Programmation Fonctionnelle
Cours 1
Premiers pas et types de base

Delia Kesner

Deux modes de travail avec OCaml

- Compiler:
  - On obtient des exécutables autonomes
  - Plus adapté pour les connaisseurs
  - Compiler vers du bytecode :
    1. Exécuter `ocamlc myprogram.ml -o myprogram`
    2. Ensuite exécuter `ocamlrun myprogram`
  - Compiler vers du code natif :
    1. Exécuter `ocamlopt myprogram.ml -o myprogram`
    2. Ensuite exécuter `./myprogram`
- Interpréter:
  - On peut expérimenter avec le langage
  - Plus adapté pour les premiers mois
  - Exécuter la commande `ocaml` directement sur votre console.
  - Mieux : Dans `emacs` utiliser le mode `tuareg`

Le mode d'interprétation

- L'utilisateur tape une *requête* (on dit encore une *phrase*) *Caml* : une *expression* terminée par deux points-virgules consécutifs.
- *Caml* analyse la syntaxe, affiche un message d'erreur si cette syntaxe est incorrecte.
- Si la syntaxe est correcte, l'interpréteur *infère* (c'est-à-dire calcule) le *type* de l'expression, affiche un message d'erreur si l'expression est mal typée.
- Si l'expression est bien typée, l'interpréteur *évalue* l'expression, puis affiche le type calculé et la valeur obtenue.

Premiers exemples I

```
3*(4+1)-7;;
# - : int = 8

2+3*5;;
# - : int = 17

5/2;;
# - : int = 2

5 mod 3;;
# - : int = 2
```

Premiers exemples II

```
17 +;;
17 +;;
    ^^
Error: Syntax error

42 + "hello";;
42 + "hello";;
     ^^^^^^^
Error: This expression has type string
       but an expression was expected of type int
```

Remarques

- C'est le ;; qui termine la requête, pas le saut de ligne. Les sauts de ligne à l'intérieur d'une requête sont des espaces comme les autres.
- Les ;; ne font pas partie de la syntaxe du langage *Caml* lui-même mais sont spécifiques à l'interpréteur.
- Commentaires : entre `(*` et `*)`, éventuellement sur plusieurs lignes.

Types de base

- `int`, `float`, `bool`, `char`, `string`, ...

Les entiers

- Type `int`
- Les valeurs: . . . , -2, -1, 0, 1, 2, 3, . . .
- Les opérateurs:
  - `+`: addition
  - `-`: soustraction
  - `*`: multiplication
  - `/`: division entière
  - `mod`: reste de la division entière

Exemples entiers I

```
3+4*2;;
# - : int = 11

(3+4)*2;;
# - : int = 14

mod (3+4) 2;;
mod (3+4) 2;;
^^^
Error: Syntax error

(3+4) mod 2;;
# - : int = 1
```

Les réels

- Type `float`
- Valeurs: écrites avec un point décimal, ou en notation scientifique, par exemple 4., -5.2, 4.13, 5e4, -5e-2, ...
  - Rappel notation scientifique: 5e4 veut dire $5 * 10^4$
- Les opérations:
  - `+.`, `-.`, `*.`, `/.` (avec un point!)
  - `sin`, `sqrt`, `log`, `ceil`, `floor`, ...
- Les fonctions de conversions:
  - `float_of_int`
  - `float`
  - `int_of_float`

Exemples réels I

```
3.0 +. 0.01;;
# - : float = 3.01

3e6 +. 2e5;;
# - : float = 3200000.

2e-4 *. 0.1;;
# - : float = 2e-05

2 + 3.0;;
2 + 3.0;;
    ^^^
Error: This expression has type float
       but an expression was expected of type int
```

Exemples réels II

```
1.0 + 2.0;;
1.0 + 2.0;;
^^^
Error: This expression has type float
       but an expression was expected of type int
```

```
4 *. 56;;
4 *. 56;;
^
Error: This expression has type int
       but an expression was expected of type float
  Hint: Did you mean ‘4.’?
```

Exemples réels III

```
float_of_int 17;;
# - : float = 17.

int_of_float 42.3;;
# - : int = 42

(float_of_int 3) +. 5.8;;
# - : float = 8.8

(float_of_int (5 * (int_of_float 2.5))) +. 0.1;;
# - : float = 10.1
```

Les booléens

- Type : `bool`
- Constantes : `true` et `false`
- Opérateurs logiques:
  - `not`: négation
  - `&&`, `&`: et séquentiel
  - `||`, `or`: ou séquentiel
- Opérateurs de comparaison:
  - `=`: égalité
  - `<`, `<=`: plus petit, plus petit ou égal
  - `>`, `>=`: plus grand, plut grand ou égal
- Opérateur conditionnel: `if` cond `then` e1 `else` e2
  - cond est une expression de type `bool`
  - e1 et e2 sont du même type
  - Il y a une valeur par défaut quand la partie *else* manque, on va revenir sur ce point plus tard.

