Maple Cheat Sheet

Based on Maple 12/13 Windows Version

Syntax

`;` Ends a command with a semicolon. *e.g.* `5+6; plot(x);`

`:` Suppresses the display of output by ending a command with a colon. Useful for lengthy outputs or loading packages. *e.g.* `with(plots): 5000!:`

`:=` Assigns an expression to a variable. *e.g.* `x:=3; x:='x';` unassigns the variable x.

`=` Defines mathematical equations. *e.g.* `y = x^2 + 3*x + 4;`

`%` Refers to the last result. n of the % symbols refers to the n^{th} previous result. *e.g.* `%%%` gives the third previous presult.

f`:=(`*x,y,...*`)->...` Defines a function. *e.g.* `f := (x,y) -> x^2+y^2;` defines the function $f(x, y) = x^2 + y^2$. `f(0,1)` evaluates $f(0, 1)$.

L`:=[`*x1, x2, ..., xn*`]` Defines a list (ordered sequence) L of expressions $x_1, x_2, \ldots, x_n$. Refer to the n^{th} list item by $L[n]$. To extract the contents of a list, use the empty selection operator []. *e.g.* `A:=[1,2,3]; A[3];` returns 3. `A[]` returns 1,2,3.

S`:={`*x1, x2, ..., xn*`}` Defines a set S of expressions x_1, x_2, $\ldots$, x_n. Use the empty selection operator[] to extract the contents of a set. *e.g.* `S:={5,3,3,2,1}; S[];` returns 1,2,3,5.

name`[`*expression*`]` Indexed name. *e.g.* `b[1]; A[x,y,z];`

`?`*topic* Displays help on topic.

`#`*expression* Comment (not interpreted as commands).

All identifiers (variables and functions) are **case sensitive**. *e.g.* `X` is different from `x`. `Pi` and `pi` are different!

In general, a function whose name begins with a capital letter is an inert form of the function who has the same name but begins with lower case. Inert functions are unevaluated and may be manipulated and printed in a prettyprinted format. *e.g.* `Int(x,x);` returns $\int x dx$ and is the inert form of `int(x,x);`, which evaluates to $x^2/2$.

Usages and Icons

Right-click	Perform context operation on math expression
!	Execute the current line
!!!	Execute the entire worksheet
[>	Insert prompt
T	Insert text paragraph

Keyboard Shortcuts

`Enter`	Evaluate and display result on new line
`Shift + Enter`	Continue on next line without executing
`Ctrl + =`	Evaluate and display inline (Document Mode)
`Ctrl + Space`	Complete symbol/command
`F5`	Toggle Math/Text entry (Document Mode)
	Toggle 2-D/1-D Math entry (Worksheet Mode)
`Ctrl + F1`	Maple help

Defined Constants

`Pi`	$\pi \approx 3.14159265...$
`I`	complex number $I = \sqrt{-1}$
`infinity`	∞
`gamma`	Euler's constant $\gamma \approx 0.5772156649...$
`Catalan`	Catalan's constant $\approx 0.915965594...$
`exp(1)`	$e \approx 2.718281828$

Commands

General

`with(`*package*`):` Loads the specified Maple package.

`unassign(`*var*`);` Deletes a value stored in the given variable. *e.g.* `unassign('a');`

`restart;` Clears internal memory. The settings of all identifiers are resetted.

`unapply(`*expression, x, y, ...*`);` Returns a functional operator from an expression and variables. *e.g.* `f := x^2 + y^2; g:=unapply(f, x, y);` returns the functional operator `g := (x,y) -> x^2 + y^2;` such that `g(1, 2)` returns 5.

Common Mathematical Operations

`x + y - z;`	addition and subtraction
`x * y;`	multiplication
`x / y;`	division
`x^y;`	power x^y
`sqrt(x);`	square root $\sqrt{x}$
`exp(x);`	exponential e^x
`ln(x);`	natural log $ln(x)$
`log[b](x);`	logarithm $log_b(x)$
`surd(x,n);`	real n^{th} root $\sqrt[n]{x}$
`sin(x); cos(x); tan(x);`	trigonometric functions
`arcsin(x); arccos(x);`	inverse trig functions
`arctan(x);`	

Numerical Manipulation

`eval(`*expression*`);` Evaluates the given expression. *e.g.* `a:=b^2; b:=c+1; c:=2; eval(a);` returns 9.

