Chapitre 3
Programmation en Maple

1. Introduction
Ce chapitre présente la syntaxe et les règles essentielles qui régissent la programmation avec Maple. Celui-ci étant un langage interprété, programmer revient à déclarer des variables de type fonction. Nous nous intéressons donc dans cette partie à l'écriture de fonctions. La syntaxe de structures de contrôle, les éléments essentiels qui interviennent dans la déclaration et l'écriture de fonctions en langage Maple.

Remarque importante: pour pouvoir entrer plusieurs lignes d'instructions Maple successives, sans exécution à la fin de chaque ligne, presser ***Shift+Entrée*** à la fin de chaque ligne.
Puis, pour valider l'ensemble des lignes ainsi écrites, presser ***Entrée*** en plaçant le curseur par exemple sur la dernière ligne écrite.

2. Les tests
Il arrive souvent que la décision d'effectuer telle ou telle tache dépende de la réalisation d'une condition, on a alors recours au **if**.

2.1 L'instruction conditionnelle simple

La syntaxe est la suivante :

```
if condition then
I1
I2
...
Ik
fi;
```

Explication : si la condition est vraie exécuter les instructions I1,…, Ik.
Exemple :

```
> a:=10;b:=25;
if (a<=b) then
print(b);
fi;
if (a>b) then
print(a);
fi;
```

```
a := 10
b := 25
  25
```

Pour éviter la répétition des instructions **if** on utilise l'instruction conditionnelle avec **else**.

2.2 L'instruction conditionnelle avec else
La syntaxe est la suivante :

```
if condition then
```

```
I1
…
Ik
else
I'1
…
I'k
fi;
```

Explication : si la condition est vraie exécuter les instructions I1, …, Ik sinon exécuter les instructions I'1, …, I'k.

Exemple :

```
> a:=10;b:=25;
if (a<b) then
print(b);
else
print(a);
fi;
```

a := 10
b := 25
25

2.3 L'instruction conditionnelle imbriquée

La syntaxe est la suivante :

```
if condition1 then
 instruction 1
elif condition2 then
 instruction2
elif condition3 then
instruction3
 ...
else instructionN
fi;
```

Explication : si la condition1 est vraie exécuter les instructions instruction1, sinon, si la condition2 est vraie exécuter les instructions instruction2, sinon si la condition3 est vraie exécuter les instructions instruction3…, sinon, exécuter Instruction N.

Ces opérations effectuent des tests selon certaines conditions: condition1, condition2, ..., conditionN. ***elif*** signifie "autrement si". La dernière instruction à exécuter doit être précédée de ***else***.

Exemple 1: résolution de l'équation du premier degré $ax + b = 0$.

Modifiez les valeurs de *a* et de *b*, puis validez les lignes suivantes:

```
> a:=7:b:=3:
> if a<>0 then print(`Une solution : x `=-b/a)
  elif b=0 then print(`Tout x est solution`)
  else print(`Pas de solution`)
  end if;
```

$$\text{Une solution : } x = \frac{-3}{7}$$

Exemple 2 : équation du second degré

```
> a:=1; b:=1; c:=1; delta:=b*b-4*a*c;
if (delta>0) then
print("deux solutions réelles");
elif (delta=0) then
print("une solution double réelle");
else
print("deux solutions complexes");
fi;
```

a := 1
b := 1
c := 1
d := -3
"deux solutions complexes"

3. Les opérateurs logiques and, or et not (ET, OU et NON)

Lorsqu'on souhaite vérifier plusieurs conditions simultanément, on a recours aux operateurs logiques.

and --> vrai si condition1 et condition2 sont satisfaites
or --> vrai si condition1 ou condition2 est satisfaite
not --> vrai si condition n'est pas satisfaite

Exemple: On veut savoir a quelle partie du domaine D = [-5,8] appartient x.

Si (x > -5) et (x < 8) alors
x est a l'intérieur du domaine
sinon not ((x=-5) ou (x = 8)) alors
x est a l'extérieur du domaine
sinon
x est sur la frontière du domaine
fin si

```
>x := 9;
x := 9
>if (x > -5 and x < 8) then
printf(`x extérieur`)
elif not (x = -5 or x = 8) then
printf(`x extérieur`)
else
printf(`x frontière`)
fi
```

x extérieur

4. Les itérations

Les commandes itératives servent à répéter une séquence d'instructions convenablement modifiées à chaque étape. Elles sont présentées dans Maple sous cinq formes : les boucles **for** … **do** et **while** qui existent dans plusieurs langages de programmation, ainsi que les commandes **seq, $ et map**.

