TP n° 1 : Rencontre avec CAML

L'objectif de ce premier TP est de vous familiariser avec l'environnement de programmation que nous utiliserons pour l'option informatique et de vous permettre de faire vos premiers pas en CAML. Le contenu de ce TP fait partie intégrante du cours et il faut le considérer comme tel. Vous n'hésiterez pas à chercher à prévoir le type et la valeur d'une expression avant de l'évaluer avec CAML.

1 Environnement de programmation

Pour les TP d'option informatique nous utiliserons EMACS avec le mode TUAREG [1]. EMACS est un éditeur de texte très populaire parmi les programmeurs chevronnés, l'un des deux belligérants de la guerre des éditeurs (son opposant étant VI).

On lance EMACS depuis un invite de commandes (un terminal ; que l'on ouvre en cliquant sur le petit carré noir le plus à droite en bas à gauche ; allez-y).

Nous vous conseillons de créer un répertoire dédié aux TP d'option informatique

```
:~> ls .            # liste de ce qui se trouve dans le répertoire courant "."
:~> mkdir opt_info  # créer un répertoire de ce nom
:~> ls .            # on peut voir que le répertoire a bien été créé
:~> cd opt_info     # aller dans ce répertoire
:~/opt_info> ls .   # il n'y a rien !
:~/opt_info> ls ..  # liste de ce qui se trouve dans le répertoire parent ".."
:~/opt_info> cd ..  # on se retrouve dans le répertoire parent ".."
:~> ls .            # ah oui !
:~> cd opt_info     # retournons dans opt_info
:~/opt_info> emacs tp1.ml  # créer et ouvrir le fichier tp1.ml
    # On peut y écrire par exemple : (* Mon premier commentaire *)
    # CTRL-X, CTRL-S pour sauvegarder
    # CTRL-X, CTRL-C pour quitter
:~/opt_info> ls .   # on doit voir tp1.ml
:~/opt_info> emacs tp1.ml  # ouvrir le fichier tp1.ml
```

Dans ce ficher `tp1.ml` il y a donc une première ligne contenant

```
(* Mon premier commentaire *)
```

Vous savez donc écrire des commentaires [2] en CAML.

Sur une nouvelle ligne écrivez maintenant :

```
let x = 42;;
```

Laissez le curseur en fin de ligne et effectuez la combinaison de commandes CTRL-C puis CTRL-E. Normalement vous devez avoir un message en bas de la fenêtre qui vous demande :

```
Caml toplevel to run : camllight
```

1. C'est aussi l'environnement proposé aux concours des ENS, les seuls pour lesquels il y a une épreuve sur machine.
2. Connaître la syntaxe des commentaires est la toute première chose à laquelle s'intéresse tout bon informaticien devant un nouveau langage.

Comme c'est ce qu'il nous faut, il suffit de valider (touche retour chariot). EMACS vous répond

```
Symbol's function definition is void : make-local-hook
```

Effectuez à nouveau la combinaison de commandes CTRL-C-CTRL-E. Vous devez alors avoir une nouvelle fenêtre qui s'ouvre avec le résultat de l'évaluation de cette liaison globale. Comprenez-vous bien la réponse de CAML ?

Sur un nouvelle ligne entrez le texte suivant :

```
x = 43;;
```

Que va répondre CAML ? Vérifiez à l'aide de CTRL-C-CTRL-E. Si vous êtes surpris, c'est normal, à la fin du TP vous aurez bien compris.

Voici les commandes les plus importantes sous EMACS :

Ouvrir/créer un fichier	CTRL-X, CTRL-F
Sauvegarder un fichier	CTRL-X, CTRL-S
Sélection	CTRL-ESPACE
Couper	CTRL-W
Copier	ALT-W
Coller	CTRL-Y
Supprimer la ligne courante	CTRL-K
Annuler (dernière action)	CTRL-_
Annuler (commande en cours)	CTRL-G
Quitter	CTRL-X, CTRL-C

Et les raccourcis spécifiques à TUAREG pour lancer CAML :

Interpréter la phrase courante	CTRL-C, CTRL-E
Interpréter la région courante	CTRL-C, CTRL-R
Tout interpréter	CTRL-C, CTRL-B
Interruption	CTRL-C, CTRL-K

2 Les types et opérations de base

Dans ce TP, nous proposons des expressions mais pas leur évaluation en CAML. Pour chaque expression, vous devez donc prévoir le résultat de son évaluation et vérifier avec CAML.

Les types de base[3] en CAML sont les types **`int`**, **`float`**, **`bool`**, **`char`** et **`string`**.

