Maple Basics — A Quick Introduction

This is a reference sheet and is ideally meant to accompany one or more computer lab demo sessions, but may be used individually, sitting at a computer with Maple running, as I believe all essential info is included.

All examples were tested on Maple V Release 3, but I have tried to make it correct for Release 5.

Please notify me of errors and omissions.

Maple does algebraic computing, or more accurately "symbolic manipulation".
It can factorise or expand, cancel terms, differentiate, integrate, plot graphs, expand functions in series, solve equations, solve differential equations, take limits, and lots more stuff.

Basic Ideas

- Maple is **interactive** — you type a command and it executes it right away. Your input is red, and Maple's output is blue (unless you changed the settings).
- The **semi-colon** (;) followed by {Enter} is needed at the end of a command to input an expression, equation, etc. Without it Maple assumes the expression (or whatever) is continuing on the next line.
- For **long input lines**, just keep typing - it will wrap to the next line. Or, if you want the line breaks in your input expression to be at convenient places, hit {Enter} (without ;) to go to the next line.
- An *assignment* has ' := ' in it and ' : ' or ' ; ' at the end — see below. As in normal (numerical) computing, variables get assigned values. But here the 'values' are not numbers, but mathematical expressions, equations, etc. — e.g. `y := 5 * (x - 8) / (x + 20);`
- An *expression* doesn't have an equals sign in it. — e.g. $x^2 + y^3$, or $\lim_{x\to 0} \sin(x)/x$.
- An *equation* has an equals sign (=) (but no :=) in it — e.g. $y = f(x)$, or $5e^{A/7} = A - 99$.
  Note that you can assign a variable to be equal to an equation. Try
  `p_max_eq := diff(p(r), r) = 0;`
- A *function* has the form $function\_name(variable)$.
- Maple knows all the **standard functions** —
  e.g. `cos(Pi)`, `tan(Pi/4)`, `ln(x)`, `arcsin(c^2/g)`, and many more such as `BesselJ`.
  Factorial is `factorial(7);` or `(x^2)!`
  Note that `ln(x)` and `log(x)` are the same function, and `log10(x)` gives base 10 logs.
  Note the exponential e^x is not entered as `e^x` — rather put `exp(x)` .
  To see a list of known functions, type `? inifcns` at the prompt.
- Many of the commands given below, will work on **both expressions and equations**, though I may have put the argument as an expression. If you're unsure, just try it out!
- **Moving the cursor** Use the mouse or the {PageUp} & {PageDown} keys or arrow keys to position the cursor. Note that using the scroll bar on the right does not move the cursor. If you then press e.g. {Enter} or an arrow key, the screen will jump back to where the cursor is. Use a mouse click to put the cursor where you want after scrolling.
- **Editing** To correct or change an input line, use the {Delete} & {Backspace} keys for deletion, or just type new stuff in. Then press {Enter} to get Maple to revise the calculation. **Note** if the subsequent calculations are affected by this change, you must re-execute all of them.

- **Block operations** Most of the usual Windows things apply — particularly *cut*, *copy* & *paste*. (Note: in release 3, the screen will not move while you are marking, so, to mark a block that is bigger than the screen, point to the start of the block, mark a small portion, page down, hold down the {Shift} key, and then click at the end of the block.)

- **Worksheets** What you have when you've finished a calculation is a Maple worksheet. You can save it in the usual Windows fashion, as a `.mws` file (`.ms` in release 3). When you start Maple again, you can load it in the usual Windows manner, and start editing it. But note that Maple will not know any of the values of the variables unless you execute it.

- **Printing** You can print the current worksheet with the print button at the top. (Make sure you have a printer connection first.) To print just a part is trickier. I suggest the best way is to mark the block to be printed in the usual windows manner, copy it, open a new worksheet, paste, and then print the new worksheet. (Marking a block in the worksheet has no effect on the print options.) Otherwise you can use *File-PrintPreview* to see what's on each page, and then select the pages in the print dialog box.