4.1 Structure de contrôle for .. to:

for variable ***from*** initiale ***to*** finale ***by*** pas ***do*** ... ***od;***

Exécute une boucle pour une variable allant d'une valeur initiale à une valeur finale, avec un pas donné.
from , by peuvent être omises ou écrites dans un ordre quelconque. Si l'on omet ***from*** , initiale vaut 1. Si l'on omet ***by*** , pas vaut 1.

Exemple 1: calcul de la somme des 100 premiers entiers naturels

```
> somme:=0:
 for k from 0 to 100 do somme:=somme+k od:
 printf(`somme `=somme);
```

somme = 5050

Exemple 2: calcul de la somme des entiers impairs inférieurs à 100
Ici, on omet ***from*** donc la première valeur de *k* est 1. La dernière valeur prise par *k* sera 99.

```
> somme:=0:
for k to 100 by 2 do somme:=somme+k od:
printf(`somme `=somme);
```

somme =2500

for variable ***in*** expression ***do*** .. ***od;***

Exemple 3: donner les nombres premiers de la liste [31,39,47,105]

```
> for k in [31,39,47,105] do
if isprime(k) then printf(k,`est premier`) end if
od;
```

31, est premier
47, est premier

L'instruction ***break*** permet de sortir de la boucle.
L'instruction **next** permet de passer à l'itération suivante dans la boucle.
Exemple :

```
> for i from 1 to 10 do
if i=4 then next;fi;
if i=6 then break; fi;
print(i);
od;
```

1 2 3 5 6

4.2. Structure de contrôle while .. do:
Il arrive souvent qu'on veuille répéter un groupe d'instructions jusqu'a ce qu'une certaine condition soit satisfaite. C'est justement le type de contrôle que permet d'exercer la commande **while**, sa structure syntaxique est la suivante :

while condition ***do <instruction> end do;***

Exécute une suite d'instructions, tant que la condition est vraie.

Exemple 1: calcul de la somme des 100 premiers entiers naturels

```
> somme:=0:k:=-1:
```

while k<100 do k:=k+1:somme:=somme+k end do:
printf(`somme `=somme);

somme = 5050

Exemples 2:

>i:=4:
while *i<=30* ***do***
printf(i);
i:=i+7;
od: *# : au lieu de ; pour ne pas afficher le i*

4 11 18 25

Exemples 3:

>restart; l:=[a,b,c,d,e]: i:=1:
while *i<=nops(l)* ***do***
print(l[i]);
i:=i+1;
od*: # : au lieu de ; pour ne pas afficher le i*

a b c d e

4.3 La commande seq

Si f(i) est le terme général d'une suite, alors la commande :

seq(f(i),i=m..n)

génère la suite : f(m), f(m+1), …, f(n).

Cette commande peut être utilisée pour calculer des sommes ou des produits. Voici un autre exemple.

Exemple : produit cartésien -- voici la suite des couples d'un produit cartésien d'ensembles A*B.

>restart :
A := {a,b,c,d} :
B := {1,2,3}:
Seq(seq([i,j],i=A),j=B) ;

[a,1],[b,1],[c,1],[d,1],[a,2],[b,2],[c,2],[d,2],[a,3],[b,3],[c,3],[d,3]

4.4. La commande $

La commande **$**, tout comme la commande **seq**, est utilisee lorsqu'on désire construire une suite f(i) avec un paramètre i variant entre a et b. la syntaxe de la commande **$** est la suivante :

'f(i)'$'i' = a .. b

Exemple : construisons la suite 1^2, 2^2, 3^2, 4^2, 5^2 avec la commande **$**

>'i^2'$'i'=1..5 ;

1, 4, 9, 16, 25

4.5 La commande map

La commande ***map(f,L)***, utilise également l'itération pour appliquer une procédure f aux membres d'une liste L ou, de façon plus générale, aux opérandes d'une expression.

