

# LICENCE SCIENCES & TECHNOLOGIES 1<sup>re</sup> ANNÉE

## UE INF201, INF231 ALGORITHMIQUE ET PROGRAMMATION FONCTIONNELLE

2019 / 2020

---

### UNE SÉANCE D'EXPÉRIMENTATION TYPE

---

L'objectif de ce chapitre est de vous permettre de vous familiariser avec l'environnement OCAML. Il explique comment lancer un éditeur et l'interpréteur OCAML, utiliser ces deux outils pour construire des programmes et les tester, construire le compte rendu de TP et terminer la session.

## 1 Déroulement d'une séance de TP

Pour débiter cette première séance, connectez-vous sur un serveur Linux du DLST, par exemple turing.

### 1.1 Installation de l'environnement pour le langage OCAML

Pour pouvoir travailler avec le langage OCAML, nous suggérons l'utilisation d'un environnement composé de deux outils :

- *Un éditeur de texte* basique, par exemple `gedit`. Cet outil, dont l'interface est simple et intuitive, vous permettra d'éditer vos fichiers sources.
- L'interpréteur OCAML (`ocaml`), pour évaluer votre code source et le mettre au point.

Pour mettre en place cet environnement, il faut lancer l'éditeur et l'interpréteur dans deux fenêtres différentes. Dans l'interpréteur de commandes Linux, taper :

```
1. gedit crtp1.ml &
```

`crtp1.ml` est le nom du fichier (compte rendu de TP n°1). Ne pas oublier le `&` final<sup>1</sup>. Cet éditeur démarre souvent en mode plein écran, retailler sa fenêtre si besoin.

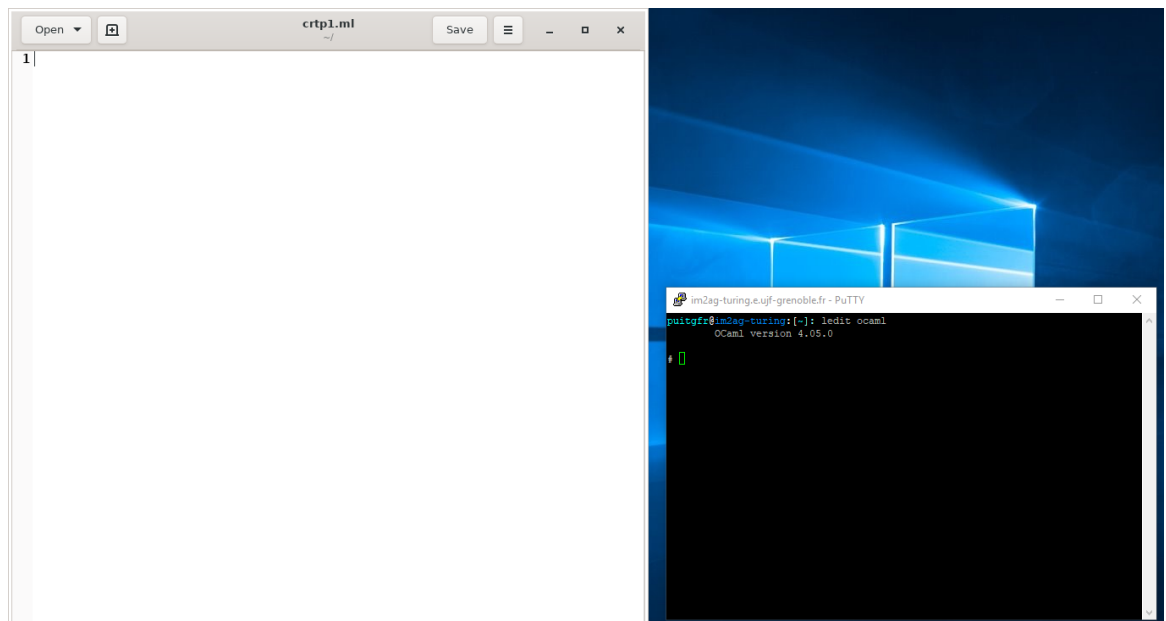
---

1. La signification du `&`, et plus généralement des commandes Linux n'est pas détaillée dans cette UE.

2. `ledit ocaml`

Nous vous conseillons la commande : `ledit ocaml` plutôt que `ocaml` tout court ; vous pourrez ainsi rappeler une phrase précédente avec la touche `↑` du clavier (flèche vers le haut).

