Chap. 0 : Présentation générale de Maple

Laurent Poinsot

12 février 2009

Plan

Plan

Plan

Plan

Plan

Ce chapitre est une présentation générale de ce qu'est le logiciel Maple que vous allez utiliser pendant cette année. Dans l'ensemble, il s'adresse plutôt aux personnes ne connaissant que peu (ou pas) Maple. Notons dès à présent qu'il est très difficile si ce n'est impossible de présenter Maple dans son ensemble, mais ce logiciel disposant d'un outil d'aide en ligne très performant, vous pourrez vous-même l'explorer en fonction de vos besoins.

Ce chapitre est une présentation générale de ce qu'est le logiciel Maple que vous allez utiliser pendant cette année. Dans l'ensemble, il s'adresse plutôt aux personnes ne connaissant que peu (ou pas) Maple. Notons dès à présent qu'il est très difficile si ce n'est impossible de présenter Maple dans son ensemble, mais ce logiciel disposant d'un outil d'aide en ligne très performant, vous pourrez vous-même l'explorer en fonction de vos besoins.

Ce chapitre est une présentation générale de ce qu'est le logiciel Maple que vous allez utiliser pendant cette année. Dans l'ensemble, il s'adresse plutôt aux personnes ne connaissant que peu (ou pas) Maple. Notons dès à présent qu'il est très difficile si ce n'est impossible de présenter Maple dans son ensemble, mais ce logiciel disposant d'un outil d'aide en ligne très performant, vous pourrez vous-même l'explorer en fonction de vos besoins.

Maple est un système de calcul formel.

Mais qu'est-ce, au juste, un "système de calcul formel" (ou "système de calcul symbolique") ?

À la différence d'un système numérique (comme les calculatrices classiques) qui ne peut manipuler que des expressions numériques, un système de calcul formel peut aussi manipuler des expressions symboliques, c'est-à-dire des expressions ne comportant pas de paramètres numériques.

Maple est un système de calcul formel.
Mais qu'est-ce, au juste, un "système de calcul formel" (ou "système de calcul symbolique") ?
À la différence d'un système numérique (comme les calculatrices classiques) qui ne peut manipuler que des expressions numériques, un système de calcul formel peut aussi manipuler des expressions symboliques, c'est-à-dire des expressions ne comportant pas de paramètres numériques.

Maple est un système de calcul formel.
Mais qu'est-ce, au juste, un "système de calcul formel" (ou "système de calcul symbolique") ?
À la différence d'un système numérique (comme les calculatrices classiques) qui ne peut manipuler que des expressions numériques, un système de calcul formel peut aussi manipuler des expressions symboliques, c'est-à-dire des expressions ne comportant pas de paramètres numériques.

On peut par exemple factoriser une expression comme $a^2 - b^2$. Si la factorisation de cette expression n'est pas un résultat qui semble extraordinnaire, Maple peut en revanche produire des résultats plus intéressants.

Par exemple, la résolution d'une équation différentielle homogène du premier ordre est des plus simples (nous reviendrons dans les chapitres suivants sur les commandes employées, le but étant ici simplement de montrer les possibilités offertes par l'outil) :

```
> dsolve({a*diff(y(x),x)+b*y(x)=0,y(0)=C},{y(x)})
```

$$y(x) = Ce^{(-\frac{bx}{a})}$$

On peut par exemple factoriser une expression comme $a^2 - b^2$. Si la factorisation de cette expression n'est pas un résultat qui semble extraordinnaire, Maple peut en revanche produire des résultats plus intéressants.
Par exemple, la résolution d'une équation différentielle homogène du premier ordre est des plus simples (nous reviendrons dans les chapitres suivants sur les commandes employées, le but étant ici simplement de montrer les possibilités offertes par l'outil) :

```
> dsolve({a*diff(y(x),x)+b*y(x)=0,y(0)=C},{y(x)})
```

$$y(x) = Ce^{(-\frac{bx}{a})}$$

On peut par exemple factoriser une expression comme $a^2 - b^2$. Si la factorisation de cette expression n'est pas un résultat qui semble extraordinnaire, Maple peut en revanche produire des résultats plus intéressants.
Par exemple, la résolution d'une équation différentielle homogène du premier ordre est des plus simples (nous reviendrons dans les chapitres suivants sur les commandes employées, le but étant ici simplement de montrer les possibilités offertes par l'outil) :

```
> dsolve({a*diff(y(x),x)+b*y(x)=0,y(0)=C},{y(x)})
```

$$y(x) = Ce^{(-\frac{bx}{a})}$$

La traduction de cette commande est la suivante :

- `diff(y(x),x)` correspond à $y'(x)$;
- `dsolve({...},{y(x)})` est la commande pour résoudre l'équation différentielle (donnée entre accolades et ici sympbolisée par `...`) par raport à la fonction inconnue y et qui a pour condition initiale (dans ce cas précis !) `y(0)=C`.

La traduction de cette commande est la suivante :

- `diff(y(x),x)` correspond à $y'(x)$;
- `dsolve({...},{y(x)})` est la commande pour résoudre l'équation différentielle (donnée entre accolades et ici sympbolisée par `...`) par raport à la fonction inconnue y et qui a pour condition initiale (dans ce cas précis !) `y(0)=C`.

La traduction de cette commande est la suivante :

- `diff(y(x),x)` correspond à $y'(x)$;
- `dsolve({...},{y(x)})` est la commande pour résoudre l'équation différentielle (donnée entre accolades et ici sympbolisée par ...) par raport à la fonction inconnue y et qui a pour condition initiale (dans ce cas précis !) `y(0)=C`.

Notons au passage que Maple adopte une forme dite *pretty print* (qui signifie "joli affichage") : il affiche ses résultats sous la forme mathémtique usuelle. De nombreuses fonctions Maple disposent de formes dites **inertes**, c'est-à-dire qu'au lieu de l'effectuer, Maple se contente d'afficher le calcul à réaliser. C'est ce que l'on verra pour les limites, les dérivées, les intégrales, *etc*.

Notons au passage que Maple adopte une forme dite *pretty print* (qui signifie "joli affichage") : il affiche ses résultats sous la forme mathémtique usuelle. De nombreuses fonctions Maple disposent de formes dites **inertes**, c'est-à-dire qu'au lieu de l'effectuer, Maple se contente d'afficher le calcul à réaliser. C'est ce que l'on verra pour les limites, les dérivées, les intégrales, *etc*.

Les objectifs de ce cours sont les suivants :

- Maîtriser un logiciel de calcul formel, en l'espèce Maple ;
- Résoudre des problèmes mathématiques à l'aide de Maple ;
- Programmer des algorithmes complexes.

Les objectifs de ce cours sont les suivants :

- Maîtriser un logiciel de calcul formel, en l'espèce Maple ;
- Résoudre des problèmes mathématiques à l'aide de Maple ;
- Programmer des algorithmes complexes.

Les objectifs de ce cours sont les suivants :

- Maîtriser un logiciel de calcul formel, en l'espèce Maple ;
- Résoudre des problèmes mathématiques à l'aide de Maple ;
- Programmer des algorithmes complexes.

Les objectifs de ce cours sont les suivants :

- Maîtriser un logiciel de calcul formel, en l'espèce Maple ;
- Résoudre des problèmes mathématiques à l'aide de Maple ;
- Programmer des algorithmes complexes.

Il est impossible de détailler ici le fonctionnement sous tous les environnements (il existe des versions de Maple disponibles sous Windows, MacOS, Linux, Unix). Pour lancer Maple :

- Sous Windows : il suffit de cliquer sur l'icône Maple ajoutée, lors de l'installation du logiciel, à la liste des programmes ;
- Sous Linux : il faut taper la commande *xmaple*.

Une fois le logiciel lancé, vous vous retrouvez face à une interface classique. Au premier abord, Maple peut sembler austère : vous vous retrouvez face à un prompt (>), après lequel il vous faudra entrer les commandes que le logiciel exécutera. Il y a aussi des icônes, bien entendu, mais la majeure partie du travail effectué sous Maple est faîte en tapant les instructions ! Attardons-nous sur les instructions à entrer.

Il est impossible de détailler ici le fonctionnement sous tous les environnements (il existe des versions de Maple disponibles sous Windows, MacOS, Linux, Unix). Pour lancer Maple :

- Sous Windows : il suffit de cliquer sur l'icône Maple ajoutée, lors de l'installation du logiciel, à la liste des programmes ;
- Sous Linux : il faut taper la commande *xmaple*.

Une fois le logiciel lancé, vous vous retrouvez face à une interface classique. Au premier abord, Maple peut sembler austère : vous vous retrouvez face à un prompt (>), après lequel il vous faudra entrer les commandes que le logiciel exécutera. Il y a aussi des icônes, bien entendu, mais la majeure partie du travail effectué sous Maple est faîte en tapant les instructions ! Attardons-nous sur les instructions à entrer.