Examples In the examples below it is often assumed that you have done the previous ones — i.e. that variables such as `X1` still have their values. I have often put several commands on the same line, to save space. It is better to use separate lines, though you can enter them this way.

Crucial Commands

Stop button Kill a calculation that's taking too long — it may not stop right away. (In release 3 you could use {Ctrl}-{Break}.)

{F1} Context sensitive help. It looks at what's near the cursor to choose the topic. (If the cursor is in an output line (blue), nothing happens.) There are two parts to the help window. The upper grey area is for topic selection, the lower yellow area gives the selected help, usually with examples and related commands at the bottom. You can drag the boundary between them up or down with the mouse. Alternatively use the help menu. **NOTE** that Maple's online help is good for mathematical operations, but dismal for system and interface.

? <*topic*> provide help on the given topic — semi-colon at end not needed — space after ? not needed. Only one of these per line is allowed,
e.g. `? diff`

<*variable*> := <*expression*>; Assign an (algebraic) value to a variable,
e.g. `A0 := a*x^2 + b*x + c;` `char_eq := a2*r^2 + a1*r + a0 = 0:`
Notice the ; at the end displays the result of the assignment, but : does not.

<*variable*>; Show the value of a variable,
e.g. `char_eq;`

<*variable*> := '<*itself*>'; Clear the value assigned to that variable — note you must use the single close quote both before and after the variable name — it's on the same key as the double quote character,
e.g. `A0 := 'A0';`

<*text*> Comment — anything after the # and before the next > is ignored by Maple, so even if the comment line wraps, you don't need another '#'. Another sort of comment is obtained by moving the cursor left until it is on the left of the '>', and then typing — it comes out in black letters — hitting {Enter} to put a new '>' on the next line.

[> button Insert a new prompt. Put the cursor in the line above where you want the new prompt, and then click on the button at the top with a > symbol on it. Alternatively, put the cursor in the line below where you want the new prompt, use the left arrow key to move the cursor to the far left,

when the left square bracket becomes fatter, then hit {Enter}. Alternatively, use the *Insert* menu. (In release 3, {Ctrl}-O inserts a prompt above the cursor line, and {Ctrl}-I inserts one below it.)

`restart;` Start over again. You can type this at any point, and the values of all variables will be cleared. This is often needed at the top of a worksheet you keep changing, as the following example demonstrates:

```
V1 := a*T^2 - b*T + c = 0;
soln := solve(V1, T);
T := soln[1];
```

Now go back up to the V1 line and correct "- b" to "+ b" and hit {Enter} 3 times to reapeat the 3 calculations — you get the new value of T in the expression for V1, and you end up with an error. You see, Maple works with whatever assignments it currently has stored in memory. Use `restart` to wipe all assignments. If you put it in the middle of a worksheet, be sure you don't want any of the assignments done above it. It's usually best at the top. I also strongly recommend giving solutions different variable names, to prevent problems like in the above example happening, e.g. `T_sol := soln[1];`

Edit-Execute-Worksheet To run the whole worksheet, use the mouse to click through the menus as indicated, or type `Alt-E, E, W` .

New sheet button To open a new blank worksheet, click on the blank page button on the top (or click *File*, *New*). Note that the old worksheet is still open, and Maple still knows all variable values from the current Maple session, so it's like a continuation. (In release 3 this closes the present worksheet and starts a new blank worksheet.)

`quit` Quit the worksheet. You can also just close the window. (In release 3 this quits Maple.)

Variable Names & Reserved Words

Variables names are made up like any computer language — they start with a letter, and can have any number of letters, digits (numbers) and underscore charaters after. (The underscore is the other character on the minus key.) They can't have spaces, and they can't start with a digit. Capital & small letters are not the same. Unfortunately, they can't be very long — the maximum is 500K characters!

