TP1: Terminal Unix, et premiers pas en OCaml

L'objectif de ce TP est de se familiariser avec un système d'exploitation type Unix, puis d'apprendre à écrire quelques programmes basiques en OCaml.

Système d'exploitation, machine virtuelle

Le travail en cours d'informatique se fait sous un système d'exploitation appelé Linux, qui fait partie de la famille plus large Unix. Les ordinateurs du lycée sont sous Windows, mais disposent d'un logiciel qui permet de **simuler** un ordinateur sous Linux, ce que l'on appelle une **machine virtuelle**. Ce logiciel est **VMWare Workstation** : localisez-le sur le bureau, puis lancez la machine virtuelle "NonOS". Lorsqu'elle est lancée, connectez-vous à la session avec le mot de passe **concours**.

Sauvegarder son travail

ATTENTION, vous n'avez pas de session personnelle sur les machines virtuelles au lycée. Si vous y laissez un fichier, aucune garantie que vous le retrouverez à la session suivante. Donc, à la fin de chaque cours/TP, **mettez votre travail sur une clé USB** ou envoyez-le vous par mail. Lorsque vous branchez votre clé USB, une fenêtre s'affiche et vous demande si vous voulez connecter votre clé à la machine virtuelle où à Windows. Connectez-la à la machine virtuelle, et vous la verrez apparaître sur celle-ci après quelques secondes. Si ce n'est pas le cas, essayez un autre port USB. Alternativement, il est aussi possible de copier-coller des fichiers de la machine virtuelle vers Windows et inversement, avec CTRL+C / CTRL+V.

Exercice 1

Créer un fichier sur le bureau de la machine virtuelle avec un clic-droit et, en utilisant l'une des méthodes précédentes (clé USB, glisser-déposer ou copier-coller), le transférer hors de la machine virtuelle.

1 Utilisation d'un terminal

Les systèmes Unix, comme celui de la machine virtuelle, disposent d'un **terminal**, qui est une fenêtre dans laquelle on peut taper des commandes. Le terminal permet donc d'interagir avec l'ordinateur en utilisant uniquement le clavier, sans avoir besoin d'interface graphique.

A Dossiers

Les ***dossiers***, ou ***répertoires*** sont des collections de fichiers. Les ordinateurs sont organisés en dossiers, avec une ***structure arborescente***, ce qui signifie que votre ordinateur contient des dossiers, qui eux même contiennent des dossiers, etc...

Il est possible de se déplacer dans l'ordinateur avec un explorateur de fichiers classique comme sur Windows ou Mac, avec la souris. Pour cela, cliquez sur "Applications", en haut à gauche, puis sélectionnez "Gestionnaire de fichiers" pour le lancer. Il y a également un raccourci sur la barre du bas.

Exercice 2

Lancez le gestionnaire de fichier. Vous vous trouvez par défaut dans /home/candidat. Vous devriez voir quelques fichiers et dossiers. Dans Linux, les fichiers dont le nom commence par un point sont considérés comme des fichiers cachés. Vous pouvez choisir de les voir ou pas en appuyant sur CTRL+H.

B Navigation dans le terminal

Dans les TP, on travaillera principalement dans un terminal de l'ordinateur. Cela signifie que l'on interagit avec l'ordinateur en tapant des commandes et pas en cliquant à la souris dans une interface graphique.

Exercice 3

Ouvrez un terminal en cliquant sur l'icône  dans la barre des raccourcis en bas de votre écran. Vous pouvez aussi cliquer sur "Applications" en haut à gauche pour y trouver le terminal (ainsi que toutes les autres applications). Fermez la fenêtre du terminal, si un message d'erreur s'affiche, sélectionnez "ne plus afficher".

Dans un terminal, on navigue de dossier en dossier, et lorsque l'on ouvre un terminal, on se situe par défaut dans le répertoire ***/home/candidat*** : c'est la "page d'accueil" du système de fichier. Dans le langage du terminal, ce répertoire a un surnom : on le note `~`, et on l'appelle le **home** (la maison).

