**How it works**

1. **Enter** your formulas and text (in quotes) into the "**Code**" box on the left.
2. Press **F5** or click  to **calculate**. The results will appear in the "**Output**" box on the right.
3. Click  to **print** or  to **copy** the output, or **export** it to **Html** , **PDF**  or **Word** .

**The language**

The Calcpad language includes the following elements (click an item to insert):

* Real numbers: digits 0 - 9 and decimal point ".";
* Complex numbers: re ± im*i* (e.g. 3 - 2*i*);
* Real vectors: [*v*1; *v*2; *v*3; ...; *v*n];
* Real matrices: [*M*11; *M*12; ... ; *M*1*n* | *M*21; *M*22; ... ; *M*2*n* ... | M*m*1; M*m*2; ... ; M*mn*];
* Variables:  
  - all Unicode letters;  
  - digits: 0 – 9;  
  - comma: “ , “;  
  - special symbols: **′** , **″** , **‴** , **⁗** , **‾** , **ø** , **Ø** , **°** , **∡** ;  
  - superscripts: **⁰** , **¹** , **²** , **³** , **⁴** , **⁵** , **⁶** , **⁷** , **⁸** , **⁹** , **ⁿ** , **⁺** , **⁻** ;  
  - subscripts: **₀** , **₁** , **₂** , **₃** , **₄** , **₅** , **₆** , **₇** , **₈** , **₉** , **₊** , **₋** , **₌** , **₍** , **₎**  
  - “**\_**“ (underscore) for subscript;  
  Any variable name must start with a letter. Names are case sensitive.
* Constants: π, e, φ, γ, g, G, ME, MS, c, h, μ0, ε0, ke, e, me, mp, mn, NA, σ, kB, R, F, γc, γs, γa, γg, γw
* Operators:

“!” - factorial;  
“^” - exponent;  
“/” - division;  
“÷” - force division bar;  
“\” - integer division;  
“**⦼**” - modulo (reminder, %%);  
“\*” - multiplication;  
“-“ - minus;  
“+” - plus;  
“**≡**” - equal to (==);  
“≠” - not equal to (!=);  
“<” - less than;  
“>” - greater than;  
“≤” - less or equal (<=);  
“≥” - greater or equal (>=);  
“**∧**” - logical “AND” (&&);  
“**∨**” - logical “OR” (||);  
“**⊕**” - logical “XOR” (^^);  
“=” - assignment;

* Custom functions type *f* (*x*; *y*; *z*; ...);
* Built-in functions:  
  ◦ Trigonometric:  
   sin(*x*) - sine;  
   cos(*x*) - cosine;  
   tan(*x*) - tangent;  
   csc(*x*) - cosecant;  
   sec(*x*) - secant;  
   cot(*x*) - cotangent;  
  ◦ Hyperbolic:  
   sinh(*x*) - hyperbolic sine;  
   cosh(*x*) - hyperbolic cosine;  
   tanh(*x*) - hyperbolic tangent;  
   csch(*x*) - hyperbolic cosecant;  
   sech(*x*) - hyperbolic secant;  
   coth(*x*) - hyperbolic cotangent;  
  ◦ Inverse trigonometric:  
   asin(*x*) - inverse sine;  
   acos(*x*) - inverse cosine;  
   atan(*x*) - inverse tangent;  
   atan2(*x*; *y*) - the angle whose tangent is the quotient of *y* and *x*;  
   acsc(*x*) - inverse cosecant;  
   asec(*x*) - inverse secant;  
   acot(*x*) - inverse cotangent;  
  ◦ Inverse hyperbolic:  
   asinh(*x*) - inverse hyperbolic sine;  
   acosh(*x*) - inverse hyperbolic cosine;  
   atanh(*x*) - inverse hyperbolic tangent;  
   acsch(*x*) - inverse hyperbolic cosecant;  
   asech(*x*) - inverse hyperbolic secant;  
   acoth(*x*) - inverse hyperbolic cotangent;  
  ◦ Logarithmic, exponential and roots:  
   log(*x*) - decimal logarithm;  
   ln(*x*) - natural logarithm;  
   log\_2(*x*) - binary logarithm;  
   exp(*x*) - exponential function;  
   sqr(*x*) or sqrt(*x*) - square root;  
   cbrt(*x*) - cubic root;  
   root(*x*; *n*) - *n*-th root;  
    
