Types structurés

Maple offre une grande[1] variété de structures permettant de regrouper des données élémentaires.

I. Séquences

C'est le type structuré fondamental de *Maple*. On désigne par séquence toute succession de termes **séparés par des " , "**.

La définition d'une séquence répond donc au motif suivant :

nom **:=** *terme1* { **,** *terme2* **,** ... **,** *termeN* } **;**

(Lui donner un nom est bien sûr facultatif.)

Par exemple : **s := a, b, c, 1, Pi, exp ;** est une séquence correctement définie en *Maple*. Les données qu'elle contient sont, typiquement, hétérogènes.

Le *type* d'un tel objet *Maple* est *exprseq* (*expr*ession *seq*uence) ([2]).

Accès aux termes

L'accès aux termes de séquences est des plus souples. Il s'effectue à l'aide de crochets : si **s** est une séquence,

- **s[i]** désigne le $i^{ème}$ terme de **s** (la numérotation commence à 1) ;
- **s[-i]** (où i > 0) est $i^{ème}$ terme de **s** depuis la fin (le dernier terme de **s** est donc tout simplement **s[-1]**) ;
- **s[i . . j]** est la "sous-séquence" formée des termes de **s** du $i^{ème}$ au $j^{ème}$ (inclus).

Les conventions de signe précédentes peuvent être associées : ainsi **s[3 . . -2]** désigne la sous-séquence des termes du $3^{ème}$ à l'avant-dernier.

Concaténation

Si l'on sépare deux séquences par une virgule, celle-ci sera indiscernable de celles qui séparent les termes contenus dans chaque séquence.

On comprend alors que la concaténation (mise bout à bout) de séquences est réalisée très simplement :

séquence1 **,** *séquence2* **,** ... **,** *séquenceN* **;**

regroupe les *N* séquences en une seule. Il n'est plus possible de retrouver les séquences originales. On peut y voir l'explication de la "caractéristique" suivante :

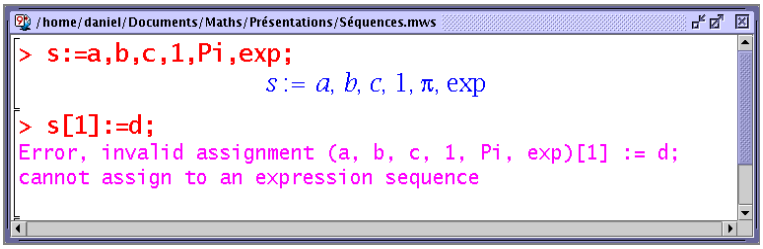
Affectation d'un élément

Cette opération est **impossible avec une séquence !**

---

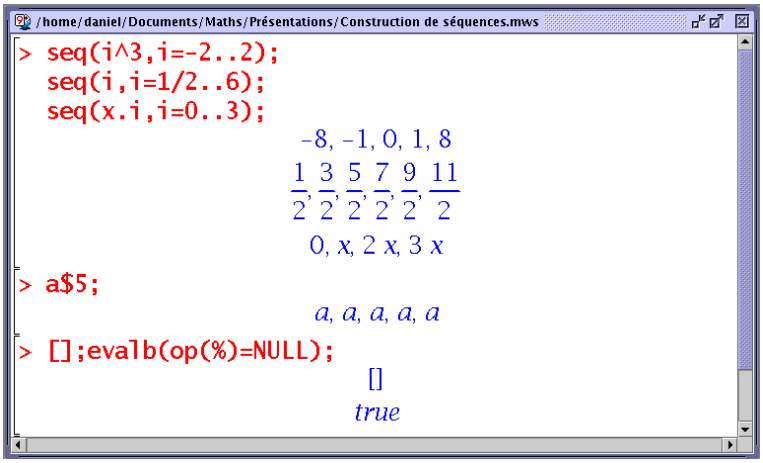
1 Qui a dit "trop grande" ?

2 Une des abréviations dont Maple a le secret -- et un bon exercice de prononciation.



Construction

L'instruction **seq** permet de former rapidement une séquence dont les termes répondent à une syntaxe régulière :



Notons le raccourci **x$n** pour **seq(x , i = 1 . . n)**, formant une séquence de n termes identiques. C'est utile lorsqu'on a à calculer les dérivées successives d'une expression, *cf.* le poly. "Expressions".

On remarque aussi le symbole **NULL** pour désigner la séquence vide. Elle ne produit aucun affichage... et c'est pourtant le résultat de certains calculs *Maple* (p. ex. la résolution d'une équation où aucune solution n'a été trouvée). Cela pourrait faire croire à tort que l'instruction n'a pas été exécutée.

En effet, *Maple* utilise lui-même des séquences pour communiquer ses résultats dans certains cas :

- résolution d'équations avec **solve** : **solve(x^3 - 6*x^2 + 11*x - 6 = 0 , x)** ;
- extraction des opérandes d'une expression avec **op** : **op(x + 2*y + 3*z^2)** (*cf.* "Expressions").

