

An Introduction to *Mathematica*

Douglas Baldwin

October 19, 2000

What is *Mathematica*?

Mathematica is the leading symbolic algebraic system on the market. It was first released in 1988 and combined numeric, algebraic, and graphical computation into one system.

From the very beginning, *Mathematica* was more than just a symbolic algebraic system. . . it was an advanced programming language. *Mathematica* supports three main programming paradigms: *procedural*, which is C- or Fortran-like; *functional*, similar to that of Lisp; and *rule-based*, which is typified by Prolog.

Programming in *Mathematica* means different things to different people. For some, it means using *Mathematica*'s 1000+ built-in functions to write "one-liners" to perform nontrivial calculations. For others, it means writing new functions to extend the capabilities of *Mathematica* even further. While still others write their functions in separate files, called a *package*, to create functions which behave like *Mathematica*'s own.

Some Basics Points about *Mathematica* Syntax

- All *Mathematica*'s built-in commands begin with a capital letter.
Examples: Solve, Factor, Plot3D, etc.
- If the function or command is more than two words, the first letter of each word is capitalized. Examples: DensityPlot, IdentityMatrix, etc.
- The arguments of all commands and functions are encased in square brackets ([. . .]).
- All arguments within the brackets are separated by a comma.
- And items are grouped using curly braces ({. . .}) or semi-colons (;). Example:

```
Plot3D[Sin[x^2+Cos[y^2]], {x, -2Pi, 2Pi}, {y, -2Pi, 2Pi}];
```

or,

```
For[i = 0, i < 10, i++,  
  r = i^2; s = i!; t = Sqrt[i];  
  Print["i=",i," i^2=",r," i!=",s," Sqrt[i]",t]];
```

Not all equals are equal in *Mathematica*

`=` represents an assignment. So, `a=7` would mean `a` would be replaced by `7` whenever it appears.

`:=` represents a delayed assignment. So, `b:=Random[]` would mean that `b` would be replaced by an unevaluated `Random[]` which would then be evaluated afresh. So, `b` might yield `0.390936` the first time it's evaluated and `0.747323` the next time it's evaluated.

`==` represents a logical test. So, `1==2` would yield `False` while `1 == 1.` would yield `True`.

`===` represents the function `SameQ[rhs,lhs]`. Which requires that the left hand side be identical to the right hand side. Such that `1 === 1.` would yield `False` because `1` is exact while `1.` is approximate.

Creating A List in *Mathematica*

- In *Mathematica*, lists are designated with curly braces: $\{\dots, \{\dots\}, \{\{\dots\}\}\}$.
Example: `list1 = {0, {1}, π , $\sqrt{2}$, a, 100!}`
- You can generate lists in numerous ways:
 1. `Table[expr, {i, imin, imax, δ_i }, ...]`
Example: `Table[i2, {i, 10}]` \equiv `Table[i2, {i, 1, 10}]` \equiv `Table[i2, {i, 1, 10, 1}]` \Rightarrow `{1, 4, 9, 16, 25, 36, 49, 64, 81, 100}`
 2. `Range[imax]` generates the list: $\{1, 2, \dots, i_{max}\}$. Or, `Range[imin, imax]` generates the list: $\{i_{min}, i_{min} + 1, \dots, i_{max}\}$.
 3. `Array[f, n]` or `Array[f, {n1, n2, ...}]`
Example: `Array[f, {2, 3}]` \Rightarrow `{{f[1, 1], f[1, 2], f[1, 3]}, {f[2, 1], f[2, 2], f[2, 3]}}`

Extracting Elements in *Mathematica*

```
list1={0,{1},π,√2,a,10!}
```

Part

- To take an element of a list, we use the function `Part[... , i]` or `{... }[[i]]`.
Example: `Part[list1,3]≡list1[[3]]⇒π`.
- If you have a nested list (or matrix) like `list2= {{a11, a12}, {a21, a22}}`, you use double indexing separated by commas.
Example: `Part[list2,2,1]≡list2[[2,1]]⇒a21`.

Drop

- `Drop[... , n]` drops the first n elements of the list, while `Drop[... , -n]` drops the last n -elements of the list.
Example: `Drop[list1,3]⇒{√2,a,3628800}`

Select

- `Select[{...}, criteria]`, picks out all the elements in the list which meet a certain criteria.
 Example: `Select[{1,4,2,7,6}, EvenQ] ⇒ {4,2,6}`, or
`Select[{23,6,2,62,673}, #>10&] ⇒ {23,62,673}`

