Introduction à Maple

Présentation - Accès.

Maple est un logiciel de calcul formel. Il peut faire des calculs comme une simple calculatrice, effectuer des calculs plus complexes nécessitant des outils mathématiques avancées, mais aussi (et surtout) travailler avec des données symboliques comme des variables, des polynômes ou des expressions. Maple dispose également de fonctions de programmation et d'outils graphiques.
Lorsque vous avez ouvert le logiciel, vous voyez apparaître une fenêtre avec une interface graphique. Le fichier dans lequel vous travaillez est nommé `Untitled(1)` ou éventuellement `sanstitre1`. Enregistrez le fichier sous le nom que vous voulez dans le fichier que vous voulez. Les feuilles de travail ont l'extension `.mws`. Lorsque vous travaillez avec Maple, **PENSEZ A ENREGISTRER REGULIEREMENT VOTRE PROJET.**

Quelques calculs.

Commencez par insérer l'invite de commande `>` dans la feuille de travail en utilisant le bouton correspondant. Commencez par **l'instruction** suivante :
`>(36+54/2)*3;`
Tapez la touche entrée à la fin pour éxécuter. On notera l'utilisation du `;` pour signifier la fin de l'instruction. Les symboles `+, - , /, *` correspondent aux opérations de base.

Exercice 1 *Calculer les choses suivantes à l'aide de Maple.*
$36 \times 57 + 8$, $12 - 45$, $\frac{4}{47} + \frac{3}{38}$, 3^{500}, $\sqrt{32}$, $\ln(e^8)$, $e^{\frac{i\pi}{2}}$.
Les commandes à utiliser sont `**` *pour la puissance,* `sqrt` *pour la racine,* `ln` *pour le logarithme,* `exp` *pour l'exponentielle,* `I` *pour le nombre complexe i tel que $i^2 = -1$ et* `Pi` *pour π.*

Pour savoir comment fonctionne précisément une commande vous pouvez utiliser **l'aide** en tapant la commande précédée d'un point d'interrogation, par exemple
`>?sqrt`.
Dans l'aide, il est souvent plus utile de regarder les exemples que de lire les explications.
On peut remplacer le `;` à la fin de l'instruction par un `:` Dans ce cas, le résultat du calcul n'est pas affiché.
ATTENTION ! Maple fait la différence entre les majuscules et les minuscules. Les commande `Pi` et `pi` désignent deux choses différentes.

Les types d'objet en Maple.

Dans l'exercice précédent, on a remarqué que Maple est capable de calculer avec des symboles comme $\sqrt{2}$ et de renvoyer un résultat exact sans les remplacer par leurs valeurs approchées. Néanmoins, il peut parfois être utile de pouvoir travailler avec des valeurs approchées.
Entrez les instructions
`>4/7;`
`>4.0/7.0;`
Le premier nombre est une fraction, le deuxième est un nombre décimal qui est une valeur approchée de cette fraction. En ayant utilisé 4.0 à la place de 4, Maple a compris qu'il fallait faire un calcul approché.
Maple fait bien la différence entre ces deux objets : la commande `whattype` permet de connaître **le type** d'un objet.
Exécutez les commandes suivantes :
`>whattype(4/7);`
`>whattype(4);`
`>whattype(4.0/7.0);`
Il est parfois utile de **remplacer un nombre par sa valeur approchée**, pour cela on utilise la commande `evalf` :
`>evalf(sqrt(3));`
Il existe beaucoup d'autres types d'objets en Maple.

Les variables et l'affectation.

