







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*_{*m**n*}**];
- Variables:
 - all Unicode letters;
 - digits: **0 – 9**;
 - comma: **" , "**;
 - special symbols: **' , " , " , " , " , - , ∅ , ∅ , ° , 4 ;**
 - 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*, *M*_E, *M*_S, *c*, *h*, *μ*₀, *ε*₀, *k*_e, *e*, *m*_e, *m*_p, *m*_n, *N*_A, *σ*, *k*_B, *R*, *F*, *γ*_c, *γ*_s, *γ*_a, *γ*_g, *γ*_w**
- Operators:
 - "!"** - factorial;
 - "^"** - exponent;
 - "/"** - division;
 - "÷"** - force division bar;
 - "\"** - integer division;
 - "⊗"** - modulo (remainder, %%);
 - "*"** - multiplication;
 - "-"** - minus;
 - "+"** - plus;
 - "≡"** - equal to (==);
 - "≠"** - not equal to (!=);
 - "<"** - less than;
 - ">"** - greater than;
 - "≤"** - less or equal (<=);
 - "≥"** - greater or equal (>=);
 - "^"** - logical "AND" (&&);
 - "v"** - logical "OR" (||);
 - "⊕"** - logical "XOR" (^);
 - "="** - assignment;

- Custom functions type $f(x; y; z; \dots)$;
- 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;
$\operatorname{csch}(x)$	- hyperbolic cosecant;
$\operatorname{sech}(x)$	- hyperbolic secant;
$\operatorname{coth}(x)$	- hyperbolic cotangent;
 - Inverse trigonometric:

$\arcsin(x)$	- inverse sine;
$\arccos(x)$	- inverse cosine;
$\arctan(x)$	- inverse tangent;
$\operatorname{atan2}(x; y)$	- the angle whose tangent is the quotient of y and x ;
$\operatorname{acsc}(x)$	- inverse cosecant;
$\operatorname{asec}(x)$	- inverse secant;
$\operatorname{acot}(x)$	- inverse cotangent;
 - Inverse hyperbolic:

$\operatorname{asinh}(x)$	- inverse hyperbolic sine;
$\operatorname{acosh}(x)$	- inverse hyperbolic cosine;
$\operatorname{atanh}(x)$	- inverse hyperbolic tangent;
$\operatorname{acsch}(x)$	- inverse hyperbolic cosecant;
$\operatorname{asech}(x)$	- inverse hyperbolic secant;
$\operatorname{acoth}(x)$	- inverse hyperbolic cotangent;
 - Logarithmic, exponential and roots:

$\log(x)$	- decimal logarithm;
$\ln(x)$	- natural logarithm;
$\log_2(x)$	- binary logarithm;
$\exp(x)$	- exponential function;
$\operatorname{sqr}(x)$ or $\operatorname{sqrt}(x)$	- square root;
$\operatorname{cbrt}(x)$	- cubic root;
$\operatorname{root}(x; n)$	- n -th root;

◦ Rounding:

<code>round(<i>x</i>)</code>	- round to the nearest integer;
<code>floor(<i>x</i>)</code>	- round to the smaller integer (towards $-\infty$);
<code>ceiling(<i>x</i>)</code>	- round to the greater integer (towards $+\infty$);
<code>trunc(<i>x</i>)</code>	- round to the smaller integer (towards zero);

◦ Integer:

<code>mod(<i>x</i>; <i>y</i>)</code>	- the remainder of an integer division;
<code>gcd(<i>x</i>; <i>y</i>; <i>z</i>...)</code>	- the greatest common divisor of several integers;
<code>lcm(<i>x</i>; <i>y</i>; <i>z</i>...)</code>	- the least common multiple of several integers;

◦ Complex:

<code>abs(<i>x</i>)</code>	- absolute value/magnitude;
<code>re(<i>x</i>)</code>	- the real part of a complex number;
<code>im(<i>x</i>)</code>	- the imaginary part of a complex number;
<code>phase(<i>x</i>)</code>	- the phase of a complex number;

