# Table of Contents

**Cette page n'est pas encore traduite entièrement. Merci de terminer la traduction**
*(supprimez ce paragraphe une fois la traduction terminée)*

# Calculateur avec variables

## Introduction

Pour chaque vue détaillées d'une géocache c:geo propose un onglet appelé “Variables” pour noter et effectuer des calculs avec les formules et les variables dont vous avez besoin pour cette cache.


Cela peut s'avérer utile si vous effectuez, par exemple, une multi-cache qui nécessite de collecter des valeurs sur le terrain puis d'effectuer des calculs avec celles-ci pour passer à l'étape suivante ou au final.

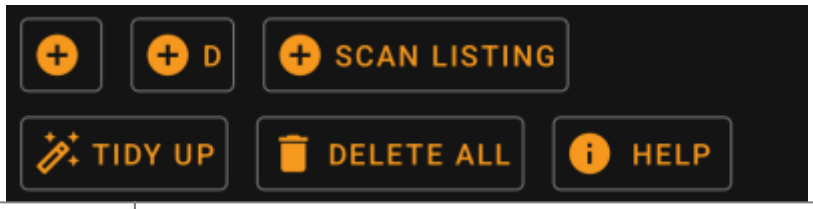
Vous pouvez utiliser cet onglet comme une aide autonome pour effectuer certains calculs ou vous pouvez également réutiliser toute variable définie ici pour générer un nouveau point de passage.

Les sections suivantes de cette page décrivent le contenu et la fonctionnalité de nouvel onglet.

## Partie "Commandes"

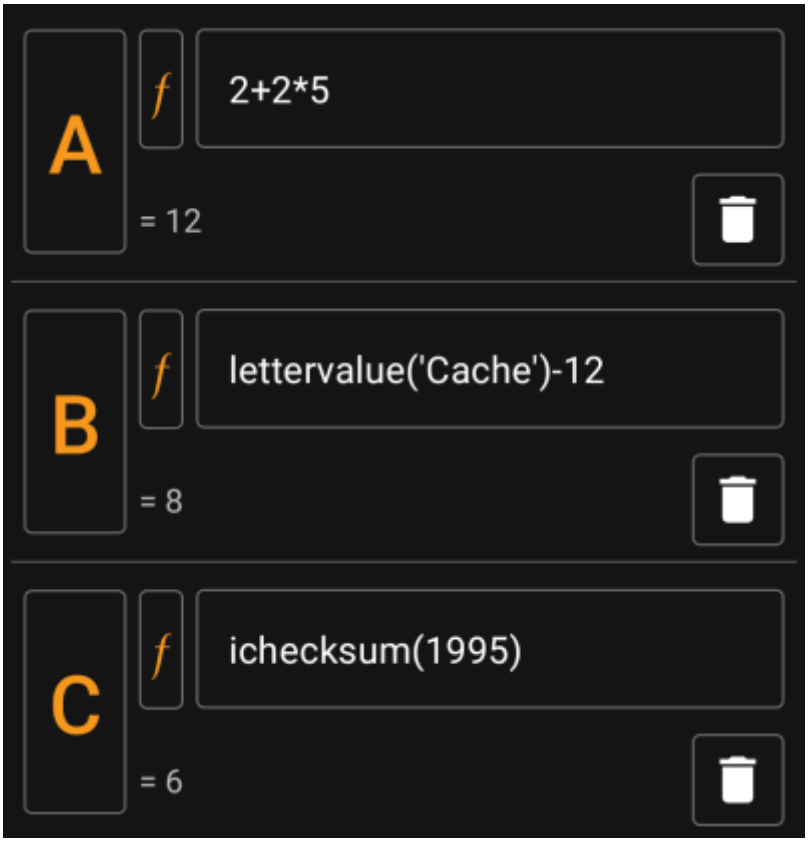
Dans la partie supérieure de l'onglet, vous verrez un ensemble de boutons proposant des fonctions pour remplir

la section variable ci-dessous :




| Bouton | Description |
|---|---|
|  | Ajoute manuellement une variable en fournissant son nom. |
|  | Ajoute automatiquement la variable suivante dans l'ordre alphabétique. |
|  | Analyse la description de la géocache à la recherche de formules potentiellement contenues et les propose pour être reprises dans l'onglet des variables. Chaque formule trouvée sélectionnée sera ajoutée dans une nouvelle variable. |
|  | Supprime toutes les variables sans valeur ou formule et peut être utilisé si vous avez accidentellement créé trop de variables ou si vous avez effacé certaines d'entre elles et n'en avez plus besoin. |
|  | Supprime toutes les variables définies et leurs valeurs. |
|  | Ouvre cette page du manuel dans un navigateur. |

