Introduction to *Mathematica*, with emphasis on tricky points

David P. Roberts

Mathematica is an extremely sophisticated calculator. To begin with, you can use it just as you would a regular calculator. So you can add, subtract, multiply, and divide by `+`, `-`, `*`, and `/`, respectively. For example,

```
In[1] := 4*12

Out[1] = 48
```

Here `In[n]` indicates the nth input line, i.e. what you type. When you are done with an input line you hit either the `enter` key or `shift-return`. You don't hit `=` as you would on many calculators. `Out[n]` is *Mathematica*'s response, i.e. the answer you want. Like in ordinary math language, in *Mathematica* a space indicates multiplication. So you could just as well have typed `4 12` as `In[1]` to get the same answer.

There are many other special symbols in *Mathematica*, which tend to resemble the symbols used in other programming languages. For example `^` indicates exponentiation, so if you type `2^5` the computer will return `32`. However, most commands in *Mathematica* are English words. For example, you type in the number π by typing `Pi`. Similarly, there is no special button for the sine function. Instead you type `Sin`. Thus,

```
In[2] := Sin[Pi/2]

Out[2] = 1
```

There is a fundamental difference between how typical calculators work and how *Mathematica* works. *Mathematica* always works at the *symbolic* level unless you force it to work at the *numeric* level. Typical calculators work only at the numeric level. Here are two examples. First,

```
In[3] := 4/12

Out[3] = 1/3
```

Note that *Mathematica* has simply reduced the fraction and that 4/12 is really exactly equal to 1/3. A typical calculator would have output 0.3333333333, which is only approximately equal to 1/3. Here's the second example.

```
In[4] := Pi

Out[4] = π
```

Here *Mathematica* basically did nothing except switch from the input font to the prettier output font. Actually *Mathematica* does a fair amount of font trickery, which I will not mention further in this quick introduction.

There are two easy ways to force *Mathematica* to work at the approximate, numeric level. One is the command `N`. For example,

```
In[5] := N[Pi]

Out[5] = 3.14159
```

This is what a typical calculator would have output in the first place. One advantage of staying at the exact symbolic level as much as possible is that you can get more precision if you want to, by using an optional second argument to `N`.

```
In[6] := N[Pi,50]

Out[6] = 3.1415926535897932384626433832795028841971693993751058209749445 9
```

The second way to force *Mathematica* to work at the numeric level is to type a decimal point. For example, the computation

```
In[7] := 2^100
```

`Out[7]` = 126765060022822940149670320537 6

takes place at the symbolic level. However, the computation

```
In[8] := 2.^100
```

`Out[8]` = 1.26765 $\times 10^{30}$

takes place at the numeric level.

Typical calculators are extremely number-based. *Mathematica* on the other hand, can help you with enormously more math. Here's how some algebra would look:

```
In[9] := Expand[(a+b)^2]
```

`Out[9]` = $a^2 + 2ab + b^2$

```
In[10] := Factor[%]
```

`Out[10]` = $(a + b)^2$

Here `%` means the previously output quantity; thus since you are typing `In[10]`, `%` means `Out[9]`. `%%` or `%8` or `Out[8]` would refer to `Out[8]`. Similarly, `%%%` or `%7` or `Out[7]` would refer to `Out[7]`, and so on.

Another way to refer to things, often better than using `%`'s, is to name things when you first introduce them. For example,

```
In[11] := thirdpower = Expand[(a+b)^3]
```

`Out[11]` = $a^3 + 3a^2b + 3ab^2 + b^3$

```
In[12] := Factor[thirdpower]
```

`Out[12]` = $(a + b)^3$

From now on in your session, `thirdpower` is a *Mathematica* object, just like all the built in *Mathematica* objects.

As I've indicated, there are often several ways to do a given thing in *Mathematica*. Sometimes the *function*`[`*argument*`]` notation is inconvenient and it is better to use the equivalent *argument* `//` *function* ("slash-slash") grammar. Here are interchanges 2 and 5 done this different way

```
In[13] := Pi/2 //Sin
```

`Out[13]` = 1

```
In[14] := Pi //N
```

`Out[14]` = 3.14159

One way to substitute in *Mathematica* is to use the `/.` "slash-dot" construction, which is somewhat similar to the slash-slash construction just discussed. For example, to plug 1 in for a in `Out[9]` you could type

```
In[15] := Out[9] /. a->1
```

`Out[15]` = $1 + 2b + b^2$

Equally well, you could have done this with optional braces

```
In[16] := Out[9] /. {a->1}
```

`Out[16]` = $1 + 2b + b^2$

When you make two substitutions at once you need those braces:

```
In[17] := Out[9] /. {a->2,b->x^2}
```

`Out[17]` = $4 + 4x^2 + x^4$

In general, commas, like the one between the 2 and the b in `In[17]` never "float" in *Mathematica*. They are stuck always in between either braces or brackets, as in `{...,...}` or `[...,...]`.

