Special Characters

.	Decimal point – depending on the number format settings in the System Preferences		
,	Decimal comma – depending on the number format settings in the System Preferences		
;	List separator		
( )	Parenthesis		
´	Number of decimal digits:	π´6	→ 3.141593
`	Number of significant digits:	10*π`4	→ 31.42
"	Initial of a comment:	sin(π/3) "a comment…	
?	Shows associated description:	?BRL	→ Brazilian Real
	of currency symbols:	?USD	→ U.S. Dollar
%	Percent postfix operator – may be attached directly to any number and has the immediate effect of dividing that number by 100 – of course, that is exactly what per cent means, e.g.:	76%	→ 0.76
	If Lazy % Syntax is enabled in the InstantCalc Preferences Panel (default), then the % operator allows for lazy addition/subtraction of the percentage of the left operand to/from itself:	100 + 5%	→ 105
		100 - 5%	→ 95
	If Lazy % Syntax is disabled, additions and subtractions behave strictly mathematical:	100 + 5%	→ 100.05
		100 - 5%	→ 99.95

Hexadecimal and Binary Conversion Operators

$	Hexadecimal number prefix:	$20 + 5 → 37
0x	Alternate hexadecimal prefix:	0x7FFF → 32767
&	Hexadecimal result format:	&10 + $1E*5 → $A0
§	Alternate hex format:	§10 + $1E*5 → 0xA0
#	Binary number prefix:	#11100101+5 → 234
\	Binary result output:	\123 → #1111011

Note: Above hexadecimal and binary result output formatting operators (&, §, \) must prefix the expression and work on integer values only, i.e., floating point values are not formatted: &12.3 = \12.3 → 12.3

Display of Fractions

ƒ convert result to a fraction: ƒ0.1875 → 3/16
ƒ must be the first character of the expression.

Note: " " (<alt><space> = NO BREAK SPACE) substitutes "**+**"
ƒ11.5 → 11 1/2
11 1/2 → 11.5

Operator Precedence

InstantCalc basically adheres to the standard calculation rules as nowadays trained in the math courses of elementary and/or secondary schools, e.g.: 1 + 2 * 3 = 3 * 2 + 1 = 1 + (2 * 3) → 7.

According to these rules, the multiplication has a higher precedence than the addition, and consequently 2 * 3 is evaluated before the multiplication result is added to 1. Therefore, there is no need of grouping the multiplication by parenthesis in the above example. The natural result is 7 in each notation.

For forcing another than the natural evaluation order, parenthesis must be used, therefore, for forcing the addition to being evaluated first, the expression needs to be (1 + 2) * 3 ➞ 9.

Besides honouring the standard precedence rules, InstantCalc evaluates everything strictly from left to the right. Note, that in some cases left to right evaluation is the same as right to left evaluation, e.g.: 1 + 2 + 3 = 3 + 2 + 1 ➞ 6.

In other cases, it makes a difference, and we need to use parenthesis for forcing a different evaluation sequence, if desired.

100 + 25% + 10% ➞ 137.5
100 + (25% + 10%) ➞ 135

2^3^4 = (2^3)^4 ➞ 4096
2^(3^4) ➞ 2.417851639e+24

Take care!

Dyadic Operators – by ascending precedence level

Level 0 – weakest binding of the operands to the operator

:= Variable assignment operator: x := 5; (x + 2)*3 ➞ 21

Variables keep their definition even upon restart until they are undefined with x := none.

Variables can also be added, removed, and changed in the User Variables panel.

Level 1 – for equations

= Equation operator, i.e. subtraction with lowest precedence. This is especially useful in combination with the

Newton Solver, in order to subtract the right side of an equation from the left side, e.g.:

solve(x := 3; x^2 - 3*x + 2 = 12) → 5

Level 2

or	Logical OR:	true or false → true
||		3.4 || 0 → true

Level 3

xor	Logical XOR:	true xor true → false
		2 xor 0 → true

Level 4

and	Logical AND:	true && false → false
&&		2 and 1 → true

Level 5

|	Bitwise OR:	5 | 3 → 7

Level 6

≈	Bitwise XOR:	5 ≈ 3 → 6

Level 7

&	Bitwise AND:	5 & 3 → 1

Level 8 (Comparisons)

