Mathematica Tips, Tricks, and Techniques

$\boxed{\text{Syntax}}$

Michael A. Morrison

(Version 3.4: February 2, 2000)

Applying computer technology is just a matter of finding
the right wrench to pound in the correct screw.

—Anonymous

Contents

1 Basic Syntax: What Everyone Should Know from the Outset. (Beginners) 3

1.1 What are the three most common syntax errors—and how can I avoid them? 3
1.2 What are the five most important aspects of *Mathematica* syntax? 3
1.3 I want a summary of the ways *Mathematica* uses parentheses, brackets, braces, etc. 3
1.4 What do I absolutely have to know about command names? 4
1.5 How can I exploit *Mathematica*'s naming convention to distinguish my commands from its commands? 4
1.6 *Mathematica* pretends not to know the name of a built-in command I just told it to execute. What's wrong? 4

2 How *Mathematica* Syntax *differs* from Mathematics Syntax. (Everybody) 4

2.1 What's a sure-fire safe way to avoid conflicts when I define a new function? 4
2.2 How do I specify negation in an equation? 4
2.3 How do I raise a quantity to a power? 4
2.4 I keep telling *Mathematica* a number is complex, but it persists in treating it as real. What's wrong? 5
2.5 *Mathematica* isn't treating my *equations* properly. What's wrong? 5
2.6 What's the correct syntax for a matrix? 5

3 Finer Points of *Mathematica* Syntax. (Intermediate) 5

3.1 How can I make sense out of the myriad equation solving commands *Mathematica* offers? . . 5
3.2 The output from my equation solving commands looks useless. How do I extract the actual answer from this output? 5
3.3 When should I use the special *Mathematica* syntax to refer to the output from a command I recently executed? 5
3.4 How do I enter text (e.g., file names, comments) in *Mathematica*? 6
3.5 *Mathematica* won't load a package I know it has access to! Am I incorrectly using `Needs`? . . 6
3.6 *Mathematica* isn't evaluating pieces of my expression in the right order. How can I control the evaluation order? 6

4 Special Syntax for Functions and Assignments. (Intermediate) 6

4.1 How can I restrict my function to selected argument ranges? 6
4.2 What's the difference between immediate and delayed assignments in function definitions? . . 7
4.3 What is *Mathematica*'s syntax for denoting arguments to functions? 7
4.4 I want to make a one-shot replacement in an expression? Do I have to define a function? . . . 7
4.5 I told *Mathematica* to implement a replacement for an expression that occurs several times but it only made the replacement once! What do I do? 7

5 Bells and Whistles. (Advanced) 7

5.1 How can I easily apply a function to a whole bunch of argument values? 7

5.2 What's a handy, powerful shorthand way to define functions? 8
5.3 Can I change the head of an expression without re-defining the expression? 8
5.4 How can I determine the head of an expression? 8
5.5 Can I define default values to my function arguments? 8
5.6 I am setting up a sequence of commands. What's the cleanest, clearest, most straightforward way to do this? 8

1 Basic Syntax: What Everyone Should Know from the Outset. (Beginners)

1.1 What are the three most common syntax errors—and how can I avoid them?

1. *Multiplication!*

   The single most common mistake newcomers make is to assume that ab means a times b. It doesn't. It's the name of the *single variable*, ab. If you want to write a times b, use the asterisk, as `a*b`.

Tip *Mathematica* allows you to indicate multiplication using either a space or an asterisk. Thus `2 Sin[x]` and `2*Sin[x]` are identical. You're on safer ground, though, with the asterisk.

2. *Functions!*

   The second most common mistake is to try to tell *Mathematica* to evaluate $\sin x$ by typing `Sin(x)`. This won't work! *Arguments to functions must be enclosed in* square braces*, not parentheses*, as `Sin[x]`.

3. *Equations!*

   The third most common mistake is to try to write the equation $a = b$ as `a=b`. That's *not* an equation; it's an **assignment**: it tells *Mathematica* to *expression* `a` identical to the *Mathematica* expression `b`. If you want to type the *equation* $a = b$, you *must* use the "double equals" notation, as `a==b`.

1.2 What are the five most important aspects of *Mathematica* syntax?

1. *Command Names!*

   There are three kinds of commands in *Mathematica*: built-in commands (like `Sqrt[ ]`), commands defined in packages (like `Laplacian`), and user-defined commands (like `myfunction`). *All built-in and package commands begin with a capital letter.* Commands you define should begin with a lower-case letter.

