Maple : Manipulation des variables

Plan

Exercice (Rappel) :

Soit l'expression mathématique :

$$x = 1/(\sqrt{2} - 1)$$

Ecrire les commandes Maple permettant de :

1. Définir x ;
2. Évaluer x à un réel ;
3. Affecter à N le numérateur de x ;
4. Affecter à D le dénominateur de x ;
5. Positionner Digits à 25 ;
6. Évaluer de nouveau le réel x ;

Les variables

Une variable est définie par :

- Nom (identificateur)
- Type
- Valeur

Choix des noms de variables : identificateur

Un identificateur est une combinaison de :

- Lettres ('a'..'z' , 'A'..'Z')
- chiffres (0..9) (pas au début)
- espace souligné _ (touche 8)
- Le caractère % (seulement au début)

Exemple :

```
X , x , x1, 1x , _1 , 1_ , %1x , 1%x
, x%1 , a_1% , a_ 1, __ , _n_ , _1_
```

Pour échapper un identificateur de la règle générale, on délimite le nom d'objet par `` (deux back quote avec les touches clavier : AltGr et 7)

Exemple :

```
[>`1_` := 10; `1%x`:=4; `b;'#à)'(-`
:= 3 :
```

Type des variables :

En Maple, on peut stocker dans une variable un autre objet Maple de n'importe quel type (numérique, fonction, polynôme, matrice,…)

On ne déclare pas les types de variables car le type de variable change selon son contenu. La valeur à affecter à une variable ne dépend pas de son type.

⇨ On peut stocker dans une même variable plusieurs valeurs de différents types.

Exemple :

```
[> a:=5; a;          # entier
[> a:=5.; a;       # réel
[> a:= "5"; a;   # chaine de
caractère
```

Type de chaine de caractère :

Pour avoir une variable de type chaine de caractères, on doit délimiter le contenu de cette variable par deux guillemets (touche 3):

Exemple :

```
[> x :=  "ceci une chaine";
```

Assignation des variables :

Assigner ou affecter un objet à une variable c'est mettre un objet en mémoire de cette variable.

Pour vérifier le contenu de variable : nom_var ;

On a deux méthodes pour assigner/affecter un objet à une variable selon son contenu:

Si la variable est non assignée (ne contient pas de valeur = vide) c'est-à-dire on va lui affecter une valeur pour la première fois :

Opérateur d'affectation :=

Syntaxe : `nom_variable := valeur;`

Exemple :

```
[> u := 4;
```

Fonction assign ;

Syntaxe : `assign (nom_variable, valeur) ;`

Exemple :

```
[> assign (u,4); u ;
[> assign (u,"4"); u ;
```

Si la variable est assignée (contient une valeur) : on va la mettre à jour.

Syntaxe : `nom_variable := valeur;`

Exemple :

```
[> u:=4;
[> u:="14" ;
```

Les variables protégées

Maple réserve certains noms de variables dont on ne peut pas réaffecter ces noms, on les appelle « Variables protégées »

Les principaux variables protégées :

- Tous les **noms de fonctions** (sqrt, irem, iquo, sin, cos, restart…)
- **I** : Le i complexe
- **infinity**
- **Pi** : La constante mathématique (3,14)
- **D** : La fonction dérivée

Déprotéger une variable

Essayez de modifier la valeur de Pi ?

```
[> Pi:=5; # Mr help
```

Pour déprotéger des variables pour une session donnée :

```
[> unprotect(Pi);
```

On peut ensuite affecter Pi comme n'importe quelle autre variable

```
[> Pi := 5;
```

! Il faut retourner Pi à sa valeur initiale : `[> restart ;`

Vérifier si la variable est assignée :

On peut vérifier si la variable est assignée (affectée) ou non assignée (ne contient pas de valeur/vide)

Syntaxe : `assigned (nom_variable) ;`

Cette fonction retourne un résultat de type booléen :

true : si la variable est assignée (non vide)
false : si la variable est non assignée (vide)

Exemple :

```
[> u:=5: assigned (u); assigned (t);
```

Evaluation des variables :

Evaluer une variable ça revient à savoir son contenu.

