# TD 1 : Programmer en OCaml

Pierre Talbot (pierre.talbot@univ-nantes.fr)

9 avril 2019

## Objectif(s)

- ✯ Installer et s'approprier l'environnement OCaml.
- ✯ Programmer dans le noyau fonctionnel vu en cours.

*Certains exercices sont largement inspirés d'exercices du cours OCaml de Charlotte Truchet et d'Emmanuel Chailloux.*

### Exercice 1 – Bourg Palette

L'aventure commence ici. Dans votre quête, vous aurez besoin d'un "carnet de route" dans lequel vous documenterez toutes difficultés rencontrées en OCaml.

1. Installer l'environnement de travail de OCaml suivant :
   - *Package manager* `OPAM` : `https://opam.ocaml.org/doc/Install.html`
   - Le compilateur avec `opam install ocaml`.
   - Le système de *build* avec `opam install dune`.
   - Un éditeur de code / environnement de développement, au choix :
     - `sublime-text` avec `Merlin`.
     - `emacs` avec `Tuareg` (`http://tuareg.forge.ocamlcore.org/`).
     - `emacs` avec `Merlin` (`https://github.com/ocaml/merlin`).
     - `Eclipse` et plug-in Ocaide (`http://www.algo-prog.info/ocaide/`).
     - `Atom script` (`https://atom.io/packages/script`).

   À l'avenir toutes nouvelles dépendances OCaml s'installera avec `opam`.
2. Tapez dans un fichier `hello.ml` le programme suivant :

   ```
   let _ = Printf.printf "Hello world\n"
   ```

   Compiler et exécuter ce programme avec `ocaml hello.ml` et puis avec

   ```
   ocamlc hello.ml -o hello
   ./hello
   ```
3. Dans un fichier `in_out.ml` demander à l'utilisateur de saisir un nombre et afficher ce nombre.
4. Dans un fichier `odd_even.ml`, demander un nombre à l'utilisateur, afficher "odd" si le nombre est impair et "even" sinon.
5. Dans un fichier `mention.ml`, demander à l'utilisateur la note $n$ qu'il a obtenu au dernier examen et afficher sa mention sachant que :
   - $n < 10$ = Echec
   - $10 \leq n < 12$ = Peut mieux faire
   - $12 \leq n < 14$ = Presque bien
   - $14 \leq n < 17$ = Bien

— $17 \leq n$ = Votre enfant a t'il une vie ?

6. Dans un fichier `numbers.ml`, demander un entier $n$ à l'utilisateur. Afficher tous les entiers de $n$ à 1 (un par ligne), par ordre décroissant et puis croissant. Si l'utilisateur rentre 2 on affiche :

```
2
1
1
2
```

7. Dans un fichier `rectangle2D.ml`, demander deux entiers $n$ et $m$ à l'utilisateur. Afficher un rectangle $n \times m$ uniquement composé d'étoiles (*).

## Exercice 2 – Inférence de type

1. Annoter les expressions suivantes avec leurs types (ainsi que leurs sous-expressions). Par exemple :

```
fun x -> (x - 3) > 0     ----->
  (fun (x:int) -> ((((x - 3):int) > 0):bool):(int -> bool))
```

Faites de même pour les expressions suivantes :

```
2 + 40
not false
"aoi" ^ "sora"
fun x -> x + 1
fun x -> x + 1
fun x -> fun y -> x + y
fun x y -> x + y
let f = (fun x y -> x) 3
(fun x y -> x + y) 3
if x then y else z + 1
if (x = 3) then 2.
fun x -> x *. 3.
fun x -> fun f -> (f (x + 1)) - 2
fun x -> fun f -> (f (x + 1)) -. 2.
```

2. Pour chacun des types ci-dessous, écrire au moins deux expressions (significativement différentes) de votre choix pour lesquelles OCaml infère le type considéré :

```
int
char -> int
int -> int
float -> int
float -> float -> float
(int -> int) -> int
int -> int -> int
(int -> int -> int) -> int -> (int -> int)
```

## Exercice 3 – Décomposition en fonctions

1. Dans un fichier `guess.ml`, on va écrire un mini-jeu de hasard. Choisissez 3 entiers au hasard. Demandez un entier à l'utilisateur. Si il est strictement supérieur à exactement 2 des 3 choisis (donc plus petit que le troisième), il gagne deux points. Si il est égal à au moins un des 3 choisis, il gagne un point. Sinon il ne gagne rien. Choisissez un nombre entre 0 inclus et 7 exclus avec la fonction `Random.int 7`. Le programme ne réalise qu'une seule manche.

2. Dans un fichier `guess-fun.ml`, découper le programme `guess.ml` en plusieurs sous-fonctions dont :

```
ask_number_to_user : unit -> int
(* 'count_points n' where 'n' is the user's number. *)
count_points: int -> int
print_points_earned: int -> unit
round : unit -> unit
```

3. **Supplémentaire.** Permettre à l'utilisateur de faire plusieurs manches à ce jeu et comptabiliser les points. Afficher le score total au fur et à mesure du jeu. Permettre à l'utilisateur de quitter.
4. Reprenez `rectangle2D.ml` et factoriser le en proposant et utilisant une fonction :

```
print_n_times: int -> (int -> string) -> unit
```

   Soit `print_n_times 4 (fun x -> (string_of_int x))` cette fonction affiche `1234`. Notez qu'on peut simplement écrire `print_n_times 4 string_of_int` (mais qu'est-ce c'est bô).
5. **Supplémentaire.** Dans un fichier `empty_rectangle2D.ml`, demander deux entiers $n$ et $m$ à l'utilisateur. Afficher un rectangle $n \times m$ composé d'étoiles (*) sur les bords mais dont le centre est vide. Créer des fonctions pour vous aidez, par exemple :

```
print_line: int -> unit
print_border: int -> unit
print_rectangle: int -> int -> unit
```

   Essayer votre programme avec $1 \times 4$, $4 \times 1$, $0 \times 10$.

## Exercice 4 – Récursion terminale

1. Définissez une fonction calculant la factorielle de manière naïve et puis de manière récursive terminale.
2. Définissez une fonction calculant la puissance de manière naïve et puis de manière récursive terminale.