Maple uses variable names that start with an underscore as global variables — e.g. `dsolve` gives constants of integration as `_C1` , `_C2` , etc.
It also uses `%1` , `%2` , `%3` , etc for component expressions which occur several times in the output; and it then defines them at the bottom — see the `convert( ... , radical);` example below.
You should be careful how you use these types of variables.

There are some variables with special meaning that you can't use, e.g:

`Pi`	$\pi = 3.14159...$
`E`	Logarithmic constant $e = 2.71828...$
`I`	Square root of -1, i.e. i.
`infinity`	∞.

There are also various keywords, such as `do` , `read` , etc, that can't be used, and various protected words, such as `sin` , `cos` , `W` , that can't be changed. For lists of such words, do each of these (on a separate line): `? ininames` `? keywords` `? protect`

Basic Commands

`expand(`<*expression*>`);` Expand multiplied brackets, trig functions, etc,

e.g. `E1 := (P + Q)^2*(A + B)^2;` `X1 := expand(E1);`
`E2 := (P + Q)*(A - B)/(A^2 - 2*A*B + B^2);` `X2 := expand(E2);`
works on equations too: `Eq2 := expand(sin(4*theta) = cos(5*phi));`

`normal(`<*expression*>`);` Put into normal form — viz. polynomial/polynomial ,

e.g. `normal(1/(x + y) - 3/(w - z));` `normal(X2);`
terms may be cancelled between numerator & denominator, and the denominator may be factorised, but only if the expression is simple.

`simplify(<`*expression*`>);` Simplifies the expression. There is a whole host of techniques it tries, so it can take quite long on big expressions. (Sometimes the result will seem less simple than the input.)
e.g. `asx := arcsin(sin(x)); simplify(asx); y7 := ln(7*exp(y)); simplify(y7);`

`subs(<`*equation_list*`>, <`*expression*`>)` Substitute the equations listed into the expression,
e.g. `subs(A = 0, X2); subs(B = A, P = A, Q = A, X1);`
Sometimes there's little or no evaluation or simplification done after the substitution — see `eval` later.

`numer(<`*expression*`>);` Take the numerator of the given expression,
e.g. `n1 := numer(X1/X2);`

`denom(<`*expression*`>);` Take the denominator of the given expression,
e.g. `d1 := denom(X1/X2);`

`factor(<`*expression*`>);` Factorise the expression — great for smallish expressions, but takes enormously long for large expressions, and can fail for really big expressions, even when there are factors,
e.g. `factor(n1); factor(d1);`

`simplify(<`*expression*`>, symbolic);` Simplifies some expressions that the normal `simplify` can't, especially if funny powers are involved,
e.g. `A1 := G^(1/(1 - p)) * ( H^((q - p)/(q + p)) * G^(1 + p) )^((3 + q)/(1 + p));`
Try each of `normal(A1); simplify(A1); expand(A1);` and `factor(A1);` and then try `simplify(A1, symbolic);`

`convert(<`*expression*`>, parfrac, <`*variable*`>);` Do a partial fraction expansion on the given expression, with the given main variable. Gives enormous simplification of certain expressions
e.g. `Px := (x^2 + 17*x + 71)/(x^3 + 23*x^2 + 176*x + 448);`
Try each of `normal(Px); simplify(Px); expand(Px);` and `factor(Px);` and then try `convert(Px, parfrac, x);`

`solve(<`*equation_list*`>, <`*variable_list*`>);` solve the given equation(s) for the given variable(s),
e.g. `solve(A = 0, x); soln := solve(E2 = 0, P);`
`X1s := solve(X1, B); simplify(X1s[1]);`
`solnxy := solve({x^2 + y^2 = 25, x*y = 8}, {x, y});` notice there are two solutions:
`solnxy1 := solnxy[1]; solnxy2 := solnxy[2];`
WARNING: Maple does not always present solutions in the same order when you re-run a worksheet — so using the first solution may be wrong the next time you run the calculation. I don't know a simple way round this problem.