Les séquences interviennent également dans la définition et la manipulation des fonctions de plusieurs variables.

Par exemple : **v := (x , y) : f(v) ;** est correct (les parenthèses autour de la séquence **x,y** ne servant qu'au groupage).

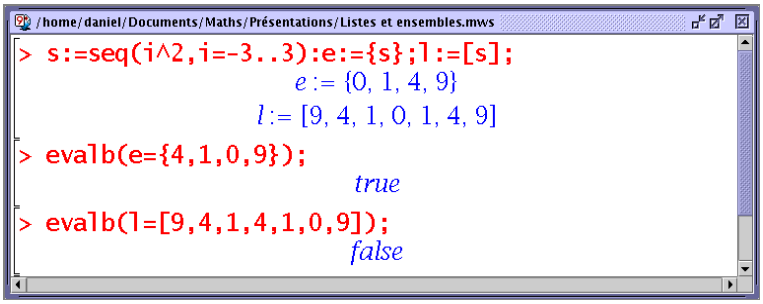
II. Listes et ensembles

On obtient deux nouveaux types de données importants en plaçant une séquence entre des délimiteurs convenables :

délimiteurs	type obtenu	équivalent mathématique	ordre	répétitions
[ ]	*list*	*n*-uplets	oui	possibles

{ }	*set*	ensembles	non	éliminées

Les exemples suivants illustrent ces différences :



Maple "trie" ([3]) automatiquement les éléments des ensembles, afin de pouvoir comparer deux d'entre eux.

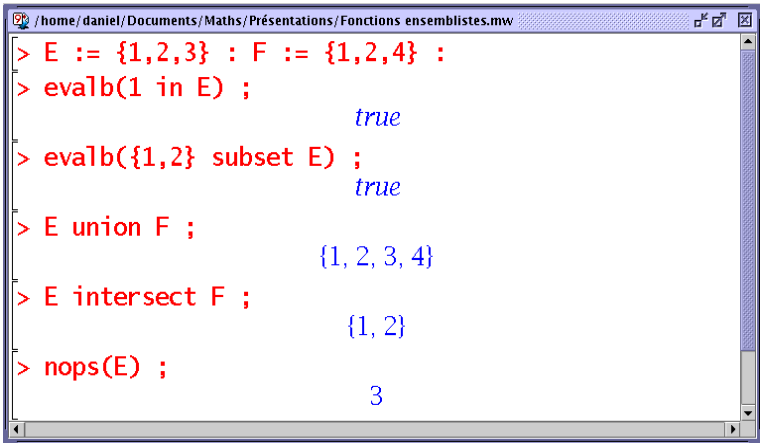
Fonctions communes

Les fonctions suivantes peuvent être appliquées aux deux types de données. Cependant leur application aux ensembles n'est guère pertinente.

- **nops** : nombre d'éléments ;
- **op** : *séquence* de tous les éléments (on a ainsi un moyen de convertir une liste ou un ensemble en séquence) ;
- **L[i]** ou **op(i , L)** : i[ème] élément de **L** (**l'affectation directe est possible** dans le cas des listes) ;
- **subsop(i =** *expression* **, L)** : remplace le i[ème] élément de **L** par *expression* ([4]) ;
- **sort** trie une liste ([4]).

Fonctions ensemblistes

Le prédicat d'appartenance se note **in**, l'inclusion **subset**, la réunion **union**, l'intersection **intersect** :



Noter la syntaxe *infixe* particulière. Pour une notation préfixée plus habituelle en *Maple*, on doit inclure la commande entre deux *backquotes*.
P. ex. : **` intersect ` (***E1* **,** *E2* **, ... ,** *En***)** (*cf.* "Évaluation").

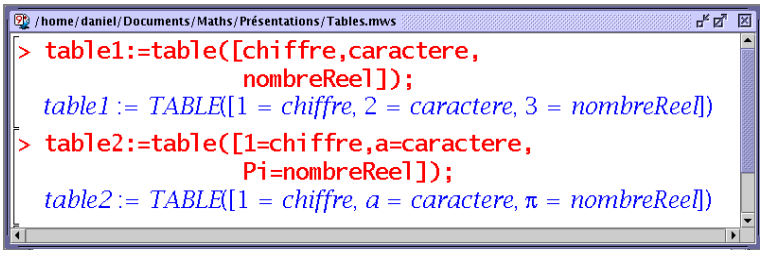
Bien sûr, dans le cas des ensembles, **nops** dénote simplement le *cardinal*.

3 Attention, l'ordre choisi par *Maple* n'est *pas nécessairement l'ordre mathématique*.
4 Attention, le résultat n'est *pas affecté* à **L**. C'est une *nouvelle liste* qui est produite, que l'on peut nommer à nouveau **L** ou autrement.

