Calcul formel avec Maple

Aide : placer le curseur sur le mot-clé, puis *Help -> Help on*
Autres possibilités :

1. -> help(sujet) ; ou -> ?sujet pour avoir de l'aide sur *sujet*.
2. *Help -> Topic Search* ou *Help -> Glossary*.

Variable : une variable a un nom, est a priori libre (sans valeur) ou affectée (avec une valeur qui peut être une expression contenant des variables libres ou nom).
:= pour l'affectation : $>x := 2*y\hat{}2+3*y-1;$
Définition d'une fonction : on utilise $>nomdefonction := variable -> expression;$
Exemple : $>f := x->2*x\hat{}2+3*x-1;$
Si on ne veut pas d'affichage, on utilise : au lieu de ;.
Pour deux ou plusieurs affichages sur une même ligne, on peut utiliser une virgule ,.
Valeur prise en x : $>f(x);$
Valeur prise en 1 : $>f(1);$
Valeur approchée : $>evalf(f(1));$
evalf(*nombre*) permet l'affichage de *nombre* avec 10 chiffres.
evalf(*nombre*,n) permet l'affichage de *nombre* avec n chiffres.
Autre écriture : evalf[n](*nombre*).
Digits :=nd; permet de fixer le nombre de chiffres retournés dans les calculs à *nd* (valeur par défaut : 10).
Exemple : Digits :=20 ;
Digits :=10 ; pour revenir à la valeur par défaut.
Le nombre maximal de chiffres pris en compte est donné par *kernelopts(maxdigits) ;* Il dépend de l'ordinateur utilisé.
Composition : @ pour $\circ$ (f@g, (f@g)(x)).
unapply permet de transformer une expression en fonction.
e :=x^2+x+3 : f :=unapply(e,x) ;
Numérateur, dénominateur d'une fraction : *numer*, *denom*.
Dito : % pour utiliser le résultat précédent (*ditto* en anglais).
π (le nombre) : Pi
π (le symbole) : pi
Infini : $infinity$ pour $+\infty$ et $-infinity$ pour $-\infty$.
Développement : $>expand((x+1)*(x-3),x);$
Factorisation : $>factor(x\hat{}2-5*x+6);$
Factoriser avec option : $>factor(x\hat{}2-x-1,sqrt(5));$
Simplification des expressions (voir l'aide) : *simplify, normal, combine, sort*
Résolution d'équation $f(x)=0$: $>solve(f(x));$ ou $>solve(f(x),x);$
Exemple : $>solve(x\hat{}2-x-1,x);$
Ensemble des solutions : $>solutions := \{solve(x\hat{}2-x-1,x)\};$

Solutions : $>solution1 := solutions[1];\ solution2 := solutions[2];$

Assignation de la dernière solution : $>eq := x**2 - x - 1 = 0; S := solve(eq); assign(S); x;$

Représentation graphique sur $[-1, 3]$: $>plot(f(x), x = -1..3)\,;$

Autre possibilité : $>plot(f, -1..3)\,;$

Représentation de plusieurs fonctions : $>plot(\{f(x), 2 * x + 1\}, x = -2..1)\,;$

On peut préciser

1. la plage des ordonnées : $>plot(\{f(x), 2 * x + 1\}, x = -2..1, y = -2..2.5)\,;$
2. la couleur : $>plot(\{f(x), 2 * x + 1\}, x = -2..1, y = -2..2.5, color = black)\,;$
3. les couleurs et le style :
   $>plot([f(x), 2 * x + 1], x = -2..1, y = -2..2.5, color = [red, blue], style = [point, line]);$
4. l'échelle : scaling= constrained (même unité sur les axes), unconstrained (unité adaptée sur chacun des axes) :
   $>plot([f(x), 2 * x + 1], x = -2..1, y = -2..2.5, color = [red, blue], scaling = constrained);$

En cliquant sur le graphique, le menu contextuel (clic droit) permet d'accéder à la légende, le style, aux axes, ...