`eval(`*expression*`, x=`*value*`);` Evaluates expression at the given point $x = value$. *e.g.* `eval(x^2+5*x, x=1);` evaluates the polynomial $x^2 + 5x$ at $x = 1$ and returns 6.

`eval(`*expression*`, {x=`*value1*`, y=`*value2*`,...});` Evaluates expression at the given points $x = value1$, $y = value2$,...

`subs(x=`*value*`,`*expression*`);` Substitutes the given value into expression. *e.g.* `subs(x=2,x^2+2*x+1);` gives 9. `subs(x=0, sin(x)/cos(x));` returns $sin(0)/cos(0)$.

`evalf(`*expression*`);` Numerically evaluates expression and returns its decimal approximation. *e.g.* `evalf(Pi);` returns 3.141592654.

`value(`*expression*`);` Evaluates the given inert expression. *e.g.* `F:=Sum(i,i=1..5); value(F);` evaluates the inert sum $\sum_{i=1}^{5} i$ and returns 15.

`assume(`*x*`, `*domain*`);` Restricts variable x to *domain*. Examples of *domain* are `positive`, `negative`, `posint`, `integer`, `real`, and `complex`. *e.g.* `assume(x, 'integer');` forces x to be an integer.

`assume(`*relation*`);` Enforces the given relational property. *e.g.* `assume(x > 0);` restricts x to be positive.

`additionally(`*x*`, `*domain*`); additionally(`*relation*`);` Places further restrictions on the given variable. Usages are similar to that for `assume`. *e.g.* `assume(x, real); additionally(x > 0);` forces x to be real as well as positive.

`map(`*f*`, `*expression*`, `*arg1*`, ..., `*argN*`);` Applies f to each of the operands or elements in *expression* along with the given arguments $arg1, \ldots, argN$. *e.g.* `map((x,y)->x+y, [0,1,2], 1);` adds one to each element in the given list to produce the result $[1, 2, 3]$.

Algebra

`simplify(`*expression*`);` Applies simplification rules to the given expression. *e.g.* `simplify(cos(Pi*cos(x)^2+Pi*sin(x)^2));` returns -1.

`collect(`*expression*`, `*variable*`);` Combines like terms in expression with respect to the given variable. *e.g.* `collect(a^2*x+b*x+5, x);` returns $5 + (a^2 + b)x$.

`normal(`*expression*`);` Simplifies and normalizes the given rational expression so that the result is of factored normal form, where the numerator and denomator are relatively prime polynomials with integer coefficients. *e.g.* `normal(1/x+x/(x+1));` returns $\frac{x+1+x^2}{x(x+1)}$.

`factor(`*expression*`);` Factors the given expression of a multivariate polynomial. Does NOT factor integers or integer coefficients in a polynomial. *e.g.* `factor(4*x^2+12*x+8);` returns $4(x + 1)(x + 2)$.

`ifactor(`*expression*`);` Factors an integer or rational number into a product of primes. *e.g.* `ifactor(24/19);` returns $\frac{(2)^3(3)}{(19)}$. `ifactor(2^10-1);` returns $(3)(11)(31)$.

`expand(`*expression*`);` Distributes the given expression. *e.g.* `expand((x+3)*(x+5));` returns $x^2 + 8x + 15$.

`solve(`*equations*`, `*variables*`);` Solves for the unknown variables in the given equations or inequalities.

e.g. `solve(x^2-25=0, x);` solves the equation $x^2 - 25 = 0$ and returns 5,-5.

e.g. `solve({x+y+z = 6, x-y+2*z = 5, 2*x+2*y+z = 9}, [x, y, z]);` solves the system of three equations and returns the solution [[x = 1, y = 2, z = 3]].

e.g. `solve(abs(x+5) > 3, x);` solves the inequality $|x + 5| > 3$ and returns $RealRange(Open(-2), infinity)$, $RealRange(-infinity, Open(-8))$.