`convert(<`*equation*`>, radical);` Converts `RootOf( ... )` to radicals and/or complex numbers, where possible,
e.g. `convert(RootOf(z^2 + 1), radical); convert(solnxy1, radical);`
`eq_list := ca = sw*sv, cw = -sa*cj, -sa*sj = sw*cv, cb*ca*cj - sb*sj = 0,`
`cb*ca*sj + sb*cj = sv, -sb*ca*sj + cb*cj = cw*cv;`
`var_list := cv, sv, cw, sw, cj, sj;`
`SOLN := solve(eq_list, var_list);` Note the variable `%1`, defined at the end
`convert(SOLN, radical);`

`assign(<`*equation_list*`>);` Assign the values given on the right of the equations to the variables on the left,
e.g. `assign(subs(x = xs, y = ys, solnxy1)); xs; yx;`
Notice we use `subs` to avoid assigning values to `x & y`, which could create the problem shown in the `restart` example.

`plot(<`*expression_list*`>, <`*variable*`> = <`*range*`>);` Plot the function(s) over the given range,
e.g. `plot(x^5 - 270*x^3 + 6000*x, x = -15..15);`

e.g. `plot({sin(x), cos(x)}, x = -pi..pi);`
(In release 3, the plot would appear in a separate window. To put the plot in the worksheet, in the plot window click *Edit*, *Copy* (or {Ctrl}-C), then in the worksheet window put the cursor on line with the `plot` command, and click *Edit*, *Paste* (or {Ctrl}-V)). Do `? plot` to see all the options.

`seq(`<*indexed expression*>`, `<*range*>`)` Create a sequence - list of expressions,
e.g. `seq1 := seq(i^2, i = [2, 5, 6, 11]);` `seq(x^j/j!, j = 0..6);`
`seq(A, A = "a".."z");` `seq(Z, k = 1 .. 10);` `seq1[3];`
Sequence order is preserved. The `$` operator does similar things. Do `? $` .

`sum(`<*expression*>`, `<*variable*>` = `<*range*>`);` Sum the expression over each value of the variable in the given range,
e.g. `sum(x, x = 1..100);` `sum('1/J!', J = 0..infinity);` `sum(y^i, i = 1..5);`
But you can't do `sum(seq1)` , which seems ridiculous.

`product(`<*expression*>`, `<*variable*>` = `<*range*>`);` Do the product of the expression over each value of the variable in the given range,
e.g. `product(y, y = 1..100);` `product(j!*(10-j)!, j = 0..10);`
`product(y^k, k = 3..9);`

`.` The concatenation operator — just a full stop. Very useful for creating a sequence of variable names, especially if you're writing Maple programs,
e.g. `Var1 := a;` `Var2 := 3;` `Var3 := b;` `Var1.Var2.Var3 := Var1*Var2*Var3;`
The variable on the left of the first dot is not evaluated, the rest are. To get all to evaluate, do
``` ``.Var1.Var2 := Var1^2*Var2^2; ``` where ``` `` ``` is the null string.
Note the quotes used here: `sum('a.i*x^i', i = 0..10);` They aren't the same quotes! These are for delayed evaluation, needed here so the sum is written out first — i.e. the `i`'s become 1, 2, 3, ... — and concatenation done second. (Try it without them.) (Do `? '`)

`diff(`<*expression*>`, `<*variable_list*>`);` Differentiate the expression with respect to the listed variables in turn. Note that these are <u>partial</u> derivatives. Also if you don't say that f depends on x then Maple takes its derivative as 0,
e.g. `diff(ln(x), x);` `diff(ln(x), x, x);` `diff(x^2*y^2, x, y);`
`diff(ln(x/y), x, y);` `diff(a*f, x);` `diff(a*f(x,y), x);` `diff(a*f(x), x$5);`
Construct your own total derivatives — a little untidy, as Maple won't write 'd' rather than '∂':
e.g. `dx_dt := diff(x(t), t);` `dy_dt := diff(y(t), t);`
`df_dt := diff(f(x,y), x)*dx_dt + diff(f(x,y), y)*dy_dt;`

`Diff(`<*expression*>`, `<*variable_list*>`);` "Inert" differentiation — the derivative is not evaluated — sometimes useful. Convert to active form with `value` ,
e.g. `Diff(ln(x), x);` `value(Diff(sin(x), x));`

`int(`<*expression*>`, `<*variable*>`)` Integrate the given expression with respect to the given variable. This is <u>indefinite</u> integration, except that <u>Maple gives no constant of integration</u>, you must add it yourself, if needed:
e.g. `int(ln(x^2), x);` `int(1/(1 + exp(x)), x);` `int(f(x), x);`

`int(`<*expression*>`, `<*variable*>` = `<*range*>`)` Integrate the given expression with respect to the given variable over the given range. This is <u>definite</u> integration.
e.g. `int(sin(x)/x, x = 0..infinity);`
(Be warned: there are many standard definite integrals for which you'll get odd but correct results.)

`Int(`<*expression*>`, `<*variable*>`)` "Inert" integration — the integral is not evaluated — sometimes useful — see the student package function `intparts` , given later. This command has definite & indefinite integration versions too. Convert to active form with `value` ,
e.g. `Int(x^2, x = 0..5);` `value(Int(sin(x), x));`
```

