1.) Le langage CAML.

1.1.) Présentation générale.

- On prononce Caml avec le "ca" de café et le "mel" de melba.
- Caml est un acronyme qui signifie "Categorical Abstract Machine Language" c'est à dire langage de la machine abstraite catégorique. la CAM est une machine abstraite capable de définir et d'exécuter des fonctions.
- On écrit généralement "Caml" avec un "C" majuscule et le reste en minuscule.
- C'est un langage de programmation à la fois facile à apprendre et très expressif.
- C'est un langage développé et distribué par l'INRIA depuis 1984.
- Il existe deux dialectes de Caml : Caml Light et Objective Caml. Caml light est un sous-ensemble de d'objective Caml, plus spécialement adapté à l'enseignement et à l'apprentissage de la programmation. En plus du cœur du langage de Caml Light, Objective Caml comporte un puissant système de modules, des objets et un compilateur optimisant.
- Caml est utilisé dans l'enseignement de la programmation principalement au CNAM, pour l'option informatique des classes préparatoires aux grandes écoles, à l'Ecole Polytechnique et à l'Ecole Normale Supérieure.

1.2.) Caractéristiques du langage.

- Sûreté. Caml est un langage très sûr. Le compilateur fait de nombreuses vérifications avant la compilation et évite ainsi de nombreuses erreurs telles que la confusion de types de données ou l'accès erroné à l'intérieur de données.
- Types de données. Caml propose la gamme des types de données classiques :
  - Types de bases : entiers, flottants, booléens, caractères, chaînes de caractères.
  - Types complexes : n-upplet, tableaux, listes, ensembles, tables de hachage, files, piles, flux de données.
  - En plus de ces types prédéfinis, Caml permet de définir de nouveaux types.
- Fonctions. Caml est un langage de programmation fonctionnel. On peut passer librement des fonctions en argument ou en retourner comme résultat.
- Gestion de la mémoire automatisée et incrémentale. Caml offre une gestion automatique de la mémoire. L'allocation ou la libération de mémoire est laissé à la charge du compilateur. Il n'y a donc pas de corruption inattendue des structures de données manipulées.
- Traits impératifs. Caml permet également la programmation impérative par le biais des boucles, des variables...
- Modules. Caml propose une compilation séparée des fichiers.
- Interactivité. Caml propose un système interactif très pratique pour apprendre le langage ou pour corriger ses programmes.
- Traitement des erreurs. Caml propose un mécanisme général de traitement des exceptions, pour traiter ou corriger les erreurs ou les situations exceptionnelles.
- Polymorphisme. Certains types peuvent rester indéterminés. Ainsi les fonctions qui sont d'usage général peuvent s'appliquer à n'importe quel type sans exception.

1.3.) Types.

Le concept de type est à rapprocher de la notion mathématique d'ensemble. Le type d'une expression représente l'implémentation de l'ensemble auquel appartient la valeur obtenue par l'évaluation de cette expression.

Un type de donnée peut également être vu comme une collection d'objets qui ont des propriétés identiques, indépendamment de toute représentation.

1.3.1.) LES TYPES DE DONNEES PREDEFINIS.

Int	Implémente un sous-ensemble fini des nombres entiers dans l'intervalle [-2^{30}, $2^{30}-1$].
Float	Implémente un sous-ensemble fini des nombres décimaux.
Boolean	Implémente les valeurs booléennes true et false.
Char	Implémente un ensemble de caractères codés par les entiers 0 à 255. Les codes 0 à 127 correspondent aux caractères ASCII. Les valeurs de ce type sont notées au moyen du caractère qu'elles représentent, placé entre quottes ou au moyen du numéro de code à trois chiffres ASCII précédé de \ et placé entre quottes (`\065`) ou encore au moyen d'un symbole spécial (`\n` nouvelle ligne, `\t` tabulation).
String	Le type string implémente un sous ensemble des chaînes de caractères de longueur [0, 2^{16}]. Les chaînes de caractères sont notées entre guillemets.
Unit.	Le type unit ne possède qu'une seule valeur, la valeur () qui signifie "rien".

Caml est un langage fortement typé, ce qui rend possible un contrôle rigoureux du typage des expressions permettant de déceler certaines erreurs de programmation. Pour ce faire, le langage doit connaître (déclaration de types) ou reconnaître (synthèse de types) le type des objets qui lui sont proposés.

La déclaration de type est la technique mise en œuvre dans certains langages tels que Pascal, C ou Ada. Dans ce cas, chaque objet figurant dans le programme est préalablement déclaré avec la mention explicite du type auquel il appartient.

La synthèse de type dispense de toute déclaration préalable du type de l'objet. Le type de l'objet est déduit du contexte par l'application de règles d'inférence de type.

Les identificateurs qui servent à nommer les types tels que `int, float, char...` sont des **constructeurs de types**.

1.3.2.) LES ELEMENTS DU LANGAGE.