Pour associer des éléments graphiques de types différents, on peut utiliser la libraire *plots* et en particulier *display* :

```
> with(plots) :
p1 :=plot(tan(x),x=-Pi/2..Pi/2,y=-3*Pi/2..3*Pi/2,color=black,scaling=constrained) :
p2 :=plot([x,arctan(x)],x=-3*Pi/2..3*Pi/2,color=[red,blue],scaling=constrained) :
p3 :=plot([Pi/2,t,t=-3*Pi/2..3*Pi/2],color=green) :
p4 :=plot([-Pi/2,t,t=-3*Pi/2..3*Pi/2],color=green) :
p5 :=plot({-Pi/2,Pi/2},x=-3*Pi/2..3*Pi/2,color=green) :
display ({p1,p2,p3,p4,p5}) ;
```

On peut aussi écrire : plots[display]({p1,p2,p3,p4,p5}) ;

Titre et légende : *title* et *legend*

plot([sin, cos], -Pi..Pi, title="Fonctions trigonométriques", legend=["sinus", "cosinus"]) ;

Couleurs : color=...

Couleurs prédéfinies : aquamarine, black, blue, navy, coral, cyan, brown, gold, green, gray, grey, khaki, magenta, maroon, orange, pink, plum, red, sienna, tan, turquoise, violet, wheat, white, yellow.

On peut aussi définir ses propres couleurs (système RGB ou HUE).

Exemple :

macro(palegreen=COLOR(RGB, .5607, .7372, .5607)) ;

plot(sin, color=palegreen) ;

Limite de f : *limit(f(x),x=0) ;* ou *limit(f(x),x=infinity) ;*.

Dérivée de f : $>diff(f(x), x)\,;$

Dérivée d'ordre 2 : $>diff(f(x), x, x)\,;$ ou $>diff(f(x), x\$2)\,;$

Dérivée d'ordre n : $>diff(f(x), x\$n)\,;$

Fonction dérivée : $> D(f)\,;$

Fonction dérivée seconde : $> (D@D)(f)\,;$

Fonction dérivée d'ordre n : $> (D@@n)(f)\,;$

Primitive : $>int(f(t), t = 0..x)$ pour $\int_0^x f(t)\,dt$.

$F := x- > int(f(t), t = 0..x)$ pour créer la fonction primitive F qui s'annule en 0.

Intégrale : $>int(f(t), t = 0..1)$ pour $\int_0^1 f(t)\,dt$.

Développement limité : $>taylor(f(x), x, n)\,;$

Substitution : la fonction subs permet de substituer à une variable libre une expression quelconque dans une expression.

$>\text{subs}(x = 2, 2 * x\hat{}2 - 3 * x + 2);$

$>\text{subs}(x = a, x + sqrt(x);$

$>\text{subs}(\cos(x) = y, 1 + \cos(x);$

Fonctions de référence

1. pour e^x : $>\exp(\text{x})\,;$
2. pour $\cos x$: $>\cos(\text{x})\,;$
3. pour $\sin x$: $>\sin(\text{x})\,;$
4. pour $\tan x$: $>\tan(\text{x})\,;$
5. pour $\arccos x$: $>\arccos(\text{x})\,;$
6. pour $\arcsin x$: $>\arcsin(\text{x})\,;$
7. pour $\arctan x$: $>\arctan(\text{x})\,;$
8. pour $\ln x$: $>\ln(\text{x})\,;$
9. pour $\sqrt{x}$: $>\text{sqrt}(\text{x})\,;$
10. pour $|x|$: $>\text{abs}(\text{x})\,;$
11. pour *signe de* x : $>\text{csgn}(\text{x})\,;$

Fonction définie par intervalles : $> piecewise(condition_1, f_1, condition_2, f_2, ..., sinon)$

Exemple : $> f := x- > piecewise(x < 0, -1, x > 0, 1, 0);$

Séquence, liste, ensemble

1. Séquence : collection ordonnée d'expressions
   $> s := 1, x, x\hat{}2;$
   $> seq(i + x, i = 1..3);$
   $NULL$ désigne la séquence vide.

