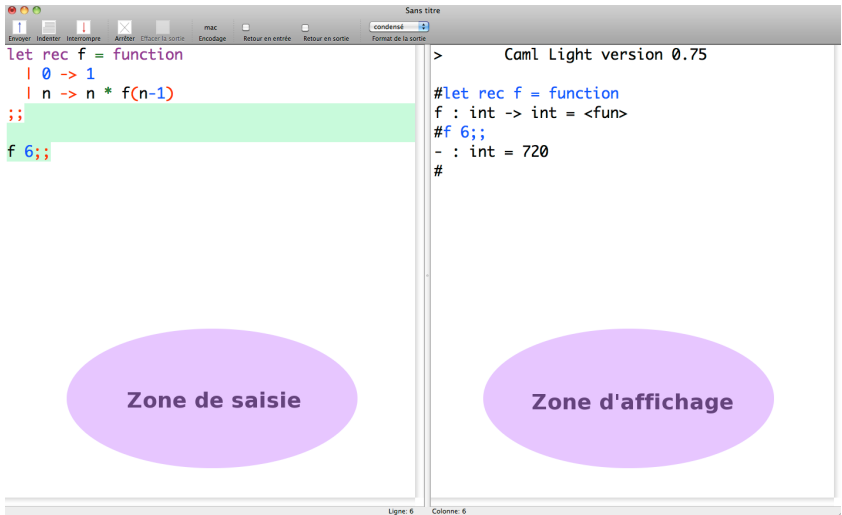


# Prise en main de CaML Light - 1

**Lycée Thiers 2015**

# L'interface



The screenshot shows the Caml Light 0.75 interface. The window is titled "Sans titre". The menu bar includes "Envoyer", "Indenter", "Interrompre", "Arrêter", "Effacer la sortie", "mac", "Encodage", "Retour en entrée", "Retour en sortie", and "Format de la sortie". The left pane, labeled "Zone de saisie" (input zone), contains the following code:

```
let rec f = function
  | 0 -> 1
  | n -> n * f(n-1)
;;
f 6;;
```

The right pane, labeled "Zone d'affichage" (output zone), shows the output of the code:

```
> Caml Light version 0.75

#let rec f = function
f : int -> int = <fun>
#f 6;;
- : int = 720
#
```

At the bottom of the window, the status bar indicates "Ligne: 6" and "Colonne: 6".

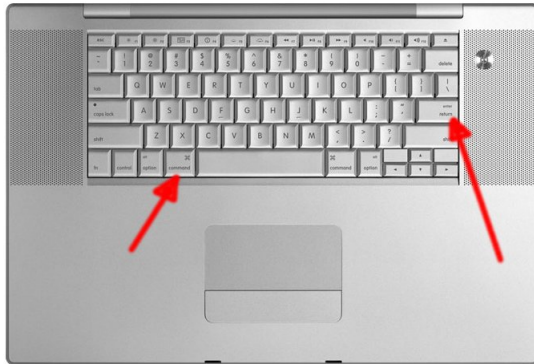
# Validation $\neq$ Saut de ligne

- 1 Phrases terminées par `;;` puis validation
- 2 Une phrase peut occuper plusieurs lignes

# Sur PC ...



# Sur Mac ...



# Premier essai !

```
2 * 5;;  
- : int = 10
```

```
2.0 *. 5.0;;  
- : float = 10.0
```

```
2. * 5.;;  
^^
```

Cette expression est de **type** float,  
mais est utilisée avec le **type** int.

# Qu'est-ce qu'un TYPE ?

Définition informelle :

***Ensembles de valeurs, partageant des propriétés communes ...***

Analogie avec les maths :

- nombres entiers naturels
- nombres réels
- couples d'entiers relatifs
- sous-ensembles de  $\mathbb{R}$
- applications de  $\mathbb{R}$  dans  $\mathbb{R}$

# Principaux types “simples”

- int  $[-2^n, 2^n - 1]$ , calcul exact, overflow
- float  $\pm m \times 2^e$ , calcul approché
- bool {true, false}
- char Caractères
- string Chaînes de caractères
- unit ... Rien !



# Le type int et l'overflow

- En maths :

$$100000^3 = 10000000000000000$$

$$100000^4 = 100000000000000000000$$

- Avec CaML (architecture 64 bits) :

```
100000 * 100000 * 100000;;  
- : int = 10000000000000000
```

```
100000 * 100000 * 100000 * 100000;;  
- : int = -145709240540253388
```

# Le type float et les arrondis

- En maths :

$$1 + 2^{-52} \neq 1$$

$$1 + 2^{-53} \neq 1$$

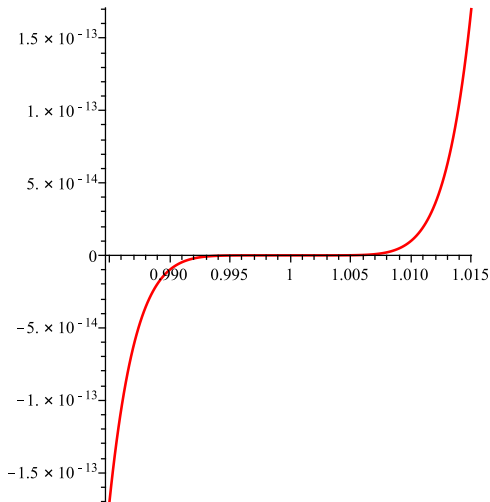
- Avec CaML :

```
1. +. 2. ** (-52.) = 1.;;  
- : bool = false
```

```
1. +. 2. ** (-53.) = 1.;;  
- : bool = true
```

