The REBOL Documentation Project

-- FR - Documentation REBOL - Manuels --

Manuels

Manuel de l'utilisateur - Chapitre 11 - Maths

Philippe Le Goff

Première publication : 22 septembre 2005, et
mis en ligne le jeudi 22 septembre 2005

Résumé :

Ce document est la traduction française du Chapitre 11 du User Guide de REBOL/Core, qui concerne les expressions mathématiques.

Ce document est la traduction française du Chapitre 11 du User Guide de REBOL/Core, qui concerne les expressions mathématiques.

- 1 Historique de la traduction
- 2 Présentation
- 3 Types de données scalaires
- 4 Ordre de l'évaluation
- 5 Fonctions et Opérateur standards
  - 5.1 absolute
  - 5.2 add
  - 5.3 complement
  - 5.4 divide
  - 5.5 multiply
  - 5.6 negate
  - 5.7 random
  - 5.8 remainder
  - 5.9 subtract
- 6 Conversion de type
- 7 Fonctions de comparaison
  - 7.1 equal
  - 7.2 greater
  - 7.3 greater-or-equal
  - 7.4 lesser
  - 7.5 lesser-or-equal
  - 7.6 not equal to
  - 7.7 same
  - 7.8 strict-equal
  - 7.9 strict-not-equal
- 8 Fonctions Logarithmiques
  - 8.1 exp
  - 8.2 log-10
  - 8.3 log-2
  - 8.4 log-e
  - 8.5 power
  - 8.6 square-root
- 9 Fonctions Trigonométriques
  - 9.1 arccosine
  - 9.2 arcsine
  - 9.3 arctangent
  - 9.4 cosine
  - 9.5 sine
  - 9.6 tangent
- 10 Fonctions logiques
  - 10.1 and

- 10.2 or
- 10.3 xor
- 10.4 complement
- 10.5 not

- 11 Erreurs
  - 11.1 Tentative de division par zéro
  - 11.2 Débordement de calcul
  - 11.3 Nombre positif requis
  - 11.4 Impossible d'utiliser l'opérateur avec un type de données

1 Historique de la traduction

Date	Version	Commentaires	Auteur	Email
1er juin 2005 19:33	1.0.0	Traduction initiale	Philippe Le Goff	lp—legoff—free—fr

2 Présentation

REBOL permet d'assurer un ensemble complet d'opérations mathématiques et trigonométriques. La plupart de ces opérateurs peuvent manipuler plusieurs types de données, comme les nombres entiers et décimaux, les tuples, des valeurs de type temps et date. Certains de ces types de données peuvent même être mélangés ou forcés.

3 Types de données scalaires

Les fonctions mathématiques de REBOL agissent de façon régulière sur une grande variété de types de données scalaires (numériques).

Ces types de données incluent :

| Type de données | Description | | Integer ! | nombres sur 32 bits sans décimales | | Decimal ! | nombres sur 64 bits avec décimales | | Money ! | valeurs monétaires sur 64 bits avec décimales | | Time ! | heures, minutes, secondes, dixièmes jusqu'au millième de secondes | | Date ! | jour, mois, année, heure, fuseau horaire | | Pair ! | coordonnées graphiques ou taille | | Tuple ! | versions, couleurs, adresses réseau |

Voici ci-dessous quelques exemples pour illustrer un certain nombre d'opération sur les types de donnée scalaires. Notez que les opérateurs renvoient les résultats adéquats pour chaque type de données.