2.1 Le type `int`

Les entiers sont représentés par le type **`int`** :

```
1;;
```

On dispose des opérateurs `+`, `-`, `*`, `/` et **`mod`**

3. Il en manque un que nous verrons plus tard.

```
1 + 1;;
3 - 5;;
2 * 10;;
10 / 3;;  (* Attention : division entière ! *)
10 mod 3;;
```

Vous aurez bien entendu prévu et vérifié la réponse de CAML pour toutes ces expressions avant de passer à la suite.

Attention : on met des espaces autour de *tous* les opérateurs binaires.

Les opérateurs `*`, `/` et `mod` sont prioritaires par rapport à `+` et `-`, sinon la règle d'association à gauche s'applique. Ajoutez les parenthèses implicites :

```
1 + 42 mod 3 + 42 / 2 * 5 / 5
```

Il est important de connaître les priorités des opérateurs et constructions CAML. Mais en cas de doute : utiliser des parenthèses. Lorsque les parenthèses ne sont pas nécessaires, la règle est la suivante : « On écrit les parenthèses si et seulement si le code s'en trouve plus lisible ».

En CAML les entiers sont codés sur 31 bits [4] (dont un bit de signe).

QUESTION 1 En CAML :

- Que vaut $1024 \times 1024 \times 1024$?
- Que vaut $1024 \times 1024 \times 1024 - 1$?

Expliquez. Et en PYTHON ?

2.2 Le type `float`

Certains [5] réels sont représentés par le type `float` :

```
1.0;;
1.;;
2e10;;
```

Les opérations précédentes sur les entiers ne peuvent pas s'appliquer sur des flottants. C'est une des conséquences du fait que CAML est fortement typé. Il faut utiliser les opérateurs sur les flottants qui sont obtenus en suffixant un "." : `+.`, `-.`, `*.`, `/.` et l'on dispose de plus de l'opérateur d'exponentiation `**` (sans point mais qui n'est applicable qu'aux flottants [6] !). Détectez les erreurs :

```
2.0 + 2.0;;
1. / 1.;;
2. +. 2;;
2.0 ** 3;;
(2.0 + 1.) *. (3. ** 3.);;
2 ** 10;;
```

Le caractère fortement typé de CAML interdit toute conversion implicite. Lorsque cela est vraiment nécessaire, on dispose cependant de fonctions qui permettent une conversion explicite :

4. Ou 63 bits sur les processeurs 64-bits.
5. Pourquoi certains ?
6. Il n'y a pas d'équivalent pour les entiers, mais nous allons bientôt en écrire.

```
int_of_float 10.;;
float_of_int 10;;
```

QUESTION 2

- A-t-on toujours `float_of_int (int_of_float x) = x;;` ?
- Et `int_of_float (float_of_int x) = x;;` ?
- Quel est l'arrondi pratiqué par CAML ?

On dispose de nombreuses fonctions à variables « réelles ». On rappelle qu'en CAML on utilise une notation proche de celle mathématique pour le sinus. Ainsi on n'écrit pas `sin(x)` mais bien `sin (x)` ou plutôt `sin x` lorsque les parenthèses sont inutiles. Voici quelques fonctions usuelles. En commentaires est indiqué le parenthésage implicite CAML et, attention, il y a quelques erreurs volontaires :

```
sqrt 25.;;            (* (sqrt 25.0) *)
cos 0.0 + sin 0.0;;   (* (cos 0.0) + (sin 0.0) *)
tan 1;;
acos (cos 1.);;       (* et pas acos cos 1. qui est (acos cos) 1. *)
sin asin 0.0;;        (* vous voyez ? *)
let pi = 4.0 *. atan 1.0;;
log (exp 1.) -. exp (log 1.);;
```

Attention, comme souvent en informatique `log` désigne bien le logarithme népérien (sinon on note plutôt $\log_2$ ou $\log_{10}$ si l'on souhaite, respectivement, les logarithmes en base 2 et 10).