## Partie "Variables"




Cette section permet de saisir la valeur ou la formule des variables générées. Plusieurs commandes sont disponibles :

| Bouton | Action |
|---|---|
|  | Un appui sur le nom de la variable permet de le changer. |
|  | Un appui sur le bouton “fonction”permet de préremplir le contenu de la fonction avec une fonction supportée. |
|  | Permet de rentrer manuellement dans le contenu de la fonction une valeur ou une formule en utilisant la syntaxe adapatée. |
|  | Permet de supprimer la variable. |

Le texte situé sous le champ de valeur affichera un aperçu du résultat. Il peut s'agir du résultat concret de la formule ou d'indications concernant les erreurs de syntaxe ou les valeurs manquantes.

## Syntaxe des formules

Le champ de valeur de chaque variable peut contenir différents types de valeurs ainsi que d'autres variables. Il supporte de nombreuses opérations mathématiques ainsi que plusieurs fonctions numériques et de chaînes de caractères (en partie liées au géocaching) décrites ci-dessous.

N'ayez pas peur de la syntaxe. Bien qu'elle prenne en charge des opérations plutôt complexes,

elle peut également être utilisée pour des calculs simples et ordinaires, comme vous avez l'habitude de le faire avec n'importe quelle calculatrice. Certaines fonctions prises en charge sont probablement réservées aux utilisateurs avancés.

La syntaxe est expliquée en détail dans les sous-chapitres suivants. Cependant, pour avoir un aperçu rapide de ce qui est pris en charge, des exemples sont donnés ici :

- `2*2+3` donne 7
- `2*(2+3)` donne 10
- `3*sin(90)` donne 3

- `4+length('test')` donne 8
- `rot13('abc')` donne nop
- `lettervalue('cache')` donne 20
- `checksum(lettervalue('cache'))` donne 2

- `A + A*2` avec `A=3` donne 9
- `AA(A+1)` avec `A=3` donne 334
- `$hello + 1` avec la variable `hello=24` donne 25
- `$hello(A+1)` avec `A=3` et `hello=24` donne 244
- `${hello}8A` avec `A=3` et `hello=24` donne 2483

## Types de valeur

La syntaxe prend en charge trois types de valeurs. La saisie est libre, en général le résultat de la formule essaiera de s'adapter le mieux possible aux valeurs données.

| Type | Description | Syntaxe | Exemple |
|---|---|---|---|
| Nombre entier | Nombre sans chiffres après la virgule | Utiliser les chiffres | `1234`, `-3` |
| Nombre decimal | Nombre avec des chiffres après la virgule | Utiliser les chiffres avec un point ou une virgule | `3.14`, `-3.14`, `3,14` |
| Chaine de caractère | Texte | Valeurs entourées par `'` ou `"` pour utiliser `'...'` or `"..."` eux-mêmes, saisir `''` ou `""` | `'test'`, `"test"` `“Il a dit ””oui””!”` |

## Opérateurs numériques

The following numeric operators are supported:

| Operator | Function | Example |
|---|---|---|
| + | Addition | 2+4 evaluates to 6 |
| - | Subtraction (or negating a number) | 6-4 evaluates to 2 `-(5-2)` evaluates to -3 |
| * | Multiplication | 3*4 evaluates to 12 |
| / | Division | 12/3 evaluates to 4 |
| % | Modulo | 12%5 evaluates to 2 |
| ^ | Potentiate | 3^3 evaluates to 9 |
| ! | Factorize | 4! evaluates to 24 |

## Relational operators and conditions

Relational operators like < or == can be used to compare two values with each other. In general, such an

operation will return the value 1 if the comparison yields true and the value 0 if it yields false.

For example, the expression `3 < 4` will compute to the value 1.