==	is equal:	sin(π) == cos(π) → false
!=	is not equal:	0 != 0.0 → false
<>		1 <> 2 → true
<	is less than:	have(3;6;7;9) < ave(3;6;7;9) → true

<=	is less or equal than:	exp(2) <= 10 → true
>	is greater than:	5 > 3 → true
>=	is greater or equal than:	stdev(3;6;7;9) >= 2 → true

Level 9

<<	Bitwise SHL (shift left):	128 << 2 → 512
>>	Bitwise SHR (shift right):	32 >> 4 → 2

Level 10

+	Addition	1 + 2 → 3
-	Subtraction	178 - 13.5 → 164.5

Level 11

* ·	Multiplication:	1*1 → 1 or 2·2 → 4
/	Division:	π/4 → 0.785398163…
div	Integer division:	17 div 5 → 3
mod	Modulo:	17 mod 5 → 2

Level 12

^	Power: evaluates left to right:	3.5^5 → 525.21875 2^3^2 = (2^3)^2 → 64
√	Root	3√5 → 1.709975947…

Level 13 – very strong binding of the operands to the operator

@	2D distance √$(x^2+ y^2)$:	x:=6; y:=8; x @ y → 10 x @ y = 2√(x^2 + y^2)

Monadic Operators – ultimate precedence *Level 14*

-	Negation:	x := 6; -x → -6
√	Square Root:	√3 = sqrt(3) = 2√3 → 1.73205…
! **not**	Logical NOT:	!true → false not false → true !0 → true !0.0 → true not 5.5 → false !inf → false !nan → nan

Angle conversions – useful for the trigonometric functions:

°	Radian → Degree: °x = x·180/π	°0.7853981634 → 45
®	Degree → Radian: ®x = x·π/180	®90 → 1.570796327

Dots per inch conversions – useful in metric applications:

ç	pt → cm:	*çx= x·2.54/72*	ç96 → 3.386666667
¶	cm → pt:	*¶x = x·72/2.54*	¶1.5 → 42.51968504

~	Bitwise inverse:	~5 → -6

All the bitwise operators can only be used with the 64 bit integer operands, and it affect ALL bits, including the sign bit and the leading zeros. For this reason a bitwise inverse of a positive number gives allways a negative number and vice versa:

\~5 → #11111111111111111111111111111111
11111111111111111111111111111010

Common Functions with one Argument

abs(x) Absolute value: abs(-5) = abs(5) → 5

round(x) to integer:
round(1.5) → 2
round(1.3) → 1
round(2.5) → 3
round(-3.5) = -round(3.5) → -4

trunc(x) Truncate to integer:
trunc(1.5) → 1
trunc(-3.5) = -trunc(3.5) → -3

sqrt(x) Square root: sqrt(2) = √2 → 1.414213562

sqr(x) Power of 2 x^2: sqr(1 + 2) = 3*3 → 9

cube(x) Power of 3 x^3: cube(-5) = -5^3 → -125

sign(x) Signum function {-1 for x < 0; 0 for x = 0; 1 for x > 0}
sign(-123) → -1
sign(ln(1)) → 0
sign(inf) → 1
sign(nan) → nan

Exponential Functions

exp(x) Base e e^x: exp(1) → 2.718281828

expm1(x) Base e $e^x - 1$: expm1(1) → 1.718281828

exp10(x) Base 10 10^x: exp10(4.5) → 31622.7766

exp2(x) Base 2 2^x: exp2(12) → 4096

Logarithms

ln(x)	Base e (**l**ogarithmus **n**aturalis):	ln(2) → 0.6931471806
ln1p(x)	Base e:	ln1p(5) = ln(5 + 1) → 1.7917595
lg(x)	Base 10:	lg(1000) → 3
lb(x)	Base 2:	lb(64) → 6

Trigonometric Functions – Arguments in Radian

sin(x)	Sine:	sin(π/3) → 0.866025404
cos(x)	Cosine:	cos(®45) → 0.707106781
tan(x)	Tangent:	tan(π/8) → 0.414213562
cot(x)	Cotangent:	1/tan(π/8) = cot(π/8) → 2.414213562

Inverse Trigonometric Functions – Results in Radian

asin(x)	Inverse Sine:	asin(0.5) → 0.5235987756
acos(x)	Inverse Cosine:	°acos(0.5) → 60
atan(x)	Inverse Tangent:	°atan(inf) → 90
acot(x)	Inverse Cotangent:	°acot(1) → 45
atan2(y; x)	Inverse tan with 2 arg's:	atan2(1.5;3) → 0.463...

