Tutorial Lisp

Valvassori Moïse

27 novembre 2001

Table des matières

Ce texte est sensé donner les bases pour se débrouiller lors d'un TP de LISP. Dans un premier temps, je considère que le lecteur n'a jamais touché XBVL de sa vie et n'est pas très famillié avec l'environement X-Window. J'essaye de présenter pas à pas l'interpréteur. Puis on se lance dans la programmation en Lisp. En commençant par le plus simple...

Je rappelle que XBVL et disponible aux adresses suivantes :
http://valvassori.free.fr/download/ et
http://www.ai.univ-paris8.fr/Xbvl/xbvl-home-fr.html.

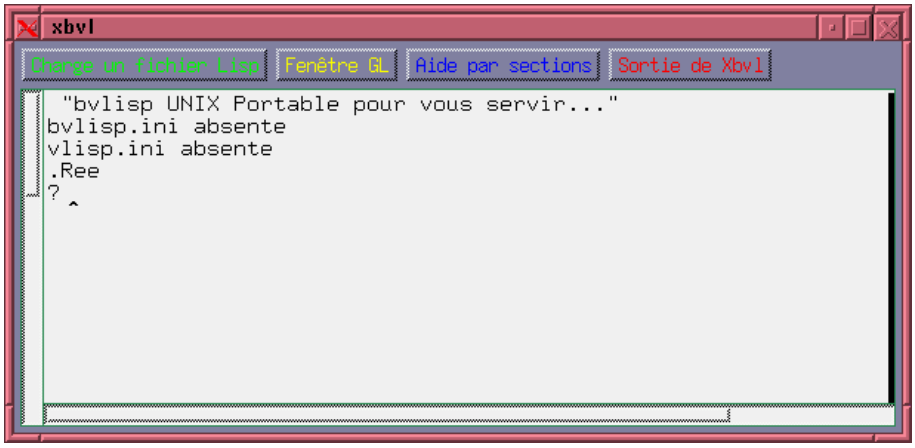
1 Lisp et Xbvl

1.1 Au commencement

Vous venez de vous connecter. Vous arrivez sur une fenêtre de terminal. Tapez dans le terminal :

```
$ xbvl &
```

Tout d'abord, je vous rappelle qu'il ne faut pas taper le «$». Ce signe n'est là que pour symboliser le terminal. Donc une fois que vous avez taper cette ligne, vous devez voir apparaître sous vos yeux la fenêtre Xvbl.



Qu'est ce qu'on voit dans cette fenêtre :

– En haut, une barre de menu. On trouve dans cette barre de menu plusieurs boutons :

Charge un fichier lisp

Ce bouton permet de charger un programme que l'on a déjà tapé.

Fenêtre GL

Ouvre une fenêtre «*Open GL*». Ca permet de voir des objets en trois dimensions.

Aide par section

Cette option est merveilleuse. Elle vous donne de l'aide sur XBVL. Par exemple, si vous voulez avoir de l'aide sur l'aide, vous cliquez sur «*Rechercher*». Ensuite, on vous demande de taper le mot à rechercher. Vous tapez «*aide*» puis vous cliquez à nouveau sur «*Rechercher*». Vous avez l'aide sur l'aide.

Tous les étudiants devraient connaître cette option.

Sortie de Xbvl

On clique dessus quand on veut sortir de Xbvl.

– Dans la partie inférieure de la fenêtre se trouve la zone de travail. C'est dans cette partie que vous allez taper vos expressions Lisp.

1.2 À l'aide

ToDO (J'ai pas encore tapé ce paragraphe... C'est bête mais c'est le plus important)

2 Mes premiers programmes

2.1 À nous de jouer

Maintenant que nous connaissons un petit peu l'interface de XBVL, il va être temps de nous en servir. Notre premier contact va consister à donner des expressions à Lisp et de regarder ce qu'il nous répond.

Tapons le nombre `5647` dans la fenêtre xbvl :

```
 "bvlisp UNIX Portable pour vous servir..."
bvlisp.ini absente
vlisp.ini absente
.Ree
? 5647
= 5647
?
 ^
```

Lisp nous répond par la ligne «`= 5647`» et nous redonne la main. Cela se voit au «`?`» sur la ligne suivante. Ce «`?`» s'appelle l'*invite* ou *prompt* en anglais. Ce nom vient du fait que Lisp nous invite à taper une nouvelle commande.