Vous pouvez arranger les fenêtres pour obtenir un écran analogue à celui ci-dessous, dans lequel l'éditeur est placé en haut à gauche de l'écran ; il contiendra in fine le compte rendu. L'interpréteur OCAML apparaît en bas à droite de l'écran.



Il est agréable de travailler avec les deux fenêtres pleinement visibles. Pour cela :

- réduire la taille de la fenêtre de l'interpréteur ;  
un clic droit sur la barre de titre permet de choisir une taille de police de 9 dans le menu `Change Settings > Appearance > Font`.
- demander à `gedit` d'afficher une marge à la colonne 80 ;  
menu burger `☰`, `Preferences`, cocher « `Display right margin at column 80` » ; retailer la fenêtre de `gedit` de façon à laisser juste apparaître cette marge ; lors de l'édition du code, il est conseillé de ne pas trop dépasser cette limite.

## 1.2 Édition du fichier de compte rendu

Commencez tout d'abord par insérer un entête de compte rendu, comme par exemple (n'oubliez pas de modifier les noms, ...) :

```
(* crtp1.ml : TP n°1
*****
*
*   PARTIE I : LANGAGES DES EXPRESSIONS ET DES FONCTIONS
*
*)
```

```

*
*****
* Date      : 1er janvier 2020
* Groupe   : 03
* Étudiant 1 : Basset Nicolas
* Étudiant 2 : Puitg François
*****
*)

```

Avant de continuer, sauvegarder le fichier, par exemple sous le nom `crtp1.ml`. Pensez à sauvegarder régulièrement votre compte rendu au cours de la séance, il arrive qu'il y ait des coupures de courant.

### 1.3 Utilisation de l'interpréteur OCAML

L'interpréteur OCAML (dans l'exemple en bas à droite de l'écran) affiche le caractère d'invite suivi du curseur : `# _.`

Lorsque vous taperez une expression terminée par deux points virgules (`;;`), celle-ci sera évaluée par l'interpréteur, qui en retour affichera un résultat. Tapez par exemple l'expression suivante dans la fenêtre de l'interpréteur OCAML :

```
3 + 12 ;;
```

Après avoir appuyé sur la touche `[Entrée]` vous obtiendrez le résultat suivant :

```
- : int = 15
#
```

L'interpréteur a évalué l'expression et a affiché comme résultat la valeur correspondante (15) ainsi que le type de l'expression (`int` pour «integer»). Après chaque évaluation, l'interpréteur affiche de nouveau un caractère d'invite (`#`) et attend une nouvelle expression.

Tapez maintenant :

```
x * 3
```

puis appuyez immédiatement sur `[Entrée]` sans taper les caractères `;;`. Le curseur passe à la ligne, mais aucun résultat n'est affiché : vous pourrez ainsi entrer des expressions complexes en utilisant plusieurs lignes. Tant que vous ne taperez pas les deux points virgules (`;;`), l'interpréteur ne fera rien, si ce n'est attendre la fin de l'expression à évaluer. Les deux points virgules permettent en fait de délimiter l'expression.

Tapez les deux points virgules puis `[Entrée]` et observez la réponse du système :

```
# x * 3
;;
Error : Unbound value x
#
```

ce qui signifie que la variable `x` n'est pas liée (`unbound`) : elle est absente du contexte d'évaluation courant. Dans cet exemple, l'interpréteur a répondu par un message d'erreur et souligne l'endroit présumé de l'erreur dans la ligne. Après avoir appuyé sur `[Entrée]` à la fin d'une ligne, vous pouvez la rappeler en appuyant sur la touche `[↑]`.