Relational operators are especially used in the `if` function. This function evaluates its first parameter. If this parameter is true (means: has a value > 0 or is a non-empty string) then it returns its second parameter. Else, and if it has a third parameter, the third parameter is returned.

The `if` function accepts any number of parameters and interprets them in an “if-then-if-then-if-then-…-else” cascade.

This means, that if the function was given 5 parameters then: * If the first parameter is true, then the second is returned * Else if the third parameter is true, then the fourth parameter is returned * Else the fifth parameter is returned

For example `if(A==5;50;A==4;40;30)` will evaluate to `50` if `A=5`, to `40` if `A=4` and to `30` for any other value of A.

| Operator | Meaning | Example |
|---|---|---|
| == | Checks for equality | 2==2 evaluates to 1(=true) |
| <> | Checks for inequality. | 3<>2 evaluates to 1(=true) |
| < | Is less than | 3<4 evaluates to 1(=true) |
| ⇐ | Is less or equal than | 3⇐3 evaluates to 1(=true) |
| > | Is greater than | 3>4 evaluates to 0(=false) |
| >= | Is greater or equal than | 5>=5 evaluates to 1(=true) |

## Functions

Functions all start with a letter, contain only letters and digits and have a directly attached parameter list in parenthesis. Multiple parameters are separated using `;`.

An example for a one-parameter function call is `sin(90)`. An example for a two-parameter function call is `rot('test'; 13)`.

The following functions are defined:

| Function | Synonyms | Description | Parameter 1 | Parameter 2 | Example |
|---|---|---|---|---|---|
| `sqrt` | - | calculates square root of given parameter | numeric parameter | - | `sqrt(9)` evaluates to 3 |
| `sin/cos/tan` | - | calculates sinus/cosinus/tangens of given parameter | numeric parameter in degree(!) | - | `sin(90)` evaluates to 1 |
| `abs` | - | calculates absolute value | numeric parameter | - | `abs(-34)` evaluates to 34 |
| `round` | - | rounds decimal values mathematically | value to round | optional: number of places to round to | `round(4.65)` evaluates to 5, `round(4.65;1)` evaluates to 4.7 |
| `if` | - | evaluates conditions and returns conditional values | list of if-then-else-values. See previous section for details | - | `if(3<4;5;6)` evaluates to 5 |

| Function | Synonyms | Description | Parameter 1 | Parameter 2 | Example |
|---|---|---|---|---|---|
| checksum | cs | calculates checksum of given numeric value. Calculates lettervalue if given paramter is of type text | positive integer or text | - | checksum(345) evaluates to 12 |
| ichecksum | ics | calculates iterative checksum of given numeric value. Starts from lettervalue if given parameter is of type text | positive integer or text | - | ichecksum(345) evaluates to 3 |
| lettervalue | lv, wordvalue, wv | calculates lettervalue of given string value | string | - | lettervalue('test') evaluates to 64 |
| rot | - | calculates rotated string of given string value | string | count to rotate by | rot('abc'; 1) evaluates to 'bcd' |
| rot13 | - | calculates rotated-13 of given string value | string | - | rot13('abc') evaluates to 'nop' |
| roman | - | scans a given string value as a roman number and returns its decimal value | string | - | roman('VI') evaluates to 6. |
| vanity | vanitycode, vc | returns the vanity code of a string | string | - | vanity('cgeo') evaluates to 2436. |

## Variables

Variables are used in a formula as placeholders for values. When a formula containing a variable is evaluated, it needs to be passed a value for each of the contained variables in order to be correctly evaluated.

Variable names are case sensitive and have to start with an alphanumeric char. Remaining chars can be alphanumeric or digits. Examples for legal variable names are: `Test`, `T1`, `t`, `Tt123`. Examples for non-legal variable names are: 1a, 2

One-letter-variables can just be typed into the formula and will be evaluated along. For example, the formula A + 2 is valid. If A has the value 5, the formula will evaluate to 7.

If multiple chars are concatenated within a formula, they will be interpreted as individual one-letter-variables. For example, the formula AA + 2 will be interpreted as variable A concatenated two times and adding 2 afterwards. If A=4, this formula will evaluate to 44 + 2 = 46. See following section for more details wrt concatenation.

