Évaluation

Le mécanisme de l'évaluation est au coeur du fonctionnement de tout système de calcul formel.

I. Affectation

Rappelons la syntaxe générale :

variable **:=** *expression* **;**

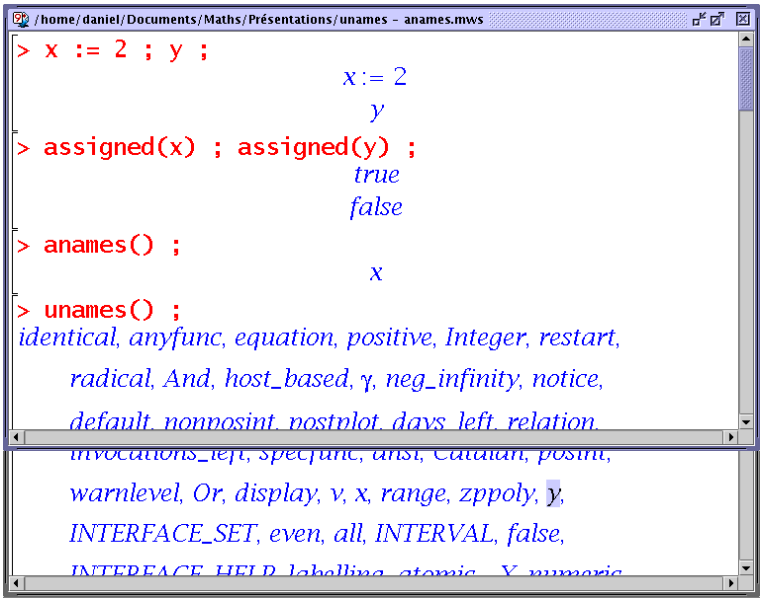
Dans cette situation :

- *expression* est évaluée -- l'objet de cette section,
- puis son résultat est stocké dans *variable*.

Ainsi, une variable *non encore affectée* se comporte comme si son contenu était simplement son *nom*[1].

On dispose des fonctions suivantes pour savoir si une variable est affectée ou non :

- **anames()** (fonction sans argument) donne la séquence des variables affectées ;
- **unames()** donne la séquence des variables non affectées ;
- **assigned(***variable***)** renvoie *true* ou *false* selon l'état d'affectation de la *variable* ;
- **unassign(' ** *variable* **')** efface la *variable* (voir plus loin la nécessité des apostrophes) ;
- **restart** efface *toutes* les variables (de l'utilisateur).



Comme on le voit, **unames()** retourne un grand nombre de constantes et variables systèmes, en plus de celles (non affectées) de l'utilisateur.

II. Évaluation

L'évaluation, c'est le "calcul" d'une quantité algébrique. Essentiellement, il s'agit de remplacer par leur contenu les variables qui en ont un.

1 L'évaluation de cette variable ne produit rien d'autre que l'affichage de son nom.

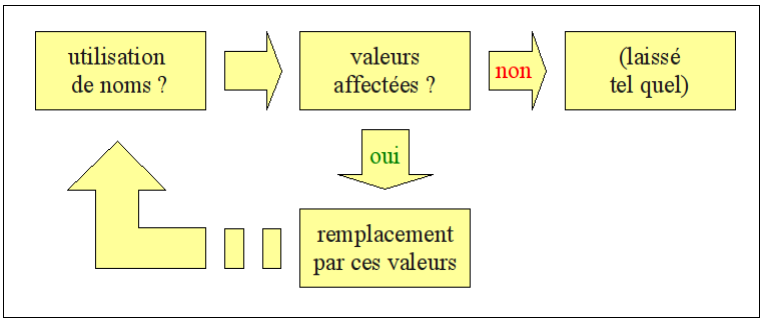
L'évaluation peut être

- *complète* ou *partielle*,
- *immédiate* ou *différée*.

Le cas général est celui d'une évaluation **complète et immédiate**.

(a) Cas général

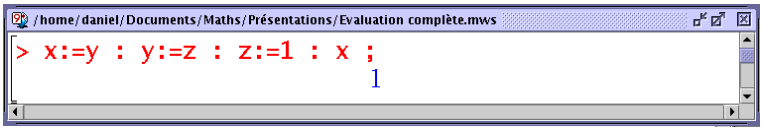
Voici ce qu'il advient d'une expression dans l'évaluation complète et immédiate :



Tant qu'il subsiste des noms de variables ayant un contenu ("*complète*"), le remplacement de ces variables par leurs contenus est effectué dans délai ("*immédiate*").