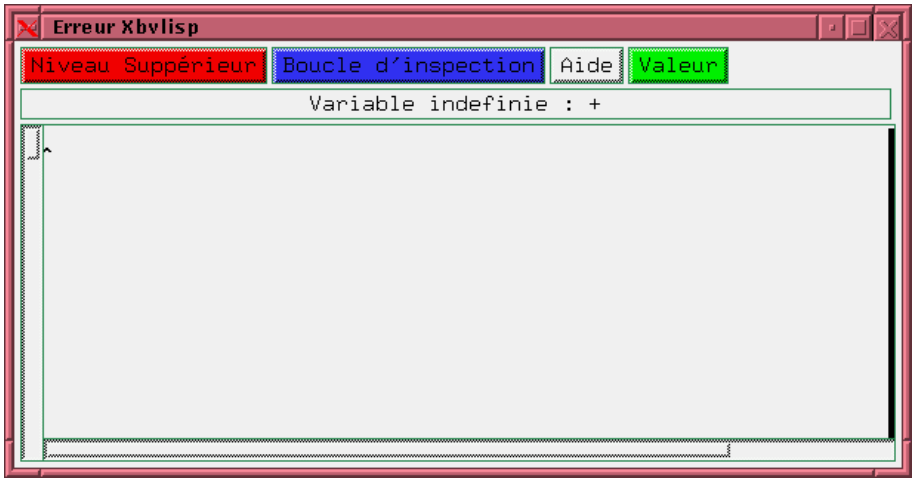
Maintenant que nous répond Lisp si nous tapons le nombre `2.8` ?

```
 "bvlisp UNIX Portable pour vous servir..."
bvlisp.ini absente
vlisp.ini absente
.Ree
? 5647
= 5647
? 2.8
= 2.8
?
 ^
```

Il nous répond bêtement «`= 2.8`». On pourrait taper tout les nombres que l'on connaît, Lisp répondrait toujours «=» suivit du nombre que l'on viens de taper. En fait, Lisp ne se contente pas, comme on pourrait le croire, de répéter tout ce qu'on lui dit. Il fait des calculs sur tout ce qu'on tape et nous donne le résultat de ce calcul. Quand on tape un nombre, Lisp calcule la valeur de ce nombre et affiche ce qu'il a trouvé. La valeur de `2.8` est bien *2,8* donc Lisp nous affiche «`= 2.8`». Le signe «=» est là pour nous dire que c'est le résultat donné par Lisp.

J'ai dit que Lisp calculait tous ce qu'on lui tapait. Il devrait donc être possible de lui faire additionner deux nombres. Essayons de lui taper `1 + 2` et observons :

```
? 1 + 2
= 1
 error-ubv + nil nil
```



Visiblement, Lisp n'a pas apprécié notre demande. Il vient de faire une erreur. Voyons de plus prés ce qu'il nous dit. Dans la première ligne, Lisp nous dit qu'il a compris «1» et il nous répond «`= 1`». Mais à la ligne suivante, il nous signale l'erreur : « `error-ubv + nil nil`». Cette ligne est trop mystérieuse pour l'instant. Laissons la de côté et intéressons nous à la nouvelle fenêtre que XBVL vient d'ouvrir. C'est une fenêtre d'erreur. La seconde ligne nous donne des renseignements sur l'erreur : «*Variable indéfinie : +*». Donc si on décrypte ce message, on peut comprendre que Xbvl ne comprend pas le signe «+». Maintenant que l'on sait pourquoi ça ne marche pas, nous allons fermer la fenêtre d'erreur. Pour ce faire, il suffit de cliquer sur le bouton rouge «Niveau Supérieur». Xbvl affiche d'autres messages d'erreur, mais oublions les pour cet exemple.

2.2 Utilisons des fonctions

En Lisp, lorsque l'on veut faire des opérations, on les tape de la manière suivante :

on ouvre un parenthèse,
on tape l'opération que l'on veut effectuer
puis on fait suivre ce sur quoi on applique l'opération,
et surtout on oublie pas de fermer la parenthèse
Entre l'opération et tous les paramètres, on met un espace .

C'est la règle d'or du Lisp. Cette règle s'applique toujours. Celui qui a compris cela, a presque tout compris au Lisp. En Lisp, l'opération s'appelle *opérateur* et ce sur quoi on applique l'opérateur s'appelle les *arguments* ou bien encore les *paramètres*. L'ensemble opérateur, arguments et parenthèses forme une expression.

Voyons ce que donne la règle d'or avec notre opération «*1+2*». Ici, l'opération est «+» (on veut faire l'opération «addition»). On applique l'addition sur les deux nombres 1 et 2. En faisant pas à pas la règle d'or, on en vient à taper dans Xbvl :

```
? (+ 1 2)
= 3
```

Comme vous le voyez, c'est pas compliqué. Il s'agit juste d'une habitude à prendre. Pour bien se familiariser avec ce mécanisme, nous allons faire plusieurs exemples avec d'autres opérations mathématiques de base.

```
? (+ 6 5.5)
= 11.5
? (- 7 2)
= 5
? (* 3 -100)
= -300
? (/ 23 2)
= 11.5
```

Comme vous l'avez remarqué, le signe «`*`» désigne la multiplication et le signe «`/`», la division.