Lorsqu'on effectue différents calculs, il est parfois utile de récupérer un résultat pour l'utiliser plus loin. La commande % permet de **rappeler le résultat précédent** :
```
>48*35+2;
>sqrt(%)/12;
```
Néanmoins, pour faire un calcul, on a souvent besoin d'autres valeurs que celles que l'on vient juste de calculer. Pour remédier à cela, on peut introduire des **variables**. Certaines sont déja définies dans Maple, par exemple `Pi`. Les autres sont définies au moment où elles sont affectées. On utilise **la commande d'affectation** `:=`
```
>a:=7;
>a;
```
Pour rendre la variable à nouveau formelle, on peut utiliser la commande suivante : `>a:='a';`
`>a;` Il est alors possible de faire des calculs à l'aide de ces variables :
`>b:=58; c:=97; >b*c;` On peut mettre plusieurs instructions sur la même ligne à condition de les séparer par un `;` ou par un `:` L'instruction `restart` remet tout à zéro, un peu comme si on ouvrait Maple, en particulier les variables affectées deviennent à nouveau formelles. Usuellement, on utilise `restart` au début de chaque feuille de travail Maple.
```
restart;
b; c;
```

Exercice 2 *Affecter la valeur* 36 *à la variable a,* $\sqrt{2}$ *à la variable b, affecter à c la somme des variables a et b.*

Exercice 3 *Affecter deux valeurs de votre choix à a et b. Echanger les valeurs de a et b.*

Il est également possible d'effectuer des calculs avec des variables formelles :
```
>polynome:=(3*c+5)*(4*c-2)+6;
>simplify(%);
```
La commande `simplify`, comme son nom l'indique **simplifie une expression**. `polynome` est tout simplement un nom de variable comme `a, b, c` ;

Exercice 4 *Calculer les expressions suivantes en simplifiant le résultat :*
$\cos^2(x)+\sin^2(x)$, $\frac{3x^2-5x+2}{x-1}$, $e^{x+ln(x)}$, $\cosh^2(x)-\sinh^2(x)$.

Listes - Séquences.

En associant des objets de type simple, on peut construire des objets plus structurés. Une **séquence** est une suite d'objets quelconques séparés par des virgules :
```
>S:=0, 1, x, Pi+1, 3*y**3+7;
>whattype(S);
```
La séquence précédente comporte cinq objets, il est possible de créer une séquence avec 0 objet en utilisant la commande `NULL` :
```
>T:=NULL;
```
Il est souvent utile **d'ajouter des éléments à une séquence**, ou de **concaténer deux séquences** :
```
>T:=T,x;
>T:=y,T;
>U:=S,T;
```
La commande `seq` est utile pour construire des séquences régulières :
```
>seq(2*i,i=1..8);
```

Exercice 5 *Construire les séquences suivantes :*
$x, z, a, 0$ - $1, 11, 21, 1211, 111221, 312211, 13112221$ - $\pi, 2\pi, 3\pi, 4\pi, 5\pi$ - $2, 5, 8, 11, 14, \ldots, 89, 92$
$a+1, a+2, \ldots a+10, b+2, b+4, \ldots, b+20$
$1, 2, 3, \ldots, 10, 2, 4, 6, \ldots, 20, 3, 6, 9, \ldots, 30, 4, \ldots, \ldots, 81, 90, 10, 20, \ldots, 80, 90, 100.$

Une **liste** est une séquence d'objets encadrée par des crochets.
```
>L:=[x, y, 1, 8, 31.5];
>whattype(L);
```
La différence est essentielle pour Maple, qui comprend la liste comme un unique objet.
On peut connaître **le nombre d'éléments d'une liste** en utilisant la commande `nops` :
```
>nops(L);
```
On peut **accéder et modifier les éléments d'une liste** de la façon suivante :

```
>L[2];
>L[2]:=0; L;
```
On peut **tranformer une liste en séquence** de la facon suivante :
```
>S:=op(L);
```
Pour **transformer une séquence en liste**, il suffit de remettre des crochets :
```
>M:=[S];
```

Exercice 6 *Construire une liste qui contient les nombres impairs entre* 0 *et* 100.

Programmation.

Comme tous les langages de programmation, Maple dispose des instructions fondamentales `if, for, while` auxquelles correspond une syntaxe particulière qu'il faudra assimiler.