Il est impossible de détailler ici le fonctionnement sous tous les environnements (il existe des versions de Maple disponibles sous Windows, MacOS, Linux, Unix). Pour lancer Maple :

- Sous Windows : il suffit de cliquer sur l'icône Maple ajoutée, lors de l'installation du logiciel, à la liste des programmes ;
- Sous Linux : il faut taper la commande *xmaple*.

Une fois le logiciel lancé, vous vous retrouvez face à une interface classique. Au premier abord, Maple peut sembler austère : vous vous retrouvez face à un prompt (>), après lequel il vous faudra entrer les commandes que le logiciel exécutera. Il y a aussi des icônes, bien entendu, mais la majeure partie du travail effectué sous Maple est faîte en tapant les instructions ! Attardons-nous sur les instructions à entrer.

Il est impossible de détailler ici le fonctionnement sous tous les environnements (il existe des versions de Maple disponibles sous Windows, MacOS, Linux, Unix). Pour lancer Maple :

- Sous Windows : il suffit de cliquer sur l'icône Maple ajoutée, lors de l'installation du logiciel, à la liste des programmes ;
- Sous Linux : il faut taper la commande *xmaple*.

Une fois le logiciel lancé, vous vous retrouvez face à une interface classique. Au premier abord, Maple peut sembler austère : vous vous retrouvez face à un prompt (>), après lequel il vous faudra entrer les commandes que le logiciel exécutera. Il y a aussi des icônes, bien entendu, mais la majeure partie du travail effectué sous Maple est faîte en tapant les instructions ! Attardons-nous sur les instructions à entrer.

Il est impossible de détailler ici le fonctionnement sous tous les environnements (il existe des versions de Maple disponibles sous Windows, MacOS, Linux, Unix). Pour lancer Maple :

- Sous Windows : il suffit de cliquer sur l'icône Maple ajoutée, lors de l'installation du logiciel, à la liste des programmes ;
- Sous Linux : il faut taper la commande *xmaple*.

Une fois le logiciel lancé, vous vous retrouvez face à une interface classique. Au premier abord, Maple peut sembler austère : vous vous retrouvez face à un prompt (>), après lequel il vous faudra entrer les commandes que le logiciel exécutera. Il y a aussi des icônes, bien entendu, mais la majeure partie du travail effectué sous Maple est faîte en tapant les instructions ! Attardons-nous sur les instructions à entrer.

Il est impossible de détailler ici le fonctionnement sous tous les environnements (il existe des versions de Maple disponibles sous Windows, MacOS, Linux, Unix). Pour lancer Maple :

- Sous Windows : il suffit de cliquer sur l'icône Maple ajoutée, lors de l'installation du logiciel, à la liste des programmes ;
- Sous Linux : il faut taper la commande *xmaple*.

Une fois le logiciel lancé, vous vous retrouvez face à une interface classique. Au premier abord, Maple peut sembler austère : vous vous retrouvez face à un prompt (>), après lequel il vous faudra entrer les commandes que le logiciel exécutera. Il y a aussi des icônes, bien entendu, mais la majeure partie du travail effectué sous Maple est faîte en tapant les instructions ! Attardons-nous sur les instructions à entrer.

Il est impossible de détailler ici le fonctionnement sous tous les environnements (il existe des versions de Maple disponibles sous Windows, MacOS, Linux, Unix). Pour lancer Maple :

- Sous Windows : il suffit de cliquer sur l'icône Maple ajoutée, lors de l'installation du logiciel, à la liste des programmes ;
- Sous Linux : il faut taper la commande *xmaple*.

Une fois le logiciel lancé, vous vous retrouvez face à une interface classique. Au premier abord, Maple peut sembler austère : vous vous retrouvez face à un prompt (>), après lequel il vous faudra entrer les commandes que le logiciel exécutera. Il y a aussi des icônes, bien entendu, mais la majeure partie du travail effectué sous Maple est faîte en tapant les instructions ! Attardons-nous sur les instructions à entrer.

Il est impossible de détailler ici le fonctionnement sous tous les environnements (il existe des versions de Maple disponibles sous Windows, MacOS, Linux, Unix). Pour lancer Maple :

- Sous Windows : il suffit de cliquer sur l'icône Maple ajoutée, lors de l'installation du logiciel, à la liste des programmes ;
- Sous Linux : il faut taper la commande *xmaple*.

Une fois le logiciel lancé, vous vous retrouvez face à une interface classique. Au premier abord, Maple peut sembler austère : vous vous retrouvez face à un prompt (>), après lequel il vous faudra entrer les commandes que le logiciel exécutera. Il y a aussi des icônes, bien entendu, mais la majeure partie du travail effectué sous Maple est faîte en tapant les instructions ! Attardons-nous sur les instructions à entrer.

Il est impossible de détailler ici le fonctionnement sous tous les environnements (il existe des versions de Maple disponibles sous Windows, MacOS, Linux, Unix). Pour lancer Maple :

- Sous Windows : il suffit de cliquer sur l'icône Maple ajoutée, lors de l'installation du logiciel, à la liste des programmes ;
- Sous Linux : il faut taper la commande *xmaple*.

Une fois le logiciel lancé, vous vous retrouvez face à une interface classique. Au premier abord, Maple peut sembler austère : vous vous retrouvez face à un prompt (>), après lequel il vous faudra entrer les commandes que le logiciel exécutera. Il y a aussi des icônes, bien entendu, mais la majeure partie du travail effectué sous Maple est faîte en tapant les instructions ! Attardons-nous sur les instructions à entrer.

Plan

Dans un premier temps, il convient de faire la différence entre les commandes (aussi appelées *instructions* ou encore *procédures*) et les fonctions `Maple`.

Les commandes peuvent ne pas posséder d'arguments (par ex., `quit,done`), alors qu'une fonction admet toujours au moins un argument (comme la fonction sinus qui devient en langage `Maple sin(x)` ou `x` est l'argument).

On entend par **argument** une expression ou une valeur qui va être transmise à la fonction. Dans le langage `Maple`, les arguments sont les expressions entrées entre parenthèses après le nom de la fonction de laquelle ils dépendent.

La syntaxe générale d'une fonction est par conséquent : `fonction(arguments)`.

Dans un premier temps, il convient de faire la différence entre les commandes (aussi appelées *instructions* ou encore *procédures*) et les fonctions `Maple`.
Les commandes peuvent ne pas posséder d'arguments (par ex., `quit,done`), alors qu'une fonction admet toujours au moins un argument (comme la fonction sinus qui devient en langage `Maple sin(x)` ou `x` est l'argument).
On entend par **argument** une expression ou une valeur qui va être transmise à la fonction. Dans le langage `Maple`, les arguments sont les expressions entrées entre parenthèses après le nom de la fonction de laquelle ils dépendent.
La syntaxe générale d'une fonction est par conséquent : `fonction(arguments)`.

Dans un premier temps, il convient de faire la différence entre les commandes (aussi appelées *instructions* ou encore *procédures*) et les fonctions `Maple`.
Les commandes peuvent ne pas posséder d'arguments (par ex., `quit,done`), alors qu'une fonction admet toujours au moins un argument (comme la fonction sinus qui devient en langage `Maple sin(x)` ou `x` est l'argument).
On entend par **argument** une expression ou une valeur qui va être transmise à la fonction. Dans le langage `Maple`, les arguments sont les expressions entrées entre parenthèses après le nom de la fonction de laquelle ils dépendent.
La syntaxe générale d'une fonction est par conséquent :
`fonction(arguments)`.

Dans un premier temps, il convient de faire la différence entre les commandes (aussi appelées *instructions* ou encore *procédures*) et les fonctions `Maple`.
Les commandes peuvent ne pas posséder d'arguments (par ex., `quit,done`), alors qu'une fonction admet toujours au moins un argument (comme la fonction sinus qui devient en langage `Maple sin(x)` ou `x` est l'argument).
On entend par **argument** une expression ou une valeur qui va être transmise à la fonction. Dans le langage `Maple`, les arguments sont les expressions entrées entre parenthèses après le nom de la fonction de laquelle ils dépendent.
La syntaxe générale d'une fonction est par conséquent : `fonction(arguments)`.