Si la variable est assignée (affectée) elle est évaluée à sa valeur.

Exemple:

```
[>restart: x:=5: # Affecter 5 à x
[> assigned(x); #tester si l'objet
x est assigné
[> x; # évaluer l'objet x
[> assign(y,x); #Affecter x à y
[> assigned(y); #tester si l'objet
y est assigné
[> y; # évaluer l'objet y
[> eval(x); eval(y);
```

Si la variable est non assignée (vide) elle est évaluée à son **nom**

Exemple:

```
[> restart: x:=5: # Affecter 5 à x
[> assigned(x); #tester si l'objet
x est assigné
```

```
[> x; # évaluer l'objet x
[> assigned(y); #tester si l'objet
y est assigné
[> y; eval(y);# évaluer l'objet y
```

Niveaux d'évaluation des variables

Jusqu'à où Maple va-t-il chercher dans ses mémoires ?

Exemple :

```
 [> x:=y; x ; # On met y en
Mémoire de x
 [> y:=5;y ; # On met 5 en Mémoire
de y
 [> x; # On demande x
```

Maple parcourt toutes les affectations

Pour fixer le niveau d'évaluation (jusqu'à quel niveau Maple doit aller chercher dans ses mémoires)

Syntaxe : `eval(nom_var, niveau)`

Exemple :
```
[>eval(x,1); eval(x,2);
eval(x,10); eval(x);
```

L'absence de deuxième argument d'eval permet à Maple d'effectuer une évaluation totale

Exemple :
```
[> eval(x);
```

Exemple :
```
[> a:=z: z:=b: b:=c: c:=d:
d:=3:
[> eval(a);
[> eval(a,eval(a));
[> eval(a,eval(a)+2);
```

Retarder l'évaluation des variables :

Pour retarder l'évaluation d'une expression, on l'entoure par des simples cotes ' (Touche 4)
Après chaque évaluation un niveau de côte est enlevé.
De d'extérieur vers l'intérieur

Exemple :
```
[> a:=1: '' a '' ; % ; % ;
[> x:=5; 'x' ; x; "x" ;
```

Le simple cote sert à :

- Empêcher l'évaluation d'une variable
- Désaffecter une variable (paragraphe suivant)

Exemple:

```
[> restart :
[> x:=1; y:=''x'+'2'';
[> %;
[> %;
[> %;
```

Dé assignation (Vider) des variables (4 méthodes) :

Syntaxe :

```
nom_variable := 'nom_variable';
```

```
nom_variable := evaln (nom_variable);
```

```
unassign ('nom_variable','h','gf');
```

```
restart ;
```

La commande restart permet de :

- Débuter une nouvelle session (Libérer la mémoire)
- dé assigne toutes les variables de la session : x , y, nom_variable,…
- dé assigne toutes les variables globales : Digits,…

Exercice :

Soit x, y et c : trois noms de variables
En utilisant deux méthodes :

1. Affecter à la variable x la valeur 2
2. Affecter à la variable y la valeur 3
3. Affecter à c le produit de x par y en retardant l'évaluation de c deux fois.
4. Désaffecter toutes les variables

Exercice :

```
[> restart: d:=5!; e:=''d'';
e; %; d:=eval(d); e; restart:
y:=x; x:=4;  'x+x+2*(y+1)';
%;

[> restart : a:=1;
x:=''a'+b'; a:=2: 'x'/x;%;%;
```

Evaluation d'une expression relationnelle;

```
[> evalf (3/4);
[> eval (x^2 +1, x=3);
[> evalb (4=5);
```

Exercice :

```
[> restart : 2>3; evalb(%);
prepa := 3>10; evalb(prepa);
eq:=2=2;
[> restart: iquo (1515,732, 'r');
% * 732 + r;  irem(%, 732,
'q');q*732+%;%;
```