Travaux Pratiques d'AP3 n°1

—Licence Informatique, 3ième année, 1ier semestre—

L'interpréteur, les fonctions, la conditionnelle

Le but de ce TP est de vous familiariser avec les concepts de base du langage OCaml, d'apprendre à manipuler l'interpréteur avec Emacs, de programmer vos premières fonctions récursives ou non, d'utiliser correctement la conditionnelle et les définitions temporaires. Vous **devez** toujours donner des exemples d'utilisation des fonctions que vous définissez.

---

1 Prise en main de l'interpréteur

Voici tout d'abord quelques indications pour démarrer :

- ✖ Pour écrire des programmes O'Caml, on pourra utiliser l'éditeur de textes Emacs, qui intègre un environnement pour lancer l'interpréteur O'Caml. Dans un terminal, on écrit une commande `emacs monpgm.ml &`.
- ✖ Une fois l'éditeur lancé, celui-ci doit être dans le "mode caml". On voit cela sur la barre au bas de la fenêtre de l'éditeur où le mot "Tuareg" doit apparaître et un menu "Tuareg" doit apparaître dans la barre des menus.
- ✖ Dans l'éditeur, tapez une phrase O'Caml comme par exemple `let a = 25 ;;`. En utilisant la combinaison de touches `Ctrl-x Ctrl-e`, vous évaluez cette phrase. Lors de la première évaluation une ligne apparaît au bas de la fenêtre pour vous demander quel interpréteur Caml vous voulez lancer. Celui par défaut convient très bien, appuyez sur la touche "Entrée".
- ✖ La fenêtre de l'éditeur doit être maintenant coupée en deux. En haut, vous avez une fenêtre où vous pouvez taper des programmes O'Caml. En bas, vous avez un interpréteur O'Caml.
- ✖ De cette façon vous allez écrire vos programmes O'Caml directement dans un fichier (`monpgm.ml`) et vous conserverez ainsi la trace de votre session de l'interpréteur O'Caml. N'oubliez pas de sauvegarder régulièrement votre travail!!! (Avec la combinaison de touches `Ctrl-x Ctrl-s`, par exemple)
- ✖ Pour découvrir plus de fonctionnalités du "mode Caml", explorez le menu "Tuareg".

2 Les arguments et les valeurs de retour

En OCaml, une façon d'indiquer qu'une fonction à plusieurs arguments est de les écrire sous la forme d'un n-uplet. De même, une fonction peut retourner plusieurs arguements sous la forme d'un n-uplet. Cela s'écrit d'une manière très proche de celle que l'on utilise en mathématiques. Par exemple, la fonction `rotation` prend un argument les coordonnées d'un point du plan et un angle et retourne les coordonnées d'un nouveau point qui correspond à la rotation centrée en $(0, 0)$ et définie par l'angle donné.

```
let rotation (x, y, theta : float * float * float) : float * float =
  (x *. cos(theta) +. y *. sin(theta), -. x *. sin(theta) +. y *. cos(theta))
```

Si vous essayez cette fonction, la constante `Float.pi` vous sera utile.

▸ **Exercice 1. (Fonctions à plusieurs arguments)**

1. *Écrivez une fonction* `norme` *qui calcule la norme d'un vecteur de l'espace. Un vecteur est représenté par 3 coordonnées* (x, y, z) *et sa norme se calcule avec la formule* $\sqrt{x^2 + y^2 + z^2}$.

▸ **Exercice 2. (Retourner plusieurs valeurs)** *Pour deux points A et B dans le plan de coordonnées respectives* (a_x, a_y) *et* (b_x, b_y)

1. *Les coordonnées du milieu des points A et B sont données par :*

$$(\frac{1}{2}(a_x + b_x), \frac{1}{2}(a_y + b_y))$$

   Écrivez une fonction `milieu` *qui prend en argument deux points du plan et qui renvoie les coordonnées du milieu de ces deux points.*

2. *Une équation cartésienne de la médiatrice de A et B est donnée par :*

$$2(b_x - a_x)x + 2(b_y - a_y)y + (a_x + b_x)(a_x - b_x) + (a_y + b_y)(a_y - b_y) = 0$$

   Écrivez une fonction `mediatrice` *qui prend en argument deux points du plan et qui renvoie les coefficients de l'équation de leur médiatrice comme défini précédemment.*

3 La conditionnelle

Comme dans tous les langages, le langage OCaml fournit une expression conditionnelle

```
if b then e1 else e2
```

où `b` est une expression qui s'évalue en un booléen et `e1` et `e2` sont des expressions.