<table>
<tr><td>blancs</td><td>Les caractères suivants sont considérés comme "blancs" :
Espace, nouvelle ligne (LF), tabulation, retour chariot, saut de page.</td></tr>
<tr><td>commentaires</td><td>(* Ceci est un commentaire en Caml *).
Les commentaires imbriqués sont traités correctement par Caml.</td></tr>
<tr><td>Identifiants</td><td>Les identifiants sont des séquences de lettres, de chiffres et de '_' commençant par une lettre.
Les identifiants ne peuvent contenir deux caractères soulignés ('_') adjaçants.
Les identificateurs doivent être différents des mots clés du langage Caml.
La taille limite d'un identifiant est (en fonction des implémentations) au moins égale à 256 caractères.</td></tr>
<tr><td>Littéraux entiers</td><td>Séquence de un on plusieurs chiffres pouvant être précédée du signe moins ('-'). Par défaut, les littéraux entiers sont en base 10. Il est cependant possible de les donner en basse 16, 8 ou 2 en les préfixant de la manière suivante :
<table>
<tr><th>Base</th><th>Préfixe</th><th>Exemple</th></tr>
<tr><td>16</td><td>0x ou 0X</td><td>0xA19B</td></tr>
<tr><td>8</td><td>0o ou 0O</td><td>0o765</td></tr>
<tr><td>2</td><td>0b ou 0B</td><td>0b1001011</td></tr>
</table></td></tr>
<tr><td>Littéraux en virgule flottante</td><td>Contiennent une partie entière, une partie décimale et une partie "exposant".
La partie entière consiste en une séquence de chiffres éventuellement précédée par un signe moins.
La partie décimale est composée d'un point suivie par zéro, un ou plusieurs chiffres.
La partie "exposant" est composée de la lettre E (suivie éventuellement du signe moins) et de un ou plusieurs chiffres.
La partie décimale ou la partie "exposant" peuvent être omises (mais pas les deux en même temps pour éviter la confusion avec un entier).</td></tr>
<tr><td>Littéraux 'caractères'</td><td>Les caractères sont délimités par des simples quottes.
Les quottes peuvent contenir n'importe quel caractère différent de ' et de \ ou alors les séquences suivantes :
<table>
<tr><th>Séquence</th><th>Signification</th></tr>
<tr><td>\\</td><td>backslach (\)</td></tr>
<tr><td>\'</td><td>quotte (')</td></tr>
<tr><td>\n</td><td>Nouvelle ligne (LF)</td></tr>
<tr><td>\r</td><td>Retour chariot (CR)</td></tr>
</table></td></tr>
<tr><td>Littéraux "chaîne de caractères".</td><td>Les chaînes de caractères sont délimitées par des doubles quottes ("). Une chaîne peut contenir n'importe quel caractère (à l'exception de " et de \) ainsi que les séquences suivantes :
<table>
<tr><th>Séquence</th><th>Signification</th></tr>
<tr><td>\\</td><td>backslash (\)</td></tr>
<tr><td>\"</td><td>double quotte (")</td></tr>
<tr><td>\n</td><td>Nouvelle ligne (LF)</td></tr>
<tr><td>\r</td><td>Retour chariot (CR)</td></tr>
<tr><td>\t</td><td>tabulation (TAB)</td></tr>
<tr><td>\b</td><td>Backspace (BS)</td></tr>
<tr><td>\dddd</td><td>Le caractère de code ASCII dddd.</td></tr>
</table></td></tr>
</table>

1.3.3.) LES OPERATIONS.

Opérations sur des entiers	Addition, soustraction Multiplication, division entière modulo et logique (bit à bit) ou logique (bit à bit) ou exclusif logique (bit à bit) Complément bit à bit rotation à gauche rotation logique à droite rotation arithmétique à droite Comparaisons Conversion d'une chaîne Conversion en chaîne Conversion d'un flottant Négation	x + y x - y x * y x / y x mod y x land y. (*6 land 4 -> 4*) x lor y. (*6 lor 4 -> 6*) x lxor y. (*6 lxor 4 -> 2*) lnot x. (*lnot 4 -> -5*) x lsl y (ou lshift_left) (*5 lsl 1->10*) (*-5 lsl 1 -> -10*) x lsr y (*5 lsr 1 -> 2*) (*-5 lsr 1 -> 1073741821*) x asr y (ou lshift_right) (*5 asr 1 -> 2*) (*-5 asr 1 -> -3*) <, >, <=, >=, <>, = int_of_string string_of_int int_of_float minus
Opérations sur les flottants	Addition, soustraction Multiplication, division sinus, cosinus exponentielle, racine carrée Valeur absolue Comparaisons conversion d'une chaîne conversion en chaîne conversion d'un entier Négation	x +. y x -. y x *. y x /. y sin x cos x exp x sqrt x atan x atan2 x abs_float <., >., <=., >=., <>., =. float_of_string string_of_float float_of_int minus_float
Opérations sur les chaînes	Concaténation	^ make_string: int-> char->string string_length("coucou") sub_string string->int->int->string s.[0] s.[0] <- `a`
Opérations sur les caractères		int_of_char char_of_int
Opérations sur les booléens	et logique (conjonction) ou logique (disjonction) négation	&& || not

1.4.) Dialoguer avec Caml.

1.4.1.) PRESENTATION DU TERMINAL CAML (SOUS WINDOWS).

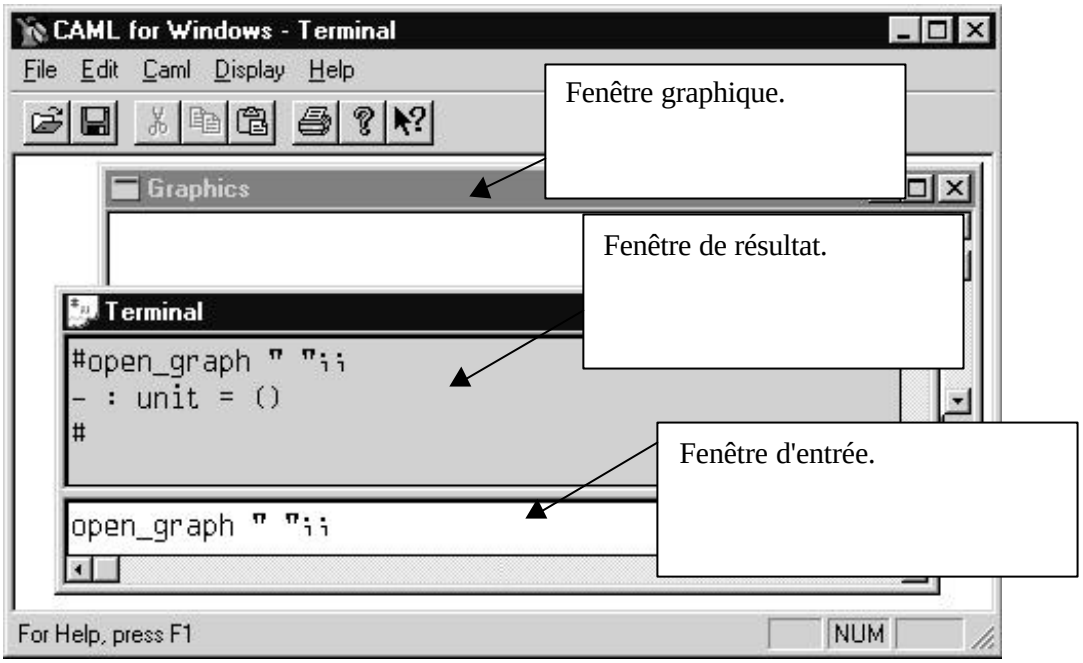


Figure 1 : Terminal Caml Light.

