

# Maple

## TP n°1 : Types composés

### Rappel 1

Pour pouvoir entrer plusieurs lignes d'instructions Maple successives, sans qu'il y ait exécution à la fin de chacune d'elles, il faut appuyer sur `Shift+Entrée` à la fin de chaque ligne.

Enfin, pour valider l'ensemble des lignes ainsi écrites, presser `Entrée` en plaçant le curseur sur n'importe quelle ligne.

### Rappel 2

Une ligne doit se terminer par un `:` ou par un `;` (il y a alors affichage de l'entrée).

### Rappel 3

Les commandes `whattype( )` ou `type( , )` permettent de connaître ou de vérifier le type d'une expression.

```
whattype(Pi);  
type(a<b,boolean);  
type(((a<b) and (c>d)),integer);
```

### Rappel 4

La commande `is( )` vérifie si une expression possède une propriété.

```
is(Pi,RealRange(0,4));  
is(Pi,fraction);  
is(cos,continuous);  
is(1265/34,integer);
```

### Rappel 5

Pour réinitialiser toutes les variables, on utilise la commande `restart;`.

## I. Suites, ensembles et listes

### 1.1 Les suites

Une suite ou séquence (type `exprseq`) est une suite d'expressions écrites dans un ordre donné et séparées par une virgule avec possibilité de répétition. C'est le type composé le plus simple.

```
s:=1,2,cos(x),a,b,Pi,3,2*x^2+1,j*k+1,ha,ha,ha;  
whattype(%);
```

On accède à chaque élément par son rang et on peut extraire des sous-suites.

```
s:=1,2,cos(x),a,b,Pi,3,2*x^2+1,j*k+1,ha,ha,ha;  
s[3];  
s[4..7];
```

L'affectation globale est possible, mais l'affectation individuelle est interdite :

```
a,b,c:=1,2,3;  
a; b; c;
```

```
s:=1,2,cos(x),a,b,Pi,3,2*x^2+1,j*k+1,ha,ha,ha;  
s[3]:=sin(x);
```

Il existe une suite particulière : la séquence vide. Elle est désignée par NULL.

```
v:=NULL;  
whattype(%);
```

Lorsqu'un élément est répété plusieurs fois de suite, on peut utiliser l'opérateur dollar \$ :

```
restart;  
Suite := a$3,b$5,c$7;
```

Pour concaténer deux suites, on utilise l'opérateur 'virgule' :

```
S1:=1,2,3;  
S2:=4,5,6;  
S3:=S1,S2;  
S4:=S3,a,S3;
```

Pour créer une suite du type  $f(1), f(2), f(3), \dots$ , on peut utiliser la commande `seq` dont la syntaxe est la suivante :

```
seq(expr,i=a..b);
```

Cette commande crée une suite en remplaçant `i` dans l'expression `expr` par les différentes valeurs de cet indice. Ici `i` varie de `a` à `b` avec un pas de 1.

```
seq(i^2,i=1..5);
```

On peut aussi faire varier `i` dans une suite (mais aussi un ensemble ou une liste).

```
s:=1,3,5,a,b;  
seq(i^2,i=s);
```

## 1.2 Les ensembles

Un ensemble (type set) est une suite d'éléments qui ne se répètent pas.

Elle est obtenue en plaçant une suite entre les accolades `{` et `}`. L'ensemble vide se note `{}`.

Il est possible que la suite contienne des doublons. Néanmoins, ceux-ci sont supprimés au passage au type ensemble. Dans le même ordre d'idée, l'ordre des termes est réévalué en fonction des éléments et non de leur ordre d'apparition. Ce qui signifie que l'ordre d'entrée n'est pas nécessairement conservé. Ceci est une source d'erreur.

```
e1:={4,1,1,2,2,3};
s:=b$2,a$3;
e2:={s};
whattype(s);
whattype(e2);
```

On peut transformer un ensemble en une suite via la commande `op`.

```
e1:={d,a,a,b,b,c,b,b,b,a,c};
op(e1);
whattype(%);
```

Remarquons donc qu'une double transformation ne nous ramène pas nécessairement au point de départ :

```
s:=b,b,a,a,a;
e2:={s};
s:=op(e2);
```

On accède à chaque élément d'un ensemble par son rang, et on peut extraire aussi des sous-ensembles.

```
e1:={d,a,a,b,b,c,b,b,b,a,c};
e1[2];
e1[2..3];
```

Il est important de voir que de telles opérations dépendent de l'ordre dans lequel l'ensemble a été rangé.