Hyperbolic Functions

sinh(x)	Hyperbolic Sine:	sinh(-2.5) → -6.050204481 sinh(2.5) → 6.050204481
cosh(x)	Hyperbolic Cosine:	cosh(-1.5) → 2.352409615 cosh(1.5) → 2.352409615
tanh(x)	Hyperbolic Tangent:	tanh(0.5493061443) → 0.5 tanh(-inf) → -1 tanh(inf) → 1
coth(x)	Hyperbolic Cotangent:	coth(0.5493061443) → 2

Inverse Hyperbolic Functions

asinh(x)	Inv. Hyperbolic Sine:	asinh(-6.05020448) → -2.5 asinh(6.050204481) → 2.5
acosh(x)	Inv. Hyperbolic Cosine:	acosh(2.352409615) → 1.5
atanh(x)	Inv. Hyperbolic Tangent:	atanh(0.5) → 0.5493061...
acoth(x)	Inv. Hyperbolic Cotangent:	acoth(2.0) → 0.5493061...

Other Functions with one Argument

Temperature conversions

FtoC(f)	Fahrenheit → Celsius:	FtoC(86) → 30
CtoF(c)	Celsius → Fahrenheit:	CtoF(37.5) → 99.5

Angle conversions – useful for the trigonometric functions

dgr(x) Radian → Degree $x \cdot 180/\pi$: dgr(asin(0.5)) → 30

rad(x) Degree → Radian: $x \cdot \pi/180$: cos(rad(60)) → 0.5

Factorial/Permutation functions – the results of are straight integer numbers for integer arguments $x \le 20$ only, otherwise, the non-integer real results of the gamma functions *gamma(x + 1)* or *gammaLn(x + 1)* are returned

fact(x) Factorial *x!*: fact(6) → 720
fact(6.01) → 733.617

factLn(x) Log-Factorial: factLn(18) = ln(fact(18)) → 36.3954

perm(n) Permutations *n!*: fact(8) = perm(8) → 40320

gamma(x) Gamma function: gamma(5.5) → 52.343

gammaLn(x) Log-Gamma function:
gammaLn(21.5) = ln(gamma(21.5)) → 43.85192586

erf(x) Error function: erf(3) → 0.9999779095

prob(s) Probability function, i.e. the integral of the Standard Normal (Gaussian) Distribution in the range of ±*s*, see also *gauss()*: prob(1) → 0.6826894921
prob(2) → 0.9544997361

Standard deviation that covers 95 % of all occasions:
solve(s := 2; prob(s) = 95%) → 1.959963985

Functions with Multiple Number of Arguments

root(n; x) nth Root $\sqrt[n]{x}$: root(5; 243) → 3

scalb(x; n) Binary scale $x \cdot 2^n$: scalb(64; 10) → 65536

pyth2D(x; y) Geometric distance $\sqrt{(x^2 + y^2)}$ – 2 dimensions
Pythagoras formular: pyth2D(1; 2) → 2.2361

pyth3D(x;y;z) Geometric distance $\sqrt{(x^2 + y^2 + z^2)}$ – 3 dimensions
Pythagoras formular: pyth3D(1;2;3) → 3.7417

gauss(x; m; s) Normal (Gaussian) distribution density function around mean *m* with standard deviation *s*:
gauss(3; 2; 1) → 0.2419707245

Note, this is the value at abscissa *x* of the gauss function, characterized by its mean m and standard deviation *s*. If we are looking for a probability value of events falling into a given range, then we need to integrate the Gaussian function in that range:

∫(z := 1; z; gauss(z; 2; 1)) → 0.6826894921 ≈ 68.3%

As a matter of fact, 68.3 % is the well known probability of an event falling into ±1 standard deviation. The Gaussian Distribution plays an essential role in all sorts of statistical analysis. For further reading see the respective Wikipedia Article.