2. Liste : séquence entre crochets.
   $>$ L :=[a,b,c] ;
   $[NULL]$ désigne la liste vide.

3. Ensemble : collection non ordonnée d'expressions, les expressions n'apparaissant qu'une fois.
   >E := {a,b,c} ;

Remarque : un ensemble ou une liste est une expression, pas une séquence (important pour *plot* ou les procédures : arguments).
Nombre d'éléments d'un ensemble ou d'une liste : > nops(L) ;
De manière plus générale, *nops* fournit le nombre d'opérandes d'une expression : nops(a+b+1) ; retourne 3.
Transformation d'un ensemble ou d'une liste en séquence (pour extraire les éléments d'un ensemble ou d'une liste) : > op(E) ;
Transformation d'une séquence en ensemble ou liste : $> s := 1, 2, 3;\ E := \{s\};\ L := [s];$
Accès au ième élément d'une séquence, d'une liste ou d'un ensemble :
$> s := 0, 1, 2, 3 :\ s[1];$ Retour : 0
Remarque : pour les séquences et les ensembles, il s'agit d'un accès en lecture seule.
Pour les listes : $> L := [0, 1, 2, 3] : L[2] = 0;$ Retour : $[0, 0, 2, 3]$
Autre possibilité : $op(i, L)$ pour obtenir le ième élément de la liste L.
Pour modifier le ième élément d'un ensemble ou d'une liste, on peut utiliser *subsop* : *subsop(i=valeur,E)*
$> E := \{0, 1, 2, 3\} : subsop(2 = 7, E);$ Retour : $\{0, 2, 3, 7\}$
member(x,L) renvoie *true* si x appartient à la liste, *false* sinon.
Concaténation de séquences : $> s1 := 1, 2, 3 :\ s2 := 3, 4, 5 :\ s := s1, s2;$ Retour : $s := 1, 2, 3, 3, 4, 5$
Opérations sur les ensembles : *union* ($\cup$), *intersect* ($\cap$), *minus* ($-$)
Remarque : on ne peut pas maîtriser l'ordre d'apparition des éléments dans un ensemble.
Pour appliquer une fonction f à tous les éléments d'un ensemble ou d'une liste : $> map(f, E);$
$> f := x- > x\hat{}2 + 1 : map(f, \{0, 1, 2, 3\}), map(f, [a, 2, b]);$ Retour : $\{1, 2, 5, 10\},\ [a^2 + 1, 5, b^2 + 1]$
zip(f,X,Y) permet d'associer les éléments de deux listes X et Y suivant une fonction *f*.
Exemple : X :=[1,2,3,4,5] :Y :=[0,2,3,2,3] :zip((x,y)->[x,y],X,Y) ; renvoie [[1, 0], [2, 2], [3, 3], [4, 2], [5, 3]]
Autre exemple : zip(igcd,[0,14,8],[2,6,12]) ; renvoie [2, 2, 4] (pgcd).

Booléen

Un booléen est une expression qui représente une proposition (énoncé qui peut s'évaluer en vrai (true) ou false (faux).
On peut évaluer un booléen grâce à *evalb*
Opérateurs de comparaison : =, <, >, <=, >=, <>.
$> evalb(1 <> 8), evalb(1 >= 8);$ Retour : $true,\ false$
Opérateurs logiques : *or*, *and*, *not*.
$> A := true :\ B := false :\ not(A)\ or\ B;$ Retour : $false$
Priorités : 1. not 2. and 3. or
$> A := true :\ B := false :\ not\ A\ or\ B;$ Retour : $false$

Bibliothèques ou librairies (packages)

1. graphique : with(plots) ; with (plottools) ; (pour les fonctions arrow, curve, line, point...)
2. ensembles : with(combinat) ;
   Fonctions *powerset* et *cartprod*.
   Exemples
   E :=1,2 ;powerset(E) ; fournit l'ensemble des parties de E.
   On peut intégrer *cartprod* à une procédure pour obtenir le produit cartésien :

```
Prodcart :=proc(E,F)
local P,Prod ;
P :=combinat[cartprod]([E,F]) ;
Prod :=NULL ;
while not P[finished] do Prod :=Prod,P[nextvalue]() end do ;
{Prod} ;
end :
Exemple :
E :={a,b} ;F :={1,2} ;Prodcart(E,F) ;Prodcart(F,E) ;
```

3. arithmétique : with(numtheory) ;
   Fonction *divisors*
4. statistique et probabilités : with(stats) ;
   sous-librairies (subpackages) : describe, fit, transform, statevalf, statplots
5. algèbre linéaire : with(linalg) ; ou with(LinearAlgebra) ;
6. ...