`fsolve(`*equations*`, `*variable*`, [complex]);` Numerically solves for the unknown *variable* in *equations*. Use the `complex` option to find a complex solution. *e.g.* `fsolve(x^2+5*x-4,x);` returns $-5.701562119, .7015621187$.

`sum(`*f*`, `*k=m..n*`);` Returns the summation $\sum_{k=m}^{n} f(k)$. *e.g.* `sum(x^2, x=1..n);` computes $\sum_{x=1}^{n} x^2$.

Calculus

`limit(`*f*`, `*x=a*`, `*dir*`);` Computes the limit of f as x approaches a. a can be any algebraic expression or `infinity`. Direction *dir* is optional and is real bidirectional by default (except for ∞ and $-\infty$). Possible values of direction are `left`, `right`, `real`, and `complex`. *e.g.* `limit(1/exp(x), x=infinity);` computes $\lim_{x\to\infty} \frac{1}{e^x}$ and returns 0.

`diff(`*f*`, `*x1*`, ..., `*xj*`);` Differentiates f with respect to variables $x_1, \ldots, x_j$: $\frac{d^j}{dx_j \ldots dx_i} f$. *e.g.* `diff(sin(x), x);` takes the first derivative of $sin(x)$. `diff(f(x,y),x,y);` computes $\frac{\partial^2}{\partial y \partial x} f(x, y)$.

`diff(`*f*`, `*x$n*`);` Computes the n^{th} derivative of f: $\frac{d^n}{dx^n} f$. *e.g.* `diff(x^4, x$2);` computes the second derivative of x^4 and returns $12x^2$.

`implicitdiff(`*eq*`, `*x1*`, ..., `*xj*`);` Implicitly differentiates *eq* with respect to variables $x_1, \ldots, x_j$. The equation *eq* defines y as a function of $x_1, \ldots, x_j$ implicitly. *e.g.* `f:= y=x^2/z^2; implicitdiff(f, y, x);` computes dy/dx and returns $2x/z^2$.

`int(f, x);` Computes an indefinite integral of f with respect to the variable x. *e.g.* `int(cos(x), x);` computes $\int cos(x)dx$ and returns $sin(x)$.

`int(f, x=a..b);` Computes the definite integral of f with respect to the variable x on the interval from a to b. *e.g.* `int(x^2, x=0..2);` computes $\int_0^2 x^2 dx$ and returns $8/3$.

Differential Equations

`dsolve(ODE, y(x));` Solves ordinary differential equations for the unknown $y(x)$. ODE can be a single differential equation, or a set or a list of equations.

e.g. `ode:=diff(y(x),x$2)=2*y(x)+1; dsolve(ode, y(x));` solves the differential equation $\frac{d^2}{dx^2}y(x) = 2y(x)+1$.

`dsolve({ODE, ICs}, y(x));` Solves ordinary differential equations ODE for $y(x)$ given initial conditions. ICs are initial conditions given in the form `y(a)=b, D(y)(c)=d, ...`.

e.g. `ics := y(0) = 1, (D(y))(0) = 0; dsolve({ode, ics}, y(x);`

`DEplot(deqns, vars, trange, inits, xrange, yrange);` In the `DEtools` package. Plots solution curves to a system of differential equations by numerical methods. *deqns* - list of first order ordinary differential equations or a single differential equation of any order. *vars* - list of dependent variables. *trange* - range of the independent variable. *inits* - list of initial conditions. *xrange*, *yrange* - range of the two dependent variables.

e.g.
```
with(DEtools):
de1:=diff(x(t),t)=-0.5*x(t)*y(t);
de2:=diff(y(t),t)=0.5*x(t)*y(t);
DEplot([de1,de2],[x(t),y(t)],t=0..50,
x=-2..2,y=-2..2,[[x(0)=0.9,y(0)=0]]);
```

Linear Algebra

`Matrix(..);` Creates a matrix.