    
  ◦ Rounding:  
   round(*x*) - round to the nearest integer;  
   floor(*x*) - round to the smaller integer (towards -∞);  
   ceiling(*x*) - round to the greater integer (towards +∞);  
   trunc(*x*) - round to the smaller integer (towards zero);  
  ◦ Integer:  
   mod(*x*; *y*) - the reminder of an integer division;  
   gcd(*x*; *y*; *z*...) - the greatest common divisor of several integers;  
   lcm(*x*; *y*; *z*...) - the least common multiple of several integers;  
  ◦ Complex:  
   abs(*x*) - absolute value/magnitude;  
   re(*x*) - the real part of a complex number;  
   im(*x*) - the imaginary part of a complex number;  
   phase(*x*) - the phase of a complex number;

◦ Aggregate and interpolation:  
 min(*A*; b⃗ ; *c*...) - minimum of multiple values;  
 max(*A*; b⃗ ; *c*...) - maximum of multiple values;  
 sum(*A*; b⃗ ; *c*...) - sum of multiple values;  
 sumsq(*A*; b⃗ ; *c*...) - sum of squares  
 srss(*A*; b⃗ ; *c*...) - square root of sum of squares;  
 average(*A*; b⃗ ; *c*...) - average of multiple value;  
 product(*A*; b⃗ ; *c*...) - product of multiple values;  
 mean(*A*; b⃗ ; *c*...) - geometric mean;  
 take(*n*; *A*; b⃗ ; *c*...) - returns the *n*-th element from the list;  
 line(*x*; *A*; b⃗ ; *c*...) - linear interpolation;  
 spline(*x*; *A*; b⃗ ; *c*...) - Hermite spline interpolation;

◦ Conditional and logical:  
 if(*cond*; *value-if-true*; *value-if-false*) - conditional evaluation;   
 switch(*cond1*; *value1*; *cond2*; *value2*; …; *default*) - selective evaluation;

[not](about:blank#0)(*x*) - logical "NOT";  
 [and](about:blank#0)(*A*; b⃗ ; *c*...) - logical "AND";  
 [or](about:blank#0)(*A*; b⃗ ; *c*...) - logical "OR";  
 [xor](about:blank#0)(*A*; b⃗ ; *c*...) - logical "XOR";

◦ Other:  
 sign(*x*) - sign of a number;  
 random(*x*) - random number between 0 and *x*;  
 getunits(*x*) - gets the units of *x* without the value. Returns 1 if *x* is unitless;  
 setunits(*x*; *u*) - sets the units *u* to *x* where *x* can be scalar, vector or matrix;  
 clrunits(*x*) - clears the units from a scalar, vector or matrix *x*;  
 hp(*x*) - converts *x* to its high performance (hp) equivalent type;  
 ishp(*x*) - checks if the type of *x* is a high-performance (hp) vector or matrix;

◦ Vector:  
 Creational:

vector(n) - creates an empty vector with length n;

vector\_hp(n) - creates an empty high performance (hp) vector with length n;

range(x1; xn; s) - creates a vector with values spanning from x1 to xn with step s;  
 range\_hp (x1; xn; s) - creates a high performance (hp) from a range of values as above;  
 Structural:

len( v⃗ ) - returns the length of the vector v⃗ ;  
 size(v⃗ ) - the actual size of the vector v⃗ (the index of the last non-zero element);  
 resize(v⃗ ; n) - sets a new length n of the vector v⃗ ;

fill(v⃗ ; x) - fills the vector v⃗ with value x;

join(A ; b⃗ ; c…) - creates a vector by joining the arguments in the list – matrices, vectors and scalars;  
 slice(v⃗ ; i1; i2) - returns the part of the vector v⃗ bounded by indexes i1 and i2 inclusive;  
 first( v⃗ ; n) - the first n elements of the vector v⃗ ;  
 last(v⃗ ; n) - the last n elements of the vector v⃗ ;  
 extract(v⃗ ; i ⃗ ) - extracts those elements from v⃗ which indexes are contained in *i*⃗ ;