{<*item 1*>, <*item 2*>, ... } A set — use curly brackets. The items can be expressions, equations, variable names, strings, etc. The order doesn't matter.
e.g. `seq2 := b, c, a;` `{seq2};` `setA := {seq1};` `setB := {5*i $ i = 1 .. 5};` `setA union setB;` `setA intersect setB;`

[<*item 1*>, <*item 2*>, ...] A list — use square brackets. The order is preserved.
e.g. `Lst2 := [seq2];` `Lst2[1];`

`dsolve(`<*differential_equation_list*>`,` <*function_list*>`)` solve the given differential equation(s) for the specified function(s),
e.g. `IVP := diff(y(x),x,x) + y(x) = cos(x), y(0) = 19;` `dsolve(IVP, y(x));`
Note that Maple gets a rather awkward solution here, which should have been simplified to $x\sin(x)/2 + C_1\cos(x) + C_2\sin(x)$.
e.g. `dsolve({diff(x(t),t) = a - b*y(t), diff(y(t),t) = c - d*x(t)}, {x(t), y(t)});`
Notice the system of DEs and the functions to be solved for are in curly brackets — they're sets.

`eval(`<*expression*>`)` Evaluate an expression further,
e.g. `DEsub := subs(y(x) = x*sin(x)/2, DE);` `eval(DEsub);`

`nops(`<*expression*>`);` Give the number of operands of the given expression — i.e. how many main parts it breaks down into,
e.g. `nops(X2);`

`op(`<*integer*>`,` <*expression*>`);` *integer* is optional. Show all operands of the given expression, or just operand number *integer*. If *integer*$= 0$ you get the operator,
e.g. `op(X2);` `op(0, X2);` `op(2, X2);` and this latter also has several parts:
`nops(op(2, X2));` `op(0, op(2, X2));` `op(op(2, X2));` but when parts are multiplied they aren't always in the order you expect — do `op(`i`, op(2, X2));` for all 4 values of i.

`series(`<*expression*>`,` <*variable*> `=` <*value*>`,` <*integer*>`);` Do a Taylor series expansion of the expression about the given value of the variable, up to the given integer order (this argument is optional) — i.e. the error term $O(x^m)$ has $m =$ <*integer*>,
e.g. `ss := series(sin(x), x = x0, 8);`

`convert(`<*series*>`, polynom);` Convert a series to a polynomial — i.e. it removes the $O(x^m)$ term, which will cause errors in many calculations,
e.g. `ss := convert(ss, polynom);`

`evalf(`<*expression*>`,` <*integer*>`);` Do a floating point evaluation of the expression to the given number of digits (this argument is optional),
e.g. `evalf(subs(x = 35, cos(x^2)));` `evalf(Pi, 60);`