gaussLn(x;m;s) Log-Gaussian distribution density function around mean *m* with standard deviation *s*, evaluated at $ln(x)$: gaussLn(3;2;1) → 0.088584

gcd(m; n) Greatest common divisor: gcd(528;112) → 16

lcm(m; n) Least common multiple: lcm(528;112) → 3696

comb(n; k) Combinations (*k* out of *n*): comb(49;6) → 13983816
i.e. binomial coefficient $n!/[k! \cdot (n-k)!]$

var(n; k) Variations $n!/(n-k)!$: var(4;2) → 12

compound(r; n) Compounded interest $(1 + r)^n$:
e.g. \$20000 at 4.5 %/year after 60 month:
20000*compound(4.5%/12; 60) → \$25035.92

annuity(r; n) Annuity (down payment) $(1 - (1+r)^{-n})/r$:
e.g. \$20000 at 4.5 %/year to be down paid in 60 m:
20000/annuity(4.5%/12; 60) → \$372.86/month

if(cnd; r1; r2) Conditional evaluation

- *r1* if condition is true: if(true; 1; 2) → 1
- *r2* if condition is false: if(false; 1; 2) → 2

if(rand < 0.5; 0; H*2+O) → 18.015
if(rand < 0.5; 0; H*2+O) → 0
if(rand < 0.5; 0; H*2+O) → 0
if(rand < 0.5; 0; H*2+O) → 18.015

linpol() Linear interpolation linpol(x; x1;y1; x2;y2)

Results in the value *y* at the position *x* on the straight line defined by the two points *x1/y1* and *x2/y2*. This is a shortcut of the general linear interpolation formula $(y2 - y1)/(x2 - x1)*(x - x1) + y1$:

linpol(80.37; 60;5.41; 100;4.79) → 5.094265

(4.79 - 5.41)/(100 - 60)*(80.37 - 60) + 5.41 → 5.094265

Functions with Variable Number of Arguments

sum()	Explicit summation	sum(1;2;3;4;5;6;7;8;9)	→ 45
prod()	Explicit product	prod(1;2;3;4;5;6;7;8;9)	→ 362880
ave()	Arithmetic average	ave(1;2;3;4;5;6;7;8;9)	→ 5
gave()	Geometric average	gave(1;2;3;4;5;6;7;8;9)	→ 4.14717
have()	Harmonic average	have(1;2;3;4;5;6;7;8;9)	→ 3.18137
stdev()	Standard deviation	stdev(1;2;3;4;5;6;7;8;9)	→ 2.73861
linreg()	Linear regression	linreg(x1;y1; x2;y2; …)	

Returns the correlation coefficient as its direct result. In addition, it populates user variables with the other results of the regression analysis:

a	intercept
b	slope
stdev_y	standard deviation of y estimated by using $a + b \cdot x$
stdev_a	standard deviation of the intercept
stdev_b	standard deviation of the slope

linreg(0;1; 5;1.0538; 12;1.1309; 20;1.2191; 28;1.3064; 34; 1.3696; 40;1.43; 44;1.4685; 48;1.5065; 50;1.5253) → 0.99969...

a	→ 1.004350452
b	→ 0.01055727928
stdev_y	→ 0.005058603831
stdev_a	→ 0.003066204759
stdev_b	→ 1.713096898e-06

Numerical Mathematical Methods

Numerical accuracy can be adjusted in the InstantCalc Preferences Panel. The numerical methods are invoked by function calls with two or three arguments which are separated by semicolon. The first argument denotes the control variable for the method, and usually it shall be set to the initial/start value by using the variable assignment operator *:=*. The last argument is the expression, that should be evaluated with respect to the control variable of the first argument. In the case of Integration, Summation, and Serial Product, the expression in the second argument should evaluate the stop value.

Newton Equation Solver **solve(var := init; eqL(var) = eqR(var))**

solve(u := 3; sin(u) = 0) ➝ 3.141592654

solve(v := 2; 3·v^2 + 6·v = 45) ➝ 3

The right side of the equation may be omitted, in which case the expression is resolved to zero. Therefore, the first example may be rewritten as solve(u := 3; sin(u)) ➝ 3.141592654.

The solver may fail in cases, i.e. the result is Not a Number:

- neither the left side, nor the right side of the equation is dependent on the control variable – revise the equation!
- poor initial guess – try to improve the initial guess, by playing around with different initial values of the control variable!
- in the course of the iteration, either side of the equation evaluate to non finite values, i.e. *inf* or *nan* – try to improve the initial guess and/or rewrite the equation, perhaps by using the conditional function *if(;;)* in order to avoid invalid function evaluations!