2. *Arguments!*

   All *Mathematica* commands and functions which take arguments share a common syntax. *The argument(s) to any function or command must be enclosed in square brackets:* `[<` $\cdots$ `>]`. For example, we denote $\sin x$ by `Sin[x]` *not* `Sin(x)`.

   This requirement holds even if the argument to a function or command is a list, as in `ListPlot[{1,3,87}]`.

3. *Lists!*

   Most of the items you enter in *Mathematica* are either (a) algebraic expressions (or equations); (b) assignments (like `a=b`); function definitions; instructions that tell *Mathematica* to perform some task (like `Integrate` or `Plot`); or lists. The **list** is extremely common, and all lists have the same structure. The items in a **list** are enclosed in curly braces. Separate elements of a list by commas, as `{a, b, c}`. Some lists contain as elements other lists. Not to worry: the structure is still the same.

4. *Parts of a List!*

   To refer to an **element of a list**, use *double square brackets.* Thus we denote the third element (called the third "part") of the list named `A` as `A[[3]]`.

5. *Iterators!*

   Many *Mathematica* commands take **iterators** as arguments. *An iterator is a list.* Its elements specify the *range of a particular argument and its increments*: i.e., the starting value, stopping value, and increment between adjacent values. The generic form of an iterator is `{i, imin, imax, istep}`.

1.3 I want a summary of the ways *Mathematica* uses parentheses, brackets, braces, etc.

Ask and ye shall receive:

arguments to functions	square brackets	`f[x,y]`
lists, vectors, & matrices	curly braces	`{x,y,z}`
indexing or extracting part of a list	double square brackets	list[[1]]
grouping in mathematical expressions	parentheses	`a(b+c)`
comments	parentheses and asterisks	`(* comment *)`

1.4 What do I absolutely have to know about command names?

Only three things—and one we've already mentioned.

1. *Capital Letters!*

   All built-in *Mathematica* **commands** begin with a capital letter.

2. *Complete Words (mostly)!*

   Almost all *Mathematica* commands are complete words (not abbreviations).

3. *One Word from Many, all Capitalized!*

   If a command consists of more than one word, the first letter of each word is capitalized, as in `ParametricPlot[ ]`. In this case, all words are adjacent: the command to construct a polar plot is `PolarPlot`, *not* `Polar Plot`.

1.5 How can I exploit *Mathematica*'s naming convention to distinguish my commands from its commands?

Begin all your variable names, functions definitions, and other expression with a lower case letter, e.g., `psi[x]`, not `Psi[x]` and `plotWaveFunction[]`, not `PlotWaveFunction`.

1.6 *Mathematica* pretends not to know the name of a built-in command I just told it to execute. What's wrong?

If the kernel just returns your command unevaluated, you probably forgot to use capitals where required in the command. Try using the Help Browser or command completion (in the `Input menu`, via the submenu `Complete Selection`.

It is possible, however, that *Mathematica* may know your command but be unable to evaluate your command. If you suspect this, check your command by inquiring via the `?` query or poke around in the Help Browser or try Command Completion

2 How *Mathematica* Syntax *differs* from Mathematics Syntax. (Everybody)

2.1 What's a sure-fire safe way to avoid conflicts when I define a new function?

In the cell where you define your function, *always* make the first statement `Clear`.

The command `Clear[f]` instructs *Mathematica* to forget any assignments you might have already made to the symbol `f`. If you've not made any such assignments, this command does nothing. It's insurance, and you should make this a firm habit from the start!

2.2 How do I specify negation in an equation?

Precede the operation to be negated by `!`. Thus `!=` signifies $\neq$.

2.3 How do I raise a quantity to a power?

Use the symbol ^ (`Shift 6` on your keyboard). Thus to generate x^3, write `x^3`.

Tip If you're using version 3.0 or higher, use `CNTL 6` on your keyboard to get a two-dimensional representation x^3. Get in the habit of always using this two-dimensional form, as it makes your input and output easier to understand and check.

2.4 I keep telling *Mathematica* a number is complex, but it persists in treating it as real. What's wrong?

Probably you're using `i` to denote $\sqrt{-1}$. In *Mathematica*, this number is denoted by a capital letter, as `I`.

Thus the complex number $a + ib$ is written `a + I*b` (note the asterisk to denote multiplication).

2.5 *Mathematica* isn't treating my *equations* properly. What's wrong?

You probably entered the equation the way you would write it down or see it in a book. This won't work. In *Mathematica*, you must specify equations using the logical operator `==`.