III. Tables

Le type *table* répond à la nécessité d'une structures ou les indices peuvent être *quelconques*.

Les indices par défaut demeurent des entiers, mais on peut spécifier des indices différents :



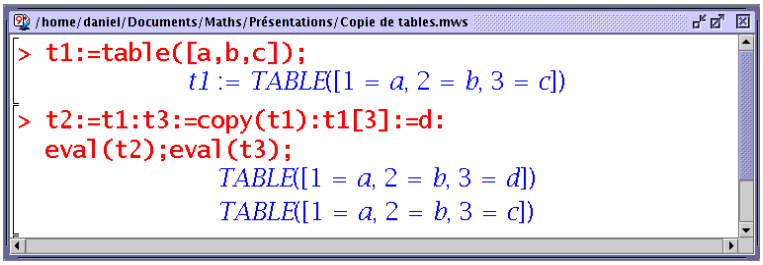
Pour connaître le contenu complet d'une table, il ne suffit pas de taper son nom suivi de "Enter". La raison réside dans les mécanismes d'évaluation de *Maple*.

Il donc faut utiliser une commande **print** ou **eval**. On fera attention à **print** qui ne produit qu'un affichage (une "image") de la table, et qui *ne convient donc pas* si le résultat doit resservir.

L'accès aux termes s'effectue comme pour les séquence et les listes.

L'affectation directe est possible : **table1[2] :=** ... mais nécessite une vigilance particulière. Les indices des tables ne faisant l'objet d'aucun contrôle, si l'on spécifie par mégarde un indice absent de la table, il lui sera simplement ajouté.

La copie de tables obéit à des règles particulières. Une table étant typiquement une structure volumineuse, *Maple* s'efforce de limiter le nombre d'exemplaire en mémoire. Une instruction comme **t2 := t1** ne dupliquera donc pas la table **t1**, mais créera en fait un pointeur (un *alias*) pour désigner les mêmes données en mémoire :



On voit donc que pour créer une copie indépendante, l'instruction **copy** est indispensable.

Maple utilise lui-même des tables pour les valeurs particulières des fonctions ou des procédures, *cf.* le poly. "Fonctions et procédures".

IV. Tableaux

Le type tableau (*array*) est un cas particulier du type *table*.

Les tableaux sont des tables indexées par *un (ou plusieurs) entier(s) décrivant un (ou plusieurs) intervalle(s)* ([5]).

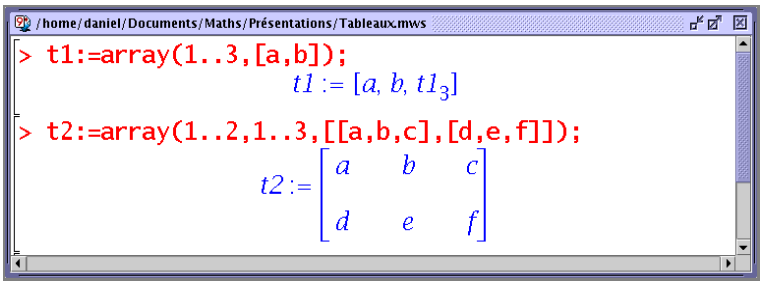
La construction de tableaux s'effectue avec l'instruction **array**

- qui *doit* comporter la spécification des intervalles d'indices ;
- qui *peut* comporter une initialisation partielle ou complète :

nom **:= array(** *interv.1* { **,** *interv.2* **,** ... **,** *interv.N* } { **,** **[** *initi.* **]** } {**,** *options*} **) ;**

5 Si **tous** les indices commencent à la valeur **1**, le tableau est du type *matrix* (d'usage fréquent en algèbre linéaire).

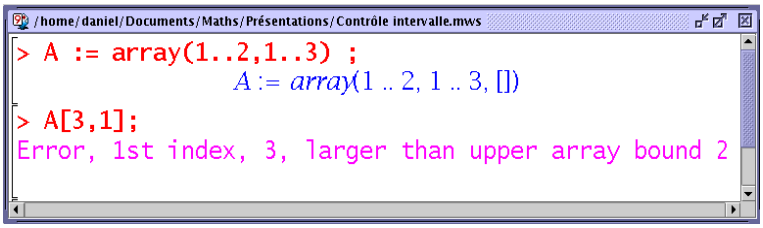
En voici deux exemples :



Les options d'initialisation seront surtout utiles en algèbre linéaire :

- **sparse** remplit le tableau de zéros ;
- (**anti**)**symmetric** / **diagonal** / **identity** correspondent aux matrices du même nom.

L'accès aux termes est en tout point identique aux tables. Toutefois *Maple* contrôlera que les indices appartiennent bien aux intervalles spécifiés lors de la définition du tableau :



On obtient certes un message d'erreur, mais c'est mieux que rien.

Les tableaux étant des tables particulières, ils sont soumis aux mêmes règles concernant la copie.