Chapitre 0 : Introduction

La compétence visée dans cette UE est :

Utiliser le paradigme de la programmation fonctionnelle
pour concevoir un programme en Caml.

1. Installation de Caml

OCaml est la principale implémentation du langage Caml. Il offre un puissant système de modules ainsi qu'une couche orientée objet. Il est livré avec un compilateur produisant du code natif pour de nombreuses plateformes.
Les différentes versions, pour toutes les plateformes, sont disponibles sur le site : https ://ocaml.org/

Exercice

Installer OCaml sur vos machines personnelles. Pour rédiger les comptes-rendus, il vous faudra un éditeur de texte capable de traiter des fichiers au format **.ml**, nous vous conseillons Notepad++.

2. La programmation fonctionnelle

2.1. Quelques repères historiques

- **1955 - 1956 :** J. John McCarthy emploie, pour la première fois, le terme *Intelligence Artificielle*. Lui et C. Shannon, M. Minsky et N. Rochester planifient une école d'été à Dartmouth pour l'année suivante, avec pour objectif de définir ce nouveau domaine et les axes de recherche associés. Lors de cet évènement, J. McCarthy a l'idée de développer un nouveau langage pour l'intelligence artificielle.
- **1958 :** Après deux ans de travail, première version de Lisp, le premier langage de programmation fonctionnelle (*list processing*).
- **Années 70-80 :** Lisp est le langage de prédilection pour l'IA.
- **1987 :** Première version de Caml (*Categorical Abstract Machine Language*), développée par l'Inria (cocorico !)
- **1996 :** Première version d'OCaml. Ce langage sera notamment utilisé pour développer Coq (un assistant de preuves) ou encore, plus récemment, Facebook Messenger.
- **Années 2000 à aujourd'hui :** La programmation fonctionnelle sort du milieu académique pour s'infiltrer dans le milieu industriel. Les développeurs ne sont pas suffisamment formés à ce paradigme, alors, les nouveaux langages (Scala 2003, Rust 2010, Kotlin 2011,...) combinent programmations orientée objet et fonctionnelle pour bénéficier des avantages de chacune et inciter les développeurs à l'utiliser.

2.2. Un nouveau paradigme

Il est important de comprendre que nous ne sommes pas en train d'apprendre *encore un autre* langage de programmation, mais bien de découvrir un **nouveau paradigme** de la programmation. À titre d'exemple, il n'y aura pas de variable, pas d'affectation, pas de boucle... un vrai changement d'habitudes !

Dans un langage fonctionnel, le programme consiste en une unique expression à évaluer.

En programmation fonctionnelle, agir c'est **évaluer** !

2.3. Premiers pas en Caml

Le programmeur tape une **requête**, et le système, après **interprétation** et **exécution** de cette requête, va afficher un résultat appelé **réplique**.
Pour signaler au programmeur que le système est prêt à accepter une nouvelle requête, Caml affiche le symbole #, appelé **invitation** ou **prompt**.
Le programmeur signale la fin d'une requête par les symboles ;;, ce qui signifie qu'il demande au système d'interpréter et d'exécuter cette requête.
La réplique de Caml est toujours constituée de plusieurs informations :

```
- : son type = sa valeur
```

3. Le processus de compilation

3.1. Quelques rappels sur les langages de programmation

Les langages dits de *haut niveau*, comme Java, C, Python, Caml etc., sont facilement utilisables par l'homme et permettent de décrire très précisément les opérations élémentaires.
Un programme écrit dans un langage de programmation est appelé **programme source**.
Rappelons qu'il existe deux techniques de compilations :

- Les **langages compilés**
  Le programme est écrit à l'aide d'un éditeur de texte, puis compilé en entier. Nous disposons alors d'un programme exécutable complet qui peut être utilisé autant de fois qu'on le souhaite.
- Les **langages interprétés**
  Le programme est écrit instruction par instruction, chaque instruction est compilée individuellement (interprétée) puis exécutée, puis... oubliée par la machine ! Il faudra donc recompiler à chaque nouvelle exécution.

Caml est un langage interprété. Il existe des compilateurs Caml, mais nous ne les utiliserons pas dans cette UE.