On peut aussi sélectionner des éléments particuliers d'un ensemble via la commande `select`.

```
s:=seq(2^i-1,i=1..10);
e3:={s};
select(isprime,e3);
```

La commande `nops` permet d'obtenir le nombre d'éléments d'un ensemble.

```
e1:={d,a,a,b,b,c,b,b,b,a,c};
e2:={};
nops(e1);
nops(e2);
```

La fonction booléenne `member` permet de savoir si une expression est membre d'un ensemble.

```
e4:={alpha,beta,gamma,delta};
member(delta,e4);
member(epsilon,e4);
```

On ne peut faire ni affectation globale, ni affectation individuelle.

```
{a,b,c}:={7,8,9};

e1:={d,a,a,b,b,c,b,b,b,a,c};
e1[3]:=c;
```

Il existe donc des commandes spécifiques pour modifier un ensemble `subs` et `subsop`. La commande `subs` agit à partir du nom de l'élément à modifier alors que la commande `subsop` agit à partir de l'index de l'élément. Ces remplacements ne sont valides que lors de l'appel de la fonction. L'ensemble retourne ensuite à son état initial.

```
e4:={alpha,beta,gamma,delta}:
subs(alpha=1,e4);
subs(alpha=epsilon,e4);
subsop(l=1,e4);
subsop(l=epsilon,e4);
e4;
```

Enfin, on a les opérations classiques sur les ensembles : l'union (`union`), l'intersection (`intersect`) et la différence (`minus`).

```
e4:={alpha,beta,gamma,delta};
e5:={gamma,delta,epsilon,zeta,eta,theta};
e6:={eta,theta,lambda};
e4 union e5;
e4 intersect e5;
e5 minus e6;
```

### 1.3 Les listes

Les listes (type `list`) comme les suites sont composées d'expressions écrites dans un ordre donné et séparées par une virgule avec possibilité de répétition.

Elles s'obtiennent d'ailleurs simplement en plaçant une suite entre les crochets `[` et `]`.

La liste vide se note `[]`. Les listes ont un comportement et des fonctions similaires aux ensembles.

```
L1:=[4,1,1,2,2,3];
s:=b$2,a$3;
L2:=[s];
whattype(s);
whattype(L1);
```

On peut aussi faire des listes de listes.

```
L1:=[4,1,1,2,2,3]:
L2:=[b$2,a$3,c$4]:
L:=[L1,L2];
```

De même que pour les ensembles, la fonction booléenne `member` permet de savoir si une expression est membre d'une liste.

```
Liste_a:=[4,1,1,2,2,3];
member(2,Liste_a);
member(5,Liste_a);
```

La commande `nops` donne le nombre d'éléments d'une liste.

```
Liste_a:=[4,1,1,2,2,3]:
nops(Liste_a);
```

On accède à chaque élément d'une liste par son rang et on peut extraire des sous-listes. Dans le cas de liste de listes, l'accès se fait à plusieurs niveaux.

```
L1[6];
L2[2..5];
L[1];
L[1,6];
```

Toutefois, contrairement aux suites l'affectation individuelle est autorisée mais l'affectation générale des éléments de la liste est interdite.

```
Liste:=[a,b,b,c,c,c]:  
Liste[1]:=e;  
Liste;
```

```
Liste := [1,2,3]:  
Liste;  
Liste := [1,2,3,4]:  
Liste;
```

```
[a,b,c] := [1,2,3];
```

Pour modifier une liste, on peut aussi utiliser les commandes `subs` ou `subsop` vues sur les ensembles. Les opérations sont les mêmes. Rappelons le caractère transitoire de ces opérations. Donc pour avoir une modification "définitive", il faut réaffecter la liste.

```
Liste:=[a,b,b,c,c,c]:  
subs(a=d,Liste);  
Liste;  
Liste:=subs(a=d,Liste):  
Liste;
```

Il est possible de convertir une liste en un ensemble et inversement avec la commande `convert`. Notez la possible perte d'information en cas de double conversion.

```
Liste_a:=[4,1,1,2,2,3]:  
Ensemble:=convert(Liste_a,set);  
Liste_b:=convert(Ensemble,list);
```

Il est possible de convertir une liste en suite avec la commande `op`. Notez qu'une double conversion ne fait pas perdre d'information.

```
Liste_a:=[4,1,1,2,2,3]:  
Suite_a:=op(Liste_a);  
Liste_b:=[Suite_a];
```

Une fonction intéressante sur les listes est la fonction de tri : `sort`.

```
Liste_a:=[4,1,1,2,2,3]:  
sort(Liste_a);
```

## II. Les Tableaux

Il existe une structure similaire aux tableaux : les tables. Nous ne l'utiliserons pas.