To understand the difference between an assignment and an equation, study the following example. In it, we use an immediate assignment `=` to assign the name `eqn` to the equation $\sin x + 7 = 0$:

```
eqn = Sin[x] + 7 == 0
```

This, of course, would be nonsense in a mathematics book.

Warning We are so used to using = to signify equations that it's very easy to fall back into this habit in *Mathematica*. Watch out.

2.6 What's the correct syntax for a matrix?

In *Mathematica*, we specify a matrix by a list of lists. Each element of the (outer) list denotes a *row* of the matrix.

Thus to specify

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

type

```
A = { {0,1},{1,0} }
```

3 Finer Points of *Mathematica* Syntax. (Intermediate)

3.1 How can I make sense out of the myriad equation solving commands *Mathematica* offers?

Easy! They all have the same syntax!

All built in commands that **solve equations**, either algebraically or numerically, have a common form indicated here by the simplest of them, the command to solve n equation algebraically:

`Solve[{`< equations you want solved >,< variables to solve for >`}]`.

3.2 The output from my equation solving commands looks useless. How do I extract the actual answer from this output?

Commands that solve equations generate *lists of replacement rules*. You can extract the solution from such lists by applying the replacement rule with the `/.` symbol. For instance, to solve the equation $x + 4 = 7$, type

```
soln = Solve[x+4==7, x]
x /. soln
```

For more about replacement rules, see below.

3.3 When should I use the special *Mathematica* syntax to refer to the output from a command I recently executed?

Never!

Tip I urge you to avoid this bad habit. In most notebooks, you'll delete cells and forget about them, so what *looks* like the previous output may not actually be the one you want—especially next time you execute the notebook. Just *name your output* using an assignment statement, then you can always refer to it.

Okay. If you must, use `%n` where `n` is the number in the desired `Out[n]` generated by your command `In[n]`.

3.4 How do I enter text (e.g., file names, comments) in *Mathematica*?

Any string of characters (e.g., a file name, a text label for a graph, a context name), must be enclosed in double quotation marks.

If you're using a notebook, just put your comments in a text or small text cell.

If you want to annotate a *Mathematica* command, enclose your **comment** in the symbols `(*` and `*)`. For example,

```
Z = 1 (* set the atomic charge *)
```

3.5 *Mathematica* won't load a package I know it has access to! Am I incorrectly using `Needs`?

Probably you just forgot that context names (e.g., the names of packages in `Needs` statements) always must be terminated by a backquote and must be enclosed in quotation marks. This backquote is part of the context name, as `Global‘`.

3.6 *Mathematica* isn't evaluating pieces of my expression in the right order. How can I control the evaluation order?

Use parentheses to group commands.

Tip Use parentheses to control the sequence in which *Mathematica* executes commands in a multi-command expression. For example, `a (b + c)` will generate the desired result $ab+ac$; if, however, you type `a b+c`, then *Mathematica* will calculate $ab + c$. If you're unsure whether *Mathematica* will execute individual operations in your expression in the order you desire, use parentheses to force it to do so. Your parentheses always override *Mathematica*'s built-in rules, and extraneous parentheses do no harm.

Warning Don't use the hierarchy of grouping symbols standard in mathematics: $\{\ [\ (\ \cdots\ )\ ]\ \}$. In *Mathematica*, square brackets denote arguments and curly brackets denote iterators; only parentheses denote grouping.

4 Special Syntax for Functions and Assignments. (Intermediate)

4.1 How can I restrict my function to selected argument ranges?

To restrict the application of a function to arguments that satisfy a certain condition, use the condition symbol `/;` in the argument list next to the variable to which it applies. After the symbol, write the condition.

For instance, to define a spherical square well potential

$$V(r) = \begin{cases} -V_0 & r \le a \\ 0 & r > a \end{cases}$$

type

```
Clear[potential]
potential[r_ /; r <= a && r >= 0] := -V0
potential[r_ /; r > a] := 0
```

4.2 What's the difference between immediate and delayed assignments in function definitions?

To define a function and assign a value to a variable, you can use use either the notation for an immediate assignment, `=`, or for a delayed assignments, `:=`. Almost always, you can use delayed assignment, `:=`, and just not worry about it.

Here's a brief introduction to the distinction between the two:

immediate assignment: Use `=` if you want *Mathematica* to evaluate the right-hand side of the assignment when you type it in.

delayed assignment: Use `:=` if you do not want *Mathematica* to evaluate the right-hand side until you use it.

Warning Remember that in *Mathematica*, `=` does not mean the same thing it means in mathematics. In *Mathematica*, `a=b+1` means "add 1 to the currently stored value of the variable [b], then assign the result to a variable named `a`."