3.2. Interprétation d'une requête

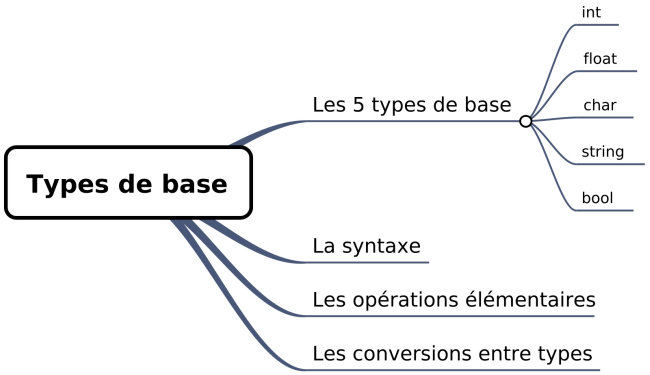
L'interprétation d'une requête s'effectue en deux temps :

- **l'analyse lexicale** consiste à rechercher des mots connus à l'intérieur de la requête (les différentes sortes de mots reconnus sont les mots-clef (réservés), les identificateurs, les symboles spéciaux et les constantes)
- **l'analyse syntaxique** consiste à étudier comment les différents mots se combinent pour former des **expressions**

Caml passe ensuite à la phase d'évaluation **type puis valeur**.

Le processus de compilation est enseigné dans tous les Masters mention Informatique ; nous distillerons tout au long de l'année des éléments, afin de vous y préparer. Ainsi, les analyses lexicales et syntaxiques sont des applications de la "Théorie des langages" (Semestre 6). L'évaluation des types sera abordée dans l'UE "Types de données, Preuves" (Semestre 6) et des valeurs ce semestre.

Chapitre 1 : Types de base



4. Les types de bases

Caml est un langage **fortement typé**, c'est à dire que toute expression possède un type, ce qui impose un certain nombre de contraintes que nous allons voir apparaître dans ce chapitre.

4.1. Les nombres entiers : le type `int`

Il s'agit des nombres entiers relatifs, ordonnés à l'aide de l'ordre usuel sur $\mathbb{Z}$. Les opérations élémentaires sur le type `int` sont :

-	le *moins* unaire (renvoie l'opposée d'une expression)	`int -> int`
+,-	addition, soustraction	`int * int -> int`
*	multiplication	`int * int -> int`
/, mod	quotient et reste de la division euclidienne	`int * int -> int`

Exercice

```
# 2147483647  ;;
-  : int = 2147483647
# 2147483647 + 1  ;;
-  : int = -2147483648
```

Sur combien de bits sont codés les entiers relatifs ?
(Souvenirs de Numération, Codages...)

4.2. Les nombres réels : le type `float`

On utilisera majoritairement la notation à **virgule fixe** avec la convention anglo-saxonne qui consiste à utiliser un point plutôt qu'une virgule (127.23). Il est également possible d'utiliser le notation à **virgule flottante** ($1.2723E02$). Les réels sont ordonnées à l'aide de l'ordre usuel sur $\mathbb{R}$. Les opérations élémentaires sont :

-.	le *moins* unaire (renvoie l'opposée d'une expression)	`float -> float`
+.,-.	addition, soustraction	`float * float -> float`
*. , /.	multiplication	`float * float -> float`
**	puissance	`float * float -> float`

En plus de ces opérations arithmétiques de base, Caml met à la disposition de l'utilisateur la plupart des fonctions mathématiques réelles courantes (exemple : $cos3.14$).

Exercice

Exécuter et commenter les requêtes suivantes. Lorsque les expressions sont incorrectes, expliquer pourquoi et corriger les !