◦ Aggregate and interpolation:

<code>min(<i>A</i>; \vec{b}; <i>c</i>...)</code>	- minimum of multiple values;
<code>max(<i>A</i>; \vec{b}; <i>c</i>...)</code>	- maximum of multiple values;
<code>sum(<i>A</i>; \vec{b}; <i>c</i>...)</code>	- sum of multiple values;
<code>sumsq(<i>A</i>; \vec{b}; <i>c</i>...)</code>	- sum of squares
<code>srss(<i>A</i>; \vec{b}; <i>c</i>...)</code>	- square root of sum of squares;
<code>average(<i>A</i>; \vec{b}; <i>c</i>...)</code>	- average of multiple value;
<code>product(<i>A</i>; \vec{b}; <i>c</i>...)</code>	- product of multiple values;
<code>mean(<i>A</i>; \vec{b}; <i>c</i>...)</code>	- geometric mean;
<code>take(<i>n</i>; <i>A</i>; \vec{b}; <i>c</i>...)</code>	- returns the <i>n</i> -th element from the list;
<code>line(<i>x</i>; <i>A</i>; \vec{b}; <i>c</i>...)</code>	- linear interpolation;
<code>spline(<i>x</i>; <i>A</i>; \vec{b}; <i>c</i>...)</code>	- Hermite spline interpolation;

◦ Conditional and logical:

<code>if(<i>cond</i>; <i>value-if-true</i>; <i>value-if-false</i>)</code>	- conditional evaluation;
<code>switch(<i>cond1</i>; <i>value1</i>; <i>cond2</i>; <i>value2</i>; ...; <i>default</i>)</code>	- selective evaluation;
<code>not(<i>x</i>)</code>	- logical "NOT";
<code>and(<i>A</i>; \vec{b}; <i>c</i>...)</code>	- logical "AND";
<code>or(<i>A</i>; \vec{b}; <i>c</i>...)</code>	- logical "OR";
<code>xor(<i>A</i>; \vec{b}; <i>c</i>...)</code>	- logical "XOR";

◦ Other:

<code>sign(<i>x</i>)</code>	- sign of a number;
<code>random(<i>x</i>)</code>	- random number between 0 and <i>x</i> ;
<code>getunits(<i>x</i>)</code>	- gets the units of <i>x</i> without the value. Returns 1 if <i>x</i> is unitless;
<code>setunits(<i>x</i>; <i>u</i>)</code>	- sets the units <i>u</i> to <i>x</i> where <i>x</i> can be scalar, vector or matrix;
<code>clrunits(<i>x</i>)</code>	- clears the units from a scalar, vector or matrix <i>x</i> ;
<code>hp(<i>x</i>)</code>	- converts <i>x</i> to its high performance (hp) equivalent type;
<code>ishp(<i>x</i>)</code>	- checks if the type of <i>x</i> 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 *x*₁ to *x*_{*n*} with step *s*;
- `range_hp(x1; xn; s)` - creates a high performance (hp) from a range of values as above;

Structural:

- `len(v)` - returns the length of the vector *v*;
- `size(v)` - the actual size of the vector *v* (the index of the last non-zero element);
- `resize(v; n)` - sets a new length *n* of the vector *v*;
- `fill(v; x)` - fills the vector *v* with value *x*;
- `join(A; b; c...)` - creates a vector by joining the arguments in the list – matrices, vectors and scalars;
- `slice(v; i1; i2)` - returns the part of the vector *v* bounded by indexes *i*₁ and *i*₂ inclusive;
- `first(v; n)` - the first *n* elements of the vector *v*;
- `last(v; n)` - the last *n* elements of the vector *v*;
- `extract(v; i)` - extracts those elements from *v* which indexes are contained in *i*;

Data:

- `sort(v)` - sorts the vector *v* in ascending order;
- `rsort(v)` - sorts the vector *v* in descending order;
- `order(v)` - the indexes of *v*, in ascending order by the elements of *v*;
- `revorder(v)` - the indexes of *v*, in descending order by the elements of *v*;
- `reverse(v)` - vector containing the elements of *v* in reverse order;
- `count(v; x; i)` - the number of elements of *v* equal to *x* with index $\geq i$;
- `search(v; x; i)` - the index of the first element in *v* with index $\geq i$ that is equal to *x*;
- `find(v; x; i)` or
- `find_eq(v; x; i)` - the indexes of all elements in *v*, after the *i*-th, that are = *x*;
- `find_ne(v; x; i)` - the indexes of all elements in *v*, after the *i*-th, that are $\neq x$;
- `find_lt(v; x; i)` - the indexes of all elements in *v*, after the *i*-th, that are $< x$;
- `find_le(v; x; i)` - the indexes of all elements in *v*, after the *i*-th, that are $\leq x$;
- `find_gt(v; x; i)` - the indexes of all elements in *v*, after the *i*-th, that are $> x$;
- `find_ge(v; x; i)` - the indexes of all elements in *v*, after the *i*-th, that are $\geq x$;
- `lookup(a; b; x)` or
- `lookup_eq(a; b; x)` - all elements of *a* for which the corresponding elements of *b* are = *x*;
- `lookup_ne(a; b; x)` - all elements of *a* for which the corresponding elements of *b* are $\neq x$;
- `lookup_lt(a; b; x)` - all elements of *a* for which the corresponding elements of *b* are $< x$;
- `lookup_le(a; b; x)` - all elements of *a* for which the corresponding elements of *b* are $\leq x$;
- `lookup_gt(a; b; x)` - all elements of *a* for which the corresponding elements of *b* are $> x$;
- `lookup_ge(a; b; x)` - all elements of *a* for which the corresponding elements of *b* are $\geq x$;

Math:

- `norm_1(\vec{v})` - L1 (Manhattan) norm of the vector \vec{v} ;
- `norm(\vec{v})` or `norm_2(\vec{v})` or `norm_e(\vec{v})` - L2 (Euclidean) norm of the vector \vec{v} ;
- `norm_p($\vec{v}; p$)` - Lp norm of the vector \vec{v} ;
- `norm_i(\vec{v})` - L ∞ (infinity) norm of the vector \vec{v} ;
- `unit(\vec{v})` - normalized form of the vector \vec{v} (with L2 norm = 1);
- `dot($\vec{a}; \vec{b}$)` - scalar product of two vectors \vec{a} and \vec{b} ;
- `cross($\vec{a}; \vec{b}$)` - cross product of two vectors \vec{a} and \vec{b} (with length 2 or 3);

○ Matrix:

Creational:

