Table of Contents

Cache-Variablen 2
Einleitung 2
Kontrollbereich 3
Variablenbereich 4
Formelsyntax 5
Typen von Variablen 6
Numerische Operatoren 6
Vergleichende Operatoren und Bedingungen 6
Functions 7
Variables 8
Concatenations 9
Overflow character 9
Range expressions 9
Comments 10

Diese Seite wurde noch nicht vollständig übersetzt. Bitte helfen Sie bei der Übersetzung.
(diesen Absatz entfernen, wenn die Übersetzung abgeschlossen wurde)

Cache-Variablen

Einleitung

c:geo bietet für jede Cache-Detail Ansicht einen Karteireiter mit dem Namen “Variablen”, der es ermöglicht Variablen und Formeln, die man für den Cache benötigt, zu notieren und Berechnungen mit ihnen durchzuführen.

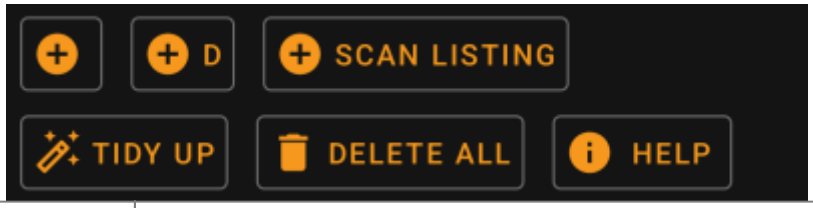
Dies kann praktisch sein, wenn du z.B. einen Multi-Cache suchst, der es erfordert draußen im Feld bestimmte Werte zu sammeln und mit diesen mathematische Berechnungen durchzuführen um zur nächsten Station oder zum Final zu gelangen.

Du kannst diese Seite mit Variablen entweder nur für sich als Helfer für Berechnungen nutzen, oder du kannst auch jede Variable, die dort definiert ist in einem berechneten Wegpunkt für diesen Cache weiternutzen.

Der folgende Abschnitt dieser Seite beschreibt den Inhalt und die Funktionen der Variablen-Ansicht.

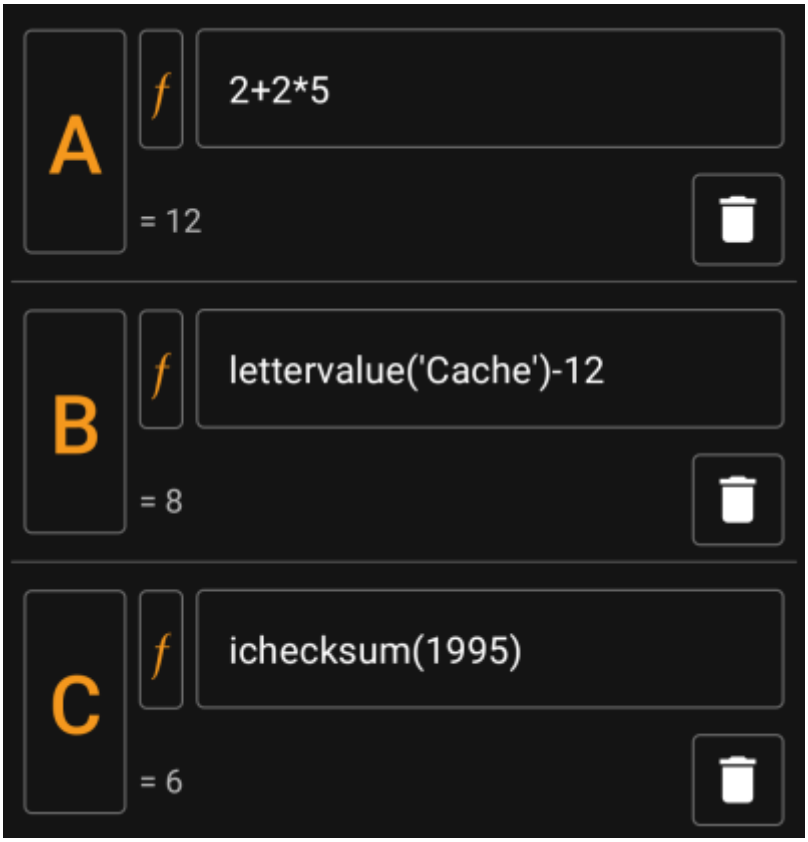
Kontrollbereich

Oben in der Ansicht sieht du einige Schaltflächen, die dir Funktionen anbieten um den darunter befindlichen Variablenbereich zu befüllen:



Schaltfläche	Beschreibung
	Eine selbst zu benennende Variable manuell zur Ansicht hinzufügen.
	Automatisch die nächste freie Variable (in alphabetischer Folge) zur Ansicht hinzufügen.
	Dieser Funktion scannt die Geocache-Beschreibung nach potentiellen Formeln und bietet an, diese in den Variablenbereich zu übernehmen. Jede ausgewählte gefundene Formel wird als Inhalt einer neuen Variable übernommen.
	Dies entfernt alle Variablen, die keinen Wert oder Formel enthalten. Die Funktion kann z.B. genutzt werden, wenn du aus Versehen zu viele Variablen angelegt hast oder du einige geleert hast, weil du sie nicht mehr benötigst.
	Dies löscht alle angelegten Variablen und ihre Inhalte.
	Öffnet diese Seite im Browser.