Data:

sort(v⃗ ) - sorts the vector v⃗ in ascending order;  
 rsort(v⃗ ) - sorts the vector v⃗ in descending order;  
 order(v⃗ ) - the indexes of v⃗ , in ascending order by the elements of v⃗ ;  
 revorder(v⃗ ) - the indexes of v⃗ , in descending order by the elements of v⃗ ;  
 reverse(v⃗ ) - vector containing the elements of v⃗ in reverse order;  
 count(v⃗ ; x; i) - the number of elements of v⃗ equal to x with index ≥ i;  
 search(v⃗ ; x; i) - the index of the first element in v⃗ with index ≥ *i* that is equal to x;  
 find(v⃗ ; x; i) or   
 find\_eq(v⃗ ; x; i) - the indexes of all elements in v⃗ , after the i-th, that are = x;  
 find\_ne(v⃗ ; x; i) - the indexes of all elements in v⃗ , after the i-th , that are ≠ x;  
 find\_lt(v⃗ ; x; i) - the indexes of all elements in v⃗ , after the i-th, that are < x;  
 find\_le(v⃗ ; x; i) - the indexes of all elements in v⃗ , after the i-th, that are ≤ x;  
 find\_gt(v⃗ ; x; i) - the indexes of all elements in v⃗ , after the i-th, that are > x;  
 find\_ge(v⃗ ; x; i) - the indexes of all elements in v⃗ , after the i-th, that are ≥ x;  
 lookup(a⃗ ; b⃗ ; x) or  
 lookup\_eq(a⃗ ; b⃗ ; x) - all elements of a⃗ for which the corresponding elements of b⃗ are = x;  
 lookup\_ne(a⃗ ; b⃗ ; x) - all elements of a⃗ for which the corresponding elements of b⃗ are ≠ x;  
 lookup\_lt(a⃗ ; b⃗ ; x) - all elements of a⃗ for which the corresponding elements of b⃗ are < x;  
 lookup\_le(a⃗ ; b⃗ ; x) - all elements of a⃗ for which the corresponding elements of b⃗ are ≤ x;  
 lookup\_gt(a⃗ ; b⃗ ; x) - all elements of a⃗ for which the corresponding elements of b⃗ are > x;  
 lookup\_ge(a⃗ ; b⃗ ; x) - all elements of a⃗ for which the corresponding elements of b⃗ are ≥ x;

Math:

norm\_1(v⃗ ) - L1 (Manhattan) norm of the vector v⃗ ;  
 norm(v⃗ ) or norm\_2(v⃗ ) or  
 norm\_e(v⃗ ) - L2 (Euclidean) norm of the vector v⃗ ;  
 norm\_p(v⃗ ; p) - Lp norm of the vector v⃗ ;  
 norm\_i(v⃗ ) - L∞ (infinity) norm of the vector v⃗ ;  
 unit(v⃗ ) - normalized form of the vector v⃗ (with L2 norm = 1);  
 dot(a⃗ ; b⃗ ) - scalar product of two vectors a⃗ and b⃗ ;  
 cross(a⃗ ; b⃗ ) - cross product of two vectors a⃗ and b⃗ (with length 2 or 3);

* + Matrix:  
     Creational:  
     matrix(m; n) - creates an empty matrix with dimensions m⨯n;  
     identity(n) - creates an identity matrix with dimensions n⨯n;  
     diagonal(n; d) - creates an n⨯n diagonal matrix and fills the diagonal with value d;  
     column(m; c) - creates a column matrix with dimensions m⨯1, filled with value c;  
     utriang(n) - creates an upper triangular matrix with dimensions n⨯n;  
     ltriang(n) - creates a lower triangular matrix with dimensions n⨯n;

symmetric(n) - creates a symmetric matrix with dimensions n⨯n;  
 matrix\_hp(m; n) - creates a high-performance matrix with dimensions m⨯n;  
 identity\_hp(n) - creates a high-performance identity matrix with dimensions n⨯n;  
 diagonal\_hp(n; d) - creates a high-performance n⨯n diagonal matrix filled with value d;  
 column\_hp(m; c) - creates a high-performance m⨯1 column matrix filled with value c;  
 utriang\_hp(n) - creates a high-performance n⨯n upper triangular matrix;  
 ltriang\_hp(n) - creates a high-performance n⨯n lower triangular matrix;