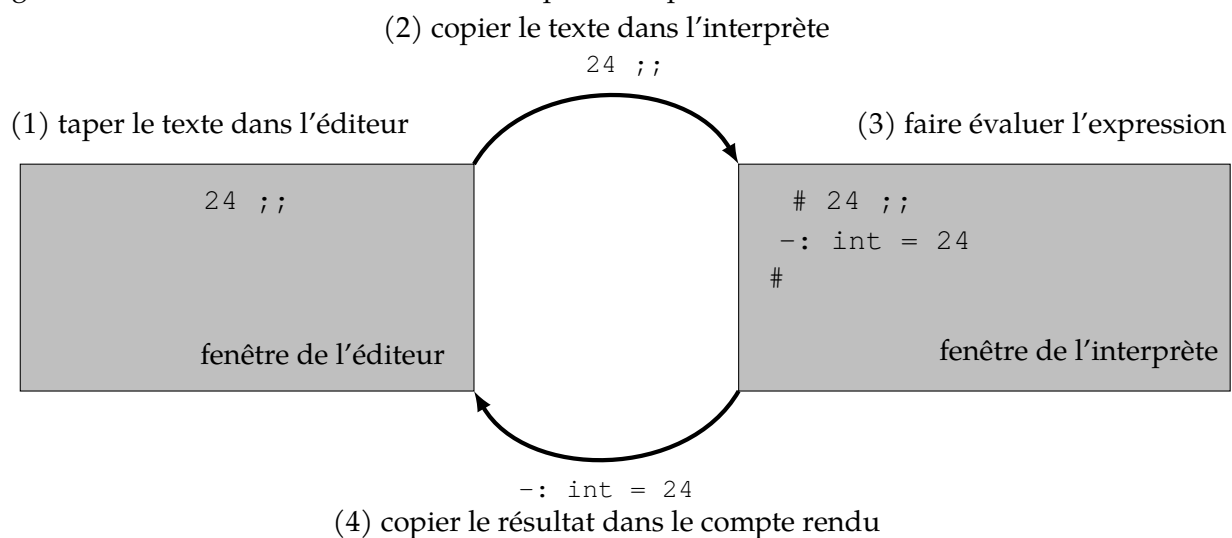
**Par ailleurs, lorsque vous quitterez l'interpréteur OCAML, tout ce que vous avez tapé sera perdu. Il est donc nécessaire d'utiliser à la fois l'interpréteur OCAML et l'éditeur.**

## 1.4 Utilisation conjointe de l'éditeur et de l'interpréteur OCAML

Au cours des différents TP, lorsque vous devrez taper des expressions ou des fonctions complexes il sera sans doute préférable d'utiliser la méthode suivante (ne rien faire pour l'instant si ce n'est comprendre la méthode). Il faudra :

1. Taper le texte souhaité dans la fenêtre de l'éditeur. Vous pourrez ainsi utiliser toutes les fonctions d'édition disponibles (couper, coller, etc.).
2. Transférer le texte dans la fenêtre de l'interpréteur afin de l'évaluer.
3. L'interpréteur OCAML donnera sa réponse.
4. Si le résultat est celui attendu, vous pourrez le copier-coller dans le fichier du compte rendu, sous la forme d'un commentaire OCAML : (\* ... \*).

La figure ci-dessous montre les différentes étapes de ce processus :



Pour réaliser les opérations de transfert entre les deux fenêtres, on utilise classiquement le copié-collé. Celui-ci peut se faire efficacement en utilisant uniquement la souris. Dans l'exemple suivant, on copie une expression simple de l'éditeur vers l'interprète.

1. Sélectionner l'expression, point-virgules compris ; dans le cas présent : `24 ;;`. Au relâchement du clic gauche, le texte a été automatiquement copié.
2. Déplacer le pointeur de la souris dans la fenêtre de l'interpréteur.
3. Effectuer un clic droit<sup>2</sup> ; le texte précédemment sélectionné est alors collé (à l'endroit indiqué par le curseur).
4. Valider en appuyant sur `Entrée`. L'interpréteur OCAML affiche sa réponse.
5. Transférer cette réponse dans le compte rendu en utilisant la méthode inverse (copier dans l'interpréteur et coller dans l'éditeur).

2. Selon l'éditeur ou le système d'exploitation utilisé, il peut s'agir plutôt d'un clic central, ou d'un clic des deux boutons si la souris ne comporte pas de troisième bouton

Apr s ces op rations, votre compte-rendu doit ressembler   cela :

```
(* Expressions           R ponses du syst me *)
24 ;;                    (* - : int = 24 *)
3+4 ;;                  (* ... *)
```

Noter les caract res (\* et \*) qui encadrent un texte destin    l'utilisateur humain : il ne sera pas  valu  par OCAML lors d'une lecture ult rieure du fichier.