2.3 Le type `bool`

Le type booléen **`bool`** comporte les deux valeurs `true` (VRAI) et `false` (FAUX). On dispose de l'opérateur unaire `not` (NON) et des opérateurs binaires `&&` (ET) et `||` (OU)[7].

```
true;;
((not true) || false) && (false && true);;
3 > 3;;
(2 = 2) || (3 <> 4) || (1 <= 2) || (2 >= 1);;
2.2 > 2.1 && 2.1 > 2.0
3.0 *. 0.1 = 0.3;;   (* alors ? *)
```

Ici vous devriez être choqués. Alors que CAML se vante d'être un langage fortement typé, comment se fait-il que = ou > soient valables pour les entiers *et* pour les flottants (et pour bien d'autres types), alors que l'on s'attendrait à des opérateurs `=.` et `>.` ? Nous expliquerons cela prochainement. Mais attention :

```
3.0 = 3;;
2 < 4.;;
```

ATTENTION : en CAML = (resp. <>) est l'opérateur de comparaison (resp. différence) qui se dit == (resp. !=) en PYTHON[a]. En PYTHON, = est l'opérateur d'*affectation*, là où l'on utilise la construction **`let`** en CAML.

a. == et != existent aussi en CAML et vous n'aurez pas d'erreur de compilation. Mais ils désignent la comparaison *physique*, c'est-à-dire le fait que les objets sont exactement au même endroit en mémoire. Ne pas les utiliser !

7. L'opérateur | se lit aussi (OU) mais son usage est celui de la définition par cas des fonctions. L'opérateur **`and`** a un autre usage que nous verrons prochainement. Les opérateurs & et **`or`** sont également valables mais vous ne devez pas les utiliser (préférez && et || qui sont plus cohérents). Ne pas se mélanger les pinceaux avec les opérateurs **`or`** et **`and`** de PYTHON ne sera pas facile au début.

Remarque 1. *Les opérateurs* `&&` *et* `||` *sont dits* paresseux, *c'est-à-dire que les expressions ne sont évaluées que si nécessaire. Ainsi, si* `expr1` *est fausse dans* `expr1 && expr2`, `expr2` *ne sera jamais évaluée. De même si* `expr1` *est vraie dans* `expr1 || expr2`, `expr2` *ne sera jamais évaluée.*

QUESTION 3

`e1 && e2` signifie `if e1 then e2 else false`

Traduisez de même `e1 || e2` à l'aide d'une expression conditionnelle.

2.4 Le type `char`

CAML dispose des caractères qui sont de type `char` :

```
'a';;
```

On dispose de fonctions de conversions depuis et vers les codes ASCII :

```
int_of_char 'a';;
char_of_int 32;;
```

2.5 Le type `string`

Il existe un type prédéfini pour les chaînes de caractères [8] :

```
let hello = "Hello world!";;
let hello = "Hello" ^ " " ^ "world" ^ "!";;
```

L'opérateur ^ permet de concaténer deux chaînes. Trois fonctions sont assez utiles :

```
string_length hello;;
nth_char hello 2;;
sub_string hello 0 5;;
```

QUESTION 4

Quel est le type de ces trois fonctions ?

Mais attention à l'encodage des caractères non ASCII :

```
let moi = "élève";;
string_length moi;;
```

2.6 Types produits

Comme en Python on peut construire des tuples. Observez bien les types obtenus :

```
(1, 2);;
("Zéro", '0', 0, 0.);;
(2, 3) = (1 + 1, 3);;
(2.0, 1) = (2, 1.0);;
```

8. Tout bon cours de programmation doit commencer par un programme affichant `"Hello world!"`.

3 Liaisons

3.1 Liaisons globales

Nous avons déjà rencontré plusieurs exemples de liaisons globales qui, rappelons-le, sont définies pour toute la durée de vie du programme.

```
let y = "y";;
y;;
x;;  (* On devrait voir 42 défini plus haut *)
```

3.2 `let` déstructurant

La construction `let` peut également être utilisée pour « déconstruire » un tuple, c'est le moyen pour accéder aux champs d'un tuple :

```
let paire = (3, 2);;
let (x, y) = paire;;
```

Lorsque l'un des champs ne nous intéresse pas, on peut utiliser la variable spéciale _ qui sert de poubelle.

```
let coordonnees = (3.0, 2.0, 10.0);;
let (abscisse, _, _) = coordonnees;;
```

3.3 Liaisons locales

Les liaisons locales masquent temporairement les liaisons englobantes et sont valables dans la portée du `in` :

```
x;;
let x = 0 in x * x;;
x;;
let schtroumpf = "bleu" in "Couleur : " ^ schtroumpf;;
schtroumpf;;
```

Les liaisons locales sont des *expressions* de la forme `let identifiant = expr1 in expr2`. C'est une expression comme une autre, on peut donc par exemple l'utiliser pour une liaison globale :

```
let quatre = (let x = 2 in x * x);;
let quatre = let x = 2 in x * x;;
```

Si `expr1` et `expr2` sont courtes on peut tout écrire sur une seule ligne. Sinon, on préfère utiliser plusieurs lignes en indentant le corps du `let`.

```
let trente =
  let quinze =
    let cinq = 2 + 2 + 1 in
    let deux = 1 + 1 in
    let sept = cinq + deux in
    cinq + deux + sept + 1
  in
  quinze + quinze
;;
```

QUESTION 5 Combien de liaisons globales et locales y a-t-il dans l'exemple précédent ?