Numerical Differentiation **∆(var := poe; expression(var))**

∆(u := 1; sin(u)) ➝ 0.5403023059 ← cos(1)

∆(v := 2; 3·v^2 + 6·v) ➝ 18

The expression shall be dependent on the control variable, otherwise, the result is simply 0. At the *poe* (point of evaluation) ± the relative tolerance (s. Preferences Panel) the expression must result in finite numbers, otherwise the numerical differentiation may fail, i.e. it reports **nan** = **n**ot **a** **n**umber.

Romberg Integration **∫(var := start; stop; expression(var))**

∫(u := 0; π; sin(u)) → 2

∫(p := -1; 1; 1/(1 + sqr(p))) → 1.570796327 ← π/2

∫(q := -2; 2; gauss(q; 0; 1)) → 0.9544997361

Double Integration **∫(p := a; b; ∫(q := c; d; f(p; q)))**

∫(p :=-3;3; ∫(q :=-2;2; gauss(p;0;1)·gauss(q;0;1)))
→ 0.9519227815

Summation (implicit) **∑(var := start; stop; expression(var))**

∑(k := 1; 9; k) → 45

Saving $750 per month over 10 years at 6.25 %/year
∑(k := 0; 119; 750*(1 + 6.25%/12)^k) → 124591.42

the above gives the same result as:
750*annuity(6.25%/12;120)*compound(6.25%/12;120)
→ 124591.42

Serial Product (implicit) **∏(var := start; stop; expression(var))**

∏(k := 1; 9; k) → 362880

Encapsulation of the numerical methods into each other, for example for double integration, or for using integration, differentiation, summation, and serial product together with the Newton Solver, is well possible:

∫(u := -1; 1; ∫(v := -1; 1; pyth2D(u; v)))
→ 3.060782866

solve(p := 1.5; ∫(q := -p; p; gauss(q;0;1)) = 0.95)

→ 1.959963985

solve(x := 2; Δ(d := x; π/2·sqr(d) + 4/d) = 0)
→ 1.08385214

solve(dp := 1000; 50000·(1+6.25%/12)^48
- dp·∑(k := 0; 47; (1+6.25%/12)^k)) → 1179.990993

50000/annuity(6.25%/12; 48) → 1179.990993

A Function without Argument

rand Equally distributed random numbers between 0 and 1
∑(k := 1; 1000000; rand)/k ≈ 0.5 ± 0.0002

Complex Number Calculations

Complex numbers are usually denoted as a sum of their real and imaginary part, i.e.: *re + i·im*

For a calculator like InstantCalc, that deals with any sort of numbers, this text book notation is not quite appropriate, as the "+" sign within the complex context, while it is related, does not exactly correspond to an arithmetic addition operation.

For InstantCalc an alternative notation scheme for complex numbers has been defined. Real and imaginary part are separated by a semicolon and embraced by curly brackets: **{re; im}**

Some mathematical functions in real space are defined only for their arguments lying in a certain range. For example, the real space square root is not defined for negative values:

√-1 → **N**ot **a** **N**umber (**nan**)

On the other hand, the complex variant of the square root is defined for negative arguments:

√{-2;0} → {0; 1.414213562}

Examples for other functions that are undefined for certain ranges of arguments in the real space but may give results in the complex space are general roots of even integer power or of any real power, as well as asin(), acos(), atan(), logarithms, etc.

InstantCalc may automatically do a real to complex conversion for said kind of functions if the real arguments lie outside of the defined ranges. The automatic real to complex conversion can be enabled or disabled in the InstantCalc Preferences Panel. By default, InstantCalc does no automatic real to complex or complex to real conversions.

The following special complex functions exist. The examples use the variable definition z := {3; 1.5}.

real(z) Real part of complex numbers: real(z) → 3

imag(z) Imaginary part of complex numbers: imag(z) → 1.5

norm(z) Euclidean Norm (distance/val.) of *z*: norm(z) → 3.354...
= sqrt(sqr(real(z)) + sqr(imag(z))) → 3.354...

conj(z) Complex Conjugate of *z*: conj(z) → {3;-1.5}

arg(z) Complex Argument (angle) of *z*: arg(z) → 0.463...
= atan2(imag(z); real(z)) → 0.463...

