Chapitre 1

Initiation à Ocaml

1 Expressions.

```
5 ;;     (* expression que l'on transmet a l'interpreteur *)
- : int = 5   (* la reponse de l'interpreteur *)

1 + 3 ;;  (* expression que l'on transmet a l'interpreteur *)
- : int = 4

6 * 4 ;;
- : int = 24

15 mod 4 ;; (*  la fonction modulo  *)
- : int = 3

78 / 5 ;;
- : int = 15
(* Toute expression a un type et une valeur *)

78.0  /. 5.0 ;;
- : float = 15.6
(* On ne melange pas les entiers et les flottants *)
  5. + 2.;;
  ^^
Error: This expression has type float but an expression
 was expected of type int
```

2 Affectation . Définition de variables.

```
let x = 3;; (* definition d'une variable globale *)
val x : int = 3
```

Propriété 1 :

La syntaxe pour définir une variable globale est :

let nom_de_la_variable = valeur_de_la_variable ;;

```
x ;;  (* utilisation de la variable globale *)
- : int = 3
x+1;;
- : int = 4
let y = x + 5 ;;  (* definition d'une autre variable globale *)
```

```
val y : int = 8
let x = 13;;  (* redefinition de x *)
val x : int = 13
x ;;  (* x est bien redefini *)
- : int = 13

y;;
- : int = 8 (* y n'est pas affecte par la redefinition de x *)

let x = 6 in x+1;;        (* utilisation d'une variable locale *)
- : int = 7       (* on n'a pas redefini de variable globale *)

x ;;
- : int = 13  ( x n'a pas change de valeur )
```

Propriété 2 :

La syntaxe pour définir une variable locale est :

let nom_de_la_variable = valeur_de_la_variable in expression ;;

Le type de cette expression est celui de expression.

Remarque : une variable locale "masque" une variable globale.

```
x;;
- : int = 13

let z = x+3;;
val z : int = 16
```

La valeur de la variable globale `x` est utilisée pour calculer la valeur de `z`.

```
let w  =  let x = 2 in x+3 ;;
```

C'est la valeur de la variable locale `x` qui est prise en compte dans le calcul de la valeur de `w` ; la variable locale `x` masque la variable globale `x`.

Exercice 1 : Pour chaque procédure, déterminer ce que renvoie l'interpréteur, puis vérifier sur l'ordinateur :

(a)
```
let nb_1 = 5 ;;
let nb_2 = 7 ;;
let nb_1 = nb_1 + nb_2;;
let nb_2 = nb_1 - nb_2 ;;
let nb_1 = nb_1 - nb_2 ;;
```

(b)
```
let z = 4 in let t = z*2 in 3*z-2*t;;
```

(c)
```
let x = 5.;;
let y = 3.;;
let z = x + y ;;
```

3 Conditionnelle . Utilisation de conditions.

Opérateurs de comparaison : $<$; $=$; $>$; $<=$; $>=$; $!=$

```
let x = 13;;
val x : int = 13
x > 6;;
-: bool = true
```

Le type bool est le type booléen. Il contient deux constantes booléennes : true et false

```
let b = true ;;
val b : bool = true

(x != 10)&&(not b);;
- : bool = false
```

Opérateurs logiques : & & ("et" logique) , || ("ou" logique) , not (négation logique).

Attention, "and" n'est pas le "et" logique, il est utilisé pour définir plusieurs variables.

```
let nb_1= 5 and nb_2 =1.5 ;;
val nb_1 : int = 5
val nb_2 : float = 1.5

if  (x > 6)|| b  then  1 else  0;;
- : int = 1
```

Propriété 3 :

Syntaxe de l'utilisation du " if ... then ... else ..." :

if cond then expr1 else expr2

avec cond une condition, expr1 et expr2 deux expressions **de même type**, qui sera le type de l'expression toute entière.

```
if (x > 20) || b  then 3.5 else 2;;
                                 ^
Error: This expression has type int
but an expression was expected of type float
```

Remarque : le "else" est obligatoire car toute expression doit avoir un type. Seul le type unit qui a une valeur unique noté (), n'exige pas de "else".

```
if (x < 20) then 2*x;;
                 ^^^
Error: This expression has type int but
an expression was expected of type unit
```

4 Fonctions.

```
let carre x = x*x;;
val carre : int -> int = <fun>
carre 9;;
- : int = 81

let minimum a b = if a > b then b else a;;
val minimum : 'a -> 'a -> 'a = <fun>
```

minimum a deux arguments, a et b, tous deux de type polymorphique (en fait plusieurs types peuvent être en argument, mais a et b restes de même type.)