Les types de données **integer !** et **decimal !** :

```
print 2 + 1
```

```
3
print 2 - 1
1
print 2 * 10
20
print 20 / 10
2
print 21 // 10
1
print 2.2 + 1
3.2
print 2.2 - 1
1.2
print 2.2 * 10
22
print 2.2 / 10
0.22
print random 10
5
```

Le type de données **time !** :

```
print 2:20 + 1:40
4:00
print 2:20 + 5
2:20:05
print 2:20 + 60
2:21
print 2:20 + 2.2
2:20:02.2
print 2:20 - 1:20
1:00
print 2:20 - 5
2:19:55
print 2:20 - 120
2:18
print 2:20 * 2
4:40
print 2:20 / 2
1:10
print 2:20:01 / 2
1:10:00.5
print 2:21 // 2
0:00
print - 2:20
-2:20
print random 10:00
5:30:52
```

Le type de données **date !** :

```
print 1-Jan-2000 + 1
2-Jan-2000
print 1-Jan-2000 - 1
31-Dec-1999
print 1-Jan-2000 + 31
1-Feb-2000
print 1-Jan-2000 + 366
1-Jan-2001
birthday: 7-Dec-1944
print ["I've lived" (now/date - birthday) "days."]
I've lived 20305 days.
print random 1-1-2000
29-Apr-1695
```

Le type de données **money !** :

```
print $2.20 + $1
$3.20
print $2.20 + 1
$3.20
print $2.20 + 1.1
$3.30
print $2.20 - $1
$1.20
print $2.20 * 3
$6.60
print $2.20 / 2
$1.10
print $2.20 / $1.10
2
print $2.21 // 2
$0.21
print random $10.00
$6.00
```

Le type de données **pair !** :

```
print 100x200 + 10x20
110x220
print 10x10 + 3
13x13
print 10x20 * 2x4
20x80
print 100x100 * 3
300x300
print 100x30 / 10x3
10x10
print 100x30 / 10
10x3
```

```
print 101x32 // 10x3
1x2
print 101x32 // 10
1x2
print random 100x20
67x12
```

Le type de données **tuple !** :

```
print 1.2.3 + 3.2.1
4.4.4
print 1.2.3 - 1.0.1
0.2.2
print 1.2.3 * 3
3.6.9
print 10.20.30 / 10
1.2.3
print 11.22.33 // 10
1.2.3
print 1.2.3 * 1.2.3
1.4.9
print 10.20.30 / 10.20.30
1.1.1
print 1.2.3 + 7
8.9.10
print 1.2.3 - 1
0.1.2
print random 10.20.30
8.18.12
```

4 Ordre de l'évaluation

Il y a deux règles à se rappeler pour l'évaluation d'expressions mathématiques :

- Les expressions sont évaluées de gauche à droite.
- Les opérateurs ont priorité sur les fonctions

L'évaluation des expressions de gauche à droite est indépendante du type d'opérateur utilisé. Par exemple :

```
print 1 + 2 * 3
9
```

Dans l'exemple ci-dessus, notez que le résultat n'est pas sept, comme cela le serait si la multiplication avait priorité sur l'addition.

Remarque importante

Le fait que les expressions mathématiques soient évaluées de la gauche vers la droite sans se soucier des opérateurs donne un comportement différent de la plupart des autres langages de programmation. Beaucoup de langages possèdent des règles de priorité où vous devez vous rappeler ce qui détermine l'ordre dans lequel les opérateurs seront évalués. Par exemple, une multiplication est faite avant une addition. Certains langages possèdent plus d'une dizaine de règles de ce genre.

En REBOL, plutôt que d'imposer à l'utilisateur de se remémorer les priorités des opérateurs, vous avez seulement à vous rappeler la règle "de-gauche-à-droite".

Encore plus important, pour un code poussé, perfectionné, comme des expressions qui manipulent d'autres expressions (de la réflectivité, par exemple), vous n'avez pas besoin de réordonner les termes sur la base d'une priorité. L'ordre d'évaluation demeure simple.

Pour la plupart des expressions mathématiques, l'évaluation de gauche à droite fonctionne assez bien et est simple à se rappeler. D'un autre côté, comme cette règle est différente d'autres langages de programmation, elle peut être la cause d'erreurs de programmation, donc soyez vigilants.

La meilleure solution est de vérifier votre travail. Vous pouvez aussi utiliser des parenthèses si nécessaire, afin de clarifier vos expressions (voir ci-dessous), et vous pouvez toujours saisir votre expression dans la console, pour vérifier le résultat.

