):

```
let carre = function x -> x*x ;;
carre : int -> int = <fun>
carre 5 ;;
- : int = 25
```

```
let f = fun x y -> 3 * x + y ;;
f : int -> int -> int = <fun>
f 2 4;;
- : int = 10
```

### Simplification de la syntaxe:

Pour simplifier l'écriture, la syntaxe suivante est acceptée pour la définition d'une fonction d'arité  $n$  (avec  $n \geq 1$ ).

Syntaxe :  $\text{let } nom\ p_1 \dots p_n = expr \ ;;$

Exemple : les deux fonctions précédentes peuvent être définies comme suit :

```
let carre x = x*x ;;
carre : int -> int = <fun>
```

```
let f x y = 3 * x + y ;;
f : int -> int -> int = <fun>
```

### 3. Ecrire en langage CAML les fonctions suivantes

- Succ qui calcule le successeur d'un entier.
- Pred qui calcule le prédécesseur d'un entier.
- Sum qui calcule la somme de 2 entiers.
- Max qui calcule le maximum de 2 réels.
- Max3 qui calcule le maximum de 3 réels de 2 façons différentes (Sans utiliser Max, puis en utilisant Max).
- MinMax qui donne le min et le max en même temps de 2 entiers.
- Carre qui calcule le carré d'un entier.
- SCarre qui calcule la somme des carrés de 2 entiers (en utilisant la fonction Carre).
- ValAbs qui calcule la valeur absolue d'un entier.
- Abs qui calcule la fonction :  $Abs(x, y) = |x - y|$ .
- Surf qui calcule la surface d'un cercle de rayon  $r$  ( $\pi = 3.14$ ).
- Pair qui retourne vrai si son argument est un entier pair, faux sinon.
- Prem qui retourne vrai si son argument est un entier premier, faux sinon.

## TP 3 : Les Fonctions Récursives

### 1. Définitions de fonctions récursives.

**Définition** : Une fonction récursive est une fonction dont la définition fait appel à elle-même.

**Syntaxe** : `let rec nom  $p_1 \dots p_n = expr$  ;;`

**Important** : Le mot réservé **rec** est obligatoire pour indiquer qu'il s'agit d'une fonction récursive.

**Remarque** : Pour pouvoir traiter les cas de bases et le cas général, il est utile d'utiliser la structure if-then-else dans la partie *expr* .

L'exemple suivant montre comment définir, en CAML, la fonction récursive « factorielle » nommée *fact* :

```
let rec fact n = if n = 0 then 1 else n * fact (n-1) ;;
```

```
fact : int -> int = <fun>
```

### 2. Ecrire en langage CAML les fonctions récursives suivantes :

- Fact : factoriel d'un entier n .
- Exp qui calcule la fonction :  $\text{Exp}(x, y) = x^y$  .
- Sigma : la somme des entiers de 0 à x .
- Sigma2 : la somme des entiers compris entre 2 entiers a et b.
- Fib : la fonction Fibonacci définie par :

$$\text{Fib}(n) = \begin{cases} 0 & \text{si } n = 0, \\ 1 & \text{si } n = 1, \\ \text{Fib}(n-1) + \text{Fib}(n-2) & \text{sinon} \end{cases}$$

- PGCD : la fonction qui calcule le PGCD de 2 entiers, en utilisant l'algorithme d'Euclide :

$$\text{PGCD}(a,b) = \text{PGCD}(b,r), \text{ où } r \text{ est le reste de la division de } a \text{ par } b$$

**NB** : Vous pourrez utiliser la fonction primitive  $(a \bmod b)$  qui renvoie le reste r.

- PGCD2 : la fonction qui calcule le PGCD de 2 entiers, en utilisant l'algorithme suivant :

$$\text{PGCD2}(a,b) = \begin{cases} a & \text{si } a = b, \\ \text{PGCD}(a-b, b) & \text{si } a > b, \\ \text{PGCD}(a, b-a) & \text{si } b > a \end{cases}$$

- SumSerie : la fonction qui calcule la somme des n premiers termes de la série harmonique de la forme :

$$1 + 1/2 + 1/3 + \dots + 1/n \quad (\text{pour } n = 0, \text{ la somme est } 0).$$