Thus in *Mathematica* we can write assignment expressions that would be senseless in regular mathematics, such as

```
b = b + 1
```

4.3 What is *Mathematica*'s syntax for denoting arguments to functions?

To denote a generic argument to functions, use a pattern.

In *Mathematica*, a pattern is a class of structurally similar expressions. We signal *Mathematica* that we are using a pattern by a the underscore symbol `_`. Most commonly, we use a **named pattern** by appending this pattern symbol to the name of the argument. For instance, to define a function `f` of the variable x, write `f[x_]`.

Other useful patterns are `x__` (double underscore), which assigns the name `x` to *one or more arguments*, and `x___` (triple underscore), which assigns the name `x` to *zero, one, or more arguments.*

4.4 I want to make a one-shot replacement in an expression? Do I have to define a function?

Nope. The easiest way is to use a replacement rule.

Replacement rules do exactly what they say: they replace one expression by another. *Mathematica* offers two kinds of replacement rules. Their different roles parallel those of the two kinds of assignments. To write a **replacement rule**, use either the notation for an **immediate replacement**, `->` (read as "goes to") or for a **delayed replacement**, `:>`. Fortunately, you can almost always use immediate replacements.

To invoke the replacement rule, use the symbol `/.` (read as "given that"). Thus `Sin[x] + x /. x -> 4` instructs *Mathematica* to replace all occurrences of x by 4 in the expression that precedes the `/.` symbol. So typing this generates $\sin(4) + 4$.

4.5 I told *Mathematica* to implement a replacement for an expression that occurs several times but it only made the replacement once! What do I do?

To force *Mathematica* to invoke the replacement over and over until it can no longer change the expression, use the notation for repeated replacement, `//.`.

5 Bells and Whistles. (Advanced)

5.1 How can I easily apply a function to a whole bunch of argument values?

To map a function onto a list, you can give the list to the function as its argument (enclosed in square brackets).

Alternatively, you can use the mapping symbol `@`.

Tip Most built-in *Mathematica* functions act on lists element-by-element. That is, *Mathematica* typically maps functions across lists. Such functions have the Attribute `Listable`. If you want to give this attribute to your own function `f`, type `SetAttributes[f,Listable]`.

5.2 What's a handy, powerful shorthand way to define functions?

Use anonymous functions.

To write an **anonymous pure function**, we use two symbols. Immediately after the function definition, use the symbol `&` to tell *Mathematica* that whatever preceded this symbol is a pure function. In the body of the function, we represent the argument by the symbol `#`.

5.3 Can I change the head of an expression without re-defining the expression?

Yep. That's what `Apply` is for.

To **change the head of an expression**, either give the expression to `Apply` or use the apply symbol `@@`. Thus `Plus @@ {a,b,c}` returns `a+b+c`, because we changed the head of `{a,b,c}` from `List` to `Plus`.

5.4 How can I determine the head of an expression?

Either use the command `Head` or, if you want the whole nine yards, use `FullForm`.

5.5 Can I define default values to my function arguments?

Sure you can. To assign a default to a pattern variable, use a colon, `:`.

Suppose we want to define a function $f(x,a) = ax^2$ and know that although usually $a = 2$, we want the flexibility to consider other values of a. We can, of course, just define

```
f[x_,a_] = a * x^2
```

and just type `f[x,2]` to invoke $2x^2$. But there is an easier way. If we define the default value $a = 2$ in the function definition,

```
f[x_,a_:2] = a * x^2
```

then *Mathematica* will interpret `f[x]` as $2x^2$.

5.6 I am setting up a sequence of commands. What's the cleanest, clearest, most straightforward way to do this?

Use piping. This is a syntax mode in which the output from one command is "piped" into the next command in the sequence. The double backslash symbol // feeds the output from the command that precedes it into the one that follows it. Thus each command appears in your notebook in the desired execution sequence.

Tip Getting used to piping takes a little practice, and usually requires the use of anonymous functions. But it's worth it! The increase in clarity of your commands will make it *much* easier to find and fix errors and to understand your work months (or years) later when you next need it. Piping is for everybody—not just advanced users!

To illustrate the improved clarity, consider the following equivalent formulations:

```
Simplify[ComplexExpand[TrigToComplex[Conjugate[psi[x]]*psi[x]]]]

Conjugate[psi[x]]*psi[x] // TrigToComplex // ComplexExpand //
   Simplify
```

Which would you prefer to debug?