S'il est nécessaire d'évaluer une expression dans un autre ordre, réordonnez-la ou utilisez des parenthèses :

```
print 2 * 3 + 1
7
print 1 + (2 * 3)
7
```

Quand des fonctions sont mélangées avec des opérateurs, les opérateurs sont évalués en premier, puis les fonctions :

```
print absolute -10 + 5
5
```

Dans l'exemple ci-dessus, l'addition est d'abord réalisée, et son résultat passé à la fonction "valeur absolue".

Dans l'exemple suivant :

```
print 10 + sine 30 + 60
11
```

l'expression est évaluée dans cet ordre :

```
30 + 60 => 90
```

```
sine 90 => 1
10 + 1 => 11
print
```

Pour changer l'ordre afin que le sinus de 30 soit calculé en premier, utilisez des parenthèses :

```
print 10 + (sine 30) + 60
70.5
```

ou réorganisez l'expression :

```
print 10 + 60 + sine 30
70.5
```

5 Fonctions et Opérateur standards

Cette section décrit les fonctions et opérateurs mathématiques standards utilisés en REBOL.

5.1 absolute

Les expressions :

```
absolute value
```

```
abs value
```

renvoient la valeur absolue de l'argument *value*. Les types de données possibles sont : integer, decimal, money, time, pair.

```
print absolute -10
10
print absolute -1.2
1.2
print absolute -$1.2
$1.20
print absolute -10:20
10:20
print absolute -10x-20
10x20
```

5.2 add

Les expressions :

```
value1 + value2

add value1 value2
```

renvoient la somme des valeurs *value1* et *value2*. S'utilise avec les types de données integer, decimal, money, time, tuple, pair, date, char.

```
print 1 + 2
3
print 1.2 + 3.4
4.6
print 1.2.3 + 3.4.5
4.6.8
print $1 + $2
$3.00
print 1:20 + 3:40
5:00
print 10x20 + 30x40
40x60
print #"A" + 10
K
print add 1 2
3
```

NdT : remarquez que l'expression

```
+ 1 2
3
```

est aussi valable, quoique moins "naturelle". Il en est ainsi avec plusieurs opérateurs.

5.3 complement

L'expression :

```
complement value
```

renvoie le complément numérique (bitwise complement) d'une valeur. Pour types de données integer, decimal, tuple, logic, char, binary, string, bitset, image.

```
print complement 10
-11
print complement 10.5
-11
```

```
print complement 100.100.100
155.155.155
```

Ndt : il est possible d'utiliser **complement** sur des caractères :

```
complement "I"
"¶"       (le symbole de saut de ligne de Ms-Word)
complement "V"
"©"       (le symbole Copyright)
```

5.4 divide

Les expressions :

```
value1 / value2

divide value1 value2
```

retournent le résultat de la division de *value1* par *value2*. S'utilise sur les types de données integer, decimal, money, time, tuple, pair, char .

```
print 10 / 2
5
print 1.2 / 3
0.4
print 11.22.33 / 10
1.2.3
print $12.34 / 2
$6.17
print 1:20 / 2
0:40
print 10x20 / 2
5x10
print divide 10 2
5
```

NdT : l'expression

```
/ 1 2
0.5
```

marche également.

5.5 multiply

Les expressions :

```
value1 * value2

multiply value1 value2
```

renvoient le résultat de la multiplication de *value1* par *value2*. Fonctionne avec les datatypes integer, decimal, money, time, tuple, pair, char.

```
print 10 * 2
20
print 1.2 * 3.4
4.08
print 1.2.3 * 3.4.5
3.8.15
print $10 * 2
$20.00
print 1:20 * 3
4:00
print 10x20 * 3
30x60
print multiply 10 2
20
```

Ndt : l'expression

```
* $10 2
$20.00
```

est également valable.

5.6 negate

Les expressions :

```
- value

negate value
```

changent le signe de la valeur. Les types de données possibles sont : integer, decimal, money, time, pair, char.

```
print - 10
-10
print - 1.2
-1.2
print - $10
-$10.00
print - 1:20
-1:20
print - 10x20
-10x-20
print negate 10
-10
```

5.7 random

L'expression :

```
random value
```

renvoie une valeur aléatoire qui est inférieure ou égale à la valeur de l'argument.