symmetric\_hp(n) - creates a high-performance symmetric matrix with dimensions n⨯n;  
 vec2diag(v⃗ ) - creates a diagonal matrix from the elements of vector v⃗ ;  
 vec2row(v⃗ ) - creates a row matrix from the elements of vector v⃗ ;  
 vec2col(v⃗ ) - creates a column matrix from the elements of vector v⃗ ;  
 join\_cols( ⃗c1;  ⃗c2;  ⃗c3...) - creates a matrix by joining column vectors;  
 join\_rows( ⃗r1;  ⃗r2;  ⃗r3...) - creates a matrix by joining row vectors;

augment(A; B; C...) - creates a matrix by appending matrices A; B; C side by side;  
 stack(A; B; C...) - creates a matrix by stacking matrices A; B; C one below the other;  
 Structural:  
 n\_rows(M) - number of rows in matrix M;  
 n\_cols(M) - number of columns in matrix M;  
 resize(M; m; n) - sets new dimensions m and n for matrix M;  
 fill(M; x) - fills the matrix M with value x;  
 fill\_row(M; i; x) - fills the i-th row of matrix M with value x;  
 fill\_col(M; j; x) - fills the j-th column of matrix M with value x;  
 copy(A; B; i; j) - copies all elements from A to B, starting from indexes i and j of B;  
 add(A; B; i; j) - adds all elements from A to those of B, starting from indexes i and j of B;  
 row(M; i) - extracts the i-th row of matrix M as a vector;

col(M; j) - extracts the j-th column of matrix M as a vector;  
 extract\_rows(M; i⃗ ) - extracts the rows from matrix M whose indexes are contained in vector i⃗ ;  
 extract\_cols(M; j⃗ ) - extracts the columns from matrix M whose indexes are contained in vector j⃗ ;

diag2vec(M ) - extracts the diagonal elements of matrix M to a vector;  
 submatrix(M; i1; i2; j1; j2) - extracts a submatrix of M, bounded between rows i1 and i2 and columns j1 and j2, incl.;  
 Data:

sort\_cols(M; i) - sorts the columns of M based on the values in row i in ascending order;  
 rsort\_cols(M; i) - sorts the columns of M based on the values in row i in descending order;  
 sort\_rows(M; j) - sorts the rows of M based on the values in column j in ascending order;  
 rsort\_rows(M; j) - sorts the rows of M based on the values in column j in descending order;  
 order\_cols(M; i) - the indexes of the columns of M in ascending order by the values in row i;  
 revorder\_cols(M; i) - the indexes of the columns of M in descending order by the values in row i;  
 order\_rows(M; j) - the indexes of the rows of M in ascending order by the values in column j;  
 revorder\_rows(M; j) - the indexes of the rows of M in descending order by the values in column j;  
 mcount(M; x) - number of occurrences of value x in matrix M;  
 msearch(M; x; i; j) - vector with the two indexes of the first occurrence of x in matrix M, starting from indexes i and j;

mfind(M; x) - the indexes of all elements in matrix M equal to x;  
 mfind\_eq(M; x) - the indexes of all elements in matrix M equal to x;  
 mfind\_ne(M; x) - the indexes of all elements in matrix M not equal to x;  
 mfind\_lt(M; x) - the indexes of all elements in matrix M less than x;  
 mfind\_le(M; x) - the indexes of all elements in matrix M less than or equal to x;  
 mfind\_gt(M; x) - the indexes of all elements in matrix M greater than x;  
 mfind\_ge(M; x) - the indexes of all elements in matrix M greater than or equal to x;