On comprend alors que la commande **x := x+1** avec **x** non affecté pose problème. On entrerait dans une succession infinie de remplacements (**x+1**, **x+2**, **x+3**...) si *Maple* n'était pas pourvu d'un mécanisme de protection.

Examinons l'exemple suivant :



À l'issue des trois affectations, les variables ont les contenus suivants :

variable	contenu
x	*y*
y	*z*
z	1

L'évaluation de **x** déclenche donc trois tours de la boucle décrite précédemment, où **x** se verra remplacé par *y*, puis *z*, puis finalement 1.

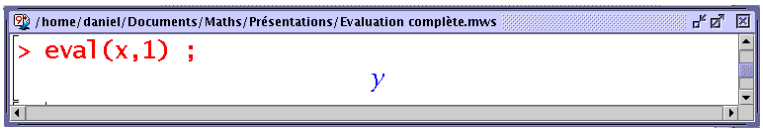
L'instruction **eval(x)** aurait eu le même effet dans ce cas[2].

L'instruction **eval(x , *n*)** permet de forcer l'évaluation à un niveau *n* donné[3].

En poursuivant l'exemple précédent, elle nous permet de constater que le contenu de **x** est bien *y* (et

2 Cette instruction existe pour forcer l'évaluation complète dans les cas exceptionnels.
3 Elle limite à *n* le nombre de tours de la boucle d'évaluation.

non 1).



Des exemples similaires, mais dans d'autres contextes, vont donner des résultats très différents :

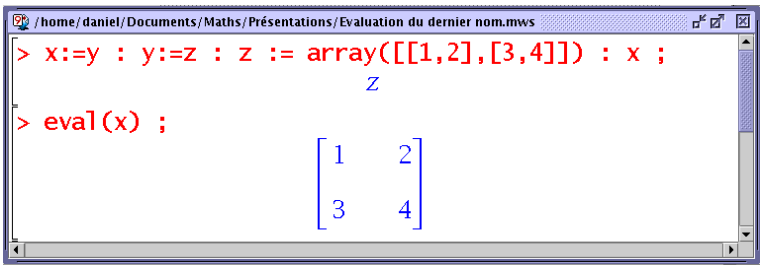
(b) Structures "volumineuses"

La première exception est constituée par les "grosses" structures de données. Il s'agit des types *array*, *table*, *matrix*, *proc*.

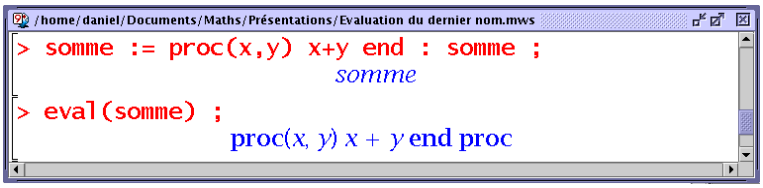
La règle dans ce cas est **l'évaluation du dernier nom**.

Cela signifie que lorsque, dans la boucle d'évaluation, *Maple* rencontre un nom de variable dont le contenu est d'un des types précédents, ce nom *n'est pas* remplacé par son contenu.

Considérons la situation suivante, qui ne diffère de l'exemple initial que par le type du contenu de **z** :



Un autre exemple, utilisant une procédure :



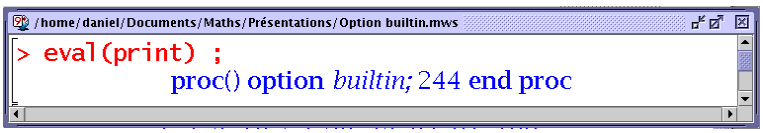
Il faut, dans ces cas, forcer l'évaluation complète avec **eval**.

