Beginner's Guide to Mathematica

Basics:

When you open up Mathematica, you will see your input screen, called a notebook. When you begin typing commands, you'll notice that brackets appear on the right side of the notebook. These are called cell brackets.

Mathematica may be different than some programming languages you have worked with in that you often don't run an entire program with one keystroke. Instead you can put several different groups of commands in different cell brackets, and you run each cell bracket individually. It's a good idea to make an effort to put the commands that tie together in the same bracket because it makes it easier to run the program again if you have fewer brackets. However, sometimes it is necessary to run one command and not the other, and in this case you have to put them in separate brackets. You can do this by clicking below the last command with your pointer until you see a horizontal line.

( ) and [ ]

One of the first stumbling points in Mathematica is the use of parentheses and brackets. In Mathematica, it's always brackets [ ] that are used to indicate the argument of a function.

\[ F[x] \quad \cos[2] \]

Parentheses ( ) cannot be used in this way. Parentheses are used in Mathematica only to group arithmetic expressions:

\[ 8 \div (2 + 3) \quad (4-a) * 6 \]

If you don't keep these straight you will have problems in your programs.

Capitalization:

Another common syntax mistake is the naming of built-in Mathematica functions. Mathematica functions are always capitalized. Forgetting to capitalize a \( \sin[x] \) or an \( \exp[y-1] \) will cause errors because Mathematica will not recognize them.

Defining Functions:

Defining a new function in Mathematica is also slightly tricky, syntax-wise. Basically when defining a function you are defining an operation that can be applied to different objects in Mathematica. In defining a function, you have to remember two basic syntax rules: you have a \( \_ \) after each variable in brackets, and you have a : before the equals sign:
F[x_] := x^2  
g[r_,s_] := r^2 (s-2)

Lists:

To be able to easily manipulate a large number of objects at once in your program, using a list is necessary. {} curly braces are used in Mathematica to denote a list:

\[
x = \{3, 5, 7\} \quad y = \{a, b, c\}
\]

After you've defined a list, it's easy to carry out operations on all of its elements at once.

\[
\begin{align*}
\text{In: } x+2 & \quad \text{Out: } \{5, 7, 9\} \\
\text{In: } x-y & \quad \text{Out: } \{3-a, 5-b, 7-c\}
\end{align*}
\]

Equality:

Most of the equality statements you use in Mathematica will be simple definitions like

\[
x = 3 \quad y = 2+x
\]

(Remember when defining variables that Mathematica is case-sensitive; there's a difference between vel and Vel)

However, sometimes rather than just defining the equality, you will want to test two expressions to find out if they are equal. You can do this in Mathematica by using two equal signs:

\[
\begin{align*}
\text{In: } 1 == 1 & \quad \text{Out: } \text{True} \\
\text{In: } 1==2 & \quad \text{Out: } \text{False}
\end{align*}
\]

A more frequent use of the double equals sign is in the Solve expressions, where the computer uses the equality test to solve equations.

\[
\text{In: } \text{Solve}[2x == 6, x] \\
\text{Out: } \{x->3\}
\]

Forgetting the second equals sign in the Solve expression is a common mistake which is easy to miss when you're searching for your error.

% operator

If you want to use the last result you obtained in the next expression, you can use % to represent it.

\[
\begin{align*}
\text{In: } 2+3 & \quad \text{Out: } 5 \\
\text{In: } 2*\% & \quad \text{Out: } 10
\end{align*}
\]
This can be a convenient shorthand. However, it is important to remember if you are debugging and rerunning parts of your program, that the % refers to the last result Mathematica obtained, and not necessarily to the result appearing above the next expression on the screen.

**Substitution rule:**

The substitution rule is a very powerful tool in Mathematica. It removes the need to constantly redefine variables to substitute into expressions. A substitution rule is written like this:

\[ \text{Expression} \; / \; \text{Variable} \rightarrow \text{value} \]

\[ \text{In: } x^2 \; / \; x \rightarrow 2 \]
\[ \text{Out: } 4 \]

By doing this you performed the operation without permanently assigning 2 to x.

Another valuable use for substitution rules is in working with solutions to equations.

\[ \text{In: sol = Solve}[x^2-3y == 0, x] \]
\[ \text{Out: } \{\{x \rightarrow -(3^{0.5}) \times (y)^{0.5}\}, \{x \rightarrow (3^{0.5}) \times (y)^{0.5}\}\} \]
\[ \text{In: } x^2 \; / \; \text{sol} \]
\[ \text{Out: } \{3y, 3y\} \]

It allows you to easily substitute other values into the solution.