Dans un premier temps, il convient de faire la différence entre les commandes (aussi appelées *instructions* ou encore *procédures*) et les fonctions `Maple`.
Les commandes peuvent ne pas posséder d'arguments (par ex., `quit,done`), alors qu'une fonction admet toujours au moins un argument (comme la fonction sinus qui devient en langage `Maple sin(x)` ou `x` est l'argument).
On entend par **argument** une expression ou une valeur qui va être transmise à la fonction. Dans le langage `Maple`, les arguments sont les expressions entrées entre parenthèses après le nom de la fonction de laquelle ils dépendent.
La syntaxe générale d'une fonction est par conséquent : `fonction(arguments).`

Dans un premier temps, il convient de faire la différence entre les commandes (aussi appelées *instructions* ou encore *procédures*) et les fonctions `Maple`.
Les commandes peuvent ne pas posséder d'arguments (par ex., `quit,done`), alors qu'une fonction admet toujours au moins un argument (comme la fonction sinus qui devient en langage `Maple sin(x)` ou `x` est l'argument).
On entend par **argument** une expression ou une valeur qui va être transmise à la fonction. Dans le langage `Maple`, les arguments sont les expressions entrées entre parenthèses après le nom de la fonction de laquelle ils dépendent.
La syntaxe générale d'une fonction est par conséquent :
`fonction(arguments).`

Dans un premier temps, il convient de faire la différence entre les commandes (aussi appelées *instructions* ou encore *procédures*) et les fonctions `Maple`.
Les commandes peuvent ne pas posséder d'arguments (par ex., `quit,done`), alors qu'une fonction admet toujours au moins un argument (comme la fonction sinus qui devient en langage `Maple sin(x)` ou `x` est l'argument).
On entend par **argument** une expression ou une valeur qui va être transmise à la fonction. Dans le langage `Maple`, les arguments sont les expressions entrées entre parenthèses après le nom de la fonction de laquelle ils dépendent.
La syntaxe générale d'une fonction est par conséquent :
`fonction(arguments).`

Plan

Essayons de faire un calcul simple :

```
> 1+2
Warning, premature end of input
```

On voit que le système, à l'issue du traitement de la commande, renvoie un message d'erreur (on identifie facilement les messages d'erreur ou d'avertissement que le système nous adresse car ils débutent soit par `warning` soit par `error`).

Dans ce cas l'erreur est dûe à l'absence d'un point-virgule final.

Toutes les instructions `Maple` se terminent par un point-virgule ou par deux points " :"

```
> 1+2;
```

Essayons de faire un calcul simple :

```
> 1+2
Warning, premature end of input
```

On voit que le système, à l'issue du traitement de la commande, renvoie un message d'erreur (on identifie facilement les messages d'erreur ou d'avertissement que le système nous adresse car ils débutent soit par `warning` soit par `error`).

Dans ce cas l'erreur est dûe à l'absence d'un point-virgule final.

Toutes les instructions `Maple` se terminent par un point-virgule ou par deux points " :"

```
> 1+2;
```

Essayons de faire un calcul simple :

```
> 1+2
Warning, premature end of input
```

On voit que le système, à l'issue du traitement de la commande, renvoie un message d'erreur (on identifie facilement les messages d'erreur ou d'avertissement que le système nous adresse car ils débutent soit par `warning` soit par `error`).

Dans ce cas l'erreur est dûe à l'absence d'un point-virgule final.

Toutes les instructions Maple se terminent par un point-virgule ou par deux points " :"

```
> 1+2;
```

Essayons de faire un calcul simple :

```
> 1+2
Warning, premature end of input
```

On voit que le système, à l'issue du traitement de la commande, renvoie un message d'erreur (on identifie facilement les messages d'erreur ou d'avertissement que le système nous adresse car ils débutent soit par `warning` soit par `error`).

Dans ce cas l'erreur est dûe à l'absence d'un point-virgule final.

Toutes les instructions `Maple` se terminent par un point-virgule ou par deux points " :"

```
> 1+2;
```

Essayons de faire un calcul simple :

```
> 1+2
Warning, premature end of input
```

On voit que le système, à l'issue du traitement de la commande, renvoie un message d'erreur (on identifie facilement les messages d'erreur ou d'avertissement que le système nous adresse car ils débutent soit par `warning` soit par `error`).

Dans ce cas l'erreur est dûe à l'absence d'un point-virgule final.

Toutes les instructions `Maple` se terminent par un point-virgule ou par deux points " :"

```
> 1+2;
```

Essayons de faire un calcul simple :

```
> 1+2
Warning, premature end of input
```

On voit que le système, à l'issue du traitement de la commande, renvoie un message d'erreur (on identifie facilement les messages d'erreur ou d'avertissement que le système nous adresse car ils débutent soit par `warning` soit par `error`).

Dans ce cas l'erreur est dûe à l'absence d'un point-virgule final.

Toutes les instructions `Maple` se terminent par un point-virgule ou par deux points " :"

```
> 1+2;
```

Essayons de faire un calcul simple :

```
> 1+2
Warning, premature end of input
```

On voit que le système, à l'issue du traitement de la commande, renvoie un message d'erreur (on identifie facilement les messages d'erreur ou d'avertissement que le système nous adresse car ils débutent soit par `warning` soit par `error`).

Dans ce cas l'erreur est dûe à l'absence d'un point-virgule final.

Toutes les instructions `Maple` se terminent par un point-virgule ou par deux points " :"

```
> 1+2;
```

Le point-virgule permet d'exécuter la commande et d'afficher le résultat. Notons tout de même que, si l'on veut simplement exécuter les commandes sans voir le résultat s'afficher, il faut remplacer le point-virgule par deux points.

```
> 1+2 :
```

calcule le résultat, sans que ce dernier soit affiché. Bien que ce ne soit pas d'une grande utilité dans le cas présent, cela s'avère parfois très intéressant.

Le point-virgule permet d'exécuter la commande et d'afficher le résultat. Notons tout de même que, si l'on veut simplement exécuter les commandes sans voir le résultat s'afficher, il faut remplacer le point-virgule par deux points.

```
> 1+2 :
```

calcule le résultat, sans que ce dernier soit affiché. Bien que ce ne soit pas d'une grande utilité dans le cas présent, cela s'avère parfois très intéressant.

Le point-virgule permet d'exécuter la commande et d'afficher le résultat. Notons tout de même que, si l'on veut simplement exécuter les commandes sans voir le résultat s'afficher, il faut remplacer le point-virgule par deux points.

```
> 1+2 :
```

calcule le résultat, sans que ce dernier soit affiché. Bien que ce ne soit pas d'une grande utilité dans le cas présent, cela s'avère parfois très intéressant.

On a déjà dit que ">" représente le **prompt**. Plusieurs instructions peuvent être regroupées sous un même prompt : elles forment un **bloc d'instructions**. Les instructions formant un bloc seront exécutées à l'aide d'une seule validation (par la touche Entrée). Pour entrer plusieurs commandes dans le même bloc d'instructions, il suffit de taper Maj puis Entrée.

On a déjà dit que ">" représente le **prompt**. Plusieurs instructions peuvent être regroupées sous un même prompt : elles forment un **bloc d'instructions**. Les instructions formant un bloc seront exécutées à l'aide d'une seule validation (par la touche Entrée). Pour entrer plusieurs commandes dans le même bloc d'instructions, il suffit de taper Maj puis Entrée.

On a déjà dit que ">" représente le **prompt**. Plusieurs instructions peuvent être regroupées sous un même prompt : elles forment un **bloc d'instructions**. Les instructions formant un bloc seront exécutées à l'aide d'une seule validation (par la touche `Entrée`). Pour entrer plusieurs commandes dans le même bloc d'instructions, il suffit de taper `Maj` puis `Entrée`.

On a déjà dit que ">" représente le **prompt**. Plusieurs instructions peuvent être regroupées sous un même prompt : elles forment un **bloc d'instructions**. Les instructions formant un bloc seront exécutées à l'aide d'une seule validation (par la touche `Entrée`). Pour entrer plusieurs commandes dans le même bloc d'instructions, il suffit de taper `Maj` puis `Entrée`.

On a déjà dit que ">" représente le **prompt**. Plusieurs instructions peuvent être regroupées sous un même prompt : elles forment un **bloc d'instructions**. Les instructions formant un bloc seront exécutées à l'aide d'une seule validation (par la touche `Entrée`). Pour entrer plusieurs commandes dans le même bloc d'instructions, il suffit de taper `Maj` puis `Entrée`.

Il faut bien noter que Maple respecte la casse des lettres c'est-à-dire fait la différence entre majuscules et minuscules. Les fonctions et autres commandes doivent être écrites avec attention, comme le montre l'exemple suivant :

```
> sin(0);
```

$$0$$

```
> Sin(0);
```

$$Sin(0)$$

Maple ne reconnaît pas la fonction sinus lorsque l'on remplace le *s* minuscule par une majuscule.

Il faut bien noter que Maple respecte la casse des lettres c'est-à-dire fait la différence entre majuscules et minuscules. Les fonctions et autres commandes doivent être écrites avec attention, comme le montre l'exemple suivant :

```
> sin(0);
```

$$0$$

```
> Sin(0);
```

$$Sin(0)$$

Maple ne reconnaît pas la fonction sinus lorsque l'on remplace le *s* minuscule par une majuscule.

Il faut bien noter que Maple respecte la casse des lettres c'est-à-dire fait la différence entre majuscules et minuscules. Les fonctions et autres commandes doivent être écrites avec attention, comme le montre l'exemple suivant :

```
> sin(0);
```

$$0$$

```
> Sin(0);
```

$$Sin(0)$$

Maple ne reconnaît pas la fonction sinus lorsque l'on remplace le *s* minuscule par une majuscule.