hlookup(M; x; i1; i2) - the values from row i2 of M, for which the elements from row i1 are equal to x;  
 hlookup\_eq(M; x; i1; i2) - the values from row i2 of M, for which the elements from row i1 are equal to x;  
 hlookup\_ne(M; x; i1; i2) - the values from row i2 of M, for which the elements from row i1 are not equal to x;  
 hlookup\_lt(M; x; i1; i2) - the values from row i2 of M, for which the elements from row i1 are less than x;  
 hlookup\_le(M; x; i1; i2) - the values from row i2 of M, for which the elements from row i1 are less than or equal to x;  
 hlookup\_gt(M; x; i1; i2) - the values from row i2 of M, for which the elements from row i1 are greater than x;  
 hlookup\_ge(M; x; i1; i2) - the values from row i2 of M, for which the elements from row i1 are greater than or equal to x;  
 vlookup(M; x; j1; j2) - the values from column j2 of M, for which the elements from column j1 are equal to x;  
 vlookup\_eq(M; x; j1; j2) - the values from column j2 of M, for which the elements from column j1 are equal to x;  
 vlookup\_ne(M; x; j1; j2) - the values from column j2 of M, for which the elements from column j1 are not equal to x;  
 vlookup\_lt(M; x; j1; j2) - the values from column j2 of M, for which the elements from column j1 are less than x;  
 vlookup\_le(M; x; j1; j2) - the values from column j2 of M, for which the elements from column j1 are less than or equal to x;  
 vlookup\_gt(M; x; j1; j2) - the values from column j2 of M, for which the elements from column j1 are greater than x;  
 vlookup\_ge(M; x; j1; j2) - the values from column j2 of M, for which the elements from column j1 are greater than or equal to x;  
 Math:  
 hprod(A; B) - Hadamard product of matrices A and B;  
 fprod(A; B) - Frobenius product of matrices A and B;  
 kprod(A; B) - Kronecker product of matrices A and B;  
 mnorm(M) or  
 mnorm\_2(M) - L2 norm of matrix M;  
 mnorm\_1(M) - L1 norm of matrix M;  
 mnorm\_2(M) - Frobenius norm of matrix M;  
 mnorm\_i(M) - L∞ norm of matrix M;  
 cond(M) or  
 cond\_e(M) - condition number of M based on the Euclidean norm of the matrix;  
 cond\_1(M) - condition number of M based on the L1 norm;  
 cond\_2(M) - condition number of M based on the L2 norm;  
 cond\_i(M) - condition number of M based on the L∞ norm;  
 det(M) - determinant of matrix M;  
 rank(M) - rank of matrix M;  
 trace(M) - trace of matrix M;  
 transp(M) - transpose of matrix M;  
 adj(M) - adjugate of matrix M;  
 cofactor(M) - cofactor matrix of M;  
 eigenvals(M; nе) - the first nе eigenvalues of matrix M (or all if omitted);   
 eigenvecs(M; nе) - the first nе eigenvectors of matrix M (or all if omitted);  
 eigen(M; nе) - the first nе eigenvalues and eigenvectors of M (or all if omitted);

cholesky(M) - Cholesky decomposition of a symmetric, positive-definite matrix M;  
 lu(M) - LU decomposition of matrix M;  
 qr(M) - QR decomposition of matrix M;  
 svd(M) - singular value decomposition of M;

inverse(M) - inverse of matrix M;  
 lsolve(A; b⃗ ) - solves the system of linear equations Ax ⃗ = b⃗ using LDLT decomposition for symmetric matrices, and LU for non-symmetric;  
 clsolve(A; b⃗ ) - solves the linear matrix equation Ax ⃗ = b⃗ with symmetric, positive-definite coefficient matrix A using Cholesky decomposition;  
 slsolve(A; b⃗ ) - solves the linear matrix equation Ax ⃗ = b⃗ with high-performance symmetric, positive-definite matrix A using preconditioned conjugate gradient (PCG) method;

msolve(A; B) - solves the generalized matrix equation AX = B using LDLT decomposition for symmetric matrices, and LU for non-symmetric;  
 cmsolve(A; B) - solves the generalized matrix equation AX = B with symmetric, positive-definite coefficient matrix A using Cholesky decomposition;  
 smsolve(A; B) - solves the generalized matrix equation AX = B with high- performance symmetric, positive-definite matrix A using preconditioned conjugate gradient (PCG) method;  
 fft(M) - performs fast Fourier transform of row-major matrix M. It must have one row for real data and two rows for complex;  
 ift(M) - performs inverse Fourier transform of row-major matrix M.   
 It must have one row for real data and two rows for complex;  
 Double interpolation:  
 take(x; y; M) - returns the element of matrix M at indexes x and y;  
 line(x; y; M) - double linear interpolation from the elements of matrix M based on the values of x and y;  
 spline(x; y; M) - double Hermite spline interpolation from the elements of matrix M based on the values of x and y;  
 *Tol* - target tolerance for the iterative PCG solver.