1.2 + 1 ;;
1.2 + 2.3 ;;
-2E-1 +. 2. ;;
(sqrt(4.)+. 2.)/. 3.5 ;;
-2 * 3 ;;
2.1 +. 4.9 ;;
sqrt 9. ;;
10/3 ;;
10/.3.5 ;;
10 mod 3 ;;
2+ 3*5 ;;
-(5+1)*(-2+5)+2*3 ;;

Vous pouvez taper chaque commande dans l'éditeur, les recopier par copier/coller dans la fenêtre du terminal de Caml, puis recopier dans l'éditeur, le résultat de l'exécution.
Voici un exemple de compte-rendu pour la première expression :

```
# 1.2 + 1 ; ;
(* Error : This expression has type float but an expression was expected of
type int *)

(* Une erreur de type est signalée, en effet 1.2 est de type float (c'est un
 flottant), alors que 1 est de type int (entier) et + est un opérateur de
 type int * int -> int.
On peut le corriger en écrivant : *)

# 1.2 +. 1. ; ;
- : float = 2.2
```

4.3. Les caractères : le type `char`

Ce sont tous les symboles de la table ASCII : chiffres, lettres et symboles divers ; ils sont ordonnés dans l'ordre de cette table. Ils sont notés entre apostrophes (Attention ! Celle du 7 pour Caml Light, et celle du 4 pour OCaml...). Aucune opération interne n'est définie sur les caractères.

4.4. Les chaînes de caractères : le type `string`

Ce sont des suites de caractères écrits entre guillemets. Les chaines de caractères sont ordonnées par l'ordre lexicographique (l'ordre des mots du dictionnaire), les caractères étant ordonnés par l'ordre de la table ASCII comme nous avons vu plus haut. La seule opération sur les chaînes est la concaténation :

```
#"langage"^" Caml" ; ;
-  : string = "langage Caml"
```

Exercice

Exécuter les requêtes suivantes et commenter les lorsque c'est nécessaire.

"salut" ;;
"salut"^"à tous" ;;
"salut"^" à tous" ;;
"salut" < "bonjour" ;;
"salut" < "Salut" ;;
"A" < "a" ;;
'A'< 'a' ;;
'a';;
'a' < 'b' ;;
'a'< "bonjour" ;;
"a" < "bonjour" ;;
'a'^"près" ;;
"a"^"près" ;;
"12" > "2" ;;

4.5. Les booléens : le type `bool`

C'est l'algèbre de Boole $\{vrai, faux\}$ telle que nous l'avons étudié en Numération, Codages. Les deux valeurs logiques sont notées `true` et `false` (en minuscules). Les opérations sont :

=, <>	égalité (!), différence
<, >	inégalités strictes
<=, >=	inégalités larges

Nous disposons sur l'ensemble des booléens des trois opérations classiques : `not`, `&`, `or` (négation logique, et logique, ou logique).

Exercice

Exécuter et commenter les requêtes suivantes.

1 = 2 ;;
4 < 5 ;;
4.2 < 4.7 ;;
(1=1) = (2<1) ;;
true = 1 ;;
true or false ;;

5. Conversions de types

Il existe des fonctions de conversion entre types permettant d'adapter le type de l'expression à une opération souhaitée.
Par exemple, pour ajouter 3 et 2.5, nous devons convertir 3 en flottant.

```
#3 +. 2.5 ;;
(* Error : This expression has type int but an expression was expected of type float *)

#float_of_int(3) +. 2.5 ;;
- : float = 5.5
```

Les principales fonctions de conversions sont :

`float_of_int`	`int -> float`	convertit un entier en réel de même valeur*
`int_of_float`	`float -> int`	donne l'entier obtenu par troncature*
`int_of_char`	`char -> int`	donne le code ASCII du caractère
`char_of_int`	`int -> char`	donne le caractère correspondant au code ASCII
`int_of_string`	`string -> int`	convertit une chaîne de caractère en entier
`string_of_int`	`int -> string`	convertit un entier en chaîne de caractères.

**ces deux premières fonctions en particulier nous serons utiles dès le prochain chapitre, ne les oubliez pas !!*

Exercice

Exécuter et commenter les requêtes suivantes.

int_of_float ;;
int_of_float(4.0) ;;
int_of_float(4.25) ;;
int_of_float(-4.25) ;;
string_of_int(-235) ;;

int_of_string "345" ;;
int_of_string "34.5" ;;
float_of_string "34.5" ;;
float_of_string "999999999999999.9" ;;