A generally better way to make substitutions is by *defining functions* as follows:

```
In[18] := f[x_] := x^2 - x - 1
```

Here the underscore on the x and the := may look weird, but you'll get used to them! Having defined f, you can substitute to your heart's content

```
In[19] := f[0]

Out[19] = -1

In[20] := f[2]

Out[20] = 1

In[21] := f[w^2]

Out[21] = w^4 - w^2 -1
```

The function `f`, like the expression `thirdpower`, can be used at any future point in your session.

Mathematica is very finicky, far more finicky than any instructor you'll ever meet. One hard-and-fast rule is that words defined in *Mathematica* always begin with a capital letter. So typing `sin` instead of `Sin` or `pi` instead of `Pi` just won't work; neither will `expand` or `factor`. Usually, *Mathematica* commands are complete English words. A common exception is the command for square root:

```
In[22]:= Sqrt[100]

Out[22] = 10

In[23]:= Sqrt[99]

Out[23] = 3 √11

In[24]:= Sqrt[99.]

Out[24] = 9.94987
```

The last two interchanges illustrate again the distinction between working at the symbolic and numeric levels.

In fact, the distinction between symbolic operations and numeric operations will reappear many times in our course. For example, consider getting *Mathematica* to solve a quadratic equation. Symbolically, the command to use is `Solve`:

```
In[25] := Solve[f[x] == 0,x]
```

`Out[25] =` $\{\{x \to (1-\sqrt{5})/2\}, \{x \to (1+\sqrt{5})/2\}\}$

Numerically, the command to use is `NSolve`.

```
In[26] := NSolve[f[x] == 0,x]
```

`Out[26] =` $\{\{x \to -0.618034\}, \{x \to 1.61803\}\}$

Note that here we are using our already defined function f to save typing. We could just as well have typed `Solve[x^2-x-1 == 0,x]` and `NSolve[x^2-x-1 == 0,x]` instead.

You should realize that symbolic commands and numeric commands make *Mathematica* do very different things internally. For example, to get `Out[25]`, *Mathematica* has used the quadratic formula, just as you would do by hand. `Out[25]` would do very different things if in `In[25]` you replaced the exponent 2 by 3, 4, 5, etc. On the other hand, to get `Out[26]`, *Mathematica* has used a uniform procedure that would work the same for exponents 3, 4, 5, etc. Try it!

We will be using a little bit of *Mathematica*'s vast graphic capabilities. The basic command is `Plot`. The first argument of `Plot` is the function being plotted. The second argument indicates, in *Mathematica*-speak, the x-range over which the function is being plotted. The two arguments are separated by a comma, just as the two arguments were separated by a comma in `In[6]`. For example, `Plot[f[x],{x,-1,2}]` yields a parabola where the two roots considered in interchanges 25 and 26 are visible as x-intercepts.

An *crucial* thing to realize is that *Mathematica* uses three different types of delimiters, namely brackets `[ ]`, parentheses `( )`, and braces `{ }`. *These delimiters are never interchangeable!* Brackets get used all the time to indicate arguments of functions, both traditional functions like `Sin` and *Mathematica* functions like `N`, `Plot`, etc. Parentheses are also common; they indicate ordinary grouping like in `In[9]`. Braces are used to form lists in *Mathematica*. We'll use them mostly in connection with other commands, such as `Plot`, as discussed in the previous paragraph.

Another *crucial* thing to realize is that *Mathematica* uses three types of equals signs. We've already seen all of them: `:=`, `=`, and `==`. They are referred to in *Mathematica*=speak as `Set`, `SetDelayed`, and `Equals`. Like the three types of delimiters, *the three types of equals signs are not interchangeable!*

`Set` sets variables equal to numbers, among other things. For example,

```
In[27] := x=3

Out[27] := 3

In[28] := x^2+1

Out[28] = 10
```

If you no longer want x to be 3 you need to clear it.

```
In[29] := Clear[x]
```

Then the same input as `In[28]` works differently:

```
In[30] := x^2 + 1

Out[30] = x^2 + 1
```

`SetDelayed`, i.e. `:=` is subtle variant of `=`. We will mostly use it when defining functions, like we did for f above.

On the other hand `Equals`, i.e. `==`, is completely different from `=` and `:=`. It corresponds more to the mathematical notion of equals:

```
In[31] := 3 == 4

Out[31]=False
```

There are other commands which function grammatically the same as `==`, for example `Less`:

```
In[32] := 3 < 4

Out[32] = True
```

Mostly, we will be using `==` in connection with specific commands which require it, such as `Solve`, and `NSolve`.

There is a lot of *Mathematica* help on-line. The simplest way to access this help is by `?`. For example, if you type either `?N*` you get a list of all *Mathematica* commands starting with `N`:

```
⋮
NeedCurrentFrontEndPackagePacket
⋮
NotebookSetupLayoutInformationPacket
⋮
Numerator
⋮
```

Don't worry, we'll only be using a few of the many, many *Mathematica* predefined objects. I don't even know what the first two things I've listed do. However `Numerator` looks more promising. Type `?Numerator` and see if it tells you enough so that you can use the command `Numerator`. If it doesn't, click on `More`. The "Further Examples" at the bottom are usually particularly helpful.