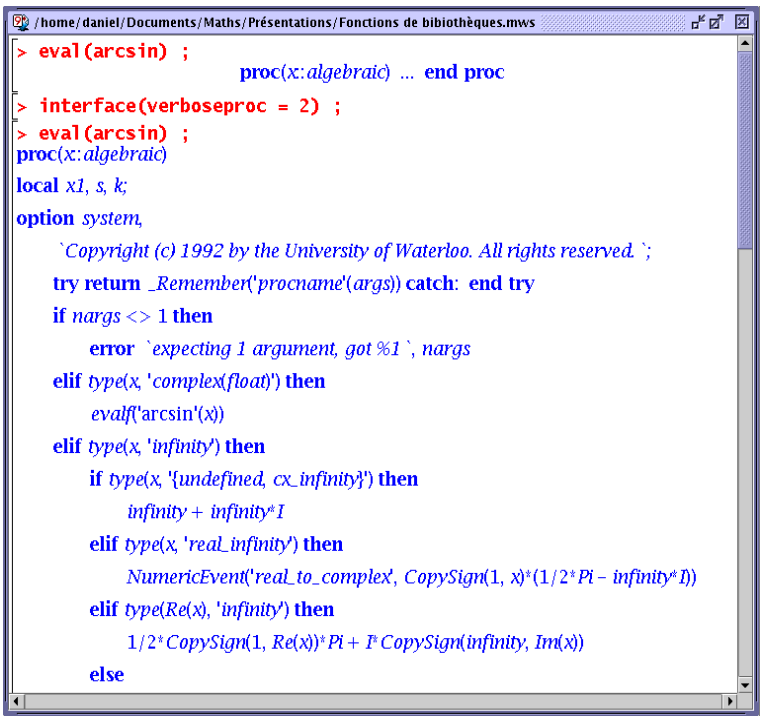
Toutefois, cela ne nous donne pas encore accès aux fonctions de bibliothèque et autres fonctions de *Maple* qui sont programmées en *Maple* même.

On peut malgré tout accéder au code de ces fonctions après avoir saisi l'incantation : **interface(verboseproc = 2)**.

On constate sur le schéma (p. suivante) que celle-ci rend *Maple* beaucoup plus disert.

Toutefois, ce réglage n'est d'aucune utilité dans le cas des fonctions incorporées de *Maple*. Celles-ci ne sont pas programmées en *Maple* ; elles utilisent l'option **builtin** et résistent à tout examen :

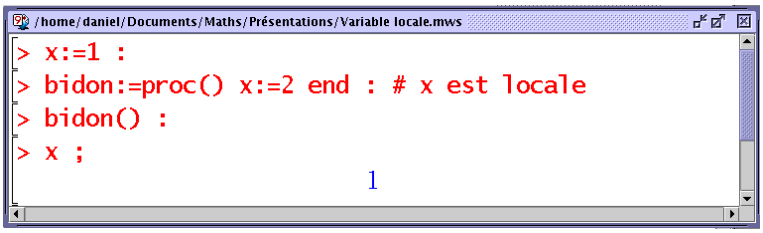




(c) Variable locales

Une autre exception survient à l'intérieur des procédures pour les variables locales. Celles-ci sont discutées dans le poly. "Fonctions et procédures".

Disons seulement que de sont des variables qui n'existent que le temps qu'une procédure soit active. Donnons-en un exemple :



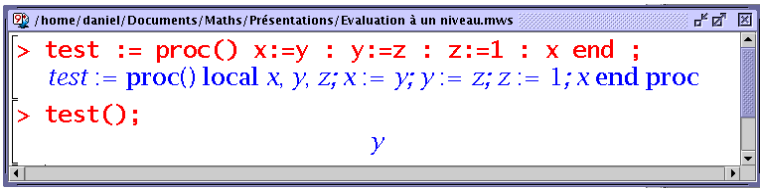
Notons que *Maple* se permet de localiser automatiquement certaines variables dans les procédures :

- celles qui apparaissent au membre de gauche d'une affectation : **x :=** ... ;
- les compteurs des boucles **for**.

La règle pour les variables locales est **l'évaluation à un niveau**.

On effectue donc un seul tour de la boucle d'évaluation.

Considérons l'exemple suivant, identique à celui du début, hormis qu'il est situé dans une procédure :

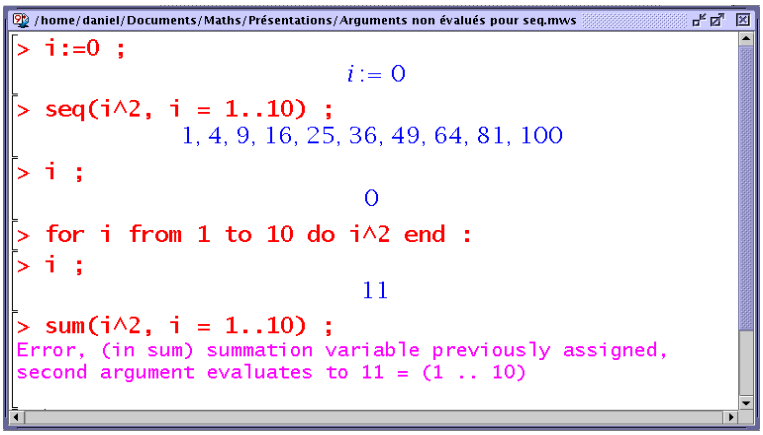


La variable **x**, locale à la procédure **test**, n'a été évaluée qu'au niveau 1.
Il faudra donc, dans les procédures, utiliser **eval** pour forcer l'évaluation complète si nécessaire.