Vector and Matrix Calculations

InstantCalc calculates expressions, each on one line of pure text. *Vectors* and *Matrices* usually consist of multiple rows and/or columns, and some definitions need to be made in order to fold the rows and columns onto a single line.

The elements of *Vectors* are separated by semicolons and all together is embraced by square brackets: **[a; b; c; …]**

Vectors may be of any dimension greater or equal to 1.

Matrices consist of rows of vectors whereby the rows are separated by semicolons and all together is embraced by square brackets. The following is an example for a

3×3 matrix: **[[a11; a12; a13]; [a21; a22; a23]; [a31; a32; a33]]**

Matrices may be of any dimensions $M \times N$ whereby M (number of rows) and N (number of columns) must be greater or equal to 1. Special cases are:

1×1 matrix: **[[a11]]**

1×N matrix: **[[a11; a12; …; a1N]]**

M×1 matrix: **[[a11]; [a21]; …; [aM1]]**

Where appropriate, InstantCalc converts 1xN or Mx1 matrices into a row vector with N elements or a column vector with M elements.

Each vector/matrix element may be of any type, including complex type, vector type, or matrix type. However, not every mathematical operation is defined with any kind of vector or matrix, and the error message "Type Mismatch" may appear when appropriate.

All scalar operators and scalar functions operate on each element of vectors and matrices, e.g.:

[1; 2; 3]*5 → [5; 10; 15]
[[3; 2]; [0; 1]]*[[1; 2]; [3; 1]] → [[3; 4]; [0; 1]]
exp([1; 2; 3]) → [2.718...; 7.389...; 20.085...]

Note that +, -, *, / are scalar operators in this respect. Special operators for Vectors and Matrices do exist:

• Operator for Vector Dot product and Matrix Product

[1;3;-5] • [4;-2;-1] → 3
[[3;2;1]; [1;0;2]] • [[1;2]; [0;1]; [4;0]] → [[7;8];[9;2]]

x Operator for Vector Cross Product
[1;2;3] x [-7;8;9] → [-6; -30; 22]

**** Operator for Matrix Division, which is not generally defined as such. In InstantCalc, Matrix Division is the shortcut for a sequence of two operations, a Matrix Inversion on the left operand (needs to be a matrix), and then the Matrix Product from the inversed matrix with the right operand (a vector or a matrix of proper dimensions). If *A* is a matrix and *b* is a vector then *A\b* would give exactly the same result as *inv(A)•b* – for examples, see the chapter Matrix Inversion below.

The Following special vector and matrix functions exist. The examples use the variable definition **A := [[1;2;0]; [2;3;0]; [3;4;1]]**

simpl(W[; thresh]) simplify a vector or matrix, i.e. recursively set all elements that are less than a threshold value to 0, and convert unidimensional vectors/matrices to scalar values, and reduce complex values whoes imaginary part is zero to real values. If the parameter *thresh* is omitted, then the machine precision *prec* is used instead:

simpl([[1;2;3]]) → [1;2;3]
simpl([[1];[2];[3]]) → [1;2;3]
simpl([1; {2;0.05}; 3]; 0.1) → [1;2;3]

elem(W; m[; n]) extract an element from a vector or a matrix:

elem(A; 2) → [2;3;0]
elem(A; 2; 1) → 2

norm(W) Euclidean Vector or Matrix Norm: norm(A) → 6.633...

trans(M) Matrix Transpose: trans(A) → [[1;2;3];[2;3;4];[0;0;1]]

det(M) Determinant of a square matrix: det(A) → -1

svd(M) Singular Value Decomposition of M, i.e. the factorization of the given matrix into $M = U\sum V^*$. Returns the vector S of the singular values on the diagonal positions of the diagonal matrix $\sum$:

svd(A) → [6.591520852;0.7116199857;0.2131897072]

inv(M) Matrix Inversion: inv(A) → [[-3;2;0];[2;-1;0];[1;-2;1]]
inv(A)•[1; -2; 5] → [-7; 4; 10]
A\[1; -2; 5] → [-7; 4; 10]

Caution: the following is not a matrix operation, but simply an element by element scalar division:

A/[1; -2; 5] → [[1;2;0];[-1;-1.5;0];[0.6;0.8;0.2]]

vec(x; n) return a vector of dimension n whose elements are set to x: vec(0.19; 3)→ [0.19;0.19;0.19]

diag(x; n) return a n x n matrix whose diagonal elements are set to x: diag(1; 3) → [[1;0;0];[0;1;0];[0;0;1]]

mat(x; m; n) return a m x n matrix whose elements are set to x: mat(2; 3; 4) → [[2;2;2;2];[2;2;2;2];[2;2;2;2]]