Il faut bien noter que Maple respecte la casse des lettres c'est-à-dire fait la différence entre majuscules et minuscules. Les fonctions et autres commandes doivent être écrites avec attention, comme le montre l'exemple suivant :

```
> sin(0);
```

$$0$$

```
> Sin(0);
```

$$Sin(0)$$

Maple ne reconnaît pas la fonction sinus lorsque l'on remplace le *s* minuscule par une majuscule.

Il faut bien noter que Maple respecte la casse des lettres c'est-à-dire fait la différence entre majuscules et minuscules. Les fonctions et autres commandes doivent être écrites avec attention, comme le montre l'exemple suivant :

```
> sin(0);
```

$$0$$

```
> Sin(0);
```

$$Sin(0)$$

Maple ne reconnaît pas la fonction sinus lorsque l'on remplace le *s* minuscule par une majuscule.

Il faut bien noter que Maple respecte la casse des lettres c'est-à-dire fait la différence entre majuscules et minuscules. Les fonctions et autres commandes doivent être écrites avec attention, comme le montre l'exemple suivant :

```
> sin(0);
```

$$0$$

```
> Sin(0);
```

$$Sin(0)$$

Maple ne reconnaît pas la fonction sinus lorsque l'on remplace le *s* minuscule par une majuscule.

Il faut bien noter que Maple respecte la casse des lettres c'est-à-dire fait la différence entre majuscules et minuscules. Les fonctions et autres commandes doivent être écrites avec attention, comme le montre l'exemple suivant :

```
> sin(0);
```

$$0$$

```
> Sin(0);
```

$$Sin(0)$$

Maple ne reconnaît pas la fonction sinus lorsque l'on remplace le *s* minuscule par une majuscule.

Il faut bien noter que Maple respecte la casse des lettres c'est-à-dire fait la différence entre majuscules et minuscules. Les fonctions et autres commandes doivent être écrites avec attention, comme le montre l'exemple suivant :

```
> sin(0);
```

$$0$$

```
> Sin(0);
```

$$Sin(0)$$

Maple ne reconnaît pas la fonction sinus lorsque l'on remplace le *s* minuscule par une majuscule.

Plan

Certains caractères (ou signes) jouent des rôles spéciaux dans le langage Maple.

Le **pourcentage** % permet de rappeler le dernier résultat calculé, %% fait référence à l'avant-dernier résultat et %%% à l'antépénultième.

Par exemple,

```
>1+1;
```

$$2$$

```
>sin(%*x);
```

$$\sin(2x)$$

```
>cos(%%*x);
```

$$\cos(2x)$$

Certains caractères (ou signes) jouent des rôles spéciaux dans le langage Maple.
Le **pourcentage** % permet de rappeler le dernier résultat calculé, %% fait référence à l'avant-dernier résultat et %%% à l'antépénultième.
Par exemple,

```
>1+1;
```

$$2$$

```
>sin(%*x);
```

$$\sin(2x)$$

```
>cos(%%*x);
```

$$\cos(2x)$$

Certains caractères (ou signes) jouent des rôles spéciaux dans le langage Maple.
Le **pourcentage** % permet de rappeler le dernier résultat calculé, %% fait référence à l'avant-dernier résultat et %%% à l'antépénultième.
Par exemple,

```
>1+1;
```

$$2$$

```
>sin(%*x);
```

$$\sin(2x)$$

```
>cos(%%*x);
```

$$\cos(2x)$$

Certains caractères (ou signes) jouent des rôles spéciaux dans le langage Maple.
Le **pourcentage** % permet de rappeler le dernier résultat calculé, %% fait référence à l'avant-dernier résultat et %%% à l'antépénultième.
Par exemple,

```
>1+1;
```

$$2$$

```
>sin(%*x);
```

$$\sin(2x)$$

```
>cos(%%*x);
```

$$\cos(2x)$$

Certains caractères (ou signes) jouent des rôles spéciaux dans le langage Maple.
Le **pourcentage** % permet de rappeler le dernier résultat calculé, %% fait référence à l'avant-dernier résultat et %%% à l'antépénultième.
Par exemple,

```
>1+1;
```

$$2$$

```
>sin(%*x);
```

$$\sin(2x)$$

```
>cos(%%*x);
```

$$\cos(2x)$$

Certains caractères (ou signes) jouent des rôles spéciaux dans le langage Maple.
Le **pourcentage** % permet de rappeler le dernier résultat calculé, %% fait référence à l'avant-dernier résultat et %%% à l'antépénultième.
Par exemple,

```
>1+1;
```

$$2$$

```
>sin(%*x);
```

$$\sin(2x)$$

```
>cos(%%*x);
```

$$\cos(2x)$$

Certains caractères (ou signes) jouent des rôles spéciaux dans le langage Maple.
Le **pourcentage** % permet de rappeler le dernier résultat calculé, %% fait référence à l'avant-dernier résultat et %%% à l'antépénultième.
Par exemple,

```
>1+1;
```

$$2$$

```
>sin(%*x);
```

$$\sin(2x)$$

```
>cos(%%*x);
```

$$\cos(2x)$$

Certains caractères (ou signes) jouent des rôles spéciaux dans le langage Maple.
Le **pourcentage** % permet de rappeler le dernier résultat calculé, %% fait référence à l'avant-dernier résultat et %%% à l'antépénultième.
Par exemple,

```
>1+1;
```

$$2$$

```
>sin(%*x);
```

$$\sin(2x)$$

```
>cos(%%*x);
```

$$\cos(2x)$$

On peut obtenir facilement des lettres grecques en tapant leur nom directement (`alpha` pour α, `beta` pour β et ainsi de suite).

On peut obtenir facilement des lettres grecques en tapant leur nom directement (`alpha` pour α, `beta` pour β et ainsi de suite).

Pour insérer un commentaire avant ou après une instruction, il suffit de mettre le signe #, et tout ce qui se trouve après ce dièse sera ignoré par Maple. Par ex.,

```
> 1+2;# Calcule la somme de 1 et de 2
```

3

Pour insérer un commentaire avant ou après une instruction, il suffit de mettre le signe #, et tout ce qui se trouve après ce dièse sera ignoré par Maple. Par ex.,

```
> 1+2;# Calcule la somme de 1 et de 2
```

3

Pour insérer un commentaire avant ou après une instruction, il suffit de mettre le signe #, et tout ce qui se trouve après ce dièse sera ignoré par Maple. Par ex.,

```
> 1+2;# Calcule la somme de 1 et de 2
```

```
3
```

Pour insérer un commentaire avant ou après une instruction, il suffit de mettre le signe #, et tout ce qui se trouve après ce dièse sera ignoré par Maple. Par ex.,

```
> 1+2;# Calcule la somme de 1 et de 2
```

```
3
```

Pour insérer un commentaire avant ou après une instruction, il suffit de mettre le signe #, et tout ce qui se trouve après ce dièse sera ignoré par Maple. Par ex.,

```
> 1+2;# Calcule la somme de 1 et de 2
```

Pour insérer un commentaire avant ou après une instruction, il suffit de mettre le signe #, et tout ce qui se trouve après ce dièse sera ignoré par Maple. Par ex.,

```
> 1+2;# Calcule la somme de 1 et de 2
```