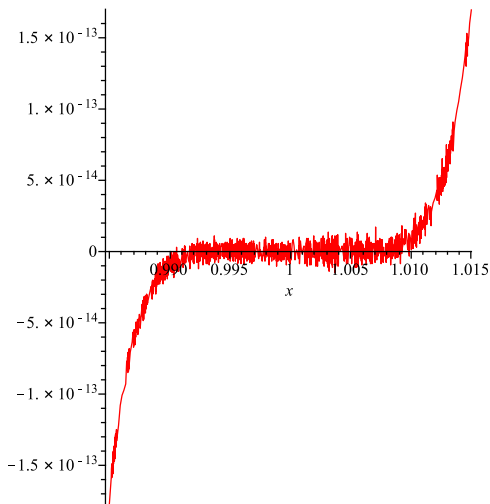
# Le type float et les arrondis

`plot((x - 1)7, x = 0.985..1.015)`



# Le type float et les arrondis

```
plot( $x^7 - 7 \star x^6 + 21 \star x^5 - 35 \star x^4 + 35 \star x^3 - 21 \star x^2 + 7 \star x - 1$ ,  $x = 0.985..1.015$ )
```



# Opérateurs pour les types simples

- Pour le type **int** :

addition	+
soustraction	-
multiplication	*
quotient	/
reste	mod

- Pour le type **float** :

addition	+.
soustraction	-.
multiplication	*.
division	/.
exponentiation	**

# Opérateurs pour les types simples

- Pour le type **bool** :

ET	&
OU (inclusif)	or
NOT	not

- Pour le type **string** :

concaténation	^
---------------	---

# Opérateurs de comparaison

Test d'égalité	=
Test d'infériorité stricte	<
Test d'infériorité large	<=
Test de supériorité stricte	>
Test de supériorité large	>=
Test d'inégalité	<>

Ces opérateurs sont **POLYMORPHES**

## ① Liaisons globales

```
let pi = 3.14;;
```

## ② Liaisons locales

```
let pi = 3.14 in  
  pi + 1.0;;
```

## ③ Liaisons locales imbriquées

```
let pi = 3.14 in  
  let r = 5.0 in  
    2.0 *. pi *. r;;
```



# Quelques types “paramétrés”

- 'a ref
- 'a \* 'b
- 'a vect
- 'a list

Référence vers une valeur

Couples

Vecteurs

Listes

→ **POLYMORPHISME**

# Utilisation des références

```
let a = ref 7;;  
a : int ref = ref 7
```

```
!a;;  
- : int = 7
```

```
a := 8;;  
- : unit = ()
```

```
a := !a + 1;;  
- : unit = ()
```

```
!a;;  
- : int = 9
```

## ❶ Syntaxe complète

```
let f = function x → 2 * x;;
```

## ❷ Syntaxe abrégée

```
let f x = 2 * x;;
```

## ❸ Application des fonctions

```
f (5);;  
- : int = 10
```

```
f 5 + 10;;  
- : int = 20
```

```
f (5 + 10);;  
- : int = 30
```

# Type d'une fonction

```
let f x = 2 * x;;  
f : int -> int = <fun>
```

```
let g x = x ** 3.0;;  
g : float -> float = <fun>
```

```
let h str = (str.[0], string_length str);;  
h = string -> char * int = <fun>
```

# Moteur d'inférence de type

– en C –

```
float f (int n, float x) {  
    int k;  
    float y;  
    k = n + 2;  
    y = x * 10;  
    if (k < 10)  
        return y;  
    else  
        return y + 1;  
}
```

– en CaML –

```
let f n x =  
  let k = n + 2 in  
  let y = x *. 10.0 in  
  if (k < 10) then  
    y  
  else  
    y +. 1.0  
;;
```

# Fonctions de plusieurs variables

Version  $A \times B \rightarrow C$

```
let f (x,y) = 2 * x + 3 * y;;  
f : int * int → int = <fun>
```

Analogie mathématique :

$$f : \mathbb{Z}^2 \rightarrow \mathbb{Z}, (x, y) \mapsto 2x + 3y$$

$$f(x, y) = 2x + 3y$$

# Fonctions de plusieurs variables

Version  $A \rightarrow C^B$

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

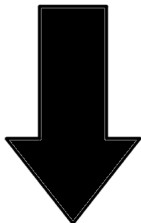
Analogie mathématique :

$$f : \mathbb{Z} \rightarrow \mathbb{Z}^{\mathbb{Z}}, x \mapsto [y \mapsto 2x + 3y]$$

$$[f(x)](y) = 2x + 3y$$

# Curryfication

```
let f (x,y) = 2 * x + 3 * y;;
```



```
let f x y = 2 * x + 3 * y;;
```



```
let evaluate u x = u x;;
```

→ **CaML**

```
let compose u v x = u (v x);;
```

→ **CaML**

The End!