Par exemple, la fonction `ordre_lexico` qui classe deux couples d'entiers selon l'ordre lexicographique donne un exemple d'utilisation de la conditionnelle.

```
let ordre_lexico ((a, b), (c, d) : (int * int) * (int * int))
  : (int * int ) * (int * int) =
  if a < c then ((a, b), (c, d))
  else if a > c then ((c, d), (a, b))
       else if b < d then ((a, b), (c, d))
            else ((c, d), (a, b))
```

L'évaluation de l'expression conditionnelle `if b then e1 else e2` se déroule de la façon suivante ; Si `b` vaut `true` alors on remplace l'expression conditionnelle par l'expression `e1` et on continue l'évaluation. Sinon, on remplace l'expression conditionnelle par l'expression `e2` et on continue l'évaluation.

Ce mécanisme d'évaluation a pour conséquence que les types des expressions `e1` et `e2` doivent être les mêmes et que ce type est le type de l'expression conditionnelle. Vous remarquerez aussi qu'une expression conditionnelle doit toujours avoir une partie "`else`" car autrement on ne pourrait pas la typer. On peut dire que le type de l'expression conditionnelle `if b then e1 else e2` est `bool * 'a * 'a -> 'a`.

Toutefois, il existe un raccourci `if b then e1` qui équivaut à `if b then e1 else ()`. Dans ce cas, on a obligatoirement `e1 : unit`. Cela est utile quand on programme dans un style impératif avec des boucles.

▸ **Exercice 3. (Sur la conditionnelle)**

1. *Écrire une fonction `signe` qui prend en argument un entier et qui renvoie une chaîne de caractères `"négatif"`, `"positif"` ou `"nul"` selon que l'entier est négatif, positif ou nul.*
2. *Écrire une fonction `max2` qui prend en argument deux entiers et qui renvoie le plus grand des deux. Écrire ensuite la fonction `min2`.*
3. *Écrire une fonction `max3` qui prend en argument trois entiers et qui renvoie le plus grand des trois. Écrire ensuite la fonction `min3`.*

4 Les définitions temporaires

Les définitions temporaires (ou locales) permettent d'éviter d'accumuler des définitions au niveau global. On peut ainsi définir un contexte d'évaluation pour une expression et simplifier l'écriture des fonctions. Une définition locale s'écrit

```
let a = e1 in e2
```

où `e1` ne dépend pas de `a` et `e2` dépend de `a`. Par exemple,

```
let nb_sol (a, b, c : float * float * float) : int =
    let d = b*.b -. 4. *. a *. c in
      if d < 0. then 0
      else if d > 0. then 2
      else 1
```

la définition temporaire de `d` évite de le recalculer trois fois. Une fonction peut aussi être définie de façon temporaire (une fonction est une expression). Dans une expression, des définitions temporaires successives peuvent être ajoutées avec le mot-clef `and`.

▸ **Exercice 4. (Sur les définitions temporaires)** *Pour deux points A et B dans le plan de coordonnées respectives (a_x, a_y) et (b_x, b_y), la longueur du segment $[AB]$ peut être calculée en utilisant le point C de coordonnées (b_x, a_y) et le théorème de Pythagore ; pour un triangle rectangle dont les longueurs des cotés sont a, b et h et où h est l'hypoténuse (le coté opposé à l'angle droit), on a $a^2 + b^2 = h^2$.*

1. *Écrivez une fonction `longueur` qui prend en argument deux points du plan et qui renvoie la longueur du segment qu'ils définissent. Vous définirez temporairement une fonction `pythagore` qui prend en argument les longueurs a et b des cotés adjacents à l'angle droit d'un triangle rectangle et qui renvoie la longueur de l'hypoténuse.*

5 Les fonctions récursives

On indique qu'une fonction est récursive en ajoutant le mot-clef `rec` juste avant son nom, voir l'exemple ci-dessous.

▸ **Exercice 5.**

1. *Que fait la fonction suivante et quel est son type ?*

```
let rec qui_suis_je (n : int) : int =
 if n = 0 then 0
 else n + (qui_suis_je (n-1))
```

2. *Écrire une fonction `somme_carre` qui calcule la somme des n premiers carrés.*

▸ **Exercice 6. (Suite de Héron)**

1. *Écrivez une fonction `heron` qui calcule la suite de Héron définie comme suit pour tout nombre réel x positif :*

$$\begin{cases} U_0 = 1 \\ U_{n+1} = \frac{1}{2} * (U_n + \frac{x}{U_n}) \end{cases}$$

2. *Cette suite converge. Écrivez une fonction `limite` qui renvoie la dernière valeur calculée lorsque la différence de deux termes consécutifs est inférieure à 10^{-8}. À quoi correspond la valeur calculée ?*
3. *Modifiez votre fonction pour choisir la précision à laquelle vous calculez la valeur limite.*

6 Les Listes

En OCaml, les listes sont les listes récursives. Elles sont construites à partir de la liste vide, notée `[]` et de l'ajout en tête de liste `_::_` dont le premier argument est l'élément que

l'on ajoute et le second élément est la liste à laquelle on ajoute l'élément. Par exemple, si l est une liste d'entiers, on lui ajoute l'entier 2 en écrivant `2::l`.