3

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : `+` (addition), `-` (soustraction), `*` (multiplication), `\` (division), `^` (puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : `<` (inférieur), `<=` (inférieur ou égal), `>` (supérieur), `>=` (supérieur ou égal), `=` (égalité), `<>` (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : $+$ (addition), $-$ (soustraction), $*$ (multiplication), $\backslash$ (division), $\hat{}$ (puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : $<$ (inférieur), $<=$ (inférieur ou égal), $>$ (supérieur), $>=$ (supérieur ou égal), $=$ (égalité), $<>$ (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : $+$ (addition), $-$ (soustraction), $*$ (multiplication), $\backslash$ (division), $\hat{}$ (puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : $<$ (inférieur), $<=$ (inférieur ou égal), $>$ (supérieur), $>=$ (supérieur ou égal), $=$ (égalité), $<>$ (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : $+$ (addition), $-$ (soustraction), $*$ (multiplication), $\backslash$ (division), $\hat{}$ (puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : $<$ (inférieur), $<=$ (inférieur ou égal), $>$ (supérieur), $>=$ (supérieur ou égal), $=$ (égalité), $<>$ (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : $+$ (addition), $-$ (soustraction), $*$ (multiplication), $\backslash$ (division), $\hat{}$ (puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : $<$ (inférieur), $<=$ (inférieur ou égal), $>$ (supérieur), $>=$ (supérieur ou égal), $=$ (égalité), $<>$ (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : $+$ (addition), $-$ (soustraction), $*$ (multiplication), \ (division), ^ (puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : $<$ (inférieur), $<=$ (inférieur ou égal), $>$ (supérieur), $>=$ (supérieur ou égal), $=$ (égalité), $<>$ (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : $+$ (addition), $-$ (soustraction), $*$ (multiplication), $\backslash$ (division), $\hat{}$ (puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : $<$ (inférieur), $<=$ (inférieur ou égal), $>$ (supérieur), $>=$ (supérieur ou égal), $=$ (égalité), $<>$ (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : $+$ (addition), $-$ (soustraction), $*$ (multiplication), $\backslash$ (division), ^ (puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : $<$ (inférieur), $<=$ (inférieur ou égal), $>$ (supérieur), $>=$ (supérieur ou égal), $=$ (égalité), $<>$ (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : $+$ (addition), $-$ (soustraction), $*$ (multiplication), $\backslash$ (division), ^ (puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : $<$ (inférieur), $<=$ (inférieur ou égal), $>$ (supérieur), $>=$ (supérieur ou égal), $=$ (égalité), $<>$ (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : $+$ (addition), $-$ (soustraction), $*$ (multiplication), $\backslash$ (division), ^ (puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : $<$ (inférieur), $<=$ (inférieur ou égal), $>$ (supérieur), $>=$ (supérieur ou égal), $=$ (égalité), $<>$ (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : $+$ (addition), $-$ (soustraction), $*$ (multiplication), $\backslash$ (division), ^ (puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : $<$ (inférieur), $<=$ (inférieur ou égal), $>$ (supérieur), $>=$ (supérieur ou égal), $=$ (égalité), $<>$ (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : $+$ (addition), $-$ (soustraction), $*$ (multiplication), $\backslash$ (division), ^ (puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : $<$ (inférieur), $<=$ (inférieur ou égal), $>$ (supérieur), $>=$ (supérieur ou égal), $=$ (égalité), $<>$ (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : $+$ (addition), $-$ (soustraction), $*$ (multiplication), $\backslash$ (division), ^(puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : $<$ (inférieur), $<=$ (inférieur ou égal), $>$ (supérieur), $>=$ (supérieur ou égal), $=$ (égalité), $<>$ (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : $+$ (addition), $-$ (soustraction), $*$ (multiplication), $\backslash$ (division), ^(puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : $<$ (inférieur), $<=$ (inférieur ou égal), $>$ (supérieur), $>=$ (supérieur ou égal), $=$ (égalité), $<>$ (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : $+$ (addition), $-$ (soustraction), $*$ (multiplication), $\backslash$ (division), ^ (puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : $<$ (inférieur), $<=$ (inférieur ou égal), $>$ (supérieur), $>=$ (supérieur ou égal), $=$ (égalité), $<>$ (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : $+$ (addition), $-$ (soustraction), $*$ (multiplication), $\backslash$ (division), ^ (puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : $<$ (inférieur), $<=$ (inférieur ou égal), $>$ (supérieur), $>=$ (supérieur ou égal), $=$ (égalité), $<>$ (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : $+$ (addition), $-$ (soustraction), $*$ (multiplication), $\backslash$ (division), ^ (puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : $<$ (inférieur), $<=$ (inférieur ou égal), $>$ (supérieur), $>=$ (supérieur ou égal), $=$ (égalité), $<>$ (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : $+$ (addition), $-$ (soustraction), $*$ (multiplication), $\backslash$ (division), ^ (puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : $<$ (inférieur), $<=$ (inférieur ou égal), $>$ (supérieur), $>=$ (supérieur ou égal), $=$ (égalité), $<>$ (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : $+$ (addition), $-$ (soustraction), $*$ (multiplication), $\backslash$ (division), ^ (puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : $<$ (inférieur), $<=$ (inférieur ou égal), $>$ (supérieur), $>=$ (supérieur ou égal), $=$ (égalité), $<>$ (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : $+$ (addition), $-$ (soustraction), $*$ (multiplication), $\backslash$ (division), ^ (puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : $<$ (inférieur), $<=$ (inférieur ou égal), $>$ (supérieur), $>=$ (supérieur ou égal), $=$ (égalité), $<>$ (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : $+$ (addition), $-$ (soustraction), $*$ (multiplication), $\backslash$ (division), ^ (puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : $<$ (inférieur), $<=$ (inférieur ou égal), $>$ (supérieur), $>=$ (supérieur ou égal), $=$ (égalité), $<>$ (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Dans les instructions, les **opérateurs** permettent de lier les **opérandes**. Ils sont constitués soit par des opérations mathématiques, soit par des liens logiques.

1. Opérateurs arithmétiques : $+$ (addition), $-$ (soustraction), $*$ (multiplication), $\backslash$ (division), ^ (puissance), `mod` (reste dans une division entière) ;
2. Opérateurs de comparaison : $<$ (inférieur), $<=$ (inférieur ou égal), $>$ (supérieur), $>=$ (supérieur ou égal), $=$ (égalité), $<>$ (différent de) ;
3. Opérateurs ensemblistes : `union` (réunion), `intersect` (intersection) ;
4. Opérateurs logiques : `and` (et logique), `ou` (ou logique) ;
5. Opérateur d'affectation : ":=".

Égalité vs Affectation

Il ne faut pas confondre "=" et ":=".
L'**égalité** "=" possède une valeur de vérité (vrai ou faux) : la proposition "$2 = 2$" est vraie, alors que la proposition "$2 = 3$" est fausse.
L'**affectation** permet de modifier la valeur d'une variable.
On écrit à gauche du signe d'affectation la variable et à droite sa **nouvelle** valeur comme $x := 3$.

Égalité vs Affectation

Il ne faut pas confondre "=" et ":=".

L'**égalité** "=" possède une valeur de vérité (vrai ou faux) : la proposition "$2 = 2$" est vraie, alors que la proposition "$2 = 3$" est fausse.

L'**affectation** permet de modifier la valeur d'une variable. On écrit à gauche du signe d'affectation la variable et à droite sa **nouvelle** valeur comme $x := 3$.

Égalité vs Affectation

Il ne faut pas confondre "=" et ":=".
L'**égalité** "=" possède une valeur de vérité (vrai ou faux) : la proposition "$2 = 2$" est vraie, alors que la proposition "$2 = 3$" est fausse.
L'**affectation** permet de modifier la valeur d'une variable. On écrit à gauche du signe d'affectation la variable et à droite sa **nouvelle** valeur comme $x := 3$.

Égalité vs Affectation

Il ne faut pas confondre "$=$" et "$:=$".
L'**égalité** "$=$" possède une valeur de vérité (vrai ou faux) : la proposition "$2 = 2$" est vraie, alors que la proposition "$2 = 3$" est fausse.
L'**affectation** permet de modifier la valeur d'une variable. On écrit à gauche du signe d'affectation la variable et à droite sa **nouvelle** valeur comme $x := 3$.

Égalité vs Affectation

Il ne faut pas confondre "$=$" et "$:=$".
L'**égalité** "$=$" possède une valeur de vérité (vrai ou faux) : la proposition "$2 = 2$" est vraie, alors que la proposition "$2 = 3$" est fausse.
L'**affectation** permet de modifier la valeur d'une variable.
On écrit à gauche du signe d'affectation la variable et à droite sa **nouvelle** valeur comme $x := 3$.

Égalité vs Affectation

Il ne faut pas confondre "$=$" et "$:=$".
L'**égalité** "$=$" possède une valeur de vérité (vrai ou faux) : la proposition "$2 = 2$" est vraie, alors que la proposition "$2 = 3$" est fausse.
L'**affectation** permet de modifier la valeur d'une variable. On écrit à gauche du signe d'affectation la variable et à droite sa **nouvelle** valeur comme $x := 3$.