Extrait de CamlWin.ini :

```
[General]
CmdLine=camlrun c:\progra~1\caml\lib\camltop -stdlib c:\progra~1\caml\lib -lang fr
HelpFile=c:\progra~1\caml\doc\refman73.hlp

[Font]
DefHeigth=18
DefWidth=8
DefItalic=0
DefUnderline=0
DefFace name=Courier

Heigth=18
Width=8
Italic=0
Underline=0
Face name=Courier
```

Lorsque l'utilisateur écrit une commande dans la fenêtre d'entrée, Caml compile l'expression puis renvoie son résultat dans la fenêtre de résultat. Ceci permet de vérifier rapidement la validité du code Caml.

`#1+2+3;;`
`- : int = 6`
type valeur
	Opérations sur des entiers.

`#1.2 +. 3.2 *. 5.0;;` `- : float = 17.2`	Opérations sur des flottants
`#4 = 5;;` `- : bool = false`	Opérations sur des booléens

Pour quitter Caml, on tape la commande quit();;.

Dans l'aide en ligne, la syntaxe du langage est donnée dans ce qu'ils appellent une "BNF-like notation". Les symboles terminaux sont donnés en caractères "machine à écrire" et les symboles non-terminaux en italique. Les crochets ([...]) indiquent des composants optionnels, les accolades ({...}) indiquent zéro ou plusieurs répétitions d'un composant. Les accolades suivis d'un signe plus ({...}+) indiquent une ou plusieurs répétitions. Les parenthèses indiquent un groupement. Par exemple, les littéraux entiers sont définis ainsi :

```
Integer-literal  ::= [-] {0...9}+
                   | [-] (0x | 0X) {0...9 | A...F | a...f}+
                   | [-] (0o | 0O) {0...7}+
                   | [-] (0b | 0B) {0...1}+
```

1.4.2.) La session Caml.

Une session Caml correspond à une boucle dans laquelle une lecture d'expressions est réalisée sur l'unité standard, chaque expression lue est soumise à une analyse lexicale puis à une analyse syntaxique, ensuite elle est typée par le système d'inférence de types, avec affichage du type inféré et pour terminer, évaluée avec affichage de la valeur obtenue en résultat.

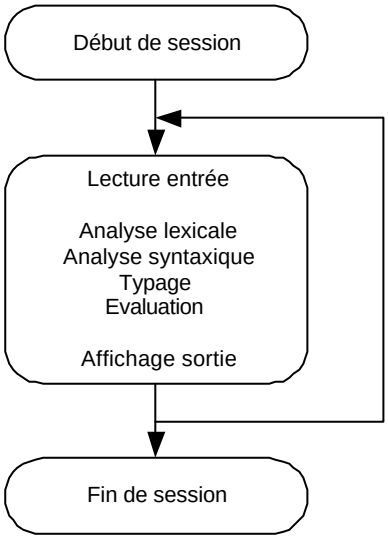


Figure 2 : Session Caml Light.

1.5.) Définitions et environnements.

Une définition construit une liaison entre un nom, la valeur à laquelle il est associé et son type. On peut considérer ce nom comme une abréviation pour la valeur qui lui est liée. Une définition n'est pas modifiable : un nom donné fait toujours référence à la même valeur, celle qu'on a calculée lors de la définition du nom. Il est impossible de changer la valeur liée à un nom; on peut seulement redéfinir ce nom par une nouvelle définition, donc un nouveau "let".

1.5.1.) Definition globale.

Les définitions globales sont valides pour une session Caml.

<définition> ::= **let** <identificateur> = <expression> | **let** <identificateur> = <expression> **and** <définition>

Exemples de définitions globales :

`let s = 1+2+3;;`
`s : string = "5"`
nom type valeur
	Soit s la somme des nombres 1 et 2 et 3

`#let s = 5;;`	Soit s = 5

La collection des définitions ainsi créées est appelée **environnement**. Les définitions successives ont donc pour effet de modifier l'environnement.

1.5.2.) DEFINITION LOCALE.

Les définitions locales n'existent que temporairement, pour la seule durée d'un calcul. Les définitions locales disparaissent à la fin de l'évaluation de l'expression où elles se trouvent.

<définition_locale> ::= <definition> **in** <expression> |<expression> **where** <definition>

`#let s = 20 in s*4;;` `- : int = 80`	
`#let titre = "L'étranger" and auteur = "A. Camus" and separateur = " : " in titre^separateur^auteur;;` `- : string = "L'étranger : A. Camus"`	
`#let h=8 and i2 = -1 in let t=3*h+7*i2 in t*t;;` `- : int = 289`	
`#let a=5 in 2*a*a+2*a-20;;` `- : int = 40`	
`#let a=5 in let b=a*a in 2*b+2*a-20;;` `- : int = 40`	

Il n'est pas nécessaire de spécifier le type de donnée. En effet, Caml ne se contente pas d'afficher le résultat d'une commande mais il détermine également son type (on dit que Caml a "*typé*" l'expression).

Cependant, pour documenter le programme, il est possible d'écrire :

```
#let (a:int) = 5;;
a : int = 5
```

On peut également définir des produits cartésiens de types (qu'on appelle des n-upplets) :

```
#let a = (125,30,12,40);;
a : int * int * int * int = 125, 30, 12, 40
```

La virgule (,) est le délimiteur des produits cartésiens et l'étoile (*) le symbole utilisé par Caml pour noter le produit cartésien.

```
#let a = (1,1.,true,"un",());;
a : int * float * bool * string * unit = 1, 1.0, true, "un", ()
```

```
#let (a : int*int) = (5,5);;
a : int * int = 5, 5
```

Exercice : Evaluer les expressions :

```
let a = 3 and b = 4 in let c = 4 and b = 1 in a*b*c;;
- : int = 12
let x = 3 in let y = x*2 in let z = x + y in let a = x-y-z in
(x*y*z)-a;;
- : int = 174
```

1.6.) Les fonctions.

Les fonctions forment les constituants élémentaires des programmes en Caml. Un programme n'est rien d'autre qu'une collection de définitions de fonctions, suivie d'un appel à la fonction qui déclenche le calcul voulu.

1.6.1.) DEFINIR UNE FONCTION.

Proche des notations mathématiques usuelles.