* Comments: "Title" or 'text' in double or single quotes. **HTML**, **CSS**, **JS** and **SVG** are allowed.
* Graphing and plotting:  
  $Plot { *f*(*x*) @ *x* = *a* : *b* } - simple plot;  
  $Plot { *x*(*t*) | *y*(*t*) @ *t* = *a* : *b* } - parametric;  
  $Plot { *f*1(*x*) & *f*2(*x*) & ... @ *x* = *a* : *b* } - multiple;  
  $Plot { *x*1(*t*) | *y*1(*t*) & *x*2(*t*) | *y*2(*t*) & ... @ *x* = *a* : *b* } - multiple parametric;  
  $Map { *f*(*x*; *y*) @ *x* = *a* : *b* & *y* = *c* : *d* } - 2D color map of a 3D surface;

*PlotHeight* - height of plot area in pixels;  
*PlotWidth* - width of plot area in pixels;   
*PlotSVG* - draw plots in vector, SVG format (= 1) or raster, PNG (= 0);  
*PlotAdaptive* - use adaptive mesh (= 1) for function plotting or uniform (= 0);  
*PlotStep* - the size of the mesh for map plotting;  
[*PlotPalette*](file:///C:\Users\proek\Documents\GitHub\CalcpadVM\Calcpad.Wpf\doc\help.html#0)  - the number of color palette to be used for surface plots (0-9);  
[*PlotShadows*](file:///C:\Users\proek\Documents\GitHub\CalcpadVM\Calcpad.Wpf\doc\help.html#0)  - draw surface plots with shadows;  
[*PlotSmooth*](file:///C:\Users\proek\Documents\GitHub\CalcpadVM\Calcpad.Wpf\doc\help.html#0)- smooth gradient coloring (= 1) or isobands (= 0) for surface plots;  
[*PlotLightDir*](file:///C:\Users\proek\Documents\GitHub\CalcpadVM\Calcpad.Wpf\doc\help.html#0) - direction to light source (0-7) clockwise.

* Iterative and numerical methods:  
  $Root { *f*(*x*) = *const* @ *x* = *a* : *b* } - root finding for *f*(*x*) = *const*;  
  $Root { *f*(*x*) @ *x* = *a* : *b* } - root finding for *f*(*x*) = 0;  
  $Find { *f*(*x*) @ *x* = *a* : *b* } - similar to above, but *x* is not required to be a precise solution;  
  $Sup { *f*(*x*) @ *x* = *a* : *b* } - local maximum of a function;  
  $Inf { *f*(*x*) @ *x* = *a* : *b* } - local minimum of a function;  
  $Area {*f*(*x*) @ *x* = *a* : *b* } - adaptive Gauss-Lobatto numerical integration;  
  $Integral {*f*(*x*) @ *x* = *a* : *b* } - Tanh-Sinh numerical integration;  
  $Slope {*f*(*x*) @ *x* = *a* } - numerical differentiation;  
  $Sum {*f*(*x*) @ *k* = *a* : *b* } - iterative sum;  
  $Product { *f*(*k*) @ *k* = *a* : *b* } - iterative product;  
  $Repeat { *f*(*k*) @ *k* = *a* : *b* } - iterative expression block with counter;  
  $While { *condition*; *expressions* } - iterative expression block with condition;

$Block { *expressions* } - multiline expression block;  
$Inline { *expressions* } - inline expression block;

*Precision* - relative precision for numerical methods [10 -2; 10 -15] (default is 10 -14).

* Program flow control:  
  Simple:  
   #if *condition*  
   *your code goes here*  
   #end if   
  Alternative:  
   #if *condition*  
   *your code goes here*  
   #else  
   *some other code*  
   #end if   
  Complete:  
   #if *condition1*  
   *your code goes here*  
   #else if *condition2*  
   *your code goes here*  
   #else  
   *some other code* #end if  
  You can add or omit as many "#else ifs" as needed. Only one "#else" is allowed.   
  You can omit this too.
* Iteration blocks:  
  Simple:  
   #repeat *number of repetitions  
   your code goes here*  
   #loop

With conditional break/continue:  
 #repeat *number of repetitions*  
 *your code goes here*  
 #if *condition*  
 #break or #continue  
 #end if  
 *some more code*  
 #loop