- `Matrix(a);` square matrix a x a filled with 0's
- `Matrix(a, b, c);` matrix a x b filled with c's
- `Matrix([[a, b, c],[d, e, f]]);` $\rightarrow \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}$

$< x_1, x_2, \ldots, x_j >$ Creates a column vector $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_j \end{bmatrix}$

`Vector`$[o]$`(..);` Creates a vector of orientation o (either `row` or `column`). Default orientation is column.

- `Vector(a, b);` column vector of size a filled with bs
- `Vector`$([a,b,c])$`;` column vector $\begin{bmatrix} a \\ b \\ c \end{bmatrix}$
- `Vector`$[row]$ $([a,b,c])$`;` row vector $\begin{bmatrix} a & b & c \end{bmatrix}$

Matrix addition, subtraction, multiplication: `A + B`, `A - B`, `A . B`

The following commands are in the `LinearAlgebra` package. Precede commands by `with(LinearAlgebra):`

`CrossProduct(u, v);` Computes the cross product of vectors u and v.

`DotProduct(u, v);` Computes the dot product of vectors u and v.

`Determinant(A);` Computes the determinant of matrix A.

`Norm(A, p);` Computes the p-norm of a matrix or vector A.

`Basis(v);` Computes the vector or set of vectors that forms a basic for the vector space spanned by v.

`Eigenvalues(A);` Computes the eigenvalues of matrix A.

`Eigenvectors(A);` Computes the eigenvectors of matrix A.

`MatrixInverse(A);` Computes the inverse of square matrix A.

`Transpose(A);` Computes the transpose of A.

Plots

`plot(f, x=xmin..xmax, options);` Creates a **two-dimensional** plot of the real function $f(x)$ over the horizontal range from $xmin$ to $xmax$. Options are specified in the form `option=value` (see box below).

- f is an expression or function with an independent variable. *e.g.* `plot(x^2, x=-5..5);`.
- f is represented parametrically: `[x(t),y(t),t=t0..t1]`. *e.g.* `plot([cos(t),sin(t),t=-2*Pi..2*Pi]);`
- f is a list of functions to be graphed on the same plot: `[f1, f2, ..., fn]`. *e.g.* `plot([1,x,x^2], x=-2..2);` puts the functions $y = 1$, $y = x$, and $y = x^2$ on the same plot.

`implicitplot(eqn, x=xmin..xmax, y=ymin..ymax, options);` In the `plots` package. i.e. Must be preceded by `with(plots):` Creates the two-dimensional plot of an implicitly defined curve *eqn* on the specified intervals: $[xmin, xmax]$ and $[ymin, ymax]$. Options are specified in the form `option=value` (see box below). *e.g.* `implicitplot(x^2+y^2=1, x=-1..1, y=-1..1);`.

`inequal(ineqs, x=xmin..xmax, y=xmin..xmax, options);` In the `plots` package. Plots regions defined by inequalities *ineqs* in the specified x and y intervals. Options are in the form `optionsfeasible / optionsopen / optionsclosed / optionsexcluded` = (`optionsList`), where *optionsList* is of the format (`option=value, option2=value2, ...`). *e.g.* `inequal({x+y>0, x-y<=1}, x=-3..3, y=-3..3, optionsexcluded=(color=blue,thickness=2));`

`plot3d(f, x=a..b, y=c..d, options);` Creates a **three-dimensional** plot of the real function $f(x, y)$ over the horizontal range $[a, b]$ and vertical range $[c, d]$. Options are specified in the form `option=value` (see box below).

- f is an expression or function with two independent variables. *e.g.* `plot3d(sin(x+y), x=-1..1, y=-1..1);`.
- f is represented parametrically: `[f1(x,y), f2(x,y), f3(x,y)]`. *e.g.* `plot3d([x*sin(x)*cos(y), x*cos(x)*cos(y), x*sin(y)], x=0..2*Pi, y=0..Pi);`
- f is a list of functions to be graphed on the same plot: `[f1(x,y), f2(x,y), ..., fn(x,y)]`. If there are three functions, use the `plotlist` option to avoid a parametric plot. *e.g.* `plot3d([sin(x*y),cos(x*y),x+y], x=-1..1, y=-1..1, plotlist);` puts the functions $z = sin(xy)$, $z = cos(xy)$, and $z = x + y$ on the same plot.