`let carre x = x*x;;`	`#let carre = function x->x+1;;` `carre : int -> int = <fun>`
`let incremente x = x+1;;`	`#let incremente = function x->x+1;;` `incremente : int -> int = <fun>`
`let puissance4 x = carre(x) * carre(x);;`	`#let puissance4 = function x->carre x * carre x;;` `puissance4 : int -> int = <fun>`

On peut se rapprocher encore de la définition mathématique en écrivant :

```
#let (carre : int -> int) = function x -> x*x;;
carre : int -> int = <fun>

#let (incremente : int -> int) = function x -> x+1;;
incremente : int -> int = <fun>
```

1.6.2.) APPLICATION D'UNE FONCTION.

Une application est une expression formée d'une fonction associée à un ou plusieurs arguments. L'application d'une fonction à son argument suit aussi la convention mathématique :

`#carre 2;;` `- : int = 4`	Les parenthèses autour de l'argument sont facultatives lorsque l'argument est une variable ou une constante.
`carre (-1);;` `- : int = 1`	Pour une expression plus compliquée, les parenthèses servent à grouper l'expression.
`carre (2*3);;` `- : int = 36`	carre (2*3) est équivalent à carre(6)
`#carre 2 * 3;;` `- : int = 12`	carre 2 * 3 est équivalent à carre(2) *3 c'est à dire à 4 * 3.
`carre (carre 2);;` `#- : int = 16`	L'argument d'une fonction peut être le résultat d'une autre fonction.

1.6.3.) APPLICATION PARTIELLE.

Soit la fonction f suivante :

```
let f s1 s2 = s1 ^ s2;;
```

Considérons les cas suivants :

```
f "Hello " "World"
- : string = "Hello World'
f "Hello "
- : string -> string = <fun>
```

La seconde évaluation de la fonction f renvoie une valeur fonctionnelle. A partir de cette valeur fonctionnelle on peut définir une nouvelle fonction :

```
let g = f "Hello "
g : string -> string = <fun>
g "Dolly";;
- : string = "Hello Dolly"
```

1.6.4.) DEFINITION LOCALE DE FONCTIONS.

Rien n'empêche de définir localement une fonction :

`let pred x = x-1 in (pred 3) * (pred 4);;` `#- : int = 6`	La fonction pred n'est définie que pendant le calcul du produit de pred 3 et pred 4.

`let pred_carre x =`
`pred ( x*x) where pred x = x-1;;`
`#pred_carre : int -> int = <fun>`	

`#let puissance4 x =`
`let carre x = x*x in carre(carre x);;`
`puissance4 : int -> int = <fun>`	

Exercice : Conversion de degrés Celsius en degrés Fahrenheit sachant que :

$$F = 9 * \frac{C}{5} + 32$$

1.6.5.) FONCTIONS A PLUSIEURS ARGUMENTS.

Les fonctions possédant plusieurs arguments ont simplement plusieurs noms d'arguments dans leur définition.

`#let moyenne a b c d = (a+b+c+d) / 4;;` `moyenne : int -> int -> int -> int -> int = <fun>`	`#let moyenne = function a -> function b -> function c -> function d -> (a+b+c+d)/4;;` `moyenne : int -> int -> int -> int -> int = <fun>`
`#let perimetre longueur largeur = 2 * (longueur+largeur);;` `perimetre : int -> int -> int = <fun>`	`#let perimetre = function longueur -> function largeur -> 2 * (longueur+largeur);;` `perimetre : int -> int -> int = <fun>`

On peut également spécifier le type au moment de la déclaration de la fonction :

```
#let (perimetre : int->int->int) = function longueur->function
largeur -> 2 * (longueur+largeur);;
perimetre : int -> int -> int = <fun>
```

1.6.6.) FONCTIONS DE CONVERSION.

Il existe des fonctions de conversion de types. Elles sont toujours de la forme type1_of_type2. Par exemple :

```
#let a = string_of_int 5;;
a : string = "5"
#let a = int_of_string "5";;
a : int = 5
#let a = int_of_char `A`;;
a : int = 65
#let a = char_of_int 72;;
a : char = `H`
```

1.6.7.) FONCTIONS CURRYFIEES.

Une fonction qui reçoit ses arguments un par un est dite curryfiée (du nom du logicien Haskel Curry) :

```
let perimetre longueur largeur = 2 * (longueur+largeur);;
perimetre : int -> int -> int = <fun>
```

Par contre, une fonction qui reçoit tous ses arguments en même temps est dite non-curryfiée :

```
#let perimetre (longueur,largeur) = 2 * (longueur+largeur);;
perimetre : int * int -> int = <fun>
```

Le néologisme "curryfiée" vient du nom du logicien Haskel Curry.

Une fonction curryfiée est donc une fonction qu'on peut appliquer partiellement.

Exercice :

a) Ecrire une fonction qui calcule la solution de l'équation `ax+b = 0`.

```
#let sol a b = -.b /. a;;
sol : float -> float -> float = <fun>
```

b) Ecrire une fonction qui calcule les solutions de l'équation `(ax+b)(cx+d) = 0.`

```
#let sol1 a b c d = (sol a b, sol c d);;
sol1 : float -> float -> float -> float -> float * float = <fun>
```

1.6.8.) FONCTIONS ANONYMES.

En Caml, on n'est pas obligé de nommer les fonctions. On peut utiliser des fonctions anonymes comme dans l'exemple suivant :

```
#function x -> x+1;;
- : int -> int = <fun>
```

L'application d'une fonction anonyme se fera ainsi :

```
#(function x -> x+1) 2;;
- : int = 3
```

Quand on a plusieurs arguments :

```
#(function x -> function y -> function z ->x*y*z) 3 2 4;;
- : int = 24

#let superieur a b = a>b;;
superieur : 'a -> 'a -> bool = <fun>
```

Une fonction anonyme est donc une expression fonctionnelle.

1.6.9.) FONCTIONS PARTIELLES.

Une fonction partielle est une fonction qui n'est pas totalement définie sur l'ensemble des valeurs du type correspondant à l'ensemble de définition.

Par exemple, la fonction inverse qui à x associe 1/x n'est pas définie pour x=0.

1.6.10.) VARIABLES DE TYPE.