Le répertoire où l'on se trouve lorsque l'on utilise un terminal s'appelle le **répertoire courant**, et il s'affiche sur chaque ligne à gauche de l'invite de commande.
Par exemple, si l'invite de commande ressemble à :

```
candidat@ordi:~/TP1/exo4$
```

Cela signifie que l'utilisateur actuel (candidat) utilise le terminal, dans le répertoire courant `/home/candidat/TP1/exo4`.

Voyons quelques commandes utiles du terminal :

- Pour afficher le contenu du répertoire courant, on utilise la commande ***ls***.
- Pour créer un fichier dans le répertoire courant, on utilise la commande ***touch*** suivi du nom du fichier à créer.
- Pour créer un dossier dans le répertoire courant, on utilise la commande ***mkdir*** suivi du nom du dossier à créer.
- Pour se déplacer vers un autre dossier, on utilise la commande ***cd***. Par exemple, si l'on se trouve dans le dossier "parent", qui contient un sous dossier "parent/enfant", on peut accéder au sous-dossier en tapant `cd enfant` puis, pour revenir en arrière, on utilise la commande `cd ..` (les deux points font partie de la commande). En fait, "`..`" signifie "Le répertoire qui me contient" dans le langage du terminal.
  En tapant uniquement `cd` sans aucun argument, on se téléporte dans le répertoire `~`.

Exercice 4

Lancez un terminal puis utilisez des commandes pour répondre aux requêtes suivantes :

Q1. Créez un répertoire appelé "TP1" puis accédez-y.

Q2. Créez dans ce répertoire un sous-répertoire appelé "terminal" et accédez y.

Q3. Créez un fichier texte appelé ***blabla.txt***.

Q4. Affichez le contenu du dossier avec ls et vérifiez que le fichier a bien été créé.

Q5. Lancez la commande ***ls -a***. Que voyez-vous ? A votre avis, que signifie "-a" (indice : c'est l'initiale d'un mot anglais) ? Quels sont les deux dossiers qui s'affichent ici et qui ne s'affichaient pas avec ls ?

Q6. Lancez la commande ***thunar .*** (le point fait partie de la commande). Vérifiez que cela a bien ouvert un gestionnaire de fichiers dans le répertoire courant du terminal.

C Édition de texte

Pour écrire du code, on utilise un éditeur de texte. Sur la machine virtuelle, vous avez plusieurs éditeurs de texte à disposition, le plus pratique à utiliser est ***VSCode***, dans ***Applications - Développement - VS Code***. C'est un éditeur de code assez puissant, avec un système d'autocomplétion et d'autres fonctionnalités adaptées au développement. Pour ouvrir un fichier dans VSCode directement depuis le terminal, il vous suffit de taper la commande `code nom_du_fichier`. Si le fichier a une extension comme .c (fichiers C), .py (fichiers python), .ml (fichiers OCaml), ou autre, l'éditeur active automatiquement la coloration syntaxique du langage concerné.

Exercice 5

Dans un terminal, créez un fichier "prog.py", puis ouvrez le avec VSCode et tapez votre fonction python préférée. Vérifiez que les mots-clés (def, etc...) sont colorés.

2 OCaml

Comme nous le verrons en cours, un programme OCaml est une grande expression mathématique. Il n'y a pas de variables modifiables comme en Python, et l'on ne dit pas que l'on "exécute" un programme OCaml, mais que l'on **évalue** une expression OCaml.
Pour le moment, on peut donc voir OCaml comme une sorte de calculatrice avancée. On peut évaluer des expressions OCaml directement dans le terminal, en utilisant un interpréteur appelé **utop**.

Exercice 6

Dans un terminal, lancez la commande `utop`, et vérifiez que cela lance un programme. Vous pouvez fermer utop avec CTRL + D, puis le relancer.

Dans utop, on peut taper des expressions mathématiques classiques, qui sont alors évaluées. On doit taper `;;` pour signaler la fin d'une expression. Par exemple, pour évaluer l'expression $(1+2) \times (3+4)$, on tapera :

```
1 (1+2) * (3+4) ;;
```

Exercice 7

Q1. Évaluez l'expression précédente dans utop, et notez ce qui est affiché par l'interpréteur.

Q2. Évaluez les expressions suivantes et vérifiez que les valeurs obtenues sont cohérentes :

- 4
- 12×5
- $(-5) \times (-6)$
- $11/3$

On remarque qu'OCaml affiche toujours "int" après les expressions précédentes : c'est le **type** de l'expression. OCaml détecte et affiche automatiquement le type de tout ce que l'on écrit.