Notez que pour les nombres entiers, **random** commence à 1, pas à 0, et va inclure la valeur de l'argument fourni. Ceci permet à **random** d'être utilisé directement dans des fonctions comme **pick**.

Quand un nombre décimal est utilisé, le résultat est du type décimal, arrondi à un entier.

Le raffinement **/seed** réinitialise le générateur de nombres aléatoires. Utilisez d'abord le raffinement **/seed** avec **random** si vous voulez générer un nombre unique aléatoire. Vous pouvez utiliser la date et l'heure courante pour fabriquer une base unique :

```
random/seed now
```

S'utilise avec les types de données integer, decimal, money, time, tuple, pair, date, char, string, et block.

```
print random 10
5
print random 10.5
2
print random 100.100.100
79.95.66
```

```
print random $100
$32.00
print random 10:30
6:37:33
print random 10x20
2x4
print random 30-Jun-2000
27-Dec-1171
```

5.8 remainder

Les expressions :

```
value1 // value2

remainder value1 value2
```

renvoient le reste de la division de *value1* par *value2*. Fonctionne avec les datatypes integer, decimal, money, time, tuple, pair .

```
print 11 // 2
1
print 11.22.33 // 10
1.2.3
print 11x22 // 2
1x0
print remainder 11 2
1
```

5.9 subtract

Les expressions :

```
value1 - value2

subtract value1 value2
```

renvoient le résultat de la soustraction entre *value2* et *value1*. S'utilise avec les types de données integer, decimal, money, time, tuple, pair, date, char.

```
print 2 - 1
1
print 3.4 - 1.2
2.2
print 3.4.5 - 1.2.3
2.2.2
```

```
print $2 - $1
$1.00
print 3:40 - 1:20
2:20
print 30x40 - 10x20
20x20
print #"Z" - 1
Y
print subtract 2 1
1
```

6 Conversion de type

Lorsque des opérations mathématiques sont réalisées entre des types de données différents, normalement, le type de données non entier ou non décimal est retourné. Quand des entiers sont combinés avec des nombres décimaux, c'est le datatype décimal qui est retourné.

7 Fonctions de comparaison

Toutes les fonctions de comparaison renvoient une valeur logique : **true** ou **false**.

7.1 equal

Les expressions :

```
value1 = value2

equal? value1 value2
```

renvoient **true** si la première et la seconde valeur sont égales. Fonctionne avec les types de données : integer, decimal, money, time, date, tuple, char et series.

```
print 11-11-99 = 11-11-99
true
print equal? 111.112.111.111 111.112.111.111
true
print #"B" = #"B"
true
print equal? "a b c d" "A B C D"
true
```

7.2 greater

Les expressions :

```
value1 > value2

greater? value1 value2
```

retournent **true** si la première valeur est supérieure à la seconde valeur. S'utilise avec les types de données integer, decimal, money, time, date, tuple, char et series.

```
print 13-11-99 > 12-11-99
true
print greater? 113.111.111.111 111.112.111.111
true
print #"C" > #"B"
true
print greater? [12 23 34] [12 23 33]
true
```

7.3 greater-or-equal

Les expressions :

```
value1 >= value2

greater-or-equal? value1 value2
```

retournent **true** si la première valeur est supérieure ou égale à la seconde valeur. S'utilise avec les types de données integer, decimal, money, time, date, tuple, char et series.

```
print 11-12-99 >= 11-11-99
true
print greater-or-equal? 111.112.111.111 111.111.111.111
true
print #"B" >= #"A"
true
print greater-or-equal? [b c d e] [a b c d]
true
```

7.4 lesser

Les expressions :

```
value1
lesser? value1 value2
```

retournent **true** si la première valeur est inférieure à la seconde valeur. S'utilise avec les types de données integer, decimal, money, time, date, tuple, char et series.

```
print 25  true
print lesser? 25.3 25.5
true
print $2.00  true
print lesser? 00:10:11 00:11:11
true
```