`map(`<*operation*>`,` <*expression*>`,` <*optional arguuments*>`);` Apply a function/operation/procedure to each operand of an expression separately. If <*operation*> takes more than one argument, they go in <*optional arguments*>,
e.g. `map(Diff, a*b*c);` `map(x -> x^2, x + y);`
`S8 := normal((U + a)^4/(x - y) + 2*(V - b)^3/(x + y));`
`S9 := convert(S8, parfrac, x);` `factor(S9);` `map(factor, S9);`

`map2(`<*operation*>`,` <*argument 1*>`,` <*expression*>`,` <*optional arguuments*>`);` Like `map` , but makes the expression the second argument of the function/operation/procedure. Do `?` `map2` .

`subsop(`<*sub 1*>`,` <*sub 2*>`, ...` <*expression*>`);` Do a substitution in a part of an expression. Each of the <*sub i*> has the form <*operand*> = <*substitution expression*> , and <*operand*> is what you'd put in an `op` command,
e.g. `ZA := x*y + u/v;` `subsop([2,1] = cos(r), ZA);`

WARNING: With complicated expressions, Maple does not always arrange terms and factors the same way each time it runs the worksheet, so you could easily substitute for the wrong part, or get an error.

`applyop(<`*function*`>, <`*operand 1*`>, <`*operand 2*`>, ... <`*expression*`>, <`*optional arguments*`>);`
Apply a function/operation to a part of an expression. Use <*optional arguments*> if <*function*> takes more than one argument,
e.g. `Pp := A*x^2 + B*y - C/z;` `applyop(f, [3,2], Pp);`
`Qq := sin(x^2 - 1)*cos(21*x^2 - 2*x - 55);` `applyop(factor, [2,1], Qq);`

`select(<`*condition*`>, <`*expression*`>, <`*optional arguuments*`>);` Pull out parts of an expression or list with given property,
e.g. `XPR := exp(y) + b*x + c*x^2 + d*x^3 + cos(z)^4;`
`select(has, XPR, d);` `select(isprime, {10, 11, 12, 13, 14, 15});`
`select(type, XPR, function);` `select(type, XPR, `^`);`
Also check out `remove` and selectremove .

The Student Package

`with(student);` Load the student package.

`intparts(Int(<`*expression*`>, <`*variable*`>), <`*factor*`>);` Integrate the given integral by parts, taking the given factor (expression) as the part that is differentiated,
e.g. `intparts(Int(x*sin(x), x), x);` or if you want the integral to be carried out, don't use inert integration: `intparts(int(x*sin(x), x), x);`

The Linear Algebra Package

`with(linalg);` Load the linear algebra package — for matirx manipulation, solving sets of linear equations, etc.

`matrix(<`*element_list*`>)` put the elements listed into a matrix,
e.g. `MM := matrix([[1, exp(y)], [7, f(z)], [-x, 1/w]]);`
`uu := matrix([[1, 0], [3, 8]]);` `vv := matrix([[exp(-y)], [44]]);`

`vector(<`*element_list*`>)` put the elements listed into a row vector,
e.g. `vec := vector([y^2, y^3]);`

`&*` Matrix multiplication. (Addition, subtraction, & raising to the power do not have special symbols.)
e.g. `ww := MM &* vv;` `w2 := MM &* vec;` but this is no good: `w3 := vec &* MM;`
Notice that the result is not displayed. In fact it is not even calculated. Which is why you need the next command:

`evalm(<`*matrix_expression*`>)` evaluate the matrix expression and display it in full,
e.g. `evalm(ww);` `evalm(w2);` `evalm(w3);` `evalm(uu &* vv);`

`det(<`*square_matrix*`>)` Give the determinant of the matrix,
e.g. `det(uu);`

`transpose(<`*matrix*`>)` Give the transpose of the matrix,
e.g. `transpose(vv);`

`trace(<`*square_matrix*`>)` Give the trace of the matrix,
e.g. `trace(uu);`

`inverse(<`*square_matrix*`>)` Calculate the inverse of the given matrix,
e.g. `inverse(uu);` or `evalm(uu^(-1));`

