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
Funktionen 7
Variablen 8
Verkettungen 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.

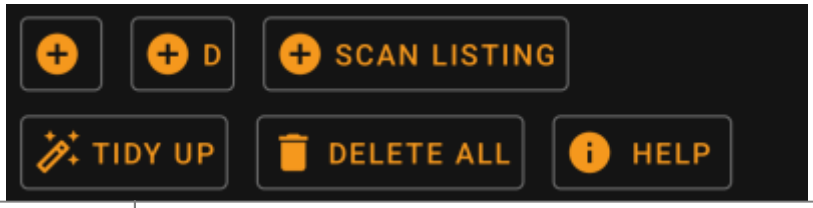
Dies kann praktisch sein, wenn du z.B. einen Multi-Cache suchst, der es erfordert draußen im Feld bestimmte Werte zu sammeln und mit diesen mathematische Berechnungen durchzuführen um zur nächsten Station oder zum Final zu gelangen.

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.

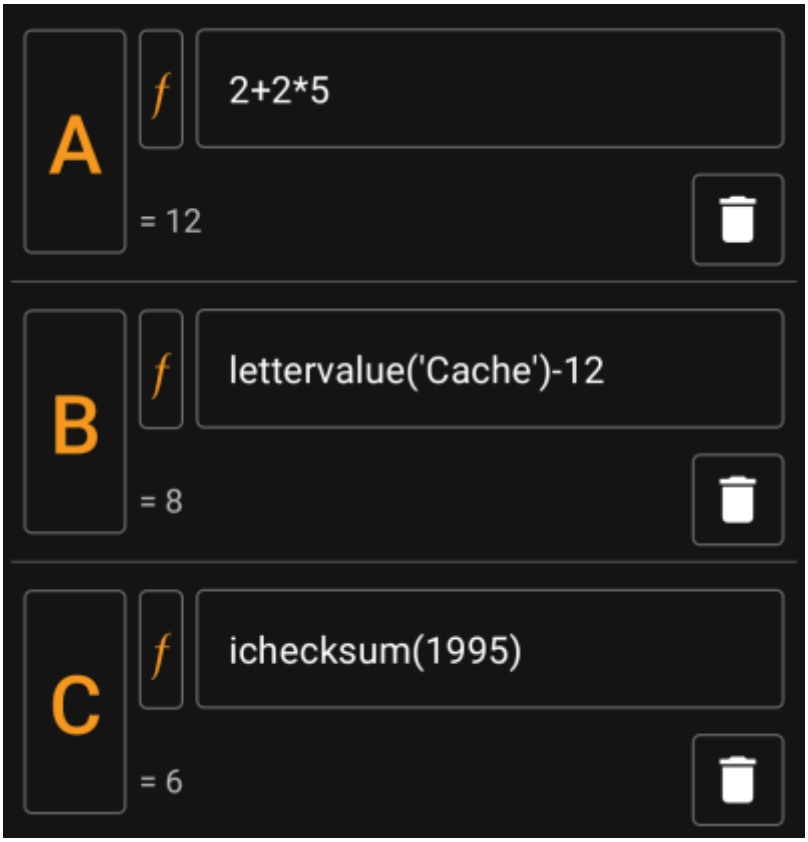
Kontrollbereich

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
Ganzzahlen (Integer)	Zahlen ohne Dezimalstellen	Nutze Zahlen	`1234`, `-3`
Dezimalzahlen (Decimal)	Zahlen mit Dezimalstellen	Nutze Zahlen mit Punkt oder Komma	`3.14`, `-3.14`, `3,14`
Text (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	Bedeutung	Beispiel
==	Prüft Gleichheit	2==2 ergibt 1(=wahr)
<>	Prüft Ungleichheit	3<>2 ergibt 1(=Wahr)
<	Ist kleiner als	3<4 ergibt 1(=wahr)
⇐	Ist kleiner oder gleich als	3⇐3 ergibt 1(=wahr)
>	Ist größer als	3>4 ergibt 0(=falsch)
>=	Ist größer oder gleich als	5>=5 ergibt 1(=wahr)

Funktionen

Funktionen starten alle mit einem Buchstaben, enthalten nur Buchstaben oder Ziffern und haben eine direkt daran angeschlossene Liste von Parametern, die in Klammern stehen. Mehrere Parameter werden dabei durch `;` getrennt.

Ein Beispiel für einen Funktionsaufruf mit einem Parameter ist `sin(90)`. Ein Beispiel für einen Funktionsaufruf mit zwei Parametern ist `rot('test'; 13)`.

Die folgenden Funktionen sind definiert:

Funktion	Synonym	Beschreibung	Parameter 1	Parameter 2	Beispiel
`sqrt`	-	Berechnet die Quadratwurzel des gegebenen Parameters	Numerischer Parameter	-	`sqrt(9)` ergibt 3
`sin/cos/tan`	-	Berechnet Sinus/Cosinus/Tangens des gegebenen Parameters	Numerischer Parameter in Grad(!)	-	`sin(90)` ergibt 1
`abs`	-	Berechnet den absoluten Wert	Numerischer Parameter	-	`abs(-34)` ergibt 34
`round`	-	Rundet Dezimalwerte mathematisch	Zu rundender Wert	Optional: Anzahl der Stellen für die Rundung	`round(4.65)` ergibt 5, `round(4.65;1)` ergibt `4.7`