Par exemple :

```
> x :=2; # On donne la valeur 2 à la variable x
```

$$x := 2$$

```
> evalb(x=2); # evalb permet de vérifier si la proposition donnée est vraie ou fausse : ici elle est vraie.
```

$$true$$

```
> x :=3 :
> evalb(x=2);
```

$$false$$

```
> x=2 :
> evalb(x=2);
```

$$false$$

Par exemple :

```
> x :=2; # On donne la valeur 2 à la variable x
```

$$x := 2$$

```
> evalb(x=2); # evalb permet de vérifier si la proposition donnée est vraie ou fausse : ici elle est vraie.
```

true

```
> x :=3 :
> evalb(x=2);
```

false

```
> x=2 :
> evalb(x=2);
```

false

Par exemple :

```
> x :=2; # On donne la valeur 2 à la variable x
```

$$x := 2$$

```
> evalb(x=2); # evalb permet de vérifier si la proposition donnée est vraie ou fausse : ici elle est vraie.
```

true

```
> x :=3 :
> evalb(x=2);
```

false

```
> x=2 :
> evalb(x=2);
```

false

Par exemple :

```
> x :=2; # On donne la valeur 2 à la variable x
```

$$x := 2$$

```
> evalb(x=2); # evalb permet de vérifier si la proposition donnée est vraie ou fausse : ici elle est vraie.
```

true

```
> x :=3 :
> evalb(x=2);
```

false

```
> x=2 :
> evalb(x=2);
```

false

Par exemple :

```
> x :=2; # On donne la valeur 2 à la variable x
```

$$x := 2$$

```
> evalb(x=2); # evalb permet de vérifier si la proposition donnée est vraie ou fausse : ici elle est vraie.
```

true

```
> x :=3 :
> evalb(x=2);
```

false

```
> x=2 :
> evalb(x=2);
```

false

Par exemple :

```
> x :=2; # On donne la valeur 2 à la variable x
```

$$x := 2$$

```
> evalb(x=2); # evalb permet de vérifier si la proposition donnée est vraie ou fausse : ici elle est vraie.
```

true

```
> x :=3 :
> evalb(x=2);
```

false

```
> x=2 :
> evalb(x=2);
```

false

Par exemple :

```
> x :=2; # On donne la valeur 2 à la variable x
```

$$x := 2$$

```
> evalb(x=2); # evalb permet de vérifier si la proposition donnée est vraie ou fausse : ici elle est vraie.
```

$$true$$

```
> x :=3 :
> evalb(x=2);
```

$$false$$

```
> x=2 :
> evalb(x=2);
```

$$false$$

Par exemple :

```
> x :=2; # On donne la valeur 2 à la variable x
```

$$x := 2$$

```
> evalb(x=2); # evalb permet de vérifier si la proposition donnée est vraie ou fausse : ici elle est vraie.
```

true

```
> x :=3 :
> evalb(x=2);
```

false

```
> x=2 :
> evalb(x=2);
```

false

Par exemple :

```
> x :=2; # On donne la valeur 2 à la variable x
```

$$x := 2$$

```
> evalb(x=2); # evalb permet de vérifier si la proposition donnée est vraie ou fausse : ici elle est vraie.
```

true

```
> x :=3 :
> evalb(x=2);
```

false

```
> x=2 :
> evalb(x=2);
```

false

Par exemple :

```
> x :=2; # On donne la valeur 2 à la variable x
```

$$x := 2$$

```
> evalb(x=2); # evalb permet de vérifier si la proposition donnée est vraie ou fausse : ici elle est vraie.
```

true

```
> x :=3 :
> evalb(x=2);
```

false

```
> x=2 :
> evalb(x=2);
```

false

Par exemple :

```
> x :=2; # On donne la valeur 2 à la variable x
```

$$x := 2$$

```
> evalb(x=2); # evalb permet de vérifier si la proposition donnée est vraie ou fausse : ici elle est vraie.
```

true

```
> x :=3 :
> evalb(x=2);
```

false

```
> x=2 :
> evalb(x=2);
```

false

Par exemple :

```
> x :=2; # On donne la valeur 2 à la variable x
```

$$x := 2$$

```
> evalb(x=2); # evalb permet de vérifier si la proposition donnée est vraie ou fausse : ici elle est vraie.
```

true

```
> x :=3 :
> evalb(x=2);
```

false

```
> x=2 :
> evalb(x=2);
```

false

Fonctions mathématiques disponibles sous Maple

1. Fonctions trigonométriques : `sin(x)`, `cos(x)`, `tan(x)`, `arcsin(x)`, `arcos(x)` et `arctan(x)` ;
2. Fonctions trigonométriques hyperboliques : `sinh(x)`, `cosh(x)`, `tanh(x)`, `arcsinh(x)`, `arccosh(x)`, `arctanh(x)` ;
3. Logarithme naturel : `ln(x)` ;
4. Racine carrée : `sqrt(x)` ;
5. Valeur absolue (et module complexe) : `abs(x)` ;
6. Factorielle : `n!` ;
7. Coefficients binomiaux C_n^k : `binomial(n,k)` ;
8. Quotient de la division euclidienne de `a` par `b` : `iquo(a,b)` ;
9. Reste de la division euclidienne de `a` par `b` : `irem(a,b)` ;
10. Max et min d'un ensemble : `max(E)` et `min(E)`.

Fonctions mathématiques disponibles sous Maple

1. Fonctions trigonométriques : `sin(x)`, `cos(x)`, `tan(x)`, `arcsin(x)`, `arcos(x)` et `arctan(x)` ;
2. Fonctions trigonométriques hyperboliques : `sinh(x)`, `cosh(x)`, `tanh(x)`, `arcsinh(x)`, `arccosh(x)`, `arctanh(x)` ;
3. Logarithme naturel : `ln(x)` ;
4. Racine carrée : `sqrt(x)` ;
5. Valeur absolue (et module complexe) : `abs(x)` ;
6. Factorielle : `n!` ;
7. Coefficients binomiaux C_n^k : `binomial(n,k)` ;
8. Quotient de la division euclidienne de `a` par `b` : `iquo(a,b)` ;
9. Reste de la division euclidienne de `a` par `b` : `irem(a,b)` ;
10. Max et min d'un ensemble : `max(E)` et `min(E)`.

Fonctions mathématiques disponibles sous Maple

1. Fonctions trigonométriques : `sin(x)`, `cos(x)`, `tan(x)`, `arcsin(x)`, `arcos(x)` et `arctan(x)` ;
2. Fonctions trigonométriques hyperboliques : `sinh(x)`, `cosh(x)`, `tanh(x)`, `arcsinh(x)`, `arccosh(x)`, `arctanh(x)` ;
3. Logarithme naturel : `ln(x)` ;
4. Racine carrée : `sqrt(x)` ;
5. Valeur absolue (et module complexe) : `abs(x)` ;
6. Factorielle : `n!` ;
7. Coefficients binomiaux C_n^k : `binomial(n,k)` ;
8. Quotient de la division euclidienne de `a` par `b` : `iquo(a,b)` ;
9. Reste de la division euclidienne de `a` par `b` : `irem(a,b)` ;
10. Max et min d'un ensemble : `max(E)` et `min(E)`.

Fonctions mathématiques disponibles sous Maple

1. Fonctions trigonométriques : `sin(x)`, `cos(x)`, `tan(x)`, `arcsin(x)`, `arcos(x)` et `arctan(x)` ;
2. Fonctions trigonométriques hyperboliques : `sinh(x)`, `cosh(x)`, `tanh(x)`, `arcsinh(x)`, `arccosh(x)`, `arctanh(x)` ;
3. Logarithme naturel : `ln(x)` ;
4. Racine carrée : `sqrt(x)` ;
5. Valeur absolue (et module complexe) : `abs(x)` ;
6. Factorielle : `n!` ;
7. Coefficients binomiaux C_n^k : `binomial(n,k)` ;
8. Quotient de la division euclidienne de `a` par `b` : `iquo(a,b)` ;
9. Reste de la division euclidienne de `a` par `b` : `irem(a,b)` ;
10. Max et min d'un ensemble : `max(E)` et `min(E)`.

Fonctions mathématiques disponibles sous Maple

1. Fonctions trigonométriques : `sin(x)`, `cos(x)`, `tan(x)`, `arcsin(x)`, `arcos(x)` et `arctan(x)` ;
2. Fonctions trigonométriques hyperboliques : `sinh(x)`, `cosh(x)`, `tanh(x)`, `arcsinh(x)`, `arccosh(x)`, `arctanh(x)` ;
3. Logarithme naturel : `ln(x)` ;
4. Racine carrée : `sqrt(x)` ;
5. Valeur absolue (et module complexe) : `abs(x)` ;
6. Factorielle : `n!` ;
7. Coefficients binomiaux C_n^k : `binomial(n,k)` ;
8. Quotient de la division euclidienne de `a` par `b` : `iquo(a,b)` ;
9. Reste de la division euclidienne de `a` par `b` : `irem(a,b)` ;
10. Max et min d'un ensemble : `max(E)` et `min(E)`.

Fonctions mathématiques disponibles sous Maple

1. Fonctions trigonométriques : `sin(x)`, `cos(x)`, `tan(x)`, `arcsin(x)`, `arcos(x)` et `arctan(x)` ;
2. Fonctions trigonométriques hyperboliques : `sinh(x)`, `cosh(x)`, `tanh(x)`, `arcsinh(x)`, `arccosh(x)`, `arctanh(x)` ;
3. Logarithme naturel : `ln(x)` ;
4. Racine carrée : `sqrt(x)` ;
5. Valeur absolue (et module complexe) : `abs(x)` ;
6. Factorielle : `n!` ;
7. Coefficients binomiaux C_n^k : `binomial(n,k)` ;
8. Quotient de la division euclidienne de `a` par `b` : `iquo(a,b)` ;
9. Reste de la division euclidienne de `a` par `b` : `irem(a,b)` ;
10. Max et min d'un ensemble : `max(E)` et `min(E)`.