Lors de la détermination du type d'une fonction, Caml recherche toujours le type le plus général. Parfois, il n'est possible de déterminer le type des arguments :

```
#let compare a b = a>b;;
compare : 'a -> 'a -> bool = <fun>

#let g a = a;;
g : 'a -> 'a = <fun>
```

Dans ce cas, le symbole `'a` représente n'importe quel type. Dans l'exemple suivant a,b,c et d sont de n'importe quel type mais a est du même type que b et c du même type que d. Caml utilisera donc deux variables de type :

```
#let compare a b c d = (a>b) && (c>d);;
compare : 'a -> 'a -> 'b -> 'b -> bool = <fun>

#let g a b c = (a,b,c);;
g : 'a -> 'b -> 'c -> 'a * 'b * 'c = <fun>
```

1.6.11.) VARIABLES DE TYPE FAIBLES.

En plus des variables de type, Caml introduit également des variables de type faibles qu'il note `_a, `_b, `_c.. . Il s'agit de types qu'on pourrait qualifier de "types inconnus en attente d'être connus".

```
#let a = ref [];;
a : '_a list ref = ref []
```

1.6.12.) RESULTAT D'UNE FONCTION.

Dans l'instruction suivante, print_string "a" retourne le type "unit", tandis que 5*3 retourne le nombre 15 de type entier :

`#print_string "a";;` `a- : unit = ()`	`#5*3;;` `- : int = 15`

Si on combine les expression ainsi, Caml ignore le type unit().

```
#let f () = (print_string "a";5*3);;
f : unit -> int = <fun>
```

Par contre, si on les combine ainsi, Caml prévient que le type int sera ignoré.

```
#let f () = (5*3;print_string "a");;

Entrée interactive:
>let f () = (5*3;print_string "a");;
>            ^^^
Attention: cette expression est de type int,
mais est utilisée avec le type unit.
f : unit -> unit = <fun>
```

1.6.13.) L'INSTRUCTION "CONDITIONNELLE".

Une construction de base dans un langage de programmation est le branchement conditionnel :

`if` *`condition`* **`then`** *`action1`* **`else`** *`action2`*

Si le booleen *condition* est vrai, c'est l'expression *action1* qui sera évaluée sinon c'est *action2*.

<conditionnelle> ::= if <expression> **then** <expression>
else <expression>

`#let inverse x =` `if x<>0 then 1 / x else 0;;`	Fonction inverse.

`#let xor x y =`
`if (x=true) && (y=true) then false`
`else`
`if (x=true) && (y=false) then true`
`else`
`if (x=false) && (y= true) then true`
`else false;;`
	Définition de la fonction xor.

Exercice :

Ecrire la fonction `valeur_absolue(x)` qui renvoie |x|.

1.6.14.) L'ALTERNATIVE A N BRANCHES.

```
#let f i =
match i with
| 1 -> "Un"
| 2 -> "Deux"
| 3 -> "Trois"
| 4 | 5 | 6-> "Quatre, cinq ou six"
| i when i mod 2 = 0 -> "Nombre pair"
| _ -> "Autre nombre";;
f : int -> string = <fun>
#f 3;;
- : string = "Trois"
```

La forme "`i when i mod 2 = 0`", est appelée "*filtrage gardé*" ou "*surveillant*".

Exercice :

Ecrire une fonction qui reçoit l'année et le mois en paramètres et renvoie le nombre de jours dans le mois.

1.7.) Les effets.

Il existe des fonctions dont le but n'est pas d'effectuer des calculs mais de produire des *effets*, c'est à dire une action comme l'affichage d'un message ou l'écriture dans un fichier. En voici quelques-unes unes :

print_string	print_string "essai";;
print_char	print_char `a`;;
print_int	print_int 10;;
print_float	print_float 10.5;;
print_newline	print_newline();;
read_line	let s = read_line();;
read_int	let i = read_int();;
read_float	let f = read_float();;

Comme toutes les fonctions, ces fonctions renvoient une valeur. Celle-ci est généralement égale à la valeur "unit" qui signifie "rien".

```
#print_string "machin";;
machin- : unit = ()
```

L'ordre d'exécutions des effets dans une expression n'est pas garanti par le compilateur :

```
#(print_int 2;2)*(print_int 3;3)+(print_int 4;4)+(print_int 5;5);;
5432- : int = 15

      #let a = (print_int 2;2) and b =(print_int 3;3)
      and c=(print_int 4;4) and d = (print_int 5;5) in a*b*c*d;;
      2345- : int = 120
```

Exercice :

Ecrire une fonction qui reçoit jour,mois,année en paramètres et écrit la date sous la forme "le jj mmmm yyyy").

1.8.) Les délimiteurs.

Caml permet l'emploi des délimiteurs ou de la paire `begin...end`. On s'en servira pour :

- Regrouper des instructions séparées par des ;

```
#let f a =
begin
      print_string "La valeur de a
      est ";
      print_int a;
      print_string " celle de 2*a
      est ";
      print_int (2*a);
      print_newline();
end;;
f : int -> unit = <fun>
```

```
#let f a =
(
print_string "La valeur de a est ";
print_int a;
print_string " celle de 2*a est ";
print_int (2*a);
print_newline();
);;
f : int -> unit = <fun>
```

On peut très bien écrire :

```
#begin 2 + 3 end * 4;;
- : int = 20
```

- Limiter la portée d'une définition locale.

```
#let juillet97() =
let s = "britannique" in
let s = "chinoise" in
print_string   "Hong-Kong   est   une
ville ";
print_string s;
print_string " sous souveraineté ";
print_string s;
print_newline();;
juillet97 : unit -> unit = <fun>

#juillet97();;
#Hong-Kong  est  une  ville  chinoise
sous souveraineté chinoise
- : unit = ()
```

```
#let juillet97() =
let s = "britannique" in
begin
let s = "chinoise" in
print_string   "Hong-Kong   est   une
ville ";
print_string s;
end;
print_string " sous souveraineté ";
print_string s;
print_newline();;
juillet97 : unit -> unit = <fun>

#juillet97();;
Hong-Kong  est  une  ville  chinoise
sous souveraineté britannique
- : unit = ()
```

1.9.) Différences définition/variable.

```
let b = 2;;
b : int = 2
let f x = x + b;;
f : int->int = <fun>
f 1;;
- : int = 3
let b = 3;;
f 1
- : int = 3
```

1.10.) Conseils de programmation.

1.10.1.) BIEN PENSER L'ARCHITECTURE D'UN PROGRAMME.