**Remarque** Pour aller plus vite, il est possible d' tendre la s lection jusqu'au retour   la ligne, ce qui  vite d'avoir ensuite   taper sur  dans l'interpr teur.

Le paragraphe 1.8 sugg re une autre fa on de travailler en utilisant l'environnement de d veloppement `emacs`. Vous essaierez pendant la s ance de TP non encadr e et choisirez ensuite la m thode qui vous convient le mieux.

## 1.5 Construction du compte rendu de TP

### 1.5.1 Objectifs

Dans tous les cas de figure, construire le compte rendu est essentiel :

- il doit vous permettre de garder la trace des exp rimentations effectu es et donc de vous pr parer aux examens ;
- il doit permettre   vos enseignants d' valuer le travail effectu  pendant chaque s ance et de vous donner des conseils pour les prochaines s ances.

Le compte rendu est un fichier OCAML (dont le nom se termine par `.ml`) contenant les d finitions des objets que vous avez  labor es au cours d'une s ance **ainsi que les tests effectu s**.

Lorsque vous sortirez de l'interpr teur OCAML, toutes les d finitions seront perdues, seul le contenu du fichier de compte rendu (que vous aurez sauvegard ) sera conserv .

**Pour charger   nouveau les d finitions de fonction lors des s ances suivantes, tapez la commande : `#use "crtpl.ml" ; ;` NB : il faut taper le # ci-dessus en plus du # de l'interpr teur.**

Cette directive permet aussi de v rifier qu'il n'y a pas d'erreur de syntaxe dans le fichier. Elle peut  tre utilis e juste avant la fin d'une s ance de TP pour laisser le fichier dans un  tat stable, et/ou au d but de chaque s ance pour v rifier que l'on repart sur des bases saines.

### 1.5.2 Structure du compte rendu

 laborer un compte rendu est un travail aussi important qu'utile. Comme le fichier r sultant doit pouvoir  tre interpr t  par OCAML (pour pouvoir le relire lors des s ances suivantes), toutes les informations qui ne correspondent pas   des expressions ou des fonctions OCAML valides doivent  tre mises entre commentaires (c'est   dire entre (\* et \*) en OCAML).

Dans le compte rendu, pour chaque fonction vous devrez fournir :

1. *La spécification de la fonction.* Il est impératif de donner la spécification *AVANT* de débiter la réalisation.
2. *Une réalisation en OCAML* avec des commentaires significatifs. La présentation des fonctions OCAML doit être soignée. En particulier, il est impératif que la mise en page fasse apparaître la structure (indentation).
3. *Les jeux de tests* validant la correction de la fonction réalisée. Chaque jeu d'essais doit être accompagné d'un commentaire justifiant la pertinence des données choisies.
4. Dans certains cas *la trace de l'évaluation de la fonction* vous sera demandée.

### 1.5.3 Exemple de compte-rendu



```
(* crtp1.ml : TP n°1
*****
*
*      PARTIE I : LANGAGES DES EXPRESSIONS ET DES FONCTIONS
*
*****
* Date      : 1er janvier 2020
* Groupe    : 03
* Étudiant 1 : Basset Nicolas
* Étudiant 2 : Puitg François
*****
*)
(* EXERCICE 1.2
  somme3 : int -> int -> int -> int
  Sémantique : somme de trois entiers
  Exemples et propriétés : ...
  Algorithme : utilisation de +
*)
let somme3 (a:int) (b:int) (c:int) : int =
  a + b + c
;;
(* Tests : *)
assert ((somme3 1 2 3) = 6) ;;          (* cas général *)

assert ((somme3 (-1) 2 (-1)) = 0) ;;  (* entiers négatifs *)
assert ((somme3 0 0 0) = 0) ;;        (* cas zéro *)
```

Le fichier de compte-rendu doit être conservé : il constituera un support de révision.

## 1.6 Fin de la séance

N'oubliez pas avant de quitter l'éditeur de vérifier que le compte-rendu est syntaxiquement correct (directive `#use`).