Fonctions mathématiques disponibles sous Maple

1. Fonctions trigonométriques : `sin(x)`, `cos(x)`, `tan(x)`, `arcsin(x)`, `arcos(x)` et `arctan(x)` ;
2. Fonctions trigonométriques hyperboliques : `sinh(x)`, `cosh(x)`, `tanh(x)`, `arcsinh(x)`, `arccosh(x)`, `arctanh(x)` ;
3. Logarithme naturel : `ln(x)` ;
4. Racine carrée : `sqrt(x)` ;
5. Valeur absolue (et module complexe) : `abs(x)` ;
6. Factorielle : `n!` ;
7. Coefficients binomiaux C_n^k : `binomial(n,k)` ;
8. Quotient de la division euclidienne de `a` par `b` : `iquo(a,b)` ;
9. Reste de la division euclidienne de `a` par `b` : `irem(a,b)` ;
10. Max et min d'un ensemble : `max(E)` et `min(E)`.

Fonctions mathématiques disponibles sous Maple

1. Fonctions trigonométriques : `sin(x)`, `cos(x)`, `tan(x)`, `arcsin(x)`, `arcos(x)` et `arctan(x)` ;
2. Fonctions trigonométriques hyperboliques : `sinh(x)`, `cosh(x)`, `tanh(x)`, `arcsinh(x)`, `arccosh(x)`, `arctanh(x)` ;
3. Logarithme naturel : `ln(x)` ;
4. Racine carrée : `sqrt(x)` ;
5. Valeur absolue (et module complexe) : `abs(x)` ;
6. Factorielle : `n!` ;
7. Coefficients binomiaux C_n^k : `binomial(n,k)` ;
8. Quotient de la division euclidienne de `a` par `b` : `iquo(a,b)` ;
9. Reste de la division euclidienne de `a` par `b` : `irem(a,b)` ;
10. Max et min d'un ensemble : `max(E)` et `min(E)`.

Fonctions mathématiques disponibles sous Maple

1. Fonctions trigonométriques : `sin(x)`, `cos(x)`, `tan(x)`, `arcsin(x)`, `arcos(x)` et `arctan(x)` ;
2. Fonctions trigonométriques hyperboliques : `sinh(x)`, `cosh(x)`, `tanh(x)`, `arcsinh(x)`, `arccosh(x)`, `arctanh(x)` ;
3. Logarithme naturel : `ln(x)` ;
4. Racine carrée : `sqrt(x)` ;
5. Valeur absolue (et module complexe) : `abs(x)` ;
6. Factorielle : `n!` ;
7. Coefficients binomiaux C_n^k : `binomial(n,k)` ;
8. Quotient de la division euclidienne de `a` par `b` : `iquo(a,b)` ;
9. Reste de la division euclidienne de `a` par `b` : `irem(a,b)` ;
10. Max et min d'un ensemble : `max(E)` et `min(E)`.

Fonctions mathématiques disponibles sous Maple

1. Fonctions trigonométriques : `sin(x)`, `cos(x)`, `tan(x)`, `arcsin(x)`, `arcos(x)` et `arctan(x)` ;
2. Fonctions trigonométriques hyperboliques : `sinh(x)`, `cosh(x)`, `tanh(x)`, `arcsinh(x)`, `arccosh(x)`, `arctanh(x)` ;
3. Logarithme naturel : `ln(x)` ;
4. Racine carrée : `sqrt(x)` ;
5. Valeur absolue (et module complexe) : `abs(x)` ;
6. Factorielle : `n!` ;
7. Coefficients binomiaux C_n^k : `binomial(n,k)` ;
8. Quotient de la division euclidienne de `a` par `b` : `iquo(a,b)` ;
9. Reste de la division euclidienne de `a` par `b` : `irem(a,b)` ;
10. Max et min d'un ensemble : `max(E)` et `min(E)`.

Fonctions mathématiques disponibles sous Maple

1. Fonctions trigonométriques : `sin(x)`, `cos(x)`, `tan(x)`, `arcsin(x)`, `arcos(x)` et `arctan(x)` ;
2. Fonctions trigonométriques hyperboliques : `sinh(x)`, `cosh(x)`, `tanh(x)`, `arcsinh(x)`, `arccosh(x)`, `arctanh(x)` ;
3. Logarithme naturel : `ln(x)` ;
4. Racine carrée : `sqrt(x)` ;
5. Valeur absolue (et module complexe) : `abs(x)` ;
6. Factorielle : `n!` ;
7. Coefficients binomiaux C_n^k : `binomial(n,k)` ;
8. Quotient de la division euclidienne de `a` par `b` : `iquo(a,b)` ;
9. Reste de la division euclidienne de `a` par `b` : `irem(a,b)` ;
10. Max et min d'un ensemble : `max(E)` et `min(E)`.

Plan

En plus des fonctions par défaut, Maple dispose de fonctions supplémentaires qui appartiennent à des bibliothèques, appelées **packages**. Pour pouvoir uiliser ces fonctions, il faut charger en mémoire ces bibliothèques. Ce chargement se fait à l'aide de la commande `with` :

```
> with(package);
```

Par exemple, le chargement du package `linalg` se fera par :

```
> with(linalg);
```

[*BlockDiagonal*, *GramSchmidt*, *JordanBlock*, *LUdecomp*, . . .]

Cette méthode a un léger inconvénient : toutes les fonctions chargées sont affichées à l'écran, d'où l'intérêt d'utiliser les ":".

En plus des fonctions par défaut, Maple dispose de fonctions supplémentaires qui appartiennent à des bibliothèques, appelées **packages**. Pour pouvoir uiliser ces fonctions, il faut charger en mémoire ces bibliothèques. Ce chargement se fait à l'aide de la commande `with` :

```
> with(package);
```

Par exemple, le chargement du package `linalg` se fera par :

```
> with(linalg);
```

[*BlockDiagonal*, *GramSchmidt*, *JordanBlock*, *LUdecomp*, . . .]

Cette méthode a un léger inconvénient : toutes les fonctions chargées sont affichées à l'écran, d'où l'intérêt d'utiliser les ":".

En plus des fonctions par défaut, Maple dispose de fonctions supplémentaires qui appartiennent à des bibliothèques, appelées **packages**. Pour pouvoir uiliser ces fonctions, il faut charger en mémoire ces bibliothèques. Ce chargement se fait à l'aide de la commande `with` :

```
> with(package);
```

Par exemple, le chargement du package `linalg` se fera par :

```
> with(linalg);
```

[*BlockDiagonal*, *GramSchmidt*, *JordanBlock*, *LUdecomp*, . . .]

Cette méthode a un léger inconvénient : toutes les fonctions chargées sont affichées à l'écran, d'où l'intérêt d'utiliser les ":".

En plus des fonctions par défaut, Maple dispose de fonctions supplémentaires qui appartiennent à des bibliothèques, appelées **packages**. Pour pouvoir uiliser ces fonctions, il faut charger en mémoire ces bibliothèques. Ce chargement se fait à l'aide de la commande `with` :

```
> with(package);
```

Par exemple, le chargement du package `linalg` se fera par :

```
> with(linalg);
```

[*BlockDiagonal*, *GramSchmidt*, *JordanBlock*, *LUdecomp*, . . .]

Cette méthode a un léger inconvénient : toutes les fonctions chargées sont affichées à l'écran, d'où l'intérêt d'utiliser les ":".

En plus des fonctions par défaut, Maple dispose de fonctions supplémentaires qui appartiennent à des bibliothèques, appelées **packages**. Pour pouvoir uiliser ces fonctions, il faut charger en mémoire ces bibliothèques. Ce chargement se fait à l'aide de la commande `with` :

```
> with(package);
```

Par exemple, le chargement du package `linalg` se fera par :

```
> with(linalg);
```

[*BlockDiagonal*, *GramSchmidt*, *JordanBlock*, *LUdecomp*, . . .]

Cette méthode a un léger inconvénient : toutes les fonctions chargées sont affichées à l'écran, d'où l'intérêt d'utiliser les ":".

En plus des fonctions par défaut, Maple dispose de fonctions supplémentaires qui appartiennent à des bibliothèques, appelées **packages**. Pour pouvoir uiliser ces fonctions, il faut charger en mémoire ces bibliothèques. Ce chargement se fait à l'aide de la commande `with` :

```
> with(package);
```

Par exemple, le chargement du package `linalg` se fera par :

```
> with(linalg);
```

[*BlockDiagonal*, *GramSchmidt*, *JordanBlock*, *LUdecomp*, . . .]

Cette méthode a un léger inconvénient : toutes les fonctions chargées sont affichées à l'écran, d'où l'intérêt d'utiliser les ":".

En plus des fonctions par défaut, Maple dispose de fonctions supplémentaires qui appartiennent à des bibliothèques, appelées **packages**. Pour pouvoir uiliser ces fonctions, il faut charger en mémoire ces bibliothèques. Ce chargement se fait à l'aide de la commande `with` :

```
> with(package);
```

Par exemple, le chargement du package `linalg` se fera par :

```
> with(linalg);
```

[*BlockDiagonal*, *GramSchmidt*, *JordanBlock*, *LUdecomp*, . . .]

