16 - Les types de Maple (exemples)

1. Types de base

Voici quelques types numériques :

```
> whattype(1999), whattype(-1), whattype(1/3), whattype(12.345) ;
```

$$integer, integer, fraction, float$$

Le contrôle de type s'effectue après évaluation et/ou simplification automatique.

Ceci explique que $1. - 1$, simplifié en 0, soit de type *integer*.

De même, $2. - 1$ est évalué en 1. (notez le point décimal) et est donc un $float$.

Enfin $\frac{45}{9}$ est simplifié en 5, et est donc de type *integer*.

```
> whattype(1.-1), whattype(2.-1), whattype(45/9) ;
```

$$integer, float, integer$$

Tant qu'un nom de variable est non assigné, il est de type *symbol*.

Sinon il est évalué en son contenu qui peut être une somme, un produit, une puissance, etc.

```
> restart : whattype(x) ;
  x :=a+b : y :=c*d : z :=e^f : map(whattype,[x,y,z]) ;
```

$$symbol$$

$$[+, *, \hat{}]$$

L'expression $a + b * c$ est la somme de a et de $b * c$: elle est donc de type `+`.

En revanche, $(a + b) * c$ est un produit : c'est une expression de type $*$.

$A\&*B$ représente un appel non évalué à la fonction `&*`.

```
> map(whattype,[a+b*c,(a+b)*c,A&*B]) ;
```

$$[+, *, function]$$

On teste ici le type d'expressions qui sont des égalités ou des inégalités :

```
> map(whattype,[a=b,a<>b,a<b,a<=b,a>b,a>=b]) ;
```

$$[=, <>, <, <=, <, <=]$$

Voici trois expressions logiques, de types respectifs *and*, *or* et *not*.

©EduKlub S.A.
Tous droits de l'auteur des œuvres réservés. Sauf autorisation, la reproduction ainsi que toute utilisation des œuvres autre que la consultation individuelle et privée sont interdites.

Extrait gratuit de document, le document original comporte 10 pages.

```
> map(whattype,[a and b, a or b, not a]) ;
```

$$[and,\ or,\ not]$$

$[a, b, c]$ est une liste, $\{a, b, c\}$ est un ensemble, et a, b, c est une séquence d'expressions...

```
> whattype([a,b,c]), whattype({a,b,c}), whattype(a,b,c) ;
```

$$list,\ set,\ exprseq$$

Quand une variable contient une table (un tableau), il faut comme toujours utiliser `evalm` pour accéder au contenu de cette variable et en tester le type :

```
> x :=table() : y :=array(4..9) : map(whattype,[x,y,eval(x),eval(y)]) ;
```

$$[symbol,\ symbol,\ table,\ array]$$

3..8 est un intervalle (type *range*), $b[3]$ est un nom indexé, $a.3$ est de type *symbol* (car évalué en le nom $a3$) et $a.(\frac{3}{4})$ est de type *point* car ici la concaténation ne peut pas s'effectuer.

```
> 3..8,b[3],a.3,a.(3/4) ; map(whattype,[%]) ;
```

$$3..8,\ b_3,\ a3,\ a.(\frac{3}{4})$$

$$[..,\ indexed,\ symbol,\ .]$$

L'expression '$2 + 3$' s'évalue en 5 et est donc de type *integer*.

En revanche ''$2 + 3$'' s'évalue en '$2 + 3$' et est de type *uneval*.

```
> whattype('2+3'), whattype(''2+3'') ;
```

$$integer,\ uneval$$

exp(1) se simplifie en e uniquement pour l'affichage, mais reste considéré comme un appel non évalué à la fonction exp : c'est donc un objet de type *function*.

Il est de même de $\sin\frac{\pi}{99}$ qui n'est pas simplifié.

En revanche, on lit le type de exp(0) et $\sin\frac{\pi}{3}$ après simplification.

```
> x :=[exp(1),exp(0),sin(Pi/99),sin(Pi/3)] ; map(whattype,x) ;
```

$$x := [\mathrm{e},\ 1,\ \sin(\frac{1}{99}\,\pi),\ \frac{1}{2}\,\sqrt{3}]$$

$$[function,\ integer,\ function,\ *]$$

On utilise maintenant `type` pour vérifier si une expression est d'un type particulier.

L'expresion $3 * 12$, n'est pas de type $*$, car elle est évaluée en 36.

`Pi` est une constante symbolique : ce n'est pas un *float*.

L'expresion a/b n'est pas de type `/`, tout simplement parce que ce type n'existe pas, mais aussi parce a/b, codé $a * b^{-1}$ en mémoire, est de type $*$.