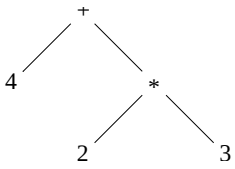
Maintenant, nous allons explorer un peu plus loin le concept de la règle d'or, en calculant *4+(2*3)*. Avant de se lancer sur son clavier, il convient de se poser quelques questions. Quelle est l'opérateur ? Quels sont les paramètres ? La réponse n'est pas aussi simple que pour 1+2 où l'opérateur est + et les arguments sont 1 et 2. Mais ici, nous voyons qu'il y a deux opérateurs (* et +). Notre expression sera elle de la forme «(+ ...» ou de la forme «(* ...» ? C'est tout un problème. Mais si nous regardons de plus prés, nous nous apercevons que nous calculons 4 plus quelque chose. Notre expression devrait donc ressembler à «`(+ 4` *quelque chose*`)`». Or le quelque chose qui nous manque est le résultat de l'opération 2*3 (qui s'écrit «`(* 2 3)`»). Il nous reste plus qu'à remplacer le quelque chose qui manque par `(* 2 3)`. L'expression devient : `(+ 4 (* 2 3))`. Testons dans Xbvl :

```
? (+ 4 (* 2 3))
= 10
```

Nous venons de voir que les arguments d'une expression peuvent être eux même des expressions.

Quand Lisp calcule une expression, il calcule toujours en premier tous les arguments de la fonctions puis il vérifie que l'opérateur existe. Si il n'a pas de problème, il applique les arguments à l'opérateur, puis rend le résultat de l'opération. Si un des arguments est aussi une expression, il applique la même règle. Il faut toujours se souvenir qu'**une expression Lisp rend toujours un résultat**. C'est ce que l'on voit apparaître après de le signe «=».

Voyons ce que cela donne avec `(+ 4 (* 2 3))`. Lisp trouve une expression avec deux arguments : `4` et `(* 2 3)`. Il calcule le résultat de ces deux arguments. Pour le premier, il renvoie simplement 4. Par contre le second est un aussi une expression. Il va donc devoir calculer son résultat. Les arguments sont 2 et 3. Leur calcul ne pose pas de problème. Maintenant, Lisp regarde si l'opération «*» existe. Comme elle existe, il effectue le calcul et renvoie le résultat de l'opération. A cette étape, Lisp se retrouve exactement dans la même situation que si on avait tapé `(+ 4 6)`. Sauf qu'il connaît déjà les arguments (on vient juste de les calculer). Il lui reste plus qu'à vérifier que l'opération «+» existe et calculer le résultat. Une fois celui ci à sa disposition et comme c'était le dernier calcul, il affiche le résultat à l'écran.



3 Donnons un nom aux choses

Maintenant que nous sommes familier avec l'évaluation des expressions. Nous allons nous pencher sur un autre problème. Imaginons que cherchions à calculer le périmètre d'un cercle de rayon 10. Nous appliquons la formule $2\pi R$. Ce qui donne :

```
? (* 10 (* 2 3.14))
= 62.8
```

Très bien. Lisp nous donne le bon résultat. Maintenant, nous voulons calculer le périmètre du cercle de rayon 33. Nous allons taper `(* 33 (* 2 3.14))`. Cala nous donnera le bon résultat. Mais si nous avons beaucoup de périmètre à calculer cela risque vite de devenir pénible de taper *3.14*. Cela serait pratique si on pouvait écrire «*pi*» à la place. En Lisp, on réalise cela avec la fonction *setq*. Cette fonction s'utilise de la manière suivante : `(setq` *nom expression*`)`. Lisp évalue l'expression et associe le résultat de l'expression au nom. Après avoir évalué une fonction `setq`, le *nom* sera toujours remplacé par le résultat de l'*expression*. Ce qui donne dans notre cas :

```
? (setq pi 3.14)
= 3.14
? pi
= 3.14
? (* 10 (* 2 pi))
= 62.8
```

Sur la troisième et la cinquième ligne, quand on cherche à calculer `pi`, on voit très bien qu'il a remplacé `pi` par la valeur de 3.14. Maintenant, si nous voulons une valeur plus précise de π, il suffit de le redéfinir :

```
? (setq pi 3.14159)
3.14159
```

Mais nous pouvons pousser le concept encore plus loin. Cela serait encore mieux si au lieu de taper «`(* 2 pi)`», nous tapions «`2pi`».

```
? (setq 2pi (* 2 pi))
= 6.28318
```

Lisp a évalué l'expréssion «`(* 2 pi)`» et a associé le résultat au nom «`2pi`».