Un tableau (type `array`) est une structure indexée, qui peut avoir plusieurs dimensions et dont les indices sont des entiers appartenant à un intervalle  $a..b$  où  $a$  et  $b$  sont des entiers. Un tableau est créé par la commande `array`. Il n'y a pas nécessairement d'affectation. Par exemple :

```
V:=array(1..10);
```

Cette commande a crée un tableau à une dimension de longueur 10 mais sans entrée spécifique.

Pour les tableaux de dimension 2, la commande `A := array(1..m,1..p)` ; crée un tableau  $A$  à  $m$  lignes et  $p$  colonnes. La syntaxe générale de la commande est la suivante :

```
array(type_tableau, borne, liste_des_valeurs);
```

`type_tableau` est optionnel. Il permet de décrire le tableau : `symmetric`, `antisymmetric`, `sparse`, `diagonal` ou `identity`.

L'une des deux informations `borne` ou `liste_des_valeurs` est obligatoire mais pas nécessairement les deux.

`borne` est la suite des intervalles de chacune des dimensions du tableau. Si `borne` n'est pas précisée, c'est la liste des valeurs qui est utilisée pour déduire ces intervalles.

`liste_des_valeurs` est la liste des valeurs initiales du tableau, elle peut être une liste d'équations, une liste de valeurs ou une liste de listes! `Liste_des_valeurs` peut être incomplète.

```
A:=array(1..2,1..2,[[1,2],[3,4]]);
V:=array(identity,1..3,1..3);
B:=array(1..2,1..2,[[1,2]]);
M:=array(1..2,1..2);
```

Pour afficher un tableau, on peut aussi utiliser la commande `print`. De plus, si les entrées sont toutes définies, il y a aussi la commande `eval`. Notez que dans une liste où les entrées sont incomplètes, les valeurs non définies sont remplacées par des points d'interrogation lors de l'appel de la commande `eval`.

```
print(A);
print(V);
print(B);
print(M);
```

```
eval(A);
eval(B);
eval(M);
```

Notez que, même si l'affichage est le même, les fonctions `print` et `eval` ne renvoient pas le même type de résultat.

```
whattype(print(A));
whattype(eval(A));
```

Même s'ils peuvent gérer les mêmes objets, un tableau et une liste de listes ont leurs fonctions propres.

```
B:=array([[5,6],[7,8]]);
L:[[5,6],[7,8]];
```

```
whattype(B);
whattype(eval(B));
```

```
whattype(L);  
whattype(eval(L));
```

La différence entre tableau et liste devient difficilement perceptible lorsqu'il s'agit d'un tableau à une dimension.

```
C:=array([9,10,11]);  
whattype(%);  
L:=[9,10,11];  
whattype(%);
```

On peut convertir un tableau en liste de listes via la commande `convert`.

```
E:=array(1..3,1..3,[[1,2,3],[a,b,c],[4,5,6]]):  
convert(E,`listlist`);  
whattype(%);
```

Réciproquement une liste peut-être convertie en tableau par la même commande `convert`.

```
L:=[[cos(x), sin(x),tan(x)],[x,x^2,x^3]];  
convert(L,`array`);  
whattype(%);
```

L'affectation dans un tableau se fait par référence.

De même que dans une liste, on peut sélectionner une entrée dans un tableau :

```
E:=array(1..3,1..3,[[1,2,3],[a,b,c],[4,5,6]]):  
E[2,3];
```

Toutefois, rappelons que dans une liste de listes, on peut extraire des éléments qui sont des listes. Ceci n'est pas possible pour les tableaux.

```
L:=[[cos(x), sin(x),tan(x)],[x,x^2,x^3]]:  
L[1];  
L[1,2];
```

### **Exercice 1**

Soient les listes  $[a,b,c]$  et  $[d,e,f,g]$ .

Quelle est la commande qui permet de concaténer ces deux listes en une seule ici  $[a,b,c,d,e,f,g]$ ?

### **Exercice 2**

Quelle est la commande qui permet de connaître le nombre d'élément d'une suite  $s$ ?

### **Exercice 3**

En utilisant uniquement les fonctions de ce TP, trouver le nombre de nombres premiers entre 600 et 1000.

### **Exercice 4**

1. Créer une liste des dix premiers multiples non nuls de 1.  
Idem pour les multiples de 2,3 et 4.
2. Mettre ces listes sous forme d'un tableau à deux dimensions..