Procédures

Une des possibilités majeures de Maple est de pouvoir créer des programmes ou **procédures** utilisant ces instructions. Une procédure permet d'éxécuter un certains nombres d'instructions ou commandes à l'aide d'une seule commande qu'on aura programmée. Une procédure se présente de la façon suivante :

```
> <nom>:=proc(<arg1>, <arg2>, ... )
     local <var1>, <var2>, ... ;
    <instruction1>;
    <instruction2>;
    ...
    return <resultat>;
  end proc;
```

Les crochets désignent des noms d'objets qu'il faudra remplacer.
- `<nom>` est le **nom de la procédure**.

- `<arg1>, <arg2>, ...` sont les **arguments de la procédure**, dont la procédure aura besoin pour fonctionner. LES ARGUMENTS NE PEUVENT PAS ETRE MODIFIES A L'INTERIEUR DE LA PROCEDURE.

- `<var1>, <var2>, ...` sont des **variables locales** que la procédure utilise et que l'on doit déclarer.

- `<instruction1>, <instruction2>, ...` sont des instructions que la procédure éxécute.

- l'instruction `return` permet de renvoyer un résultat en fin de procédure en l'occurrence celui qui se trouve dans la variable `<resultat>`.

- `end proc` permet de terminer la procédure.

Voici un premier exemple :

```
> division:=proc(a, b)
     local c;
     c:=evalf(a/b);
     return c;
  end proc;
> division(10,13);
```

Exercice 7 *Taper la procédure précédente, pour cela il faut passer à la ligne sans nouvel invite de commande* `>`. *Il faut utiliser la combinaison de touches Shift+Entrée. Que fait la procédure ? Quels sont les arguments ? Quelles sont les variables locales ?*

L'exemple précédent n'est pas vraiment révélateur de ce qu'on peut faire avec une procédure car il est nettement plus facile de réaliser une division directement.
Pour utiliser la procédure une fois écrite, il suffit de taper son nom et de lui donner les arguments.

Exercice 8 *Créer une procédure* `toto` *qui prend en argument un entier* n *et qui renvoie une liste des entiers de* 1 *à* n.

L'instruction conditionnelle `if`

La syntaxe est la suivante :

```
if   <condition>
   then    <instruction1>;
           <instruction2>;
            ...;
   [else   <instructionA>;
           <instructionB>;
            ...;]
fi;
```

Si la condition est vraie, les instructions `<instruction1>, <instruction2>, ...` sont éxécutées, sinon ce sont les instructions `<instructionA>, <instructionB>, ....` L'éxécution du programme reprend ensuite. La partie du programme concernant la commande `else` a été mise entre crochets car elle est optionnelle (Eventuellement, on ne fait rien si la condition n'est pas vérifiée). On termine l'instruction par un `fi;` ou par `end if;`. La condition est un nouveau type d'objet, il s'agit d'un **booléen**. En résumé il s'agit d'une variable qui est vraie ou fausse, la commande `evalb` permet de connaître la valeur du booléen.

```
>u:=1;
>evalb(u>0); evalb(u>2);
```

Les symboles `<, >, >=,<=, <>, =` sont utilisés pour les comparateurs usuels (inférieur, supérieur, supérieur ou égal, ...).
ATTENTION. IL NE FAUT PAS CONFONDRE le symbole `=` qui teste l'égalité entre deux valeurs avec le symbole d'affectation `:=`. Il s'agit d'une source d'erreur fréquente !

Exercice 9 *Créer une procédure qui prend en argument un nombre a ,qui renvoie* -1 *si ce nombre est négatif et* 1 *s'il est positif.*

La boucle `for`

La syntaxe est la suivante :

```
for <compteur> from <début> to <fin> [by <pas>]
  do <instruction1>;
     <instruction2>;
     ...
  od;
```

Le programme éxécute les `<instruction1>, <instruction2> ...` puis revient au début de la boucle et incrémente le `<compteur>` en fonction du `<pas>`. Par défaut, le `<pas>` est égal à 1. La boucle est parcourue autant de fois que le compteur en a besoin pour aller de `<début>` à`<fin>`. Le `<compteur>` est un nom de variable qui pourra être utilisé dans les instrutions présentes dans la boucle.