Cette méthode a un léger inconvénient : toutes les fonctions chargées sont affichées à l'écran, d'où l'intérêt d'utiliser les ":".

En plus des fonctions par défaut, Maple dispose de fonctions supplémentaires qui appartiennent à des bibliothèques, appelées **packages**. Pour pouvoir uiliser ces fonctions, il faut charger en mémoire ces bibliothèques. Ce chargement se fait à l'aide de la commande `with` :

```
> with(package);
```

Par exemple, le chargement du package `linalg` se fera par :

```
> with(linalg);
```

[*BlockDiagonal*, *GramSchmidt*, *JordanBlock*, *LUdecomp*, . . .]

Cette méthode a un léger inconvénient : toutes les fonctions chargées sont affichées à l'écran, d'où l'intérêt d'utiliser les ":".

En plus des fonctions par défaut, Maple dispose de fonctions supplémentaires qui appartiennent à des bibliothèques, appelées **packages**. Pour pouvoir uiliser ces fonctions, il faut charger en mémoire ces bibliothèques. Ce chargement se fait à l'aide de la commande `with` :

```
> with(package);
```

Par exemple, le chargement du package `linalg` se fera par :

```
> with(linalg);
```

[*BlockDiagonal*, *GramSchmidt*, *JordanBlock*, *LUdecomp*, . . .]

Cette méthode a un léger inconvénient : toutes les fonctions chargées sont affichées à l'écran, d'où l'intérêt d'utiliser les ":".

Plan

L'aide Maple et extrêmement importante ; en effet, aucun livre ne décrit **toutes** les fonctions qui composent le système. Quand on veut des renseignements précis sur une fonction peu connue, il faut donc utiliser l'aide Maple. Pour activer cette aide, plusieurs possiblilités s'offrent à vous :

1. Aller dans le menu Help ;
2. Taper directement sur une ligne de commande :

   ```
   ? fonction
   ```

   Cette méthode, beaucoup plus rapide que la précédente, suppose de connaître le nom de la fonction que l'on cherche.

L'aide Maple et extrêmement importante ; en effet, aucun livre ne décrit **toutes** les fonctions qui composent le système. Quand on veut des renseignements précis sur une fonction peu connue, il faut donc utiliser l'aide Maple. Pour activer cette aide, plusieurs possiblilités s'offrent à vous :

1. Aller dans le menu Help ;
2. Taper directement sur une ligne de commande :

   ```
   ? fonction
   ```

   Cette méthode, beaucoup plus rapide que la précédente, suppose de connaître le nom de la fonction que l'on cherche.

L'aide Maple et extrêmement importante ; en effet, aucun livre ne décrit **toutes** les fonctions qui composent le système. Quand on veut des renseignements précis sur une fonction peu connue, il faut donc utiliser l'aide Maple. Pour activer cette aide, plusieurs possiblilités s'offrent à vous :

1. Aller dans le menu Help ;
2. Taper directement sur une ligne de commande :

   ```
   ? fonction
   ```

   Cette méthode, beaucoup plus rapide que la précédente, suppose de connaître le nom de la fonction que l'on cherche.

L'aide Maple et extrêmement importante ; en effet, aucun livre ne décrit **toutes** les fonctions qui composent le système. Quand on veut des renseignements précis sur une fonction peu connue, il faut donc utiliser l'aide Maple. Pour activer cette aide, plusieurs possiblilités s'offrent à vous :

1. Aller dans le menu Help ;
2. Taper directement sur une ligne de commande :
   ```
   ? fonction
   ```
   Cette méthode, beaucoup plus rapide que la précédente, suppose de connaître le nom de la fonction que l'on cherche.

L'aide Maple et extrêmement importante ; en effet, aucun livre ne décrit **toutes** les fonctions qui composent le système. Quand on veut des renseignements précis sur une fonction peu connue, il faut donc utiliser l'aide Maple. Pour activer cette aide, plusieurs possiblilités s'offrent à vous :

1. Aller dans le menu Help ;
2. Taper directement sur une ligne de commande :
   ```
   ? fonction
   ```
   Cette méthode, beaucoup plus rapide que la précédente, suppose de connaître le nom de la fonction que l'on cherche.

L'aide Maple et extrêmement importante ; en effet, aucun livre ne décrit **toutes** les fonctions qui composent le système. Quand on veut des renseignements précis sur une fonction peu connue, il faut donc utiliser l'aide Maple. Pour activer cette aide, plusieurs possiblilités s'offrent à vous :

1. Aller dans le menu Help ;
2. Taper directement sur une ligne de commande :

   ```
   ? fonction
   ```

   Cette méthode, beaucoup plus rapide que la précédente, suppose de connaître le nom de la fonction que l'on cherche.

L'aide Maple et extrêmement importante ; en effet, aucun livre ne décrit **toutes** les fonctions qui composent le système. Quand on veut des renseignements précis sur une fonction peu connue, il faut donc utiliser l'aide Maple. Pour activer cette aide, plusieurs possiblilités s'offrent à vous :

1. Aller dans le menu Help ;
2. Taper directement sur une ligne de commande :

   ```
   ? fonction
   ```

   Cette méthode, beaucoup plus rapide que la précédente, suppose de connaître le nom de la fonction que l'on cherche.

L'aide Maple et extrêmement importante ; en effet, aucun livre ne décrit **toutes** les fonctions qui composent le système. Quand on veut des renseignements précis sur une fonction peu connue, il faut donc utiliser l'aide Maple. Pour activer cette aide, plusieurs possiblilités s'offrent à vous :

1. Aller dans le menu Help ;
2. Taper directement sur une ligne de commande :
   ```
   ? fonction
   ```
   Cette méthode, beaucoup plus rapide que la précédente, suppose de connaître le nom de la fonction que l'on cherche.

Plan

Une feuille d'aide Maple se décompose en quatre parties. La première partie donne la syntaxe (ou les syntaxes) d'une fonction donnée et détaille ses arguments. Ensuite, vient une description de la fonction : à quelle fonction mathématiques elle correspond. Puis une partie où figurent de nombreux exemples qui sont souvent d'une aide précieuse et qui permettent de bien saisir comment utiliser la fonction. Enfin, en général sur la dernière ligne, figure une liste des rubriques connexes auxquelles on peut accèder en cliquant sur le mot : ce sont des liens hypertextes.

Une feuille d'aide Maple se décompose en quatre parties. La première partie donne la syntaxe (ou les syntaxes) d'une fonction donnée et détaille ses arguments. Ensuite, vient une description de la fonction : à quelle fonction mathématiques elle correspond. Puis une partie où figurent de nombreux exemples qui sont souvent d'une aide précieuse et qui permettent de bien saisir comment utiliser la fonction. Enfin, en général sur la dernière ligne, figure une liste des rubriques connexes auxquelles on peut accèder en cliquant sur le mot : ce sont des liens hypertextes.

Une feuille d'aide Maple se décompose en quatre parties. La première partie donne la syntaxe (ou les syntaxes) d'une fonction donnée et détaille ses arguments. Ensuite, vient une description de la fonction : à quelle fonction mathématiques elle correspond. Puis une partie où figurent de nombreux exemples qui sont souvent d'une aide précieuse et qui permettent de bien saisir comment utiliser la fonction. Enfin, en général sur la dernière ligne, figure une liste des rubriques connexes auxquelles on peut accèder en cliquant sur le mot : ce sont des liens hypertextes.

Une feuille d'aide Maple se décompose en quatre parties. La première partie donne la syntaxe (ou les syntaxes) d'une fonction donnée et détaille ses arguments. Ensuite, vient une description de la fonction : à quelle fonction mathématiques elle correspond. Puis une partie où figurent de nombreux exemples qui sont souvent d'une aide précieuse et qui permettent de bien saisir comment utiliser la fonction. Enfin, en général sur la dernière ligne, figure une liste des rubriques connexes auxquelles on peut accèder en cliquant sur le mot : ce sont des liens hypertextes.

Une feuille d'aide Maple se décompose en quatre parties. La première partie donne la syntaxe (ou les syntaxes) d'une fonction donnée et détaille ses arguments. Ensuite, vient une description de la fonction : à quelle fonction mathématiques elle correspond. Puis une partie où figurent de nombreux exemples qui sont souvent d'une aide précieuse et qui permettent de bien saisir comment utiliser la fonction. Enfin, en général sur la dernière ligne, figure une liste des rubriques connexes auxquelles on peut accèder en cliquant sur le mot : ce sont des liens hypertextes.

Pour finir ce premier chapitre

Pour démarrer une nouvelle session `Maple`, c'est-à-dire une réinitialisation du système et l'effacement de la mémoire vive donc du contenu des diverses variables, il faut taper `restart`.