Variable names longer than one char can be declared in Unix-Bash-Style by prepending their name with $. For example, a variable named `Test` is can be referenced using `$Test`. The formulua `$Test + 2` is valid. If value for variable `Test` is 4 the formula will evaluate to 6.

In situations where variable name conflicts with following alphas/chars, the variable name can be enclosed in {} to differentiate it from following text. For example, the following expression will concatenate the value of variable `Test` with the value of variable A: `${Test}A`

Some more complex examples:

- The formula `A + $A * $Test - t` uses three variables named A, `Test` and `t`. The variable A is used in two places. Assuming A=2, `Test`=3 and `t`=1, the formula would evaluate to 7.

- The formula `AA + b + $A1` uses three variables A, b and A1. Assuming A=2, b=3 and A1=4, the formula would evaluate to 29 ( = 22 + 3 + 4)
- The formula `AB(B+1)` uses two variables A and B. Assuming A=2 and B=3, the formula would evaluate to 234
- The formula `$AB(B)(B+1)` uses two variables AB and B. Assuming AB=2 and B=5, the formula would evaluate to 256
- Using `{}` syntax, the previous example could also be written like this: `${AB}B(B+1)`

### Concatenations

If multiple expressions are concatenated directly after another with no separating operator, values are concatenated to a consecutive expression. This expression evaluates to a number if it forms a valid numeric expression, otherwise it evaluates to a text value.

Expressions, which can be concatenated, include e.g. integer digits, variables, expressions in parenthesis and the Overflow character (see next subsection).

For example, the formula `AA(A+4)55$Test(3)` contains two variables A and `Test`. Assuming A=9 and `Test=70`, it would evaluate to `991355703`.

### Overflow character

In concatenated expressions, the character _ can be used as an overflow sign. It is a placeholder for possible spillovers if numeric variables evaluate to a value with more than one digit, otherwise it is filled with 0.

An example should make the usage clear:

- The Formula 1A with A=2 will evaluate to 12
- The Formula 1_A with A=2 will evaluate to 102
- The Formula 1_A with A=23 will evaluate to 123
- The Formula 1__A with A=23 will evaluate to 1023
- The Formula 1__A with A=234 will evaluate to 1234

### Range expressions

You can specify ranges in formulas using `[]`. This is needed when variables are used in a context where a range of values should be iterated over. A prominent example is the Generate Waypoints function.

Link to anchor on waypoint calc page as soon as its updated to cover waypoint generation with ranges.

An example for a range expression is `[0-9]`. This specifies a range with 10 values (the integer values 0 to 9).

You may specify consecutive values using `,` as a delimiter. You may exclude values or value ranges by prepending a `^` to it. Ranges are parsed from left-to-right, giving an order to the elements in the range. For example the following are valid range specifications:

- `[0-2, 4]` evaluates to a range containing `0`, 1, 2 and 4.
- `[0-3, ^1-2]` evaluates to a range containing `0` and 3.
- `[0-3, ^1-2, 5]` evaluates to a range containing `0`, 3 and 5.

When a range is used in a context where only one value is allowed (this is the case in normal calculation), the first range value is used for calculation. For example, the expression `[0-9]` will evalulate to `0` in a normal

calculation context, while `[8, 0-9]` will evalulate to 8.

Ranges currently support only positive constant integer values. A range must always be evaluate to at least 1 value and a range may not evaluate to more than 20 values. For example the following ranges are invalid:

- `[]`: empty
- `[5, ^0-9]`: evaluates to empty
- `[0-1000]`: evaluates to more than 20 entries
- `[-5]`: negative int not allowed
- `[A]`: variables not allowed

A formula may include one or more range definitions mixed with normal other formula parts. For example the following formulas are valid:

- `3*[0-2]`: evaluates to values `0`, 3 and 6
- `A*[4, 7]`: for A=3 this evaluates to values 12 and 21
- `[1-2]*[3-4]`: evaluates to 3, 6, 4 and 8.

## Comments

You may enter comments into formula expressions using the # character. Comments end at next # or at end of expressions. Everything in a comment is ignored during evaluation. For example:

- `A * 5 # test comment` for A=3 evaluates to 15
- `3.14 # this is pi # * 2 # and this is two` evaluates to `6.28`