Arithmétique

1. *mod*, *modp* pour les calculs modulo n.
   Exemple : 18 mod 7 ; et modp(18,7) ; retournent 4.
2. *iquo* et *irem* : quotient et reste dans la division euclidienne.
   Exemple : iquo(18,7),irem(18,7) ; retourne 2,4.
3. *ifactor* retourne les facteurs premiers d'un entier.
4. *igcd* retourne le PGCD de deux entiers (greatest common divisor of integers).
   igcdex(a,b,'u','v') renvoie le *PGCD(a,b)*, ainsi que u et v tels que : $au+bv=PGCD(a,b)$
   Exemples : *igcd(12,15)* ; renvoie 3
   igcdex(12,15,'u','v'),u,v ; retourne 3, -1, 1
5. *ilcm* retourne le *PPCM* (least common multiple of integers)
6. *Nombres premiers* : *isprime* (test de primalité), *ithprime* : ième nombre premier, *nextprime* et *prevprime* : nombre premier suivant et précédent.

Algèbre linéaire

Deux librairies sont disponibles : *linalg* et *LinearAlgebra*.

1. *linalg*
   (a) Création d'une matrice :
   ```
   B1 :=matrix(2,2,[1,2,3,4]) ;
   B2 :=matrix(2,2,[[1,2],[3,4]]) ;
   ```
   (b) matrix(m,n) : création d'une matrice de type (m,n) dont les variables sont libres.
   ```
   B3 :=matrix(2,2) ;
   ```
   (c) evalm(B3) ; pour visualiser une matrice (par défaut, c'est le nom de la matrice qui apparaît : la taille peut être très importante).
   (d) matrix(m,n,0) : matrice de type (m,n) nulle
   (e) diag(seq(1,x=1..n)) ou diag(1$n) : matrice unité I_n.

(f) f :=(i,j)->expression de i et j ; A :=matrix(m,n,f) ; permet de définir une matrice dont les coefficients sont définis en fonction de i et j.

(g) Opérations

i. A+B pour additionner deux matrices.

ii. A-B pour soustraire deux matrices.

iii. c*A pour multiplier une matrice A par un scalaire c.

iv. multiply(A,B) pour multiplier deux matrices. multiply(A, B, C) ; pour trois matrices. On peut aussi utiliser &*, puis evalm : evalm(A&*B) ;

v. inverse(A) pour obtenir l'inverse d'une matrice inversible.

(h) A1 :=copy(A) pour créer une copie A1 de la matrice A et travailler sur A1 tout en ne modifiant pas A.
Remarque : Le nom de la matrice ou du tableau représente l'endroit où est rangé le tableau dans la machine.
Si A est un tableau ou une matrice, l'affectation A1 :=A n'a pas pour effet de recopier A dans A1, mais seulement d'attribuer à A1 la même contenance que A. Et toute modification se reporte sur le tableau ou la matrice d'origine ! Pour obtenir un tableau de contenu identique à A mais différent de A, on écrit A1 :=copy(A). On peut ensuite modifier des valeurs : A1[2,3] :=1 ; par exemple.

(i) transpose(A) retourne la matrice transposée de A (échange des lignes et des colonnes).

(j) concat(A1, A2) pour étendre A1 en l'associant à A2 *horizontalement.*

(k) stackmatrix(A1, A2) pour étendre A1 en l'associant à A2 *verticalement.*

(l) addrow(A,i,j,c) retourne une copie de la matrice A dans laquelle la ligne L_j est remplacée par $L_j + cL_i$.

(m) addcol(A,i,j,c) : comme addrow(A,i,j,c), mais pour les colonnes.

(n) mulrow(A,i,c) retourne une copie de la matrice dans laquelle la ligne i a été multipliée par le nombre c.

(o) mulcol(A,i,c) : comme mulrow(A,i,c), mais pour les colonnes.

(p) swaprow(A,i,j) renvoie une nouvelle matrice dont les lignes i et j ont été échangées.

(q) swapcol(A,i,j) : comme swaprow, mais pour les colonnes.

(r) delrows(A,plage) et delcols(A,plage) permettent de supprimer des lignes ou des colonnes.

(s) subvector(A, r, c) retourne le vecteur extrait dans la ligne r ou la colonne l.
Exemple : x1 :=subvector(A,1,1..3) ; retourne le vecteur de la ligne 1 de la matrice A (coefficients : colonne de 1 à 3).