Quelques équivalences pratiques

- `b = true` est la même chose que `b`
- `b = false` est la même chose que `not b`
- `if b then true else false` est la même chose que `b`
- `if b then false else true` est la même chose que `not b`
- `if b then true else b1` est la même chose que `b || b1`
- `if b then b1 else false` est la même chose que `b && b1`

Exemples booléens I

```
not false && false;;
# - : bool = false

not (false && false);;
# - : bool = true

true = false;;
# - : bool = false

3 = 3;;
# - : bool = true

4 + 5 >= 133;;
# - : bool = false
```

Exemples booléens II

```
2.1 *. 4. >= 7.4 ;;
# - : bool = true

if (3 <4) then 1 else 0;;
# - : int = 1

if (4 <3) then 1 else 0;;
# - : int = 0

if 1<2 then 6+7 else 67/23;;
# - : int = 13
```

Exemples booléens III

```
if 6=8 then 1 else 77.5;;
if 6=8 then 1 else 77.5;;
                   ^^^^
Error: This expression has type float
       but an expression was expected of type int
```

```
(if 6=3+3 then 3<4 else 8 > 7) && 67.8 > 33.1;;
# - : bool = true
```

```
if (if 1=1 then 2=2 else 4.0 > 3.2) then 2<3 else 3;;
if (if 1=1 then 2=2 else 4.0 > 3.2) then 2<3 else 3;;
                                                 ^

Error: This expression has type int
       but an expression was expected of type bool
```

Les caractères

- Type : `char`
- Valeurs: caractères ASCII (écrits entre apostrophes), par exemple 'b', 'Z', ' ', ...
- ASCII= *American Standard Code pour Information Interchange* (les caractères qu'on trouve sur un clavier américain)
- Échappement:
  - `\n` : saut de ligne
  - `\r` : retour chariot
  - `\t` : tabulation
  - `\'` : apostrophe
  - `\"` : guillemet
- Les fonctions de conversion:
  - `Char.code`: conversion de char à int
  - `Char.chr`: conversion de int à char
  - `Char.lowercase_ascii`
  - `Char.uppercase_ascii`

Exemples caractères I

```
'a';;
# - : char = 'a'

Char.code 'a';;
# - : int = 97

'\097';;
# - : char = 'a'

'\97';;
'\97';;
^^^
Error: Illegal backslash escape in string or character (\9)
```

Exemples caractères II

```
Char.uppercase_ascii 'd';;
# - : char = 'D'

Char.lowercase_ascii (Char.uppercase_ascii 'd');;
# - : char = 'd'
```

Les chaînes de caractères

- Type : `string`
- Valeurs: chaînes de caractères (écrites entre guillemets), par exemple "bonjour", "z w ", "a", " ", ...
- Opérateurs:
  - Concatenation: `"Je pense "^"donc je suis"`
  - `String.length`: `string -> int`
  - `String.get`: `string -> int -> char`
  - `String.make`: `int -> char -> string`
  - `String.sub`: `string -> int -> int- > string`
- Attention! `string` $\neq$ `char`

Exemples chaînes de caractères I

```
"\097\098\099";;
# - : string = "abc"

"Bonjour".[0];;
# - : char = 'B'

String.get "Bonjour" 0;;
# - : char = 'B'

"Bonjour".[1];;
# - : char = 'o'

String.get "Bonjour" 1;;
# - : char = 'o'
```

Exemples chaînes de caractères II

```
String.length "Bonjour";;
# - : int = 7

"Bonjour".[10];;
# Exception: Invalid_argument "index␣out␣of␣bounds".

String.make 10 'a';;
# - : string = "aaaaaaaaaa"

String.sub "abcdefg" 0 2;;
# - : string = "ab"
```