(d) Commandes seq et for

seq n'évalue pas ses arguments, et ne les affecte pas non plus. **for** les affecte sans les évaluer.

Cela peut produire des résultats inattendus :



Par comparaison, **sum** et **prod** affectent initialement la variable (et l'évaluent). Si celle-ci est déjà affectée, cela produit une erreur.

Il est donc parfois utile de différer l'évaluation.

(e) Évaluation différée

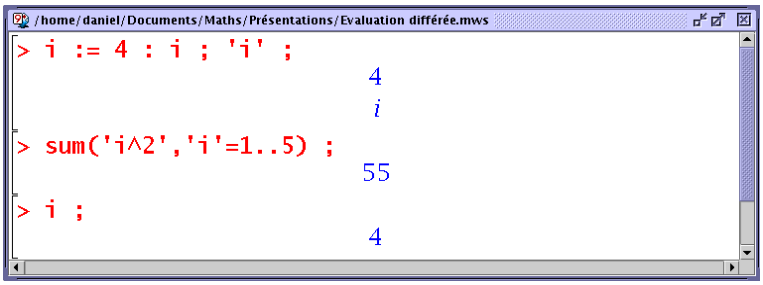
On diffère l'évaluation d'une variable à l'aide de guillemets simples (apostrophes).
On fera attention à ne pas confondre :

- **'** (sous 4) "right quote" : c'est lui qui sert à différer l'évaluation ;
- **`** (Alt Gr + 7) "left quote" ou "backquote" : délimiteur de chaînes de caractères.

La manière générale de différer l'évaluation d'une variable est donc :

' *variable* **'** ;
variable

Voici un exemple :



L'évaluation complète retire un niveau de guillemets simples **' '** . Ceux-ci ne font que *différer* l'évaluation de la variable.

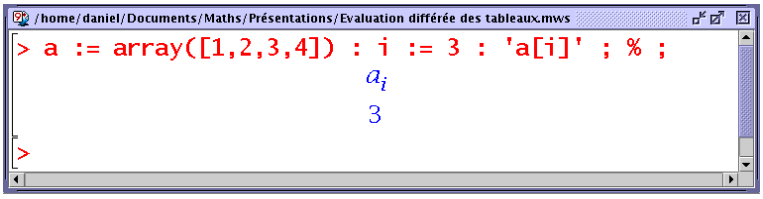
D'autre part, *ils n'empêchent pas* les simplifications automatiques.

Une application possible est l'effacement d'une seule variable **x** :

- **unassign(' x ')** ;
- ou tout simplement **x := ' x '** .

Dans les deux cas, les guillemets simples sont bien sûr indispensables[4] .

Dans le cas des tableaux p. ex. :



on peut remarquer que les guillemets simples "passent à travers" les crochets et diffèrent aussi l'évaluation de l'indice.

(f) Paramètres (arguments) des procédures

Un simple mot pour mentionner que *ce ne sont pas des variables*.

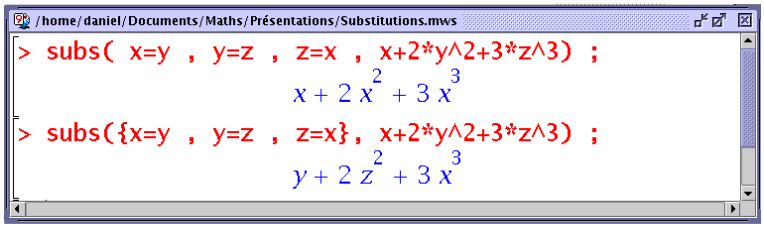
Ce sont des *valeurs* qui sont récupérées au moment de l'appel de la procédure. Il n'y a *pas* de création de variable homonyme[5].

Plus de détails dans le poly. "Fonctions et procédures", notamment sur les modalités particulières d'évaluation de ces paramètres.

III. Substitutions

Les remarques sur l'évaluation éclairent le mécanisme des substitutions, quand plusieurs ont lieu dans une même instruction **subs** :

- elles sont généralement *successives* (il y a *évaluation* après chacune d'elles) ;
- sauf si on les regroupe avec des accolades :



Dans ce second cas, on voit qu'il n'y a *pas d'évaluation intermédiaire* (entre les **{ }**), et que les substitutions sont réellement *simultanées.*

4 Sans eux, **x** serait remplacé par son contenu.
5 On parle de "*transmission par valeur*" des arguments.