**Plotting:**

Plotting is usually fairly easy; the basic syntax is Plot \[fx, \{x, xmin, xmax\}\]. If you want to plot a list of values you can use ListPlot the same way, and to make a parametric plot you just use ParametricPlot \[\{fx, fy\}, \{t, tmin, tmax\}\].

One common problem in plotting is caused when you try to plot a function that Mathematica has not first evaluated; this will give you an error message. If you are having problems plotting a function that you have defined in terms of another function, it is a good idea to try using Evaluate in your plot command: Plot \[\text{Evaluate}[fx], \{x, xmin, xmax\}\]

**Clearing variables:**

Often you will have so many variables in a program that it would be very time-consuming to write separate commands to clear each when you want to run the program again. Instead you can insert the command Remove\["Global\"\] to clear all variables. If you're having problems with your program, it is sometimes a good idea to clear all the variables and then run it again, to make sure that in debugging it you didn't mess up the values of the variables.
Using Help:

If you are having problems with a particular function of Mathematica, a valuable resource to use is the help browser, which has different categories of types of information. A good first step is to search for the function in question under the "Built-in Functions" heading. The entry here will give you basic information about the function and a few examples of usage. It will also refer you to sections in the "Mathematica Book" also in the help browser, which will give further examples and is sometimes more helpful. If the keyword you enter is not a built-in function, you can search for it under the "Master Index" heading, which will refer you to all mentions of the keyword in the help browser.

My top troubleshooting questions:
These are the "simple" things that have tripped me up the most:

- Are all built-in function names capitalized?
- Did I define my function correctly (f[x_] :=)?
- Do I have double equals in my Solve (DSolve, NDSolve) commands?
- Are my variables holding what they were intended to? (try Remove["Global "*"\])

I have spent long periods of time reworking the formulas and commands in a problematic Mathematica program only to find out I just forgot an extra = in my differential equation. Make sure in debugging you check all the simple stuff first!

Frequently used built-in functions:

**N** – Returns the numerical value of an expression. It can be written two ways:

\[
\begin{align*}
\text{In: } N[1/2] & \quad \text{Out: } 0.5 \\
\text{In: } 1/2 //N & \quad \text{Out: } 0.5
\end{align*}
\]

**Plot** – Generates a basic plot of a function. It is written in the form

\[
\text{Plot } [fx, \{x, xmin, xmax\}]
\]

**ParametricPlot** – Generates a parametric plot with x and y generated as a function of t. It is written in the form

\[
\text{ParametricPlot } [[fx, fy ], \{t, tmin, tmax\}]
\]

**ListPlot** – Plots a list of values on the y axis, considering the x coordinates to be 1, 2… It is written in the form

\[
\text{ListPlot } [[y1, y2,..]] \text{ or to specify both coordinates ListPlot } [[\{(x1,y1),(x2,y2)\..\}]]
**DSolve** – Form of the Solve function used on a differential equation. It is written in the basic form:

```
DSolve[equation, y, x], but initial conditions can also be added:
In: DSolve[{x'[t] + 4x[t] == 0, x[0] == 3, x'[0] == 0}, x, t]
Out: {{x -> 3Cos[2#1] &}}
```

**NDSolve** – Works in the same way as the DSolve, but returns a numerical solution. It is written

```
NDSolve[equation, y, {x, xmin, xmax}]
In: NDSolve[{y'[x] == y[x], y[0] == 1}, y, {x, 0, 2}]
Out: {{y -> InterpolatingFunction[{{0., 2.}},<>]}}
```

**Flatten** – This function "flattens" out nested lists by removing a set of {}, it can be useful in manipulating solutions to equations.

```
In: Flatten[{{x}, {y, z}}]
Out: {x, y, z}
```

**Simplify** – This function can put expressions into a form where they "look pretty"; it tries a bunch of different algebraic transformations like factoring and expanding to return an expression in a simpler form.

```
In: Simplify[x^2 + 2x + 1]
Out: (x + 1)^2
```

**Evaluate** – This function evaluates an expression that Mathematica would normally hold unevaluated. Evaluate is a good thing to try if you're having problems plotting a function:

```
In: Plot[Evaluate[v[t]], {t, 0, 10}]
```