4 Fonctions

On peut définir une fonction à l'aide du mot clef **function** :

```
let f = function x -> x - 1;;
```

On peut aussi utiliser la syntaxe équivalente :

```
let f x = x - 1;; (* "soit pour tout x, f(x) = x - 1", avec parenthésage Caml *)
```

On a aussi la définition par cas :

```
let nombre = function
  | 0 -> "Zéro"
  | 1 -> "Un"
  | n -> "Nombre incroyablement grand"
;;
```

On peut aussi définir des fonctions de plusieurs variables en utilisant les tuples [9] :

```
let produit = function (x, y, z) -> x *. y *. z;;
let produit (x, y, z) = x *. y *. z;;
produit (1.0, 2.0, 3.0);;
```

Bien sûr, l'expression calculée par une fonction peut être plus conséquente :

```
let sup i =
  let mpsi = "MPSI" in
  let pcsi = "PCSI" in
  if (i >= 1) && (i <= 3) then
    mpsi ^ "-" ^ (string_of_int i)
  else if (i = 4) || (i = 5) then
    pcsi ^ "-" ^ (string_of_int (i - 3))
  else
    "Pas une sup !"
;;

(sup 3) ^ " meilleure sup [19/10/16]";;
```

Ce qui nous amène aux expressions conditionnelles.

5 Conditionnelles

Une expression conditionnelle s'écrit **if** `expr1` **then** `expr2` **else** `expr3`. Bien entendu `expr1` doit être de type **bool** et `expr2` et `expr3` doivent être de même type.

```
let abs = function x -> if (x > 0.) then x else -.x;;
let grrr = if "oui" = "non" then "o" else 'n';;
```

9. Ce n'est pas la bonne manière de faire en CAML, comme nous le verrons au prochain cours.

Un expression conditionnelle est une *expression*, elle est évaluée pour produire une valeur. Inversement `expr1`, `expr2` et `expr3` peuvent être des expressions quelconques (bien typées), par exemple... des expressions conditionnelles !

QUESTION 6 Dans l'expression conditionnelle suivante :

```
if let x = 42. in x *. x > 1700. then
  if "Caml" > "Python" then 12 else 21
else if true then
  0
else let y = 1 in 1 - y * y
;;
```

identifiez `expr1`, `expr2` et `expr3`. N'hésitez pas à ajouter les parenthèses qui vous permettrons d'y voir plus clair.

6 Exercices

EXERCICE 1

Écrire une fonction `implique : bool * bool -> bool` représentant l'opérateur logique $\Rightarrow$.

EXERCICE 2

1. En utilisant une affectation locale, calculer $\dfrac{1+\sqrt{2}+\sqrt{2}^3}{e^{\sqrt{2}}-1}$.
2. Écrire une fonction `th : float -> float` qui calcule la fonction tangente hyperbolique en effectuant un unique appel à la fonction `exp`.

EXERCICE 3

En utilisant l'opérateur `**` et les fonctions de conversions, écrire une fonction `puissance : int * int -> int` telle que l'appel `puissance (n, k)` renvoie la valeur n^k. Quelle est la limite de cette approche ?

EXERCICE 4

Définir deux fonctions `est_minuscule` et `est_majuscule` qui testent si le caractère ASCII passé en entrée est une lettre minuscule, respectivement une lettre majuscule. Quel doit être leur type ? On pourra consulter sur internet la table ASCII et utiliser la fonction `int_of_char`.

Définir la fonction `majuscule : char -> char` qui renvoie la version majuscule d'une lettre si son argument est une lettre minuscule ou le caractère lui-même sinon. On pourra utiliser `char_of_int`.

EXERCICE 5

Prévoir et vérifier le résultat des expressions suivantes :

1. `let x = 1 in let x = 2 in x;;`
2. `let x = 1 in let y = x in let x = 2 in 10 * x + y;;`
3. `let y = 1 in let x = y + 1 in let y = x + 1 in x + y;;`
4. `let x = 1 in let x = 2 * x in x = 2 * x;;`
5. `let x = 1 in let x = 2 * x in let x = 2 * x;;`
6. `let x = 1 in let x = 2 * x in let x = 2 * x in x;;`
7. `let f x = 2 * x in f 3;;`
8. `let f x = 2 * x in f 3 + 4;;`