- `matrix($m; n$)` - creates an empty matrix with dimensions $m \times n$;
- `identity(n)` - creates an identity matrix with dimensions $n \times n$;
- `diagonal($n; d$)` - creates an $n \times n$ diagonal matrix and fills the diagonal with value d ;
- `column($m; c$)` - creates a column matrix with dimensions $m \times 1$, filled with value c ;
- `utriang(n)` - creates an upper triangular matrix with dimensions $n \times n$;
- `ltriang(n)` - creates a lower triangular matrix with dimensions $n \times n$;
- `symmetric(n)` - creates a symmetric matrix with dimensions $n \times n$;
- `matrix_hp($m; n$)` - creates a high-performance matrix with dimensions $m \times n$;
- `identity_hp(n)` - creates a high-performance identity matrix with dimensions $n \times n$;
- `diagonal_hp($n; d$)` - creates a high-performance $n \times n$ diagonal matrix filled with value d ;
- `column_hp($m; c$)` - creates a high-performance $m \times 1$ column matrix filled with value c ;
- `utriang_hp(n)` - creates a high-performance $n \times n$ upper triangular matrix;
- `ltriang_hp(n)` - creates a high-performance $n \times n$ lower triangular matrix;
- `symmetric_hp(n)` - creates a high-performance symmetric matrix with dimensions $n \times n$;
- `vec2diag(\vec{v})` - creates a diagonal matrix from the elements of vector \vec{v} ;
- `vec2row(\vec{v})` - creates a row matrix from the elements of vector \vec{v} ;
- `vec2col(\vec{v})` - creates a column matrix from the elements of vector \vec{v} ;
- `join_cols($\vec{c}_1; \vec{c}_2; \vec{c}_3 \dots$)` - creates a matrix by joining column vectors;
- `join_rows($\vec{r}_1; \vec{r}_2; \vec{r}_3 \dots$)` - creates a matrix by joining row vectors;
- `augment($A; B; C \dots$)` - creates a matrix by appending matrices $A; B; C$ side by side;
- `stack($A; B; C \dots$)` - 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**($M; \vec{i}$) - extracts the rows from matrix M whose indexes are contained in vector \vec{i} ;
- extract_cols**($M; \vec{j}$) - extracts the columns from matrix M whose indexes are contained in vector \vec{j} ;
- diag2vec**(M) - extracts the diagonal elements of matrix M to a vector;
- submatrix**($M; i_1; i_2; j_1; j_2$) - extracts a submatrix of M , bounded between rows i_1 and i_2 and columns j_1 and j_2 , 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**($M; x; i_1; i_2$) - the values from row i_2 of M , for which the elements from row i_1 are equal to x ;
- hlookup_eq**($M; x; i_1; i_2$) - the values from row i_2 of M , for which the elements from row i_1 are equal to x ;

- hlookup_ne**($M; x; i_1; i_2$) - the values from row i_2 of M , for which the elements from row i_1 are not equal to x ;
- hlookup_lt**($M; x; i_1; i_2$) - the values from row i_2 of M , for which the elements from row i_1 are less than x ;
- hlookup_le**($M; x; i_1; i_2$) - the values from row i_2 of M , for which the elements from row i_1 are less than or equal to x ;
- hlookup_gt**($M; x; i_1; i_2$) - the values from row i_2 of M , for which the elements from row i_1 are greater than x ;
- hlookup_ge**($M; x; i_1; i_2$) - the values from row i_2 of M , for which the elements from row i_1 are greater than or equal to x ;
- vlookup**($M; x; j_1; j_2$) - the values from column j_2 of M , for which the elements from column j_1 are equal to x ;
- vlookup_eq**($M; x; j_1; j_2$) - the values from column j_2 of M , for which the elements from column j_1 are equal to x ;
- vlookup_ne**($M; x; j_1; j_2$) - the values from column j_2 of M , for which the elements from column j_1 are not equal to x ;
- vlookup_lt**($M; x; j_1; j_2$) - the values from column j_2 of M , for which the elements from column j_1 are less than x ;
- vlookup_le**($M; x; j_1; j_2$) - the values from column j_2 of M , for which the elements from column j_1 are less than or equal to x ;
- vlookup_gt**($M; x; j_1; j_2$) - the values from column j_2 of M , for which the elements from column j_1 are greater than x ;
- vlookup_ge**($M; x; j_1; j_2$) - the values from column j_2 of M , for which the elements from column j_1 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 ;

