Fiche OCaml numéro 0, # Fiche pratique

Cette fiche s'adresse aux débutants en Caml, et donne des indications élémentaires pour développer ses premiers programmes. Avec l'expérience, on pourra envisager des approches plus sophistiquées.

1 Éditer et faire tourner vos programmes Caml

1.1 Développer vos programmes

Voici un exemple typique de code Caml:

```
let x = 3 in
let f = fun y -> y+1 in
f x
```

Écrivez vos programme dans des fichiers dont le nom est en minuscules, ne comporte pas de `-` (mais peut comporter des _), et ***se terminant par le suffixe*** `.ml`

Dialoguer avec Caml. On vous recommande d'utiliser l'éditeur de texte Emacs (cherchez-le dans vos menus), qui normalement détecte, grâce au suffixe `.ml`, que vous êtes en train de coder en Caml[1].

Emacs admet un mode nommé `tuareg` facilitant le travail avec Caml. Avec un peu de chance, le mode se met en place automatiquement à partir du moment où vous ouvrez un fichier se terminant par `.ml` .

Voici les raccourcis clavier essentiels:

`C-x C-f`	Ouvrir un nouveau fichier.	raccourcis Emacs
`C-x C-s`	Sauvegarder le fichier courant.	
`C-x b`	Changer de fichier courant.	
`C-h C-h`	Aide sur l'aide d'emacs.	
`C-c C-e`	Évaluer la phrase courante dans ocaml.	raccourcis `tuareg`
	La première fois que vous faites cette manipulation, Emacs vous pose des questions auxquelles il faut répondre en tapant sur *Entrée*, ce qui a pour effet de lancer Caml.	
`C-c C-r`	Évaluer la région courante dans ocaml.	
`C-c C-k`	Interrompre ocaml (quand ça boucle).	
`C-c h`	Afficher l'aide *OCaml* sur un symbole de la librairie standard.	
`C-c C-h`	Afficher les raccourcis de `tuareg`.	

À noter que pour aller plus vite, vous pouvez ouvrir un terminal, lancer `ocaml` (ou, mieux, `ledit ocaml`), et taper directement votre code. Si vous ne savez pas de quoi il retourne dans la phrase qui précède, ignorez-la.

Compiler. Il vous est recommandé d'utiliser l'outil `ocamlbuild`.

Deux mots à ce sujet : supposez que vous ayez écrit un programme qui se décompose en 3 fichiers :

- `titi.ml` définit une fonction `f`
- `toto.ml` définit une fonction `g`, qui fait appel à `Titi.f`
- `tutu.ml` contient la fonction principale (le "main"), et commence par `open Titi` et `open Toto`. Dans `tutu.ml`, il est fait appel à `f` et `g`.

[1] Installez Emacs sur votre ordinateur personnel, et installez avec le mode `tuareg`, qui gère le dialogue entre Emacs et Caml.

Vous tapez alors `ocamlbuild tutu.native`

et tous ces fichiers seront compilés, dans le bon ordre. S'il n'y a pas d'erreur, vous obtiendrez un programme exécutable `tutu.native`.

Remarque : on illustre ici deux manières d'appeler une fonction `f` qui a été définie dans un fichier `titi.ml` : soit vous invoquez `Titi.f`, soit vous faites d'abord `open Titi`, et ensuite vous appelez simplement `f`. La seconde solution peut être source d'ambiguïtés (si plusieurs fichiers définissent leur version de `f`).

1.2 Bêtises de notations/appellations

Caml ou OCaml? Pour ce cours, on parlera indifféremment de Caml et d'OCaml. Il ne sera a priori pas question du "O", qui fait référence à la couche objet du langage.

Les ``;;''. Caml comprend des définitions à l'aide de `let`, comme dans

```
let double = fun k -> k*2
```

Vous pouvez terminer une telle définition par `;;` pour dire "j'ai fini".

Vous pouvez aussi lui dire, sans faire de définition

```
double 16;;
```

et alors les `;;` sont nécessaires (pour dire "vas-y, calcule-moi ça").

On trouve souvent du code où il n'y a *que des définitions*, avec des choses comme

```
let _ = double (double 416)
```

Ici le "_" sert à dire "je ne donne pas de nom à la chose que je définis, seul m'importe le résultat".

Dans un style où on mélange définitions et "évaluations directes", l'absence de `;;` peut engendrer des messages d'erreur peu suaves, comme dans

```
let double = fun k -> k*2

double 26

let a = double 16
```

Ci-dessus, il faut mettre un `;;` après `k*2`, pour signaler que `double 26` est autre chose.

2 Choses à savoir faire un jour ou l'autre

Affichage (très) élémentaire.

```
print_string "resultat : ";
print_int (k+x);
print_newline ()      (* va a la ligne *)
```

À noter l'argument `()` pour `print_newline`

Un peu plus structuré: `printf`. Voici un exemple :

```
Format.printf "Le resultat apres %d essais est %f.\n" 52 3.2
```

Explications:

- on appelle la fonction `printf`, qui est définie dans la librairie `Format`;
- `%d` pour un entier, `%f` pour un flottant, `\n` signifie aller à la ligne;
- comprendre le type de `Format.printf`, c'est pour les grands (on peut très bien se contenter d'une utilisation naïve dans un premier temps).

Écrire dans un fichier. à compléter •••

3 Points "avancés": options, performances

Récupérer les options pour un exécutable. à compléter •••

Mesurer le temps d'exécution, profiler. à compléter •••

4 Liens www

La librairie standard de OCaml. La *librairie* `Pervasives`, qui est toujours ouverte dans Caml.