©EduKlub S.A.
Tous droits de l'auteur des œuvres réservés. Sauf autorisation, la reproduction ainsi que toute utilisation des œuvres autre que la consultation individuelle et privée sont interdites.

Extrait gratuit de document, le document original comporte 10 pages.

```
> type(3*12,`*`), type(a*b,`*`), type(Pi,float) ; type(a/b,`/`) ;
```

$$false, true, false$$

```
Error, type `/` does not exist
```

Ici on utilise l'instruction `zip` pour vérifier simultanément si les expressions 15, 3/2, ab et $1/b$ sont respectivement de type *integer*, *fraction*, *symbol* et *fraction*.

```
> zip(type,[15,3/2,ab,1/b],[integer,fraction,symbol,fraction]) ;
```

$$[true, true, true, false]$$

2. Réunions ou synonymes de types de base

12345 est un *integer* donc c'est un *rational*. 10/1111 est de type *fraction* donc de type *rational*.

En revanche 12.345 est de type *float*, et pas de type *rational*.

On ne le confondra donc pas avec le rationnel 12345/1000...

```
> type(12345,rational), type(10/1111,rational), type(12.345,rational) ;
```

$$true, true, false$$

Les trois nombres considérés sont tous de type *numeric*.

```
> map(type,[12345,10/1111,12.345],numeric) ;
```

$$[true, true, true]$$

On voit que tous les objets de la liste suivante sont de type *name* :

```
> restart : map(type,[a,sin,`Maple V`,a[1,2],a.3],name) ;
```

$$[true, true, true, true, true]$$

On définit deux listes x et y, contenant un certain nombre d'expressions logiques :

```
> x :=[a+1=b-2, 2<>3, evalb(2<>3), a<b, not(a< b)] ;
  y :=[ a and b, a or b, not(a<b), true or a] ;
```

$$x := [a + 1 = b - 2,\ 2 \neq 3,\ true,\ a < b,\ \textbf{not}\ (a - b < 0)]$$

$$y := [a\ \textbf{and}\ b,\ a\ \textbf{or}\ b,\ \textbf{not}\ (a - b < 0),\ true]$$

Voici quels sont les types des différents éléments de x et de y :

```
> map(whattype,x), map(whattype,y) ;
```

$$[=, <>, symbol, <, not], [and, or, not, symbol]$$

©EduKlub S.A.
Tous droits de l'auteur des œuvres réservés. Sauf autorisation, la reproduction ainsi que toute utilisation des œuvres autre que la consultation individuelle et privée sont interdites.

Extrait gratuit de document, le document original comporte 10 pages.

On teste quels éléments de x sont de type *relation*, et quels éléments de y sont de type *logical* :

```
> map(type,x,relation), map(type,y,logical) ;
```

$$[true, true, false, true, false], [true, true, true, false]$$

On voit que tous les éléments de x et de y sont de type *boolean* :

```
> map(type,x,boolean), map(type,y,boolean) ;
```

$$[true, true, true, true, true], [true, true, true, true]$$

Parmi tous ces objets, seuls les deux derniers ne sont pas de type *algebraic* :

```
> map(type,[a,sqrt(a&*b),sin(a),a/(a+b),12,sin,{a},table()],algebraic) ;
```

$$[true, true, true, true, true, true, false, false]$$

Les types $**$, *range* et *equation* sont respectivement synonymes des types `^`, `..` et `=` :

```
> zip(type,[a^b,a=b,4..7],[`**`,equation,range]) ;
```

$$[true, true, true]$$

3. Types précisés

x et y contiennent tous les deux un tableau de dimension 1 :

```
> restart : x :=array(2..8) : y :=array(1..5) :
```

Le contenu de x et y est bien de type *array*.

En revanche seul celui de y est de type *vector* (en effet l'indice de x ne débute pas à 1) :

```
> map(type@eval,[x,y],array), map(type@eval,[x,y],vector) ;
```

$$[true, true], [false, true]$$

On place maintenant dans x un tableau de dimension 3 et dans y un tableau de dimension 2 :

```
> x :=array(1..3,1..4,1..2) : y :=array(1..5,1..4) :
```

Le contenu de x et celui de y sont bien de type *array*.

En revanche seul celui de y est de type *matrix* (les indices du tableau contenu dans x débutent pourtant à 1, mais c'est un tableau de dimension 3) :

```
> map(type,[x,eval(x),eval(y)],array), map(type@eval,[x,y],matrix) ;
```

$$[true, true, true], [false, true]$$

©EduKlub S.A.
Tous droits de l'auteur des œuvres réservés. Sauf autorisation, la reproduction ainsi que toute utilisation des œuvres autre que la consultation individuelle et privée sont interdites.

Extrait gratuit de document, le document original comporte 10 pages.