With counter:  
 #for *counter* = *start* : *end  
 your code goes here*  
 #loop   
With condition:  
 #while *condition  
 your code goes here*  
 #loop

* Modules and macros/string variables:  
  Modules:  
      #include *filename* - include external file (module);  
      [#local](file:///C:\Users\proek\source\repos\Calcpad-en\Calcpad.Wpf\bin\Release\net6.0-windows\help.html#0) - start local section (not to be included);  
      [#global](file:///C:\Users\proek\source\repos\Calcpad-en\Calcpad.Wpf\bin\Release\net6.0-windows\help.html#0) - start global section (to be included);  
  Inline string variable:  
       #def *variable\_name*$ = *content*  
  Multiline string variable:  
      #def *variable\_name*$  
  *content line 1*  
  *content line 2*  
  *...*  
      #end def  
  Inline macro:  
      #def *macro\_name*$(*param1*$; *param2*$; ...) = *content*  
  Multiline macro:  
      #def *macro\_name*$(*param1$*; *param2$*; ...)  
  *content line 1*  
  *content line 2*  
  *...*  
      #end def
* Import/Export of external data:  
  Text/CSV files:  
  #read M from filename.txt@R1C1:R2C2 TYPE=R SEP=',' - read matrix M from a text/CSV file;  
  #write M to filename.txt@R1C1:R2C2 TYPE=N SEP=',' - write matrix M to a text/CSV file;  
  #append M to filename.txt@R1C1:R2C2 TYPE=N SEP=',' - append matrix M to a text/CSV file;  
  Excel files (xlsx and xlsm):  
  #read M from filename.xlsx@Sheet1!A1:B2 TYPE=R - read matrix M from an Excel file;  
  #write M to filename.xlsx@Sheet1!A1:B2 TYPE=N - write matrix M to an Excel file;  
  #append M to filename.xlsx@Sheet1!A1:B2 TYPE=N - append matrix M to an Excel file;  
  Sheet, range, TYPE and SEP can be omitted.  
  For #read command, TYPE can be either of [R|D|C|S|U|L|V].  
  For #write and #append commands, TYPE can be Y or N.
* Output control:  
  #hide - hide the report contents;  
  #show - always show the contents (default);  
  #pre - show the next contents only before calculations;  
  #post - show the next contents only after calculations;  
  #val - show only the result, without the equation;  
  #equ - show complete equations and results (default);

#noc - show only equations without results (no calculations);  
#nosub - do not substitute variables (no substitution);  
#novar - show equations only with substituted values (no variables);  
#varsub - show equations with variables and substituted values (default);

#round *n* - rounds the output to *n* digits after the decimal point;  
#round default - restores rounding to the default settings;  
#format *FFFF* - specifies custom format string;  
#format default - restores the default formatting;  
#md on - enables markdown in comments;  
#md off - disables markdown in comments.