7.5 lesser-or-equal

Les expressions :

```
value1
lesser-or-equal? value1 value2
```

retournent **true** si la première valeur est inférieure ou égale à la seconde valeur. S'utilise avec les types de données integer, decimal, money, time, date, tuple, char et series.

```
print 25  true
print lesser-or-equal? 25.3 25.5
true
print $2.29  true
print lesser-or-equal? 11:11:10 11:11:11
true
```

7.6 not equal to

Les expressions :

```
value1  value2

not-equal? value1 value2
```

retournent **true** si la première valeur n'est pas égale à la seconde valeur. S'utilise avec les types de données integer, decimal, money, time, date, tuple, char et series.

```
print 26 v 25
true
print not-equal? 25.3 25.5
true
print $2.29  $2.30
true
print not-equal? 11:11:10 11:11:11
true
```

7.7 same

Les expressions :

```
value1 =? value2

same? value1 value2
```

renvoient **true** si les deux mots font référence à la même valeur. Par exemple, lorsque vous voulez savoir si deux mots font référence à la même valeur d'index dans une série. Fonctionne avec tous les types de données.

```
reference-one: "abcdef"
reference-two: reference-one
print same? reference-one reference-two
true
reference-one: next reference-one
print same? reference-one reference-two
false
reference-two: next reference-two
print same? reference-one reference-two
true
reference-two: copy reference-one
print same? reference-one reference-two
false
```

7.8 strict-equal

Les expressions :

```
value1 == value2

strict-equal? value1 value2
```

retournent **true** si la première et la seconde valeurs sont strictement identiques. **Strict-equal** peut être utilisée comme la version sensible à la casse de **equal ?** (=) pour les chaînes de caractères et pour différencier les entiers des décimaux lorsque les valeurs sont identiques. Fonctionne avec tous les types de données.

```
print strict-equal? "abc" "ABC"
false
print equal? "abc" "ABC"
true
print strict-equal? "abc" "abc"
true
```

```
print strict-equal? 1 1.0
false
print equal? 1 1.0
true
print strict-equal? 1.0 1.0
true
```

7.9 strict-not-equal

L'expression :

```
strict-not-equal? value1 value2
```

renvoie **true** si la première et la seconde valeurs ne sont pas strictement égales. **strict-not-equal** peut être utilisée comme la version sensible à la casse de **not-equal ?** (11.4) pour les chaînes de caractères et pour différencier les entiers des décimaux lorsque leurs valeurs sont identiques. Fonctionne avec tous les types de données.

```
print strict-not-equal? "abc" "ABC"
true
print not-equal? "abc" "ABC"
false
print strict-not-equal? "abc" "abc"
false
print strict-not-equal? 1 1.0
true
print not-equal? 1 1.0
false
print strict-not-equal? 1.0 1.0
false
```

8 Fonctions Logarithmiques

8.1 exp

L'expression :

```
exp value
```

calcule E (nombre naturel) à la puissance de l'argument.

8.2 log-10

L'expression :

```
log-10 value
```

renvoie le logarithme de base 10 de l'argument.

8.3 log-2

L'expression :

```
log-2 value
```

renvoie le logarithme de base 2 de l'argument.

8.4 log-e

L'expression :

```
log-e value
```

renvoie le logarithme naturel (base E) de l'argument.

8.5 power

Les expressions :

```
value1 ** value2
```

```
power value1 value2
```

renvoient le résultat du calcul de *value1* à la puissance *value2*.

8.6 square-root

L'expression :

```
square-root value
```

renvoie la racine carrée de l'argument *value*.

9 Fonctions Trigonométriques

Les fonctions trigonométriques travaillent normalement en degrés. Il faut utiliser le raffinement **/radians** avec l'une ou l'autre des fonctions trigonométriques pour travailler en radians.

9.1 arccosine

L'expression :

```
arccosine value
```

renvoie la fonction cosinus inverse pour l'argument.