Pour sortir de l'interpréteur OCAML taper la commande `#quit ; ;` ou plus rapidement  $\wedge$ D ( .

Si l'enseignant vous le demande, envoyez-lui le compte-rendu par courrier électronique : voir paragraphe 1.7.

N'oubliez pas de quitter également la session Windows.

## 1.7 Envoi de compte rendu par courrier électronique

L'envoi de compte rendu par courrier électronique respectera les règles suivantes :

1. Le corps du message décrit de quoi il s'agit : nom du tp, nom(s) des membres du binôme, état d'avancement, et autre(s) information(s) utile(s).
2. Le compte rendu est joint sous la forme d'**un seul fichier en attachement**. S'il se compose de plusieurs fichiers, regroupez-les en un seul fichier d'archive au format tar ou zip et attachez cette archive.
3. Le champ sujet est de la forme « [INF121etd] noms des étudiants du binôme »

Exemple : envoi d'un compte rendu par binôme Alfred Mart et Auguste Dup à François Puitg.

- Un seul fichier tp3.ml à rendre → le renommer :  
`mv tp1.ml mart_dup_tp3.ml`
- Deux fichiers tp3a.ml et tp3b.ml → créer une archive :  
`tar cvf mart_dup_tp3.tar tp3a.ml tp3b.ml`

Assurez-vous que les deux variables d'environnement nécessaires soient définies : LOGNAME (votre\_login) et EDITOR (gedit ou emacs). Vérification : `echo $EDITOR ; echo $LOGNAME`.

La commande d'envoi est : `mutt -s "SUJET" -a document_a_attacher -- destinaire` soit pour notre exemple :

```
mutt -s "[INF201] mart dup INF02" -a mart_dup_tp3.ml -- puitg@imag.fr
ou
mutt -s "[INF201] mart dup INF02" -a mart_dup_tp3.tar -- puitg@imag.fr
```

Lors de la première utilisation, mutt peut proposer de créer un répertoire Mail/ → taper  ou  pour accepter.

1. Mutt affiche le destinataire : taper  pour accepter.
2. Mutt affiche le sujet : taper  pour accepter.
3. Mutt lance votre éditeur de fichier : saisissez le corps du message, sauvegardez puis quittez l'éditeur.
4. Mutt affiche toutes les composantes du message : taper  pour envoyer le message.

5. `Mutt` se termine et l'interpréteur de commande affiche le message d'invite (`login@turing : ...`) et attend la commande suivante.

Autres options de `mutt` : voir `man mutt`.

Les options `-c destinataire_en_copie` et `-b destinataire_en_copie_cachée` permettent d'ajouter un destinataire en copie de votre message.

## 1.8 L'éditeur `emacs`

Pour vous simplifier le travail, `emacs` – un puissant environnement de développement intégré – permet d'évaluer des expressions directement depuis le code source sans devoir les copier dans l'interpréteur à la main. Cela permet un gain de productivité non négligeable. Pour cela, suivez les instructions ci-dessous :

1. Lancer `emacs` sur le serveur `turing`.
2. Ouvrir un nouveau fichier par exemple `crtp1.ml`.
3. Taper dans ce nouveau buffer vierge une expression, par exemple `3 + 12 ; ;`.
4. `Ctrl-C-E` permet d'évaluer l'expression courante (c'est-à-dire celle dans laquelle se trouve le curseur) dans la boucle interactive de OCAML sans avoir à faire de copié-collé.
5. À la première utilisation de cette commande, `emacs` demande à l'utilisateur depuis le minibuffer (la ligne inférieure de la fenêtre) s'il souhaite utiliser l'interpréteur standard (`ocaml`) ; confirmer en tapant entrée.
6. Une nouvelle fenêtre doit s'ouvrir et afficher le résultat de l'évaluation. Les résultats suivants seront toujours affichés dans ce nouveau buffer nommé `*Ocaml*`.

Il est également possible de :

- n'évaluer que la sélection courante, grâce à `Ctrl-C-R` ;
- évaluer l'ensemble du buffer grâce à `Ctrl-C-B`.

Pour quitter l'interpréteur, taper `Ctrl-C-K`.