* Breakpoints for step-by-step execution:  
  #pause - calculates down to the current line and waits for the user to resume manually;  
  #input - renders an input form to the current line and waits for user input.  
  Each of the above commands is effective after the current line until the end of the report or another command that overrides it.
* Units for trigonometric functions: #deg - degrees, #rad - radians, #gra - gradians;
* Separator for target units: |;
* Return angles with units: *ReturnAngleUnits* = 1;
* Dimensionless: %, ‰, ‱, pcm, ppm, ppb, ppt, ppq;
* Angle: °, ′, ″, deg, rad, grad, rev;
* Metric units (SI and compatible):  
  Mass: g, hg, kg, t, kt, Mt, Gt, dg, cg, mg, μg, Da (or u);  
  Length: m, km, dm, cm, mm, μm, nm, pm, AU, ly;  
  Time: s, ms, μs, ns, ps, min, h, d, w, y;  
  Frequency: Hz, kHz, MHz, GHz, THz, mHz, μHz, nHz, pHz, rpm;   
  Speed: kmh;  
  Electric current: A, kA, MA, GA, TA, mA, μA, nA, pA;  
  Temperature: °C, Δ°C, K;  
  Amount of substance: mol;  
  Luminous intensity: cd;  
  Area: a, daa, ha;  
  Volume: L, daL, hL, dL, cL, mL, μL, nL, pL;  
  Force: N, daN, hN, kN, MN, GN, TN, gf, kgf, tf, dyn;  
  Moment: Nm, kNm;  
  Pressure: Pa, daPa, hPa, kPa, MPa, GPa, TPa, dPa, cPa, mPa, μPa, nPa, pPa,   
   bar, mbar, μbar, atm, at, Torr, mmHg;  
  Viscosity: P, cP, St, cSt;  
  Energy work: J, kJ, MJ, GJ, TJ, mJ, μJ, nJ, pJ,   
   Wh, kWh, MWh, GWh, TWh, mWh, μWh, nWh, pWh,  
   eV, keV, MeV, GeV, TeV, PeV, EeV, cal, kcal, erg;  
  Power: W, kW, MW, GW, TW, mW, μW, nW, pW, hpM, ks,  
   VA, kVA, MVA, GVA, TVA, mVA, μVA, nVA, pVA,   
   VAR, kVAR, MVAR, GVAR, TVAR, mVAR, μVAR, nVAR, pVAR;  
  Electric charge: C, kC, MC, GC, TC, mC, μC, nC, pC, Ah, mAh;  
  Potential: V, kV, MV, GV, TV, mV, μV, nV, pV;  
  Capacitance: F, kF, MF, GF, TF, mF, μF, nF, pF;  
  Resistance: Ω, kΩ, MΩ, GΩ, TΩ, mΩ, μΩ, nΩ, pΩ;  
  Conductance: S, kS, MS, GS, TS, mS, μS, nS, pS, ℧, k℧, M℧, G℧, T℧, m℧, μ℧, n℧, p℧;  
  Magnetic flux: Wb , kWb, MWb, GWb, TWb, mWb, μWb, nWb, pWb;  
  Magnetic flux density: T, kT, MT, GT, TT, mT, μT, nT, pT;  
  Inductance: H, kH, MH, GH, TH, mH, μH, nH, pH;  
  Luminous flux: lm;  
  Illuminance: lx;  
  Radioactivity: Bq, kBq, MBq, GBq, TBq, mBq, μBq, nBq, pBq, Ci, Rd;  
  Absorbed dose: Gy, kGy, MGy, GGy, TGy, mGy, μGy, nGy, pGy;  
  Equivalent dose: Sv, kSv, MSv, GSv, TSv, mSv, μSv, nSv, pSv;  
  Catalytic activity: kat;
* Non-metric units (Imperial/US):  
  Mass: gr, dr, oz, lb (or lbm,  lb**\_m**), kipm (or kip**\_m**), st, qr,   
   cwt (or  cwt**\_UK**, cwt**\_US**), ton (or ton**\_UK**, ton**\_US**), slug;  
  Length: th, in, ft, yd, ch, fur, mi, ftm (or ftm**\_UK**, ftm**\_US**),   
   cable (or cable**\_UK**, cable**\_US**), nmi, li, rod, pole, perch, lea;  
  Speed: mph, knot;  
  Temperature: °F, Δ°F, °R;  
  Area: rood, ac;  
  Volume, fluid: fl\_oz, gi, pt, qt, gal, bbl, or:  
   fl\_oz**\_UK**, gi**\_UK**, pt**\_UK**, qt**\_UK**, gal**\_UK**, bbl**\_UK**,  
   fl\_oz**\_US**, gi**\_US**, pt**\_US**, qt**\_US**, gal**\_US**, bbl**\_US**;  
  Volume, dry: (US) pt**\_dry**, (US) qt**\_dry**, (US) gal**\_dry**, (US) bbl**\_dry**,pk (or pk**\_UK**, pk**\_US**), bu (or bu**\_UK**, bu**\_US**);  
  Force: ozf (or oz**\_f**), lbf (or lb**\_f**), kip (or kipf, kip**\_f**), tonf (or  ton**\_f**), pdl;  
  Pressure: osi, osf psi, psf, ksi, ksf, tsi, tsf, inHg;  
  Energy/work: BTU, therm (or  therm**\_UK**, therm**\_US**), quad;  
  Power: hp, hpE, hpS;
* Custom units - .Name = expression.

Names can include currency symbols: €, £, ₤, ¥, ¢, ₽, ₹, ₩, ₪.