Variablenbereich



Dieser Bereich ermöglicht es einen Wert oder eine Formel für die erstellen Variablen einzugeben. Du kannst hier die folgenden Aktionen durchführen:

Schaltfläche	Aktion
	Tippe auf den Namen der Variable um ihn zu ändern.
	Tippe auf diese “Funktions”-Schaltfläche, um das Wertefeld mit einer der unterstützten Funktionen zu befüllen.
	Trage hier manuell einen Wert oder eine Formel unter Nutzung der Formelsyntax ein.
	Nutze das Papierkorb-Symbol um die Variable zu löschen.

Der Text unter dem Variablenfeld zeigt eine Vorschau des Ergebnisses. Dies ist entweder das konkrete Ergebnis der Formel oder gibt Hinweise bzgl. Syntax-Fehlern oder fehlenden Werten.

Formelsyntax

Das Wertefeld jeder Variable kann verschiedene Arten von Werten und auch andere Variablen enthalten. Es unterstützt vielfältige mathematische Operationen sowie einige (teilweise geocaching-bezogene) numerische und alphanumerische Funktionen, die im Folgenden beschrieben werden.

Fürchte dich nicht vor der Syntax. Sie unterstützt zwar auch relativ komplexe Operationen, can

aber auch für einfache Kalkulationen genutzt werden, wie du sie von jedem Taschenrechner kennst. Einige der Funktionen sind wahrscheinlich nur für fortgeschrittene Nutzer.

Die Syntax wird in den folgenden Unterkapiteln im Detail beschrieben. Als erste Übersicht über die unterstützten Funktionen, findest du hier eine Liste von Beispielen:

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

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

- `A + A*2` with `A=3` ergibt 9
- `AA(A+1)` with `A=3` ergibt 334
- `$hello + 1` mit der Variable `hello=24` ergibt 25
- `$hello(A+1)` mit `A=3` und `hello=24` ergibt 244
- `${hello}8A` mit `A=3` und `hello=24` ergibt 2483

Typen von Variablen

Die Formelsyntax unterstützt drei Typen von Variablen. Du kannst einfach drauf los schreiben, generell versucht die Formelberechnung die gegebenen Werte so gut wie möglich einzupassen.

Typ	Beschreibung	Anwendung	Bedeutung
Integer	Zahlen ohne Dezimalstellen	Nutze Zahlen	`1234`, `-3`
Dezimal	Zahlen mit Dezimalstellen	Nutze Zahlen mit Punkt oder Komma	`3.14`, `-3.14`, `3,14`
String	Text	Umgebe Text mit `'` oder `"` Um die Symbole `'...'` oder `"..."` selbst zu nutzen, gib `''` oder `""` ein	`'test'`, `"test"` `“Er sagt ”“ja”“!”`

Numerische Operatoren

Die folgenden numerischen Operationen werden unterstützt:

Operator	Funktion	Beispiel
+	Addition	2+4 ergibt 6
-	Subtraktion (oder eine Zahl negieren)	6-4 ergibt 2 `-(5-2)` ergibt `-3`
*	Multiplikation	3*4 ergibt 12
/	Division	12/3 ergibt 4
%	Modulo	12%5 ergibt 2
^	Potenzieren	3^3 ergibt 9
!	Faktorisieren	`4!` ergibt 24

Vergleichende Operatoren und Bedingungen

Vergleichende Operatoren wie < or == können dazu verwendet werden, um zwei Werte miteinander zu vergleichen. Generell ergibt eine solche Operation den Wert `1`, wenn der Vergleich wahr ist und den Wert `0`, wenn der Vergleich falsch ist.

Beispielsweise ergibt der Ausdruck `3 < 4` den Wert 1.

Vergleichende Operatoren werden insbesondere in der `if`-Funktion verwendet. Dieser Funktion berechnet ihren ersten Parameter. Wenn der Parameter wahr ist (bedeutet: Ergebnis ist >0 oder ein nicht leerer String) gibt die Funktion ihren zweiten Parameter zurück. Im anderen Fall, und wenn sie einen dritten Parameter hat, wird dieser dritte Parameter zurückgegeben.

Die `if`-Funktion akzeptiert jede Anzahl von Parametern und interpretiert diese in einer “if-then-if-then-if-then-...-else”-Kaskade.

Dies bedeutet, dass z.B. wenn die Funktion 5 Parameter hat: * Wenn der erste Parameter wahr ist, wird der zweite zurückgegeben * Andernfalls, wenn der dritte Parameter war ist, wird der vierte Parameter zurückgegeben. * Andernfalls wird der fünfte Parameter zurückgegeben.

Beispielsweise ergibt `if(A==5;50;A==4;40;30)` den Wert `50` wenn `A=5` ist, `40` wenn `A=4` ist und `30` für jeden anderen Wert von `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	Parameter1	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	Parameter1	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`