Exemples :

1 - *>map(D, [x ->x^2, x ->x^3, sin]) ;*

$$\left[x \to 2x, x \to 3x^2, \cos\right]$$

2- *>f := x -> log[10](x) ;*

$$x \to \log_{10}(x)$$

3- *>map (f, [1,10,100,1000]) ;*

$$\left[0, 1, \frac{\ln(100)}{\ln(10)}, \frac{\ln(1000)}{\ln(10)}\right]$$

4- *>simplify(%) ;*

$$[0, 1, 2, 3]$$

5- *> f := (i,j) -> x^((i+1)^j) ;*

$$(i,j) \to x^{(i+1)^j}$$

6- *> A := matrix(2,2,f);*

$$A := \begin{bmatrix} x^2 & x^4 \\ x^3 & x^9 \end{bmatrix}$$

7- *> map(diff,A,x);*

$$\begin{bmatrix} 2x & 4x^3 \\ 3x^2 & 9x^8 \end{bmatrix}$$

5. Les procédures en Maple

Un programme en Maple peut être organisé en sous programmes appelées **procédures**.

Une procédure est définie par les mots- clés **proc . … end proc** et peut être assignée à un nom de variable.

Ecrivons, dans un premier exemple, la procédure pour obtenir la partie entière de la moitie d'un nombre n.

Exemple : partie entière de n/2.

```
> moitie := proc(n)
trunc(n/2) ;
end proc ;
        moitie := proc(n)trunc(1/2*n) end proc
        > moitie(5);
```

2

La forme syntaxique générale de la commande **proc** est la suivante:

```
>proc(x1 :: type1, x2 :: type2, x3 :: type3, ...)
 local v1, v2, v3, .... ;
 global u1, u2, u3, ... ;
 options op1, op2, ... ;
 instructions
 end proc ;
```

Seules les commandes **proc** et **end proc** sont obligatoires dans une procédure. Toutes les autres instructions sont facultatives. Les paramètres x1, x2, … sont appelés les arguments ou les entrées de la procédure.

Les arguments d'une procédure appartiennent souvent à un type spécifique. On peut insérer pour chaque argument xi une contrainte type i sur son type en ajoutant celle-ci immédiatement après xi dans la première ligne de la procédure avec la syntaxe
xi :: typei.

Par exemple, si on veut contraindre l'entrée n à être un nombre dans la procédure moitie, on ecrit :

```
> moitie := proc(n :: numeric)
trunc(n/2) ;
end proc ;
moitie := proc(n :: numeric)trunk(1/2*n) end proc
> moitie(a);
                Error, invalid input : moitie expects its 1st argument, n, to be
                of type numeric, but received a
```

5.1. Variables locales et variables globales

L'instruction **local** dans une procédure f fournit la liste des variables dont la portée est limitée à la procédure f. Redéfinissant la procédure moitie pour donner à la variable b la valeur n/2, mais seulement à l'intérieur de la procédure :

```
> b := 100 ;
moitie := proc(n)
local b ;
b := n/2 ;
trunc(b) ;
end proc ;
                moitie := proc(n)local b ;b :=1/2*n ; trunc(b) end proc
> moitie(5) ;
                2
> b ;
                100
```

Comme on le voit, la variable b a une valeur globale de 100 et une valeur locale à l'intérieur de la procédure moitie de n/2.

La commande **global** donne la liste des variables qui ont la même valeur à l'intérieur et à l'extérieur d'une procédure.

```
> c := 70 ;
b := 100 ;
moitie := proc(n)
local b ; global c ;
```

```
b := trunc(n/2) ; c := 374 ;
print(b) ; print(c) ;
end proc :
> moitie (5) ;
        2
        374
> b ;
        100
> c ;
        374
```

5.2. Exemples

1. procédure qui calcule la somme des entiers de 1 à n

```
> somme := proc(n) # ici pas de point virgule
local s,i; # variables locales
s := 0;
for i from 1 to n do
s := s + i;
od:
s;
end;
        somme := proc (n) local s, i; s := 0; for i to n do s := s + i end do ; s
        end proc
> somme(10);
                55
> somme(100);
                5050
```

2. Procédure qui donne le factoriel de n

```
>restart;
facto := proc(n)
local p,i;
p:=1;
for i from 1 to n do
p := p*i;
od;
p;
end;
        facto := proc (n) local p, i; p := 1; for i to n do p := p´i end do ; p end
proc
> facto(5);facto(10);
                        120
                        3628800
```

5.3. **Procédures récursives**

Une procédure récursive est une procédure qui s'appelle elle-même.

Exemple : calcul du factoriel d'un nombre entier

```
> facteur := proc(n)
local f;
```

```
        if n<>1 then
        f:=n*facteur(n-1);
        else return(n);
        end if;
        > end proc;
facteur := proc(n) local f; if n <> 1 then f := n*facteur(n - 1) else return n end if; end
proc;
        > facteur(3);
```

6