Funktion	Synonym	Beschreibung	Parameter 1	Parameter 2	Beispiel
`if`	-	Wertet Bedingungen aus und gibt davon abhängige Werte zurück	Liste von wenn-dann-sonst Werten. Siehe vorheriges Kapitel für Details.	-	`if(3<4;5;6)` ergibt 5
`checksum`	`cs`	Berechnet die Quersumme des angegebenen numerischen Wertes. Berechnet den Buchstabenwert, wenn der angegebene Parameter ein Text ist	Positive Ganzzahlen oder Text	-	`checksum(345)` ergibt 12
`ichecksum`	`ics`	Berechnet die absolute/iterierte Quersumme des angegebenen numerischen Wertes. Startet beim Buchstabenwert, wenn der angegebene Parameter ein Text ist.	Positive Ganzzahl oder Text	-	`ichecksum(345)` ergibt 3
`lettervalue`	`lv`, `wordvalue`, `wv`	Berechnet den Buchstabenwert des angegebenen Textes	Text	-	`lettervalue('test')` ergibt 64
`rot`	-	Berechnet einen rotierten Text aus dem angegebenen Text	Text	Anzahl der zu rotierenden Stellen	`rot('abc'; 1)` ergibt `'bcd'`
`rot13`	-	Berechnet den ROT-13 Wert aus dem angegebenen Text	Text	-	`rot13('abc')` ergibt `'nop'`
`roman`	-	Durchsucht einen gegebenen Text nach römischen Zahlen und gibt den dezimalen Wert zurück	Text	-	`roman('VI')` ergibt 6.
`vanity`	`vanitycode`, `vc`	Gibt den Vanity-Code eines Textes zurück	Text	-	`vanity('cgeo')` ergibt 2436.

Variablen

Variablen werden in Formeln als Platzhalter für Werte verwendet. Wenn eine Formel berechnet wird, die eine Variable enthält, benötigt diese einen Wert für jede enthaltene Variable, damit sie korrekt berechnet wird.

Namen von Variablen unterscheiden Groß-/Kleinschreibung und müssen mit einem Buchstaben beginnen. Die restlichen Zeichen können Buchstaben oder Ziffern sein. Beispiel für gültige Variablennamen sind: `Test`, `T1`, `t`, `Tt123`. Beispiele für ungültige Variablennamen sind: `1a`, `2`.

Variablen, die nur aus einem Buchstaben bestehen, können einfach in Formeln eingegeben werden und werden direkt ausgewertet. Zum Beispiel ist die Formel `A + 2` gültig. Wenn `A` den Wert 5 hat, ergibt die Formel 7.

Wenn mehrere Buchstaben in einer Formel direkt hintereinander geschrieben werden, werden diese als Wiederholung dieser Ein-Buchstaben-Variable angesehen. Zum Beispiel wird die Formel `AA + 2` als zweifache Verwendung der Variable `A` interpretiert und die 2 anschließend addiert. Wenn `A=4` ist, ergibt diese Formel

somit `44 + 2 = 46`. Das folgende Kapitel gibt mehr Informationen zum Thema Verketten von Variablen.

Namen von Variablen mit mehr als einem Buchstaben können im Unix-Bash-Stil verwendet werden, in dem ihnen ein $ vorausgestellt wird. Beispielsweise kann eine Variable mit der Bezeichnung `Test` verwendet werden, in dem in Formeln `$Test` eingesetzt wird. Die Formel `$Test + 2` ist somit gültig. Wenn die Variable `Test` gleich 4 ist, ergibt die Formel eine 6.

In Situationen, wo Variablennamen in Konflikt mit darauffolgenden Buchstaben stehen, können die Variablennamen in `{}` gesetzt werden, um diese vom folgenden Text zu trennen. Beispielsweise verkettet der folgende Ausdruck den Wert der Variable `Test` mit dem Wert der Variable A: `${Test}A`

Einige weitere komplexe Beispiele:

- Die Formel `A + $A * $Test - t` nutzt drei Variablen, nämlich A, `Test` und `t`. Die Variable A wird an zwei Stellen verwendet. Angenommen A=2, `Test`=3 und `t`=1 ergibt die Formel 7.
- Die Formel `AA + b + $A1` nutzt drei Variablen, nämlich A, b und `A1`. Angenommen A=2, b=3 und `A1`=4 ergibt die Formel 29 (= 22 + 3 + 4)
- Die Formel `AB(B+1)` nutzt zwei Variablen, nämlich A und B. Angenommen A=2 und B=3 ergibt die Formel 234
- Die Formel `$AB(B)(B+1)` nutzt zwei Variablen, nämlich AB und B. Angenommen AB=2 und B=5ergibt die Formel 256
- Nutzt man die `{}`-Syntax könnte das vorherige Beispiel auch wie folgt geschrieben werden: `${AB}B(B+1)`

Verkettungen

Wenn mehrere Ausdrücke direkt hintereinander ohne trennenden Operator verkettet werden, werden deren Werte ebenso zu einem fortlaufenden Ausdruck verkettet. Dieser Ausdruck ergibt eine Zahl wenn es eine gültiger numerischer Ausdruck ist, sonst zu einem Textwert.

Ausdrücke, die verketten werden können, sind z.B. Ganzzahlen, Variablen, Ausdrücke in Klammern und das Überlaufzeichen (siehe nächstes Unterkapitel).

Beispielsweise enthält die Formel `AA(A+4)55$Test(3)` zwei Variablen A und `Test`, Unter der Annahme, dass A=9 und `Test`=70 ist, ergibt sich das Ergebnis der Formel als `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`