`Wronskian(<`*vector*`>, `*variable*`);` Calculate the Wronski matrix of the given vector, which is taken as a set of functions of the given variable — i.e. Maple must differentiate with respect to that variable,
e.g. `Wronskian([1, exp(x), x*exp(x)], x);` `Wronskian(vec, y);`

`dotprod(<`*vector*`>, <`*vector*`>)` Returns the dot product of the given vectors,
e.g. `dotprod([7, 11], vec);`

`convert(<`*list_object*`>, vector)` Converts the list object — which must be a matrix, list or array — into a vector,
e.g. `dotprod(vec, convert(vv, vector));`

`crossprod(<`*vector*`>, <`*vector*`>)` Returns the cross product of the given 3-element vectors,
e.g. `crossprod([0,0,1], [0,1,0]);`

Programming You can write Maple programs pretty much like any numerical language, using conditions, loops, etc. Get help on `do` and on `if`. Note that `for` and `while` can be combined in a `do` loop, to give a conditional loop with a counter — which is why there's one help page for both.

Once you use these, you will have to know how to combine separate command lines into one 'execution group', for which Maple help is non-existent. An execution group is a set of commands within a single left square bracket, which are all executed at once by pressing `{Enter}`. Loops etc must begin and end within one execution group — you can't execute a loop with no bottom line! Experiment with *Split or Join* in the *Edit* menu, and with `{F3}` and `{F4}` — watch how the cursor position affects what happens — to me it doesn't seem very logical or consistent between `{F3}` and `{F4}`. To insert a new prompt within an execution group, put the cursor between the prompt (>) and the left square bracket (`[`) and hit `{Enter}`. You will have already noticed that entering a command with no ';' or ':' will give you a new prompt in the same group.

Once you start writing long programs you will want to create subroutines, or rather procedures as they are called in Maple. Look up `? proc`. Essential commands for debugging programs and procedures are `printlevel` and `debug`.

More useful commands, or ones which have useful options — follow them up in the Maple help:
`combine`, `coeff`, `assume`, `expand`, `factor`, `simplify`, `convert`, `plot`, `rhs`, `lhs`, `maximize`, `minimize`, `gcd`, `lcm`, `Re`, `Im`, `laplace`, `alias`, `functional`, `pdesolve`, `D`, `collect`, `sort`, `limit`, `mtaylor`, `list`, `set`, `latex`, `library`, `linalg`, `grad`, `diverge`, `curl`, `laplacian`, `jacobian`, `spreadsheet`, `DEtools`, `student`, `geometry`, `projgeom`, `packages`.

Books The following 3 books come with Maple. They are essential info, but not as helpful as you'd like. (Indexes are much too short, and page references are sometimes wrong (e.g. for '`convert`'). Perhaps its better in the new editions.)

- *Maple V First Leaves: A Tutorial Introduction to Maple V*, B.W. Char, K.O. Geddes, G.H. Gonnet, B.L.Leong, M.B. Monaghan, S.M.Watt, Springer-Verlag, 1992.
- *Maple V Language Reference Manual*, B.W. Char, K.O. Geddes, G.H. Gonnet, B.L.Leong, M.B. Monaghan, S.M.Watt, Springer- Verlag, 1991.
- *Maple V Library Reference Manual*, B.W. Char, K.O. Geddes, G.H. Gonnet, B.L.Leong, M.B. Monaghan, S.M.Watt, Springer- Verlag, 1991.

I recommend the following, which I have. This edition covers up to release 3:

- *Maple V Quick Reference*, N.R. Blackman & M.J. Mossinghoff, Brooks/Cole, 1994.

Maple is developed at the University of Waterloo, in Ontario, Canada.
Waterloo Maple's website is at http://www.maplesoft.com

Ref: `MAPLE\MAPLEBAS.TEX`, Charles Hellaby 2003