Exercice 10 *Créer un programme prenant en arguments deux nombres entiers a et b, calculant la somme des carrés de tous les nombres entiers compris entre a et b.*
Vérifier sur des exemples que $\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$.

La boucle `while`

La boucle `while` est très similaire à la boucle `for`. Alors que la boucle `for` parcourait des instructions un nombre de fois prédéfinies à l'avance, la boucle `while` parcourt des instructions tant qu'une condition est vérifiée.

```
while <condition>
  do <instruction1>;
     <instruction2>;
     ...
  od;
```

Exercice 11 *Réécrire le programme d'avant en utilisant une boucle* `while`.

Exercice 12 *Ecrire un programme prenant comme argument un entier n et renvoyant une liste comportant les termes de la suite de Fibonacci inférieurs à n avec* $u_0 = u_1 = 1$.

Remarques

1. Lorsque vous programmez en Maple ou avec n'importe quel autre langage, bien que cela ne soit pas nécessaire, il est quasiment VITAL surtout pour des longues procédures de respecter les indentations dans les instructions de programmation `for`, `if` et `while`. Il faut également éviter de mettre trop d'instructions sur la même ligne. De manière générale tout ce qui permet d'améliorer la lisibilté d'un programme (sauts de lignes ...) est le bienvenu.

2. Lorsque un `#` est dans une ligne de commande, Maple ignorera tous ce qui est situé après. Cette commande sert à écrire des **commentaires**. Les commentaires sont très utiles, surtout si votre programme est lu par quelqu'un d'autre.

3. La commande `return` peut être utilisée à n'importe quel moment du programme (même au milieu d'une boucle), dans ce cas, Maple quitte le programme et renvoie le résultat. Elle peut par exemple être utilisé avec l'instruction `if` en testant si les données du programmes sont bien valides pour ce qu'on veut faire.

   Exercice 13 *Modifier le programme qui calcule la somme des carrés des entiers de a et b en renvoyant la différence $b-a$ si elle est négative.*

   Si le programme ne comporte pas de commande `return`, Maple renvoie le résultat de la dernière instruction effectuée. Il est toutefois préférable de mettre une commande `return` à la fin d'une procédure pour améliorer la lisibilité.

4. Il est parfois utile d'imbriquer plusieurs boucles.
5. Les commandes `for`, `if` et `while` peuvent également être utilisées en dehors des procédures :

```
> a:=1;
> for i from 1 to 20
      do  a:=a*i;
      od;
> a;
```

   Pour éviter que Maple affiche tous les résultats intermédiaires, on peut remplacer le `;` par un `:` dans l'instruction principale, c'est à dire `a:=a*i;` par `a:=a*i:`.

6. Il arrivera qu'on ait besoin de tester davantage de conditions pour `if`. Pour cela on peut utiliser l'instruction conditionnelle `elif`. qui pourrait se traduire par "sinon si".

```
>  if a=1
     then b:=2;
   elif a=2
     then b:=7;
   elif a=3
     then b:=55;
   elif a=4
     then b:=-9;
   fi;
```

7. Parfois, il n'est pas toujours possible (ou facile) d'exprimer une condition à l'aide d'une seule égalité. Pour cela on peut utiliser les instructions `and` et `or`. Imaginons que l'on veuille éxécuter une instruction tant que la variable a égale à 2 ou à 3 on utilisera la commande
   `> while a=2 or a=3.`
   Dans cet exemple, on peut aussi s'en sortir sans ces commandes en combinant les deux conditions en une seule :
   `> while (a-2)*(a-3)=0.`

Erreurs.

Messages roses.