Exercice 8

Étudions quelques types de base d'OCaml. Tapez les expressions suivantes, et notez leurs types :

1. `5`
2. `12.32`
3. `true`
4. `'h'` (guillemets simples)
5. `"bonjour"` (guillemets doubles)

En OCaml, chaque type est muni d'opérations basiques. Par exemple, nous avons vu précédemment que le type `int` est muni des opérations $+, -, /, \times$. Le type `float` est incompatible avec ces opérations : en OCaml, interdiction de mélanger les entiers et les flottants. Pour additionner deux flottants, on doit utiliser l'opérateur `+.`, qui se lit donc "addition flottante" :

```
5.3 +. 1.2;;
- : float = 6.5
```

La même syntaxe fonctionne pour les opérateurs $-, /, \times$.

Exercice 9

Écrire quelques expressions de type `float` utilisant les 4 opérateurs flottants.

Les opérateurs de comparaison $<, \leq, =, \neq, \geq, >$ se notent respectivement `<, <=, =, <>, >=, >`. Contrairement aux opérateurs arithmétiques, ils se notent pareil pour les entiers, les flottants, et même les autres types ! En revanche, on ne peut comparer que des expressions **de même type** :

```
1 + 1 <= 5 ;; OK, vaut true
1 + 10 <= 5 ;; OK, vaut false
1 + 12 <= 53.5 ;; ERREUR: pas de comparaison entre un int et un float
```

Exercice 10

Q1. Évaluer les expressions suivantes :

1. `1 + 1 = 2`
2. `1.2 < 2.05`
3. `"bla" >= "chapeau"`
4. `1 + 1 <> 3`

Q2. Déterminer l'ordre par défaut sur le type `string` et sur le type `bool`.

Le type `bool` est aussi muni d'opérations, qui permettent de faire le ET, OU et NON booléen, comme en Python :

- `&&` est le ET logique
- `||` est le OU logique
- `not` est le NON logique

Par exemple, considérons l'expression OCaml suivante :

```
(1 + 1 = 2) && not (1 < 5);;
```

Elle est de type `bool`, et s'évalue à **true**, car `1 + 1 = 2` vaut **true** mais `not (1 < 5)` vaut **false**.

Exercice 11

Taper une expression OCaml monstrueuse utilisant TOUS les opérateurs vus pour l'instant :

```
+, -, *, /, +., -., *., /., <, <=, =, <>, >=, >, ||, &&, not
```

Copiez-collez-la (clic-droit dans le terminal pour copier) dans un fichier à part, on la fera grandir plus tard.

Fonctions

OCaml est un langage **fonctionnel**, ce qui signifie entre autres que les fonctions y jouent un rôle central. Voyons notre première fonction, définie par défaut en OCaml : `int_of_float`. Cette fonction prend en entrée un flottant, et renvoie sa partie entière, le transformant en entier. C'est donc la fonction qu'on noterait en maths :

$$\texttt{int\_of\_float} : \begin{array}{ccc} \mathbb{R} & \to & \mathbb{N} \\ x & \mapsto & \lfloor x \rfloor \end{array}$$

Pour **utiliser** une fonction, en OCaml, pas besoin de parenthèses : écrire `f x` signifie "appliquer f sur x". Par exemple :

```
int_of_float 5.6;;
- : int = 5
```

A l'inverse, la fonction `float_of_int` prend en entrée un entier, et renvoie ce même entier, avec le type `float` :

```
float_of_int 5 = 5.0;;
- : bool = true
```

En OCaml, l'application de fonction est prioritaire sur presque tout, et parenthèses servent à créer des blocs unitaires pour contourner les priorités. Par exemple, si l'on veut appliquer `float_of_int` à l'expression 1+2, on doit écrire `float_of_int (1 + 2)`, car si l'on écrit seulement `float_of_int 1 + 2`, OCaml comprendra `(float_of_int 1)+ 2`, ce qui est mal typé !

Exercice 12

Vérifier sur quelques exemples qu'appliquer `float_of_int` puis `int_of_float` sur un entier n redonne n.