Propriété 4 :

Syntaxe de la définition d'une fonction :

let id id1 ... idn = expr ;;

la fonction définie est la fonction id. Elle a n arguments, représentés par id1, ... idn.
Remarque : Les fonctions de cette forme sont dites curryfiées, il est aussi possible de définir cette même fonction avec un n-uplet :

```
let minimum (a, b) = if a > b then b else a;;
val minimum : 'a * 'a -> 'a = <fun>
```

Remarque : on peut aussi écrire une fonction sans lui donner de nom, avec l'appel "function" ou "fun" pour une fonction curryfiée :

```
function x-> x+1;;
- : int -> int = <fun>
let minimum = fun a b -> if a> b then a else b;;
val minimum : 'a -> 'a -> 'a = <fun>
```

Exercice 2 : Un gérant de parc d'attractions cherche à écrire une fonction pour la gestion des entrées.
Il propose ces prix :

- 17 €par adultes.
- 13 €par enfants.

Il propose aussi les réductions suivantes :

- Si le nombre d'adultes est supérieur ou égal à 10, chaque adulte paye alors 15 €.
- Si le nombre d'enfants est supérieur ou égal à 10, chaque enfant paye alors 11 €.

Écrire la fonction cout qui prend en argument le nombre d'adultes et d'enfants, et qui renvoie le prix à payer.

5 Boucles inconditionnelles.

```
for i = 1 to 10 do
  print_int(i)
done;;
12345678910- : unit = ()
```

Propriété 5 :

Syntaxe de la boucle inconditionnelle :

for id = val1 to val2 do expr1 done ;

Exercice 3 : Ecrire la fonction `multiple` qui affiche la liste des $p+1$ premiers multiples de n, n et p étant placés en arguments.

```
val multiple : int -> int -> unit = <fun>
multiple 5 7;;
0 , 5 , 10 , 15 , 20 , 25 , 30 , 35 , - : unit = ()
```

6 Structure de pointeurs.

Les références permettent de discerner le nom d'une variable et sa valeur, et donc de modifier cette valeur.

```
let x = ref 5;;
val x : int ref = {contents = 5}
```

Propriété 6 :

Syntaxe pour un pointeur :

let id =ref valeur ;;

- L'assignement :

```
x:= 7;;
- : unit = ()
```

- Accéder au contenu :

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

Exercice 4 : Écrire la fonction `somme` qui renvoie la somme des n premiers entiers non nuls :

```
val somme : int -> int = <fun>
somme 5;;
- : int = 15
```

7 Boucles conditionnelles.

```
let i = ref 3 in
while !i <= 12 do
  print_int(3*(!i)); print_string("␣,␣"); i := !i +1
done;;
9 , 12 , 15 , 18 , 21 , 24 , 27 , 30 , 33 , 36 , - : unit = ()
```

Propriété 7 :

Syntaxe de la boucle conditionnelle :

while cond do expr1 done

Exercice 5 : On considère la suite définie par :

$$\begin{cases} u_0 &= 5 \\ u_{n+1} &= 2u_n + 3 \end{cases}$$

Ecrire la fonction `indice_max` qui renvoie la valeur du rang maximal pour lequel les valeurs de la suite sont inférieur au réel M donnée en argument.

```
val indice_max : int -> int = <fun>
indice_max 50;;
- : int = 2
```

8 Les types courants

1. int, float : attention aux différents opérateurs.
2. liste : voir chapitre listes.
3. vecteur/tableau/string : voir chapitre tableau
4. Couples, tuples :

   ```
   let x = (1,'a') ;;
   val x : int * char = (1, 'a')
   fst(x);;
   - : int = 1
   snd(x);;
   - : char = 'a'
   ```

   Tous les éléments du couple n'ont pas forcément me même type.

9 Type enregistrement

```
type personne =
{nom : string ; prenom : string ;  annee_naissance : int ;
 mutable classe : string ; frere_soeur : personne list} ;;

let eleve_1 = {nom = "Dupond" ; prenom = "Jean" ; annee_naissance=2000 ;
classe = "seconde␣2" ; frere_soeur = []};;

val eleve_1 : personne =
  {nom = "Dupond"; prenom = "Jean"; annee_naissance = 2000;
   classe = "seconde␣2"; frere_soeur = []}

eleve_1.nom;;
- : string = "Dupond"

eleve_1.classe <- "seconde␣3";;
- : unit = ()

eleve_1;;
- : personne =
{nom = "Dupond"; prenom = "Jean"; annee_naissance = 2000;
 classe = "seconde␣3"; frere_soeur = []}

eleve_1.nom <- "Dupont";;
  ^^^^^^^^^^^^^^^^^^^^^^
Error: The record field nom is not mutable
```

10 Type énumération et constructeurs de valeur

```
type couleur = Trefle | Carreau | Coeur | Pique ;;
```

Remarque : Attention à la majuscule...

```
let nombre_point (coul)= match coul with
  | Trefle -> 5
  | Carreau -> 10
  | Coeur -> 15
  | Pique -> 20;;
val nombre_point : couleur -> int = <fun>
```