Il existe une facilité d'écriture pour les listes, la liste `2::4::5::[]` peut s'écrire `[2; 3; 5]`.

Les deux observateurs sont `hd` et `tl` où `hd` retourne le tête d'une liste et `tl` retourne une liste qui contient tous les éléments de la liste donnée en argument sauf son premier élement. Ainsi, `hd([2; 3; 5])` retourne 2 et `tl([2; 3; 5])` retourne la liste `[3; 5]`.

Les deux observateurs `hd` et `tl` sont dans le module `List`. On accède a ces deux fonctions en préfixant leur noms par `List.`, on écrit `List.hd` et `List.tl`. Une autre façon de faire est d'ouvrir le module `List` en écrivant `open List ;;`. Il n'est alors plus nécessaire de préfixer les noms des observateurs.

Les listes sont des « conteneurs génériques », c'est-à-dire que les éléments qui sont dans une liste sont de n'importe quel type, on peut avoir des listes d'entiers, de caractères, de flottants, de chaînes de caractères, de listes, etc. Ainsi, le type des éléments d'une liste se note avec une variable de type `'a`. Voici les types des opérations sur les listes.

```
[]    : 'a list
_::_  : 'a * 'a list -> 'a list
hd    : 'a list -> 'a
tl    : 'a list -> 'a list
```

Généralement les fonctions qui manipulent des listes sont des fonctions récursives. Par exemple, la fonction qui calcule la longueur d'une liste s'écrit :

```
let rec longueur (l : 'a list) : int =
  if l = [] then 0
  else 1 + longueur(List.tl(l))
```

▸ **Exercice 7.**

1. *Écrire une fonction `nb_pair` qui compte le nombre d'entiers pairs d'une liste d'entiers.*
2. *Écrire une fonction `ieme` qui renvoie le ième élément d'une liste. Gérez correctement les cas où la liste est vide et où l'indice de l'élément demandé est plus grand que la longueur de la liste.*
3. *Écrire une fonction `ajt_fin` qui ajoute un élément à la fin d'une liste. Quelle est la complexité de cette fonction ?*
4. *Écrire une fonction `renverse` qui renvoie une liste dont les éléments apparaissent en ordre inverse d'une liste donnée en argument.*
5. *Écrire deux fonctions `rg_pair` et `rg_impair` mutuellement récursives qui renvoient l'une les éléments de rang pair d'une et l'autre les éléments de rang impair d'une liste donnée en argument.*
6. *Écrire une fonction `concat` qui renvoie la concaténation de deux listes données en argument.*

▸ **Exercice 8.** *On souhaite écrire une fonction* `change` *qui pour un montant entier dargent, par exemple* 1280 *euros, et une liste de billets et de pièces de monnaie (sans les centimes), par exemple* `[500; 100; 50; 10; 5]`*, renvoie la plus petite liste de billets et de pièces de monnaie dont la somme est égale au montant. Pour lexemple, on obtient la liste* `[500; 500; 100; 100; 50; 10; 10; 10]`*. (Indication : Pour bien comprendre ce qu'il faut faire déroulez lexemple précédent « à la main »).*

▸ **Exercice 9.** *Lors d'un référendum, les résultats des votes de tous les bureaux de vote d'un même canton sont rassemblés dans une liste de couples de la forme* `[("Oui", 5); ("Non", 21); ("Blanc", 6); ("Oui", 32); ("Non", 4); ... ]`

1. *Écrivez une fonction* `recap` *qui prend en argument une liste de résultats et qui renvoie un triplet dont chaque valeur correspond respectivement au nombre de "Oui", de "Non" et de "Blanc".*
2. *Écrivez une fonction* `resultat` *qui donne le résultat du vote à partir d'une liste de résultats.*
3. *Les résultats pour un département (i.e. de tous ses cantons) sont donnés sous la forme d'une liste contenant les résultats de chaque canton (i.e. une liste de listes de couples). Écrivez une fonction* `aplatit` *qui aplatit la liste de résultats d'un département en une liste de couples.*
4. *Écrivez une fonction* `resdpt` *qui donne le résultat du vote au niveau d'un département à partir de la liste de ses résultats.*