# Bases OCAML

## Chapitre 1

Auteur

# Table des matières

## ▸ Chapitre 1

# ◆ Bases OCAML ◆

Intransigeant, pas de mélange de type/transtypage !

Commentaire (* contenu *)

| Raccourcis clavier | |
|---|---|
| Ctrl-c Ctrl-s enter | démarrer OCaml dans emcas |
| Ctrl-x Ctrl-s | sauvegarde du fichier courant |
| Ctrl-c Ctrl-e | exécuter de la ligne courante |
| Ctrl-c Ctrl-b | exécuter tout le fichier |

## ▸ Integer & Float

- Pour additionner les entier : +, etc...
- Pour additionner les flottants, il faut utiliser un opérateur spécial : +.

Changer de type : float_of_int

Les fonctions usuelles mathématiques prennent en entrée uniquement des float !

## ▸ Booléens

Priorité opératoire : NOT > AND > OR

| Opérateur | |
|---|---|
| ET | && |
| OU | \|\| |
| = | Comparaison/égalité (Et non affectation !) |
| <> | Différence (inégalité) |

## ▸ String

Pas de différence notoire

| Opérateur | |
|---|---|
| ^ | Opérateur de concaténation |
| String.length « abc » | Renvoie la taille de la chaîne de caractère |

## ▸ Char

Type caractère (1 seul caractère...)

## ▸ Déclaration de variable

```
1  let a = 3;;
```

Les variables ne sont pas stockés dans la mémoire. En réalité les occurrences sont remplacés directement par la variable. Ainsi, on ne peut pas les modifier., elle sont immuables.

## ▸ Fonctions

```
1  (*Définition de la fonction*)
2  let f a = a + 2;;
3
4  (*Appel de la fonction*)
5  f 3;;
```

Les fonctions sont typés ! (Ici, int $\longrightarrow$ int)

Les returns n'existent pas en ocaml.

On peut définir une fonction avec de multiple paramètre :

```
1  let f a b = a + b;;
```

Le type sera int $\longrightarrow$ int (C'est conditionné par l'opérateur + spécifique au integer)

Il peut y avoir un indeterminé, par exemple :

```
1  let f a = a ;;
```

Cette fonction renvoie le paramètre d'entrée, peut importe son type.

Les fonctions sont des valeurs comme des autres.

Les fonctions à deux variables sont des fonctions qui revoient des fonctions qui revoient des nombres (ou des chaines de caractère)

## ▸ Alternative

```
1  let a = 1;;
2  let b = 2;;
3
4  let res = if a < b then true else false;;
```

Après un if il faut obligatoirement mettre un then ET un else

Dans le if et le else, il faut renvoyer le même type (ici, booléen)

Pas de true ou 1 par exemple.

Ici la variable *res* est représentant de l'expression *if ... false ;;*

## ▸ Fonction récursives

```
1  let rec facto n =
2    if n = 0 then
3      1
4    else n * facto(n -1)
```