Cases

- `Cases[{...}, pattern, depth]` or `Cases[expr, pattern, depth]`, picks up all elements in the list or expression which match a certain pattern. Examples:
`Cases[list1, _Integer] ⇒ {0, 3628800}`,
`Cases[list1, _Integer, Infinity] ⇒ {0, 1, 2, 3628800}`
`Cases[$\frac{1}{4} + a + e^{-x^2} + \text{Exp}[y^2] + u^2 + w^{5^x} + \frac{1}{z^2}$, $x_{-}y_{-}$] ⇒ $\{e^{-x^2}, e^{y^2}, u^2, w^{5^x}, z^{-2}\}$`

List Structure Manipulation in *Mathematica*

Flatten

- `Flatten[...]` or `Flatten[..., n]`, flattens the sub-lists into a single list or flattens the sub-lists to some level n . Examples:

`Flatten[{{a, b}, {c, d}, {{e}}}] \Rightarrow {a, b, c, d, e}`

`Flatten[{{a, b}, {c, d}, {{e}}], 1] \Rightarrow {a, b, c, d, {e}}`

Partition

- `Partition[..., n]` divides the list into non-overlapping sublists of length n .

Example:

`Partition[{a,b,c,d,e,f}, 2] \Rightarrow {{a, b}, {c, d}, {e, f}}`

- `Partition[..., n, δ]` generates sub-lists of length n which are offset by length δ .

Example:

`Partition[{a,b,c,d,e,f}, 3, 1] \Rightarrow {{a, b, c}, {b, c, d}, {c, d, e}, {d, e, f}}`

How many ways can we write a function in *Mathematica*?

function[var_,...]:=expr; Examples:

```
myFunc[n_] := n^2;
```

```
myFunc[10] ⇒ 100
```

```
myFunc2[x_, y_, z_/; z != 0] := Sqrt[x^2 + y^2]/z^2;
```

```
myFunc2[3, 4, 5] ⇒  $\frac{1}{5}$ .
```

function = Function{var_,... },expr; Example:

```
myFunc3 = Function[{x, y, z}, Sqrt[x^2 + y^2]/z^2];
```

```
myFunc3[3, 4, 5] ⇒  $\frac{1}{5}$ 
```

function = (expr)& Examples:

```
myFunc5 = (Sqrt[#^2 + #2^2]/#3^2)&;
```

```
myFunc5[3, 4, 5] ⇒  $\frac{1}{5}$ 
```

Getting Beyond The First Line

- Grouping this with parentheses:

```
myFunc6 := (Print["This is a multi-line "]; Print["Mathematica program."]);  
myFunc6 ⇒ This is a multi-line  
          Mathematica program.
```

- Using Module and Block:

```
myFunc7[n_Integer?Positive]:= (* Sets up function for positive integer n *)  
    Module[{i,sum=0}, (* Shields local variables i and sum.*)  
        For[i=0, i < n, i++,  
            sum += i]  
        Return[sum]  
    ];  
myFunc7[10] ⇒ 45
```

Nesting, Folding, Mapping, and Applying What You Have Learned

`Nest[f, expr, n]` takes a function and first applies it to an argument and then applies it to itself n times. As an example:

`Nest[f, x, 3]` yields $f[f[f[x]]]$, or

`Nest[Sin, 0.5, 4]` means `Sin[Sin[Sin[Sin[0.5]]]]` and yields 0.366099

`Fold[f, x, {...}]` takes a function and uses x as the first argument of the function and the first entry of the list as the second argument of the function. It then takes that answer and applies f to itself and the next entry in the list. Example:

`Fold[f, x, {a, b, c}]` yields $f[f[f[x, a], b], c]$

`Apply[f, expr] \equiv f @@ expr` takes whatever `expr` is and replaces its outer-most function with f . Such as:

`Apply[List, $ax^2 + bx + c$]` \equiv `List @@ $ax^2 + bx + c$` \Rightarrow `{ ax^2 , bx , c }` because the Head of $ax^2 + bx + c$ is Plus.

`Map[f, expr] \equiv f /@ expr` unlike `Apply` merely applies function f to each element on the first level in `expr`. Examples:

`Map[f, {a, b, c}]` \Rightarrow `{f[a], f[b], f[c]}`.

Okay, now that I know more about *Mathematica* than I ever wanted to know . . . how do I solve problems with it?

Problem: How many unique ordered trees can be drawn given n -nodes?

As with most problems of any complexity, the computers terminal is the worst place to start. The first step is to draw a couple of trees on the blackboard, so as to get a better understanding of the problem.

Furthermore, since our computers can't directly understand our blackboard ordered trees, we must ask ourselves what would be an easy way to represent the ordered trees in the computer.

A System For Representing Any Ordered Tree

Instead of trying to represent the tree as a set of nodes, we can represent a tree as the set of lines connecting those nodes.