<code>eigenvals(M; n_e)</code>	- the first n_e eigenvalues of matrix M (or all if omitted);
<code>eigenvecs(M; n_e)</code>	- the first n_e eigenvectors of matrix M (or all if omitted);
<code>eigen(M; n_e)</code>	- the first n_e eigenvalues and eigenvectors of M (or all if omitted);
<code>cholesky(M)</code>	- Cholesky decomposition of a symmetric, positive-definite matrix M ;
<code>lu(M)</code>	- LU decomposition of matrix M ;
<code>qr(M)</code>	- QR decomposition of matrix M ;
<code>svd(M)</code>	- singular value decomposition of M ;
<code>inverse(M)</code>	- inverse of matrix M ;
<code>lsolve(A; \vec{b})</code>	- solves the system of linear equations $A\vec{x} = \vec{b}$ using LDL ^T decomposition for symmetric matrices, and LU for non-symmetric;
<code>clsolve(A; \vec{b})</code>	- solves the linear matrix equation $A\vec{x} = \vec{b}$ with symmetric, positive-definite coefficient matrix A using Cholesky decomposition;
<code>slsolve(A; \vec{b})</code>	- solves the linear matrix equation $A\vec{x} = \vec{b}$ with high-performance symmetric, positive-definite matrix A using preconditioned conjugate gradient (PCG) method;
<code>msolve(A; B)</code>	- solves the generalized matrix equation $AX = B$ using LDL ^T decomposition for symmetric matrices, and LU for non-symmetric;
<code>cmsolve(A; B)</code>	- solves the generalized matrix equation $AX = B$ with symmetric, positive-definite coefficient matrix A using Cholesky decomposition;
<code>smsolve(A; B)</code>	- solves the generalized matrix equation $AX = B$ with high-performance symmetric, positive-definite matrix A using preconditioned conjugate gradient (PCG) method;
<code>fft(M)</code>	- performs fast Fourier transform of row-major matrix M . It must have one row for real data and two rows for complex;
<code>ift(M)</code>	- performs inverse Fourier transform of row-major matrix M . It must have one row for real data and two rows for complex;

Double interpolation:

<code>take(x; y; M)</code>	- returns the element of matrix M at indexes x and y ;
<code>line(x; y; M)</code>	- double linear interpolation from the elements of matrix M based on the values of x and y ;
<code>spline(x; y; M)</code>	- double Hermite spline interpolation from the elements of matrix M based on the values of x and y ;
<code>Tol</code>	- 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:

<code>\$Plot{$f(x)$ @ $x = a : b$}</code>	- simple plot;
<code>\$Plot{$x(t)$ $y(t)$ @ $t = a : b$}</code>	- parametric;
<code>\$Plot{$f_1(x)$ & $f_2(x)$ & ... @ $x = a : b$}</code>	- multiple;
<code>\$Plot{$x_1(t)$ $y_1(t)$ & $x_2(t)$ $y_2(t)$ & ... @ $x = a : b$}</code>	- multiple parametric;
<code>\$Map{$f(x; y)$ @ $x = a : b$ & $y = c : d$}</code>	- 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 - the number of color palette to be used for surface plots (0-9);
PlotShadows - draw surface plots with shadows;
PlotSmooth - smooth gradient coloring (= 1) or isobands (= 0) for surface plots;
PlotLightDir - 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
```

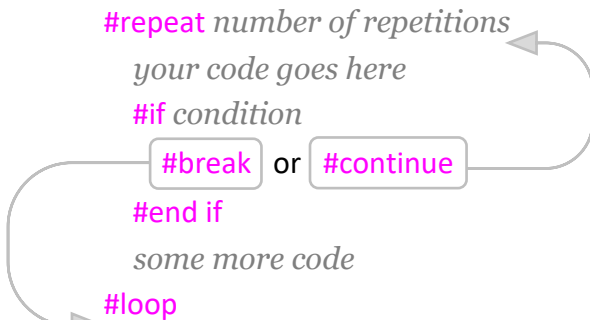
You can add or omit as many "#else ifs" as needed. Only one "#else" is allowed.
You can omit this too.

- Iteration blocks:

Simple:

```
#repeat number of repetitions
    your code goes here
#loop
```

With conditional break/continue:



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 - start local section (not to be included);
#global - 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 xlsxm):

```
#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, Ū, kŪ, MŪ, GŪ, TŪ, mŪ, µŪ, nŪ, pŪ;

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: **€**, **£**, **₣**, **¥**, **¢**, **₧**, **₹**, **₩**, **₺**.