Lorsque certaines instructions sont mal tapées, il arrivera que Maple vous renvoie un message d'erreur écrit en rose soit lors de la compilation de la procédure, soit lors de son éxécution. Les messages les plus fréquents sont les suivantes :

1. `(in rtable/Sum) invalid arguments`
   Il s'agit en général d'une commande qui est mal utilisée. (Elle ne reçoit pas en entrée le bon nombre de variables ou alors les variables ne sont pas du bon type).
2. `missing operator or ;`
   Il s'agit en général d'un point virgule oublié.
3. `reserved word ... unexpected`
   où les pointillés sont remplacés par une des commandes de programmation. Il s'agit en général d'une instruction `if`, `for` ou `while` qui est mal syntaxifiée. Ca peut par exemple être un `od` ou un `fi` oublié.

4. `Invalid subscript selector`
   Vous cherchez à modifier où à accéder à l'élément numéro k d'une liste ou k n'est pas compris entre 1 et le nombre d'éléments de la liste. En général, c'est une boucle qui est mal initialisée et Maple se retrouve à essayer de calculer des choses du style `L[0]` ou `L[-1]`.

5. `Illegal use of a formal parameter`
   Vous tentez de modifier les arguments d'une procédure, ce que Maple n'autorise pas. Il suffit en général de passer par une variable auxiliaire.

6. `attempting to assign to ... which is protected`
   Vous cherchez à affecter une variable qui est protegée par Maple. Par exemple `D:=1;` vous renvoie ce message d'erreur car D est protegée. (C'est une commande de dérivation).

La liste précédente est loin d'être exhaustive.

La fonction `trace`

Il existe beaucoup d'autres types d'erreurs. Certaines erreurs ne sont pas des erreurs de syntaxe et ne sont pas signalés par Maple, néanmoins, les résultats renvoyées par Maple vous permettent de dire qu'il y a forcément un problème. Pour une procédure la commande **trace** appliquée au nom de la procédure fait en sorte que **lors de son éxécution tous les résultats intermédiaires sont affichés**. Elle pourra vous aider à savoir où est-ce que votre programme ne fait pas ce que vous espérez. **Pour revenir à l'affichage normal** lors de l'éxécution, il faut utiliser **untrace** appliqué au nom du programme.
Vous pouvez aussi bien sûr utiliser la commande `trace` pour trouver l'endroit où se situe une erreur (avec message rose) si elle a lieu lors de l'éxécution d'une procédure (mais pas lors de la compilation).

Procédures interminables

Lorsque vous lancez une procédure, il arrive que celle-ci mette un temps anormalement trop long (ou ne s'arrête plus). Dans ce cas le bouton "interrompre l'opération actuelle" dans la fenêtre du haut (la main) vous permettra d'arrêter l'éxécution du programme. Il arrivera que Maple "plante". dans ce cas vous devrez fermer et réouvrir Maple d'où l'interêt **d'enregistrer les programmes avant leur éxécution**. Les longueurs de Maple peuvent venir de plusieurs choses, voici les principales :

1. Vous demandez à Maple quelque chose de trop long. C'est a vous de juger des performances de votre ordinateur mais vous ne pourrez en général pas parcourir une boucle `for` 100000 fois. Faites également attention lorsque vous imbriquez des boucles `for`.

2. Il arrive que la condition de sortie d'une boucle `while` ne soit jamais remplie. Dans ce cas la fonction `trace` vous permettra de savoir où est ce qu'il boucle indéfiniment et de corriger l'erreur.

3. Il peut arriver que vous oubliez d'affecter une valeur à une variable en début de procédure , dans ce cas Maple cherche à effectuer des calculs littéraux, ce qui est parfois beaucoup plus long que lorsque l'on attribue une valeur à la variable en question.

4. De manière similaire, les calculs avec les valeurs exactes sont parfois beaucoup trop coûteux en mémoire et il est parfois nécéssaire de demander à Maple de travailler uniquement avec des valeurs approchées.

Dans tout les cas, la fonction `trace` permet très souvent de détecter efficacement où se situent les problèmes.