Pour élaborer un programme, on procède en plusieurs étapes :

- Esquisser un premier "jet" de l'algorithme.
- Le traduire en pseudo-langage.
- L'améliorer pour lui donner une forme exécutable dans un langage de programmation.

Cette stratégie d'améliorations successives permet d'obtenir un programme final structuré et facile à maintenir.

1.10.2.) PROGRAMMER PAR MODULES.

Découpez un programme en fonctions élémentaires. Si une transformation devient un jour nécessaire, la partie à modifier sera plus facilement localisable.

1.10.3.) UTILISER OU ADAPTER DES PROGRAMMES EXISTANTS.

Rien n'est plus inefficace que de traiter un problème comme si personne n'avait jamais écrit une ligne sur le sujet. Il faut toujours chercher s'il n'existe pas déjà un programme traitant une partie, voire la totalité, du problème rencontré.
Inversement, il ne faut pas oublier que vos programmes seront peut-être un jour utilisé par quelqu'un d'autre et qu'ils doivent donc être facilement accessibles.

1.10.4.) FORGER SA "BOITE A OUTILS".

Conservez vos programmes, vos algorithmes en les classant par thèmes pour vous forger vos propres bibliothèques.

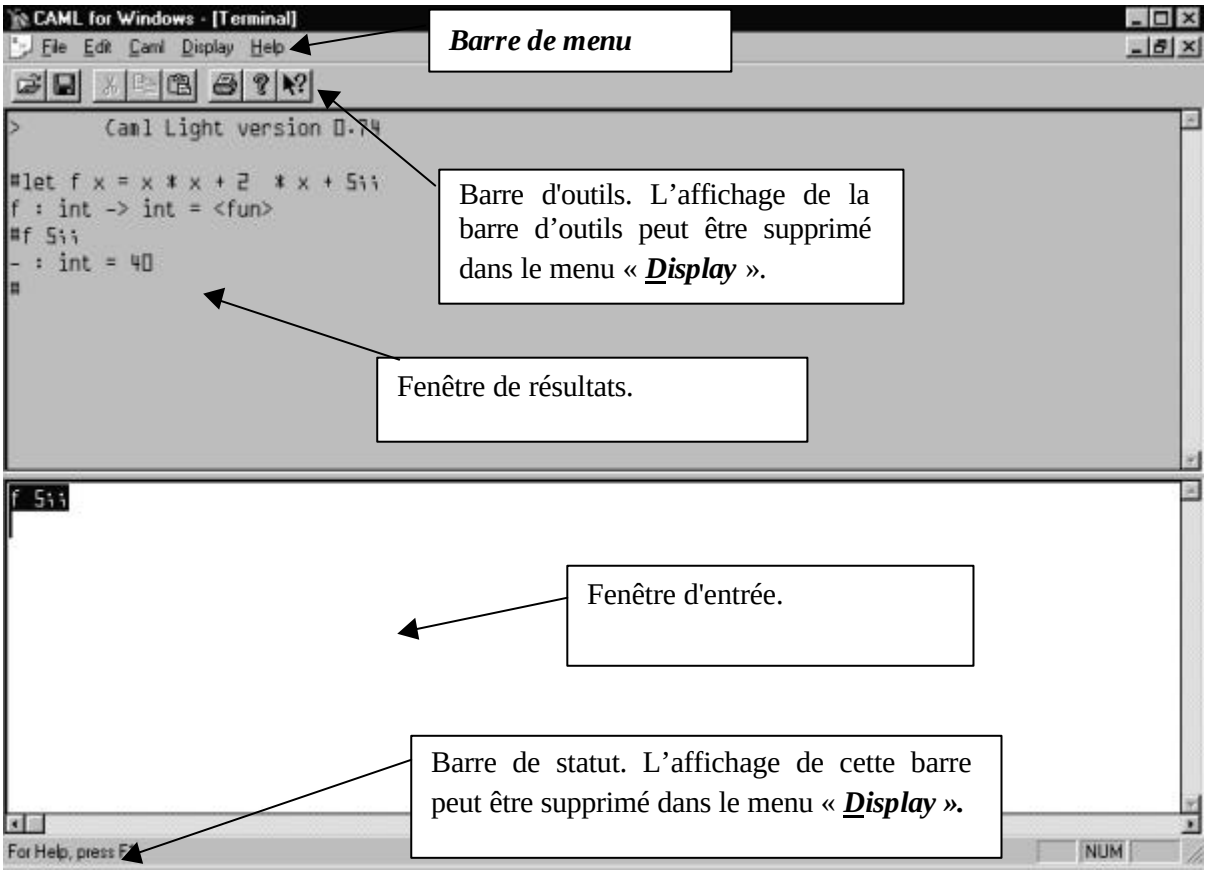
2.) Le système interactif.

Le système interactif Caml est basé sur une boucle de lecture-évaluation-impression: l'utilisateur entre une phrase, le système la compile et l'exécute au vol, puis imprime le résultat de l'évaluation. Le système interactif Caml se présente comme une application Windows nommée "Camlwin.exe".

Le "terminal" Windows se compose de deux parties :

- La partie basse dans laquelle les phrases peuvent être entrées.
- La partie haute qui affiche une copie de la phrase saisie ainsi que la ou les réponse(s) de Caml.

La combinaison des touches + envoie le contenue de la fenêtre du bas à Caml, tandis que la touche seule permet d'insérer une nouvelle ligne, sans rien envoyer à Caml. L'utilisation de ces touches peut se configurer dans le menu "Préférences". L'envoie d'une phrase à Caml peut également se faire par le menu « Caml / Send to CAML ».



Le contenu de la fenêtre d'entrée peut être édité à tout moment à l'aide des fonctionnalités « Copier/Coller » de l'interface Windows et des menus « File / Open », « File / Save ». Un historique des phrases entrées est tenu à jour et affiché dans une fenêtre séparée.

Pour quitter Caml, on utilise le menu « File / Exit » ou la fonction « quit( ) ».

A tout moment, la compilation ou l'évaluation de la phrase courante peut être interrompu en sélectionnant le menu « Caml / Interrupt CAML ».