Indexed References into the Calculation History

Rsn *R*(esult) in the history table at index *sn*

Esn *E*(xpression) in the history table at index *sn*

Predefined Constants

true **false**	Boolean values:	1 0
c0	Speed of Light in vacuum:	299792458 m/s
e	Euler´s Number: 2.71828182845904523536028747135266	
π, pi	3.14159265358979323846264338327948	
ln10	2.30258509299404590109361379290931	
gold	Golden Ratio: 1.61803398874989484820458683436564	
∞, inf	IEEE754-infinity	1/0 → inf
nan	Not a Number (type 0)	√-1 → Not a Number
none	Not a Number (type 255)	
maxshort	32767	
maxint	2147483647	
maxlong	9223372036854775807	
minreal	3.64519953188247460 3e-4951	
maxreal	1.18973149535723176 5e+4932	
prec	Numerical precision, i.e. machine epsilon: 1.08420217248550443 4e-19	

Table of Atomic Weights of the Elements in g/mol

Compiled from:

IUPAC Technical Report - Atomic weights of the elements 2009, Pure Appl. Chem., Vol. 83, No. 2, pp. 359–396, 2011.

* Note: The standard atomic weights of 10 elements having 2 or more stable isotopes have been changed to reflect this variability of atomic weight values in natural terrestrial materials. These elements are *H, Li, B, C, N, O, Si, S, Cl, Tl*. For these selected elements, the conventional atomic weights are taken.

** Note: For radioactive/artificial elements, the atomic weights of the most stable isotope has been taken.

Usage

H*2+O	→	18.015 molecular weight of water
C*2+H*5+O+H	→	46.069 molecular weight of ethanol
H*2+S+O*4	→	98.072 molecular weight of sulfuric acid
250*Cr/(Cr+O*3)´4	→	129.9992 Cr content in 250 g chromic acid

1	**H**	→	1.008	[1.00784; 1.00811]	Hydrogen*
	He	→	4.002602		Helium
2	**Li**	→	6.94	[6.938; 6.997]	Lithium*
	Be	→	9.012182		Beryllium
	B	→	10.81	[10.806; 10.821]	Boron*
	C	→	12.011	[12.0096; 12.0116]	Carbon*
	N	→	14.007	[14.00643; 14.00728]	Nitrogen*
	O	→	15.999	[15.99903; 15.99977]	Oxygen*
	F	→	18.9984032		Fluorine
	Ne	→	20.1797		Neon

3	Na	→	22.98976928	Sodium	
	Mg	→	24.3050	Magnesium	
	Al	→	26.9815386	Aluminium	
	Si	→	28.085	[28.084; 28.086]	Silicon*
	P	→	30.973762	Phosphorus	
	S	→	32.06	[32.059; 32.076]	Sulfur*
	Cl	→	35.45	[35.446; 35.457]	Chlorine*
	Ar	→	39.948	Argon	
4	K	→	39.0983	Potassium	
	Ca	→	40.078	Calcium	
	Sc	→	44.955912	Scandium	
	Ti	→	47.867	Titanium	
	V	→	50.9415	Vanadium	
	Cr	→	51.9961	Chromium	
	Mn	→	54.938045	Manganese	
	Fe	→	55.845	Iron	
	Co	→	58.933195	Cobalt	
	Ni	→	58.6934	Nickel	
	Cu	→	63.546	Copper	
	Zn	→	65.38	Zinc	
	Ga	→	69.723	Gallium	
	Ge	→	72.63	Germanium	
	As	→	74.92160	Arsenic	
	Se	→	78.96	Selenium	
	Br	→	79.904	Bromine	
	Kr	→	83.798	Krypton	

