

# Introduction à OCaml

OPTION INFORMATIQUE - TP n° 1.0 - Olivier Reynet

## À la fin de ce chapitre, je sais :

- ☞ expliquer le fonctionnement de l'inférence de type
- ☞ lire la signature d'une fonction
- ☞ définir un type simple en OCaml
- ☞ définir un type somme simple
- ☞ utiliser le filtrage de motif sur un cas simple
- ☞ utiliser une référence pour programmer impérativement
- ☞ programmer une fonction récursive simple

## A Coder en OCaml?

**En ligne** Pour utiliser OCaml en ligne, il suffit d'utiliser les bacs à sable des sites [OCaml](#) ou [TryOcaml](#).

**Sur votre machine locale** Sur votre machine, le plus simple est d'utiliser l'interprète interactifs OCaml. [On peut facilement l'installer sur n'importe quel système d'exploitation en suivant ces instructions.](#)

**Pour travailler avec un éditeur de texte** Emacs facilite l'édition de code OCaml grâce au mode Tuareg. Les commandes `M-x tuareg-mode` et `M-x run-ocaml` permettent d'activer ce mode. [Le résumé \(sic!\) des commandes est accessible en ligne. Ce tutoriel vous montre quelques commandes de base pour survivre avec Emacs.](#) Pour les puristes de la ligne de commande, ce mode Tuareg est également disponible sous Vim.

**Utiliser un IDE** Enfin, il est possible d'utiliser OCaml avec la plupart des environnements de développement : Eclipse, Visual Studio ou IntelliJ (Jet Brains).

## B Tester les commandes via l'interprète interactif

OCaml dispose d'un interprète interactif. Par défaut, c'est l'exécutable `ocaml` mais il existe également `utop`.

B1. L'inférence de type est un mécanisme puissant. Sur les éléments suivants, tenter de deviner les types de ces variables ou expressions et vérifier avec l'interprète :

- (a) `let n = 3`
- (b) `let x = 3.14`
- (c) `let c = 'a'`

- (d) `let s = "ocaml"`
- (e) `let b = n > 3`
- (f) `let bi = if n > 2 then true else false`
- (g) `let m = if bi then 4.5 else 7.`
- (h) `let elements = [1;3;5;42]`
- (i) `let elements = [true;false;true]`
- (j) `()`
- (k) `print_int 3`
- (l) `let t = (3,"clef")`
- (m) `let st = (4,5)`
- (n) `let t = (3., "clef")`
- (o) `let tt = (true, 42, 'z')`

B2. Tenter de deviner les signatures des fonctions suivantes :

- (a) `let u n = 3*n + 2`
- (b) `let f x y = (x+.y)*.(x-.y)`
- (c) `let g n x = (float_of_int n)*. x`
- (d) 

```

1      let h n x =
2          let r = ref 1. in
3              for k = 1 to n do
4                  r := !r *. x
5              done;
6              !r;;

```

---

B3. S'entraîner à utiliser ces variables et ces fonctions dans un programme principal.

## C Variables globales et locales

- C1. Créer un fichier `vars.ml`.
- C2. Définir une variable globale `rayon` et l'initialiser à la valeur `1.21`.
- C3. Coder une fonction de signature `disk_area : float -> float` qui renvoie l'aire d'un disque en fonction de son rayon.
- C4. Modifier la fonction précédente pour faire apparaître une variable locale à la fonction nommée `pi` et valant `3.14159265359`.
- C5. Quelles sont les variables locales et globales de ce code?

## D Fonctions et fonctions récursives

- D1. Créer un fichier `fact.ml`
- D2. Coder une fonction impérative de signature `i_fact : int -> int` qui renvoie  $n!$ .
- D3. Coder une fonction **récursive** de signature `fact : int -> int` qui renvoie  $n!$ .
- D4. Coder une fonction **récursive terminale** de signature `tr_fact : int -> int` qui renvoie  $n!$ . On utilisera une fonction auxiliaire interne à la fonction.
- D5. Peut-on calculer  $42!$ . Pourquoi?

## E Jouer à la bataille

On souhaite coder les fonctions élémentaires pour jouer à la bataille. On choisit de modéliser les cartes avec des types algébriques.

- E1. Créer un fichier `bataille.ml`.
- E2. Coder un type somme `couleur` capable de représenter les quatre couleurs d'un jeu de carte : trèfle, pique, carreau, cœur.
- E3. Coder un type somme `figure` capable de représenter les figures, c'est à dire le roi, la dame et le valet.
- E4. Coder un type algébrique `carte` capable de représenter une carte quelconque.
- E5. Dans le programme principal, créer quelques cartes.
- E6. **En utilisant le filtrage de motif**, coder une fonction de signature `get_value : carte -> int` qui renvoie la valeur associée à une carte. On considère que l'as vaut 14, le roi 13, la dame 12, le valet 11 et les numéros leur propre nombre.
- E7. Coder une fonction de signature `comparer : carte -> carte -> int` qui compare deux cartes d'après leur valeur. La fonction renvoie un entier valant la différence entre les valeurs des cartes.
- E8. Coder une fonction de signature `bataille : carte -> carte -> bool` qui teste s'il y a bataille entre deux cartes.

En deuxième année, vous pourrez programmer un automate qui implémentera la logique du jeu.

## F Calculer avec les entiers en OCaml

- F1. Créer un fichier `evens.ml` et écrire une fonction de signature `even : int -> bool` qui teste si un nombre est pair. En déduire une fonction qui teste si un nombre est impair. L'opérateur modulo est `mod` en OCaml.
- F2. Créer un fichier `syracuse.ml`. On considère la suite de Syracuse  $(u_n)_{n \in \mathbb{N}}$  définie par  $u_0 \in \mathbb{N}^*$  et :

$$\forall n \in \mathbb{N}, \quad u_{n+1} = \begin{cases} 3u_n + 1 & \text{si } u_n \text{ est impair} \\ \frac{u_n}{2} & \text{si } u_n \text{ est pair} \end{cases} \quad (1)$$

- (a) Écrire une fonction de signature `next_u : int -> int` qui calcule le terme suivant de la suite.
- (b) Écrire une fonction de signature `syracuse : int -> int` qui renvoie le temps de vol de la suite pour un entier  $u_0$  donné<sup>1</sup>. On utilisera une boucle `while` et deux références.
- F3. Créer un fichier `gcd.ml`. Écrire une fonction **réursive** de signature `gcd : int -> int -> int` qui calcule le PGCD de deux entiers naturels. En déduire une fonction de signature `coprime : int -> int -> bool` qui teste si deux nombres sont premiers entre eux.
- F4. On souhaite tester la conjecture de Goldbach qui affirme que *tout entier naturel pair plus grand que 2 est la somme de deux nombres premiers*. Créer un fichier `goldbach.ml`
  - (a) Écrire une fonction de signature `is_prime : int -> bool` qui teste si un nombre est premier. On rappelle que 1 n'est pas un nombre premier.
  - (b) Écrire une fonction de signature `goldbach : int -> int * int` qui teste la conjecture de Goldbach. Si le nombre entier fourni en paramètre est impair, le programme échoue en imprimant le message `"Goldbach's conjecture only on even numbers"`. Sinon, il renvoie un tuple contenant les deux nombres solution.

1. c'est à dire l'indice  $N$  de la suite pour lequel  $u_N = 1$ , s'il existe...