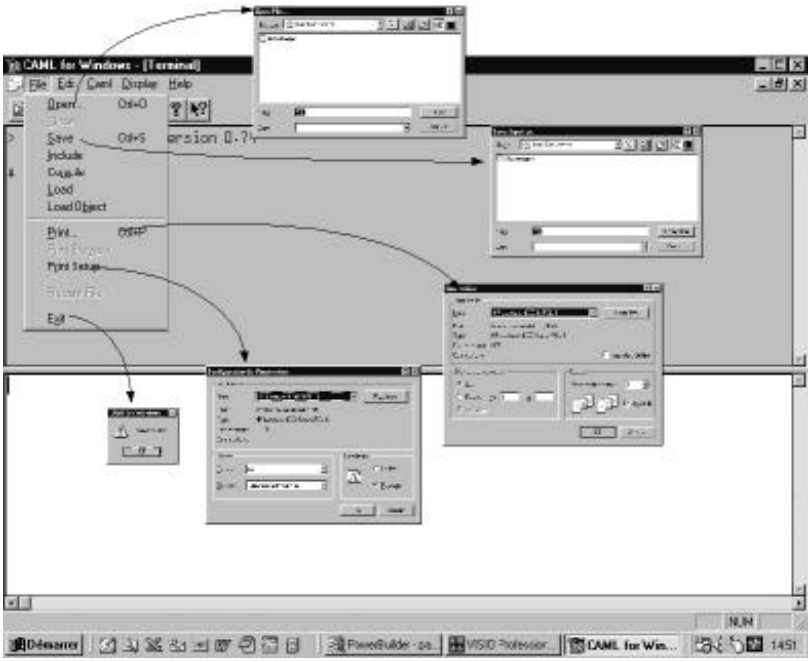


Figure 3 : Menu "File" de Caml.

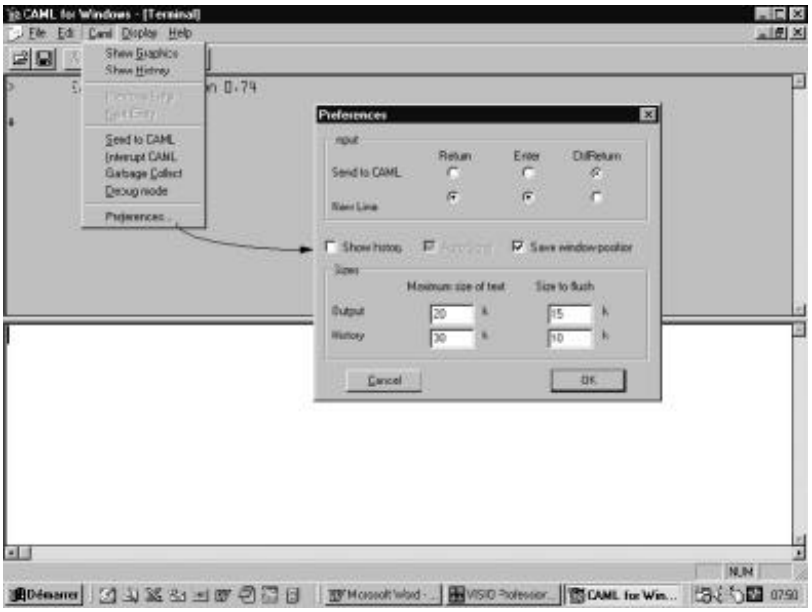


Figure 4 : Menu "Caml"

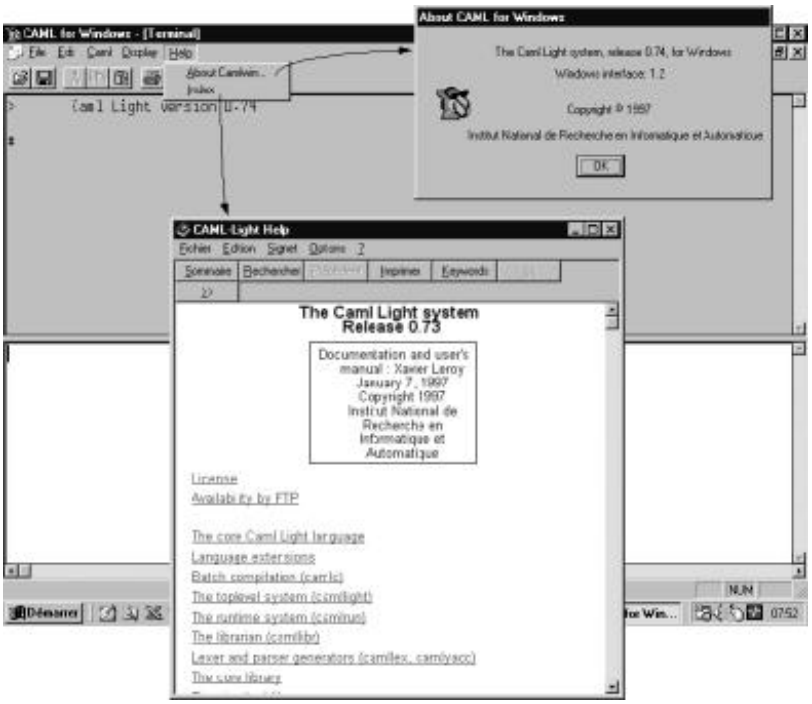
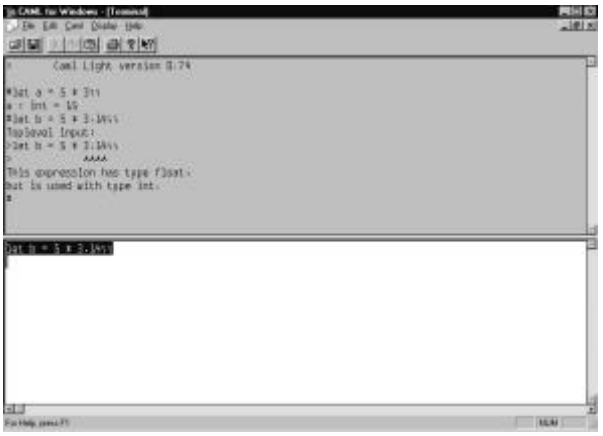


Figure 5 : Menu "Help"

Au démarrage, Caml indique le numéro de version :

Une fois qu'une phrase a été entrée, Caml répéte la phrase dans la fenêtre du haut et donne ensuite sa réponse :

Quand une erreur survient lors de l'analyse syntaxique, Caml souligne l'erreur avec des caractères " ^^^^ ".