`implicitplot3d(eqn, x=a..b, y=c..d, z=i..j, options);` In the `plots` package. Creates the three-dimensional plot of an implicitly defined surface *eqn* on the specified intervals: $x = [a, b]$, $y = [c, d]$ and $z = [i, j]$. Options are specified in the form `option=value` (see box below). *e.g.* `implicitplot3d(x^2+y^2+z^2=1, x=-1..1, y=-1..1, z=-1..1);`.

`contourplot(f, x=a..b, y=c..d, options);` In the `plots` package. Creates a 2-D contour plot of the real function $f(x, y)$ over the horizontal range $[a, b]$ and vertical range $[c, d]$. Options are specified in the form `option=value` and can be those used in the `plot` commands or the following:

Number of contours	`contours=c`
Locations of contours	`contours=[a, b, c, ...]`
Filled contours	`filledregions=true`
Gradation coloring	`coloring=[color1,color2]`

e.g. `contourplot(cos(x*y), x=-3..3, y=-3..3, contours=4, filledregions=true, coloring=[red,blue]);`

`contourplot3d(f, x=a..b, y=c..d, options);` In the `plots` package. Creates a 3-D contour plot where contours are raised to their appropriate levels. Usages are similar to `contourplot`.

`spacecurve(curves, t=a..b, options);` In the `plots` package. Creates a curve or a set of curves in 3-D space. *curves* can be a list of points, a list of the x, y, and z components, or a set of such lists. Options are similar to those in `plot3d`. *e.g.* `spacecurve({[cos(t),sin(t),t], [t,sin(t),cos(t)]},t=0..4*Pi);`

`polarplot(r(theta), theta=a..b, options);` In the `plots` package. Creates a plot for $r(theta)$ for $theta = a...b$ in polar coordinates with polar axes. *e.g.* `polarplot(sin(theta),theta=0..2*Pi);`

`transform(f);` In the `plottools` package. Creates a function that applies the given function f to all points in a plot data structure. Useful for embedding 2-D plots into 3-D ones and transforming coordinate systems. *e.g.* `g:=transform((x,y)->(x,y,-1));` creates a procedure that transforms points with two coordinates (x, y) in a plot to points with three coordinates $(x, y, -1)$.

`animate(plotcommand, plotargs, t=a..b, options);` In the `plots` package. Creates a 2-D or 3-D animation on paramter t, ranging from a to b. *plotcommand* is a Maple command that generates a 2-D or 3-D plot (*e.g.* `plot`, `plot3d`, `implicitplot`). *plotargs* is a list of arguments to the plot command. Possible options are those used in the `plot` command or the following:

Number of frames	`frames=n`
Display a trace of n frames	`trace=n`

e.g. `animate(plot, [A*sin(x), x=0..10], A=0..2, frames=50, trace=5);`

`display(L, options);` In the `plots` package. Combines the list L of plot structures into a single plot or animation. *options* are those used for `plot` or `plot3d`.

e.g.
```
with(plots):
p1:=plot3d(sin(x*y), x=-Pi..Pi, y=-Pi..Pi):
p2:=plot3d([x+y, sin(x)], x=-Pi..Pi, y=-Pi..Pi):
display([p1,p2], axes=boxed, title="test plot");
```

Options for 2D Plots

Type of axes	`axes=boxed/frame/none/normal`
Color of curves	`color=blue/black/green/red/etc.`
Determine input discontinuities	`discont=true/false`
Draw gridlines	`gridlines=true/false`
Label Axes	`labels=[x,y]`
Scaling	`scaling=constrained/unconstrained`
Line thickness	`thickness=number`
Title	`title="plot title"`
Min/max y values	`y=ymin..ymax`
View window	`view=[xmin..xmax,ymin..ymax]`

Additional Options for 3D Plots

Contours	`contours=number`
Coordinate System	`coords=cartesian/cylindrical/spherical/etc.`
Grid Dimensions	`grid=[m,n]`
Label Axes	`labels=[x,y,z]`
View window	`view=[xmin..xmax,ymin..ymax,zmin..zmax]`