TD 1

# Exercice d'initiation à Ocaml

**Correction**

Exercice 1 : Donner les résultats des expressions suivantes, sans utiliser l'ordinateur, et corriger en cas d'erreur de syntaxe :

(a) `if 2>3 then 4 ;;`

(b) `1>2 and 5>3 ;;`

(c) `3.5 + 4.0 ;;`

(d) `1+4.0 ;;`

(e) `8./.3. ;;`

(f) `8/3 ;;`

(g) `1.0<2.0 || 3>4 ;`

Exercice 2 :

1. Définir deux fonctions pour renvoyer la valeur absolue dont on donne la signature :

   `val_abs_int : int -> int = <fun>;`

   et

   `val_abs_float : float -> float = <fun>`

   Correction

Listing 1 – Les fonctions `val_abs_int` et `val_abs_float`

```
let val_abs_int x =
    if x >= 0 then x else (-x);;

let val_abs_float x =
    if x >= 0. then x else (-. x);;
```

2. Définir deux fonctions `signe` qui renvoie 1 si la donnée est positif, 0 si elle est nulle et -1 si elle est négative. :

   `signe_int : int -> int = <fun>`

   et

   `signe_float : float -> int = <fun>`

   Correction

Listing 2 – Les fonctions `signe_int` et `signe_float`

```
let signe_int x =
  if x > 0 then 1  else
    if x = 0 then 0  else (-1)   ;;

let signe_float x =
  if x > 0. then 1  else
    if x = 0. then 0  else (-1)   ;;
```

Exercice 3 :

1. Écrire une fonction `collatz` qui prend en argument un entier $n$, et rend $n/2$ si $n$ est pair, $3n+1$ si $n$ est impair.

   ```
   collatz : int -> int = <fun>
   ```

   Correction

Listing 3 – La fonction `collatz`

```
let collatz n =
  if n mod 2 = 0 then n/2 else 3*n+1;;
```

2. Écrire une fonction `syracuse` prend en argument un entier $n$ et retourne le nombre d'utilisation de la fonction `collatz` pour que la suite définie par :

$$\begin{cases} u_0 &= n \\ u_{n+1} &= \texttt{collatz}(u_n) \end{cases}$$

   atteigne la valeur 1.

   ```
   syracuse : int -> int = <fun>
   ```

   **Exemple :**

   ```
   syracuse 10;;
   - : int = 6
   ```

   Correction

Listing 4 – La fonction `syracuse`

```
let syracuse n =
  let u = ref n in
  let i = ref 0 in
  while  !u > 1  do
    u:= collatz (!u) ;
    incr(i) ;
  done ;
!i;;
```

Exercice 4 : Écrire une fonction qui prend en arguments trois entiers $a$, $b$ et $c$, et rend le plus petit des trois arguments. ( on minimisera le nombre de comparaisons.)

Correction

Listing 5 – La fonction `minima`

```
let minima a b c =
  if a < b then
      if a < c then a else c
  else if b < c then b else c ;;
```

Exercice 5 :

1. Écrire une fonction `nb_chiffre` qui renvoie le nombre de chiffres dans l'écriture décimale d'un entier positif.

```
nb_chiffre : int -> int = <fun>
```

**Exemple :**

```
nb_chiffre 5746;;
- : int = 4
```

Correction

Listing 6 – La fonction `nb_chiffre`

```
let nb_chiffre n =
  let n_2 = ref n in
  let compteur = ref 0  in
  while !n_2 !=0  do
    n_2 := !n_2 /10 ;
    incr(compteur) ;
  done ;
!compteur ;;
```

2. Écrire une fonction `nb_zero` qui renvoie compte le nombre de fois où le chiffre 0 apparait dans l'écriture d'un entier positif.

```
nb_zero : int -> int = <fun>
```

**Exemple :**

```
nb_zero 50200;;
- : int = 3
```

Correction

Listing 7 – La fonction `nb_zero`

```
let nb_zero n =
  let n_2 = ref n in
  let compteur = ref 0  in
  while !n_2 !=0  do
    if !n_2 mod 10 = 0
        then incr(compteur) ;
    n_2 := !n_2 /10 ;

  done ;
!compteur ;;
```

3. Écrire une fonction `symetrie` qui renvoie le "symétrique" d'un entier positif.

```
symetrie : int -> int
```

**Exemple :**

```
symetrie 5237;;
- : int = 7325
```

Correction

Listing 8 – La fonction `symetrie`

```
let symetrie n =
  let n_2 = ref n in (* nb a vider *)
  let sym = ref 0 in (* le symetrique *)
  while !n_2 != 0 do
    let chif_unite = !n_2 mod 10 in
    sym := !sym * 10 + chif_unite ;
    n_2 := !n_2 / 10 ;
  done ;
!sym ;;
```

Exercice 6 : On définit l'ensemble des nombres complexes avec le type suivant :

```
type complexe = { part_reelle : float ; part_imaginaire : float } ;;
```

1. Définir les complexes `zero` $= 0$ ,`un` $= 1$ , `i` , $z_1 = 3 + 2i$ et $z_2 = 5 - i$.

Correction

Listing 9 – Les complexes

```
type complexe
    ={part_reelle :float;
    part_imaginaire :float};;

let un ={part_reelle = 1. ;
    part_imaginaire = 0.};;

let i ={part_reelle = 0. ;
    part_imaginaire = 1.};;

let z_1 ={part_reelle = 3. ;
    part_imaginaire = 2.};;

let z_2 ={part_reelle = 5. ;
    part_imaginaire = -. 1.};;

let zero ={part_reelle = 0. ;
    part_imaginaire = 0.};;
```

2. Écrire les opérations usuelles sur l'ensemble des complexes.

```
somme : complexe -> complexe -> complexe
produit : complexe -> complexe -> complexe
conjugue : complexe -> complexe
inverse : complexe -> complexe
quotient : complexe -> complexe -> complexe
norme : complexe -> float (* le module *)
```

Correction

Listing 10 – Les complexes

```
let somme z_1 z_2 =
```

```
  {part_reelle = z_1.part_reelle +. z_2.part_reelle ;
part_imaginaire = z_1.part_imaginaire +. z_2.part_imaginaire};;

somme z_1 z_2;;

let produit z_1 z_2 =
  let re = z_1.part_reelle *. z_2.part_reelle -.
  z_1.part_imaginaire *. z_2.part_imaginaire in
  let im = z_1.part_reelle *. z_2.part_imaginaire +.
  z_2.part_reelle *. z_1.part_imaginaire in
   {part_reelle = re ;
    part_imaginaire = im};;

produit z_1 z_2;;

let conjugue z_1 =
{part_reelle = z_1.part_reelle  ;
part_imaginaire = -. z_1.part_imaginaire };;

conjugue z_1;;


let inverse z =
if z = zero then failwith "␣on␣ne␣divise␣pas␣par␣zero␣" else
  let norme_carre = z.part_reelle**2. +. z.part_imaginaire **2. in
  {part_reelle = z.part_reelle /. norme_carre   ;
part_imaginaire = -. z.part_imaginaire /. norme_carre  };;

inverse z_1;;
inverse i;;
inverse zero;;

let quotient z_1 z_2 =
  produit z_1 ( inverse z_2);;

sqrt(3.);;
sqrt;;

let norme z =
  let norme_carre = z.part_reelle**2. +. z.part_imaginaire **2. in
  sqrt(norme_carre);;
```