(t) submatrix(A,r,c) retourne la sous-matrice de A précisée par les lignes r et colonnes c.
Exemple : submatrix(Ae, 1..3, 4..6)

2. *LinearAlgebra*

(a) Matrix pour créer une matrice : Matrix([[1,2,3],[4,5,6]]) ;

(b) Vector pour créer un vecteur colonne : X[1] :=Vector([1,1,2,1]) ;X[2] :=Vector(4) ;# vecteur nul
Autre méthode : V :=<1,2,3> ;

(c) Vector[row] pour créer un vecteur ligne : V2 :=Vector[row]([1,2,3]) ;

(d) A :=<X[1]|X[2]> pour construire une matrice en associant des vecteurs ou des matrices.

(e) Pour extraire une sous-matrice : A[1..3,1..4] ;

(f) Matrice unité I_n : IdentityMatrix(n) ;

(g) Opérations

i. A+B pour additionner deux matrices.

ii. A-B pour soustraire deux matrices.

iii. c*A pour multiplier une matrice A par un scalaire c.

iv. Multiply pour multiplier deux matrices : Multiply(A,B) ;

v. MatrixAdd(A, B) pour additionner deux matrices.

vi. VectorAdd(A, B) pour additionner deux vecteurs.

vii. Combinaison linéaire de vecteurs ou de matrices : Add(A, B, c1, c2) ; calcule la somme c1*A + c2*B.
Autre possibilité : MatrixAdd(A, B,c1,c2) ; ou VectorAdd(A, B,c1,c2) ;

viii. ScalarMultiply pour multiplier une matrice ou un vecteur par un scalaire : ScalarMultiply(A,c) ;

ix. MatrixInverse(A) pour obtenir l'inverse d'une matrice inversible.

Statistique et probabilités

1. Librairie stats avec les sous-librairie (subpackages) :
   (a) describe : stats[describe, fonction](argument) ou describe[fonction](argument)
   stats[describe, fonction] ou describe[mean]
   variance, standarddeviation, mode, range, quantile, covariance...
   (b) transform : stats[transform, fonction](argument) ou transform[fonction](argument)
   classmark, cumulativefrequency, frequency, scaleweight, split, statsort, statvalue, tally-tallyinto
   (c) statplots : stats[statplots, fonction](argument) ou statplots[fonction](argument) ou fonction(argument)
   boxplot, histogram, scatterplot

Programmation

Traitement conditionnel :

if *booléen* then *instruction* else *instruction* end if ;

```
if booléen then instruction
elif booléen then instruction
else instruction
end if ;
```

elif permet d'éviter *else if ... end if*

Traitement itératif :

for *nom* from *expression* by *expression* to *expression* do *instruction* end do ;

nom :=*expression* ; while *nom* <= *expression* do *instruction* ; *nom* :=*nom*+1 end do ;

Procédure

```
> nom :=proc(argument1, argument2, ...) #paramètres
local variable(s) locale(s)
Corps de la procédure ; #instructions
Variable(s) résultat
end ;
```

Appel : *nom*(*valeur1*, *valeur2*, ...)

Le résultat renvoyé par une procédure est le dernier résultat qu'elle a calculé.
La commande *RETURN(expression)* permet de sortir de la procédure en retournant *expression*

args[i] : argument numéro i.

Exemple :

```
> u :=proc(u0, a,n)
local v,i ; #F5 pour retirer le >
v :=evalf(u0) ;
for i from 1 to n do v :=(v+a/v)/2 end do ;
v
end ;
```

Appel :

```
> u(2,2,5) ;
```

1.414213562

Exemple : autre version

```
> u := proc(u0,a,n)
local v,i ;
v :=evalf(u0) ;i :=1 ;
while i <= n do v :=(v+a/v)/2 ; i :=i+1 end do ;
v
end ;
```

Génération de la suite pour la représenter :

```
> suiteu := proc(u0,a,n)
local i,v,S;
v :=evalf(u0) ;S :=[[0,v]] ;i :=1 ;
while i<= n do v :=(v+a/v)/2 ; S :=[op(S),[i,v]] :i :=i+1 end do ;
S
end ;
```

op(S) pour obtenir la séquence formée des éléments la liste S.