3.) Exercices.

Qu'affiche le système interactif après :

```
#let a=
let a=3 and b=2 in
let a=a+b and b=a-b in a*a - b*b;;
a : int = 24
let b=2 in a*a+a*b+b;;
#- : int = 626
```

EXERCICE 3.0.1.:

Ecrire une fonction qui reçoit une année A en paramètre et renvoie TRUE si c'est une année bissextile et FALSE sinon. Une année est bissextile si elle est divisible par 4 et qu'elle n'est pas :

- divisible par 100 et pas par 400
- divisible par 4000.

```
#let divis_4 an =
if (an mod 4) = 0 then true else false;;
divis_4 : int -> bool = <fun>
#let divis_100 an =
if (an mod 100) = 0 then
begin
     if (an mod 400) = 0 then false else true;
end else false;;
divis_100 : int -> bool = <fun>
#let divis_4000 an =
if (an mod 4000) = 0 then true else false;;
divis_4000 : int -> bool = <fun>
#let bisextile an =
(divis_4 an) && (not (divis_100 an)) && (not (divis_4000 an));;
bisextile : int -> bool = <fun>
```

EXERCICE 3.0.2.:

En utilisant les définitions emboîtées en Caml, écrire la fonction f :

$f(x) = x^5-2x^4+3x^3-4x^2+5x-6$ pour $x \in \mathbb{N}$

```
#let f x = let p5 x = x*x*x*x*x  and
p4 x = x*x*x*x and p3 x = x*x*x and p2 x = x*x in
p5(x) - 2 * p4(x) + 3 * p3(x) - 4 * p2(x) + 5*x - 6 ;;
f : int -> int = <fun>
```

EXERCICE 3.0.3.:

En utilisant la formule de Zeller, écrire une fonction qui calcule le jour de la semaine.

```
#let js j m s a = (j+a-2*s+(a/4)+(s/4)+((26*m-2)/10)) mod 7;;
js : int -> int -> int -> int -> int = <fun>

#let jour_semaine j m a = if m>=3 then
js j (m-2) (a/100) (a mod 100) else
js j (m+10) ((a+1)/100) ((a+1)mod 100);;
```

EXERCICE 3.0.4.:

Ecrire une fonction qui calcule x^{11} en faisant le moins de multiplications possibles :

```
#let p11 x = let p2 = x*x in  let p4 = p2 * p2 in let p8 = p4 * p4
in p4 * p4 *p2*x;;
p11 : int -> int = <fun>
```

EXERCICE 3.0.5.:

Ecrire une fonction qui calcule la valeur absolue de x :

```
# #let valeur_absolue x = if x<0 then -x else x;;
valeur_absolue : int -> int = <fun>
```

EXERCICE 3.0.6.:

Ecrire une fonction qui convertit des heures/minutes en heures/centièmes.

```
#let convert (h,m) = (h, m*100/60);;
convert : 'a * 'b -> 'a * int = <fun>
```

Ecrire une fonction qui convertit des heures/centièmes en heures/minutes

```
#let convert (h,m) = (h, m*60/100);;
convert : 'a * int -> 'a * int = <fun>
```

EXERCICE 3.0.7.:

Ecrire une fonction qui calcule le lendemain d'une date.

```
#let max m a =
match m with
|1|3|5|7|8|10|12 -> 31
| 2 -> if bisextile a then 29 else 28
| _ -> 30;;
max : int -> int -> int = <fun>
#let lendemain j m a =
let jj = j+1 and maxi = max m a in
if (jj>maxi) && (m=12) then (1,1,a+1)
else
if (jj>maxi) then (1,m+1,a) else
(jj,m,a);;
lendemain : int -> int -> int -> int * int * int = <fun>
```

EXERCICE 3.0.8.:

Ecrire des fonctions permettant de calculer :

- le volume d'un cube (s^3, avec s= longueur d'une arrête),
- d'une sphère ($4*PI*r^3/3$ avec r = rayon de la sphère).

3.1.) Exercices.

EXERCICE 2.1 :

Evaluer les expressions suivantes :

```
#let a=5 in let b=2 in let c=a*b+b in a*b + c*a;;
#sqrt(a) where a=1.5*.b where
b=2. *. t where t=3.;;
#u x where x = 5 and u n = x*n where x = 3;;
```

EXERCICE 2.2 :

Calcul du poids idéal P en fonction de la taille T (en cm) et de l'âge A (en années) par les formules :

$$P = \frac{(3*T - 250)*(A + 270))}{1200}$$ pour les hommes

et

$$P = \frac{\left(\frac{T}{2} - 30\right)*(180 + A)}{200}$$ pour les femmes.

EXERCICE 2.3:

La fonction suivante, écrite en Pascal, calcule le quantième d'une date. Elle reçoit en paramètre l'année, le mois et le jour.

```
FUNCTION quantieme (y, m, d : INT) : INT;

VAR yy, mm, dd, Tmp1 : INT;

BEGIN
    yy := y;
    mm := m;
    dd := d;
    Tmp1 := (mm + 10) DIV 13;
    quantieme :=  3055 * (mm + 2) DIV 100 - Tmp1 * 2 - 91 +
                  (1 - (yy - yy DIV 4 * 4 + 3) DIV 4 +
                  (yy - yy DIV 100 * 100 + 99) DIV 100 -
                  (yy - yy DIV 400 * 400 + 399) DIV 400) * Tmp1 + dd
END;
```

L'instruction Pascal "x **DIV** y" calcule la division entière de "x" par "y". Elle correspond à l'instruction Caml "x / y". L'instruction "**:=**" est l'instruction d'affectation ; elle correspond à l'instruction "=" en Caml.

Réécrivez cette fonction en Caml.

EXERCICE 2.4 :

Ecrire une fonction qui renvoie les solutions d'une équation ax2+bx+c. Cette fonction recevra en paramètres les valeurs a, b, et c et renverra les solutions x et x'.

EXERCICE 2.5 :

La clé "C" d'un numéro Insee N est égale à (97-(N MODULO 97)). Ecrivez une fonction qui accepte en entrée les composantes du numéro Insee et calcule "C". La difficulté de l'exercice tient à l'intervalle de valeur dans lequel sont définis les entiers ([-2^{30}, 2^{30}-1], c'est à dire [-1073741824,1073741823]).