5	Rb	→	85.4678	Rubidium
	Sr	→	87.62	Strontium
	Y	→	88.90585	Yttrium
	Zr	→	91.224	Zirconium
	Nb	→	92.90638	Niobium
	Mo	→	95.96	Molybdenum
	Tc	→	97.9072	Technetium**
	Ru	→	101.07	Ruthenium
	Rh	→	102.90550	Rhodium
	Pd	→	106.42	Palladium
	Ag	→	107.8682	Silver
	Cd	→	112.411	Cadmium
	In	→	114.818	Indium
	Sn	→	118.710	Tin
	Sb	→	121.760	Antimony
	Te	→	127.60	Tellurium
	I	→	126.90447	Iodine
	Xe	→	131.293	Xenon
6	Cs	→	132.9054519	Cesium
	Ba	→	137.327	Barium
	La	→	138.90547	Lanthanum
	Ce	→	140.116	Cerium
	Pr	→	140.90765	Praseodymium
	Nd	→	144.242	Neodymium
	Pm	→	144.9127	Promethium**
	Sm	→	150.36	Samarium

	Eu	→ 151.964	Europium
	Gd	→ 157.25	Gadolinium
	Tb	→ 158.92535	Terbium
	Dy	→ 162.500	Dysprosium
	Ho	→ 164.93032	Holmium
	Er	→ 167.259	Erbium
	Tm	→ 168.93421	Thulium
	Yb	→ 173.054	Ytterbium
	Lu	→ 174.9668	Lutetium
	Hf	→ 178.49	Hafnium
	Ta	→ 180.94788	Tantalum
	W	→ 183.84	Tungsten
	Re	→ 186.207	Rhenium
	Os	→ 190.23	Osmium
	Ir	→ 192.217	Iridium
	Pt	→ 195.084	Platinum
	Au	→ 196.966569	Gold
	Hg	→ 200.59	Mercury
	Tl	→ 204.38 [204.382; 204.385]	Thallium*
	Pb	→ 207.2	Lead
	Bi	→ 208.98040	Bismuth
	Po	→ 208.9824	Polonium**
	At	→ 209.9871	Astatine**
	Rn	→ 222.0176	Radon**
7	Fr	→ 223.0197	Francium**
	Ra	→ 226.0254	Radium**

Ac	→ 227.0278	Actinium**
Th	→ 232.03806	Thorium
Pa	→ 231.03588	Protactinium
U	→ 238.02891	Uranium
Np	→ 237.0482	Neptunium**
Pu	→ 244.0642	Plutonium**
Am	→ 243.0614	Americium**
Cm	→ 247.0704	Curium**
Bk	→ 247.0703	Berkelium**
Cf	→ 251.0796	Californium**
Es	→ 252.0830	Einsteinium**
Fm	→ 257.0951	Fermium**
Md	→ 258.0984	Mendelevium**
No	→ 259.1010	Nobelium**
Lr	→ 262.1096	Lawrencium**
Rf	→ 265.1167	Rutherfordium**
Db	→ 268.125	Dubnium**
Sg	→ 271.133	Seaborgium**
Bh	→ 267.1277	Bohrium**
Hs	→ 277.150	Hassium**
Mt	→ 276.151	Meitnerium**
Ds	→ 281.162	Darmstadtium**
Rg	→ 280.164	Roentgenium**
Cn	→ 285.174	Copernicium**
Uut	→ 284.178	Ununtrium**
Fl	→ 289.187	Flerovium**
Uup	→ 288.192	Ununpentium**

Lv	→ 292.200	Livermorium**
Uus	→ 293.2	Ununseptium**
Uuo	→ 294.2	Ununoctium**

Error Messages

Syntax Error	The expression does not conform to the pertinent calculation rules. Check the selected range for missing or extra parenthesis or look out for extra operators at inappropriate places.
Semantic Error	The expression contains unknown identifiers. Check the spelling of the selected word or symbol.
Wrong Number of Parameters	The number of arguments/parameters in a function call does not match the function definition.
Error in Logical Expression	The logical expression does not conform to the pertinent evaluation rules. Look out for invalid chaining of comparison operations, like: *term1 == term2 != term3*
Type Mismatch	Some functions work only with selected types of operands, e.g. *gcd()* and *lcm()* expect integers.
Not a Number	A valid numeric result does not exist for an expression and its arguments, for example, in the case of automatic real to complex conversion is disabled, *sqrt(-1)* results in *Not a Number*.

Singular Matrix	Matrix inversion cannot be performed because the given matrix is singular.
Invalid Row Index	The row index *sn* of a *R*(esult) or *E*(xpression) reference does not exist in the calculation history.
Invalid Recursion	A reference to an *E*(xpression) in the calculation history leads back to itself.