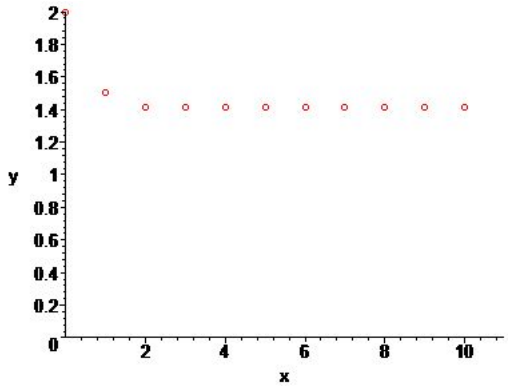
Appel :

```
> suiteu(2,2,10) ;
```

[[0, 2.], [1, 1.500000000], [2, 1.416666666], [3, 1.414215686], [4, 1.414213562], [5, 1.414213562], [6, 1.414213562], [7, 1.414213562], [8, 1.414213562], [9, 1.414213562], [10, 1.414213562]]

Représentation de la suite :

```
plot(suiteu(2,2,10), x=0..11, y=0..2,style=point,symbol=circle) ;
```



print pour afficher : $x := 3$; $print("x = ",x)$; ou $print(`x = `)$; (accent grave)
$x := 3 : print("x =", x), print(`x = `, x);$
Retour :
"x=", 3
x=, 3
printf : $x := 7 : y := x - 1/x : printf("x = \%1.0f\ y = \%5.3f\ y = \%a\backslash n", x, y, y)$;

Autre exemple :

```
F := proc() ;
> if nargs = 0 then
# Pas d'argument : annonce
print (‘Calcul de n !/(n+1) pour n entier‘)
# Argument négatif : (elif est mis pour else if)
elif args[1] < 0 then RETURN(‘Argument négatif‘)
# Calcul
else RETURN(factorial(args[1])/(args[1]+1))
end if
end ;
```

Import - export

1. *Export As ...* pour sauvegarder une image au format jpeg, EPS (Encapsulated PostScript)... (cliquer sur l'image et utiliser le menu contextuel : clic droit).
   Le cas échéant, convertir le fichier EPS exporté au format pdf (*Clic droit puis Convertir au format pdf*).
2. Exportation d'une image au format ps.
   Exemple : plotsetup(ps,plotoutput=`E :\\documents\\Fichiers maple TP\\monimage.ps`, plotoptions=`color,portrait,noborder`) ;
   p1 :=plottools[point]([3,1],color=blue,symbol=diamond,symbolsize=20) :
   p2 :=plottools[point]([1,2],color=red,symbol=circle,symbolsize=20) :
   plots[display]({p1,p2},view=[-0.5..3.5,-0.5..3.],tickmarks=[4,4]) ;
   plotsetup(inline) ;# ou plotsetup(default) ; pour revenir au mode d'affichage par défaut.
   Convertir ensuite le fichier ps au format pdf (*Clic droit puis Convertir au format pdf*).
3. Importation d'un fichier .txt dans une liste :
   data :=readdata("E :\\documents\\Fichiers maple TP\\donnees.txt",float,10) ;
   Arguments : fichier, type d'élément (integer, float...), nombre de colonnes.
   Type de sortie : liste.
4. Exportation d'une matrice A de type Matrix (LinearAlgebra) au format txt
   ExportMatrix("E :\\documents \\matriceA.txt",A,format=rectangular,delimiter=" ") ;
   Importation d'une matrice au format txt dans une matrice de type Matrix
   A1 :=ImportMatrix("E : \\documents \\Fichiers maple\\matriceA.txt",delimiter=" ") ;
   Conversion éventuelle au type matrix (linalg) :
   A1b :=convert(A1,matrix) ;

Apostrophe (quote)

Distinguer :

1. l'apostrophe droite (quote) ' ' pour réinitialiser une variable (retour à une variable libre : non affectée).
   Exemple : unassign('x') ou x :='x'.
2. l'apostrophe droite double (double quotes) " " pour les chaînes de caractères (string).
   Exemple : "abc, etc"
3. l'accent grave ou apostrophe inversée ` ` (back quote : name quote) pour les noms (names) de variable ou les symboles (symbols).
   Exemple : `a b c`

Règles d'évaluation

Une variable peut avoir deux statuts : variable affectée (ou étiquette : assigned variable) et variable non affectée (ou nom : unassigned variable).
L'affectation se fait par la commande : *variable := expression ;*
expression est composée de nombres, chaînes de caractères, listes, tableaux, fonctions ou variables affectées ou non.