Let us now look at the first couple of n -nodes:

n -nodes	Trees
2	$\{\{1, 2\}\}$
3	$\{\{1, 2\}, \{1, 3\}\}$ $\{\{1, 2\}, \{2, 3\}\}$
4	$\{\{1, 2\}, \{1, 3\}, \{1, 4\}\}$ $\{\{1, 2\}, \{1, 3\}, \{2, 4\}\}$ $\{\{1, 2\}, \{1, 3\}, \{3, 4\}\}$ $\{\{1, 2\}, \{2, 3\}, \{2, 4\}\}$ $\{\{1, 2\}, \{2, 3\}, \{3, 4\}\}$

Can you see the pattern?

Now that we have a pattern, how do we program it?

The first thing to realize that it would be difficult to get to final result directly. Instead, we can break it up into two steps. The first step is to generate all the possible lines, and then combine them into only valid trees.

$\{?, 2\}$	$\{?, 3\}$	$\{?, 4\}$	$\{?, 5\}$	$\{?, 6\}$
1	1	1	1	1
	2	2	2	2
		3	3	3
			4	4
				5

To program this in *Mathematica*, we must first look at what tools we have to work with. Our primary tools for creating lists is `Table`, `Range`, and `Array`. From the table we see that the second entry in the list ranges from 2 to n . This would hint to the use of the function `Range`, such as: `Range[2, n]` or `Drop[Range[n], 1]`.

The first entry ranges from 1 to the value of the second entry minus one. We can easily do this with *Mathematica's* `Table` function. Such that we can nest the `Table` functions as follows:

```
lines = Table[Table[{i, j}, {i, j - 1}], {j, 2, nodes}];
```

Alternatively, we can create a pure function, `(. . .)&`, using the `Table` function:

```
lines = Table[{i, #}, {i, # - 1}]& /@ Range[2,nodes];
```

Example: For `nodes=4`, *Mathematica* gives us the list:

```
{{{1, 2}}, {{1, 3}, {2, 3}}, {{1, 4}, {2, 4}, {3, 4}}}
```

Writing the instructions for our 10,000+ piece jig-saw puzzle:

So, what if we have a list of possible lines. . . how do we know which ones go in which tree?
 The answer is, we (as mathematicians) have to figure out a way to telling *Mathematica* which lines are valid and which are not. If we look back to the tables, we notice that $i_1 \leq i_2 \leq i_3 \leq \dots \leq i_{n-1}$.

n -nodes	Trees	$\{i_1, 2\}$	$\{i_2, 3\}$	$\{i_3, 4\}$	$\{i_4, 5\}$	$\{i_5, 6\}$
2	$\{\{1, 2\}\}$	1	1	1	1	1
3	$\{\{1, 2\}, \{1, 3\}\}$ $\{\{1, 2\}, \{2, 3\}\}$	1	2	2	2	2
4	$\{\{1, 2\}, \{1, 3\}, \{1, 4\}\}$ $\{\{1, 2\}, \{1, 3\}, \{2, 4\}\}$ $\{\{1, 2\}, \{1, 3\}, \{3, 4\}\}$ $\{\{1, 2\}, \{2, 3\}, \{2, 4\}\}$ $\{\{1, 2\}, \{2, 3\}, \{3, 4\}\}$	1	2	3	4	5

Coding it up in *Mathematica*

```

orderedTrees[n_Integer /; nodes > 1] :=
  Module[{lines, parts},
    lines =
      Table[{i, #}, {i, # - 1}] & /@ Drop[Range[n], 1];
    parts =
      Select[
        Partition[
          Flatten[
            Fold[Table,
              Array[ $\alpha$ , n - 1],
              Table[{ $\alpha$ [i], i},
                {i, nodes - 1}]
            ]
          ],
          nodes - 1
        ], (LessEqual @@ Table#[[i]], {i, n - 1})&];
    Return[Table[lines[[i, #[[i]]]], {i, nodes - 1}] & /@ parts]
  ];

```

Example:

orderedTrees [5] ⇒

{{{1, 2}, {1, 3}, {1, 4}, {1, 5}}, {{1, 2}, {1, 3}, {1, 4}, {2, 5}},
{{1, 2}, {1, 3}, {2, 4}, {2, 5}}, {{1, 2}, {2, 3}, {2, 4}, {2, 5}},
{{1, 2}, {1, 3}, {1, 4}, {3, 5}}, {{1, 2}, {1, 3}, {2, 4}, {3, 5}},
{{1, 2}, {2, 3}, {2, 4}, {3, 5}}, {{1, 2}, {1, 3}, {3, 4}, {3, 5}},
{{1, 2}, {2, 3}, {3, 4}, {3, 5}}, {{1, 2}, {1, 3}, {1, 4}, {4, 5}},
{{1, 2}, {1, 3}, {2, 4}, {4, 5}}, {{1, 2}, {2, 3}, {2, 4}, {4, 5}},
{{1, 2}, {1, 3}, {3, 4}, {4, 5}}, {{1, 2}, {2, 3}, {3, 4}, {4, 5}}}