9.2 arcsine

L'expression :

```
arcsine value
```

renvoie la fonction sinus inverse pour l'argument.

9.3 arctangent

L'expression :

```
arctangent value
```

renvoie la fonction tangente inverse pour l'argument.

9.4 cosine

L'expression :

```
cosine value
```

renvoie le cosinus de l'argument.

9.5 sine

L'expression :

```
sine value
```

renvoie le sinus de l'argument.

9.6 tangent

L'expression :

```
tangent value
```

renvoie la tangente de l'argument.

10 Fonctions logiques

Les fonctions logiques peuvent être effectuées sur des valeurs logiques et sur des valeurs scalaires incluant les types de données integer, char, tuple, et bitset.

Avec des valeurs logiques, les fonctions logiques renvoient des valeurs booléennes. Quand elles utilisent d'autres types de valeurs, les fonctions logiques travaillent au niveau des bits.

10.1 and

La fonction **and** compare deux valeurs logiques et renvoie **true** seulement si les deux valeurs sont elles aussi **true** :

```
print (1  true
print (1  false
```

Quand elle est utilisée avec des nombres entiers, la fonction **and** effectue une comparaison bit à bit, et renvoie 1 si chacun des bits est à 1, ou 0 si ni l'un ni l'autre n'est 1 :

```
print 3 and 5
1
```

10.2 or

La fonction **or** compare deux valeurs logiques et renvoie **true** si l'une ou l'autre des deux est **true**, ou renvoie **false** si les deux sont **false**.

```
print (1  true
print (1  true
```

```
print (3  false
```

Quand elle est utilisée avec des nombres entiers, la fonction **or** effectue une comparaison bit à bit, et renvoie 1 si l'un des bits est 1, ou 0 si l'un et l'autre bit sont à 0 :

```
print 3 or 5
7
```

10.3 xor

La fonction **xor** compare deux valeurs logiques et retourne **true** si et seulement si l'une des valeurs est **true** et l'autre **false**.

```
print (1  false
print (1  true
print (3  false
```

Utilisée avec des nombres entiers, **xor** compare bit à bit ces nombres et renvoie 1 si et seulement si un bit est à 1 tandis que l'autre est à 0. Sinon, elle renvoie 0 :

```
print 3 xor 5
6
```

5.3 complement

La fonction **complement** renvoie le complément logique ou binaire d'une valeur. Elle est utilisée pour avoir l'inverse binaire de nombres entiers ou inverser un ensemble de bits (bitsets).

```
print complement true
false
print complement 3
-4
```

10.5 not

Pour une valeur logique, la fonction **not** renverra **true** si l'argument est **false** et **false** si, au contraire, l'argument est **true**. Elle n'effectue pas d'opérations numériques binaires.

```
print not true
false
print not false
true
```

11 Erreurs

Les erreurs mathématiques sont émises quand des opérations illégales sont réalisées, ou quand un débordement de calcul (overflow) se produit.

Les erreurs suivantes peuvent être rencontrées dans les opérations mathématiques.

11.1 Tentative de division par zéro

Une tentative a été faite de diviser un nombre par 0.

```
1 / 0
** Math Error: Attempt to divide by zero
** Where: connect-to-link
** Near: 1 / 0
```

11.2 Débordement de calcul

Une tentative de calcul d'un nombre trop grand pour REBOL a été faite.

```
1E+300 + 1E+400
** Math Error: Math or number overflow
** Where: connect-to-link
** Near: 1 / 0
```

11.3 Nombre positif requis

Une tentative de calcul a été faite avec un nombre négatif sur un opérateur mathématique qui accepte juste des nombres positifs.

```
log-10 -1
** Math Error: Positive number required
** Where: connect-to-link
** Near: log-10 -1
```

11.4 Impossible d'utiliser l'opérateur avec un type de données

Une tentative de calcul entre des types de données incompatibles. Le type de données du second argument dans l'opération est retourné tel quel.

```
10:30 + 1.2.3
** Script Error: Cannot use add on time! value
```

```
** Where: connect-to-link
** Near: 10:30 + 1.2.3
```