Définir une fonction

Les fonctions OCaml sont des objets de **première classe**, ce qui signifie que l'on peut les manipuler aussi simplement que les entiers, les flottants, etc... Par exemple, une fonction seule est une expression valide :

```
1 float_of_int;;
2 - : int -> float = <fun>
```

OCaml ne peut pas afficher la valeur d'une fonction, mais est capable de déterminer son type. Ici, la fonction `float_of_int` est de type `int -> float`, lu "int donne float", ou "int flèche float", ce qui signifie qu'elle prend en entrée un entier et renvoie un flottant. C'est équivalent à la notation mathématique :

$$f : \mathbb{N} \longrightarrow \mathbb{R}$$

OCaml nous permet de définir nos propres fonctions, avec le mot clé `fun`. Par exemple, pour définir la fonction mathématique

$$f : \begin{array}{ccc} \mathbb{N} & \to & \mathbb{N} \\ x & \mapsto & 2x+1 \end{array}$$

on écrit :

```
1 fun x -> 2*x + 1;;
2 - : int -> int = <fun>
```

On peut ensuite appliquer une fonction comme précédemment : on juxtapose simplement la fonction puis son argument. Ici aussi, les parenthèses servent à faire des blocs unitaires pour forcer la priorité des opérations.

Exercice 13

Tapez l'expression suivante, vérifiez son type et sa valeur :

```
1 (fun x -> 2*x + 1) (3 * 2);;
```

Que se passe t-il si vous essayez d'appliquer la fonction sur un flottant plutôt que sur un entier ?

Exercice 14

Faire grandir votre expression monstrueuse des exercices précédents pour y ajouter deux fonctions définies avec le mot-clé `fun`. *Indication : vous pouvez écrire l'expression sur plusieurs lignes pour plus de clarté.*

Fonctions d'ordre supérieur

Une fonction d'ordre supérieur est une fonction qui prend en entrée une fonction, et/ou renvoie une fonction. Vous en avez déjà croisé une en maths/physique/SI : la **dérivée** :

$$\begin{array}{rccc} \frac{d}{dx} : & \mathcal{C}_1(\mathbb{R}) & \to & \mathcal{C}_0(R) \\ & f & \mapsto & \frac{df}{dx} \end{array}$$

Ainsi, la fonction **dérivée** appliquée à la fonction $x \mapsto x^2$ donne la fonction $x \mapsto 2x$.

Voici un exemple plus simple en OCaml :

```
fun f -> 2 * f 0;;
- : (int -> int) -> int : <fun>
```

Cette expression est une fonction qui, étant donnée $f : \mathbb{Z} \to \mathbb{Z}$, renvoie $2 \times f(0)$.

Exercice 15

Appliquer la fonction ci-dessus à la fonction $x \mapsto x + 3$ et vérifier le résultat.

Exercice 16

Pour $k \in \mathbb{Z}$, on note $f_k : x \mapsto x + k$. Autrement dit, f_k est la fonction "ajouter k". Définir en OCaml la fonction $k \mapsto f_k$, et vérifier que $f_{12}(15) = 27$.

Exercice 17

On ne peut pas définir la dérivée en OCaml, car le concept de limite ne peut pas être traduit de manière exacte en code, mais on peut faire une approximation, en prenant un pas de $dx = 0.001$ par exemple. On considèrera donc que pour une fonction f réelle dérivable, sa dérivée en $a \in \mathbb{R}$ est :

$$\frac{d}{dx}(f)(a) = \frac{f(a + 0.001) - f(a)}{0.001}$$

Implémenter la fonction dérivée en OCaml, et vérifier sur des exemples que :

1. Pour $f : x \mapsto x^2$, $\frac{d}{dx}(f)(x) = 2x$
2. Pour $f : x \mapsto \frac{1}{x}$, $\frac{d}{dx}(f)(x) = \frac{-1}{x^2}$
3. Pour $f : x \mapsto \sin(x)$, $\frac{d}{dx}(f)(x) = \cos(x)$ (les fonctions sin et cos existent par défaut en OCaml)

Essayez de faire changer le pas utilisé dans la dérivée pour voir comment varie la précision.

Exercice 18

Ajoutez quelques fonctions d'ordre supérieur à votre expression monstrueuse.