La règle d'évaluation complète (full evaluation rule)

Maple remplace toutes les variables affectées par leur valeur, y compris pour les variables affectées contenues dans les valeurs obtenues.

Exemple :
y :=x^2 ;
x :=2*a ;
y ; retourne 4a^2
a :=3 ;
x,y ; retourne 6, 36

La règle d'évaluation au dernier nom (last name evaluation rule)

Elle concerne les variables structurées comme les tableaux et les matrices.
Les variables affectées sont remplacées par leurs valeurs si ces valeurs sont des noms.
Exemple :
A :=array(1..4,[1,2,3,4]) ;
B :=A :B ;eval(B) ; retournent A, puis [1,2,3,4].
B[1] :=5 :eval(A) ; retourne [5,2,3,4].
A l'occasion de l'affectation B :=A, la valeur obtenue est le nom A et donc la même zone de mémoire et non pas le contenu de A.
Utiliser B :=copy(A) ; pour conserver le tableau original et travailler sur une ''copie''.

Réinitialisation d'une variable

Pour supprimer l'affectation d'une variable (libération ou ré-initialisation d'une variable) et ainsi retrouver sous son nom une variable non affectée, on peut utiliser :

1. restart ; (qui supprime toutes les affectations)
2. unassign('variable') ; (quotes)
3. variable :='variable' ;

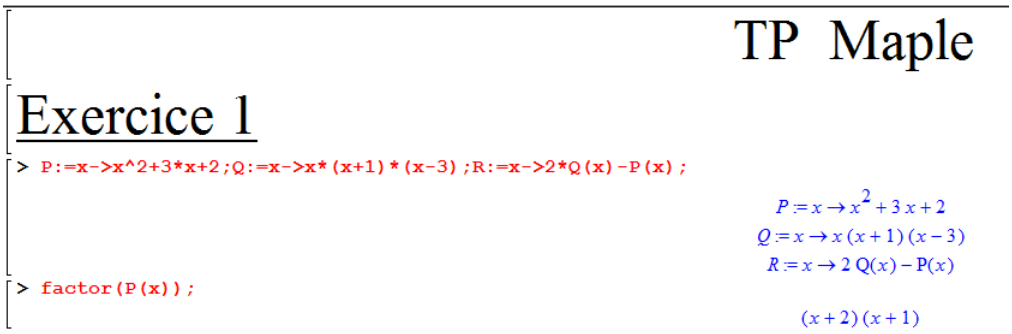
Remarque : l'affectation et la réinititalisation agit sur les feuilles ouvertes.

Maj + Entrée pour aller à la ligne sans introduire un nouveau prompt : > (par exemple dans la définition d'une procédure).
F5 pour supprimer le prompt > dans une ligne.
Insert -> Execution Group pour créer un groupe d'execution comprenant un crochet et un prompt avant le curseur (*commande en mode math*) *Before Cursor ou Ctrl K* ou après *After Cursor ou Ctrl J*.

Insertion d'une ligne de commande : *Insert -> Maple Input (Ctrl M)*
Insertion de texte (par exemple pour "Exercice 1", pour préciser la sortie, ...) : *Insert -> Text*
pour des commentaires.

Indications de présentation

Insert -> Execution Group pour créer un groupe d'execution comprenant un crochet et un prompt avant le curseur (*commande en mode math*) *Before Cursor (ou Ctrl K)* ou après *After Cursor (ou Ctrl J)*.
Insert -> Text (ou Ctrl T) pour insérer le numéro de l'exercice, ou un texte de réponse ou de commentaire (après avoir sélectionné le crochet d'invite ou non).
Insert -> Maple Input (ou Ctrl M) après avoir sélectionné le crochet d'invite pour insérer une ligne de commande.



Tutorials

1. Maple Quick Start
   https ://www.maplesoft.com/support/training/quickstart.aspx
2.