Mathematica

The Philosophy

> The programmer's time is more valuable than the processor's time. Thus, the architecture is interpreted (interactive), real-time, and goal-oriented.

"Programs you write in Mathematica may nevertheless end up being faster than those you write in compiled languages" p.506

"The internal code of Mathematica uses polynomial time algorithms whenever they are known." p.63

> Mathematica accepts code in all of the modern programming paradigms.

"All the approaches are in a sense ultimately equivalent, but one of them may be vastly more efficient for a particular problem, or may simply fit better with your way of thinking about the problem." p.487

"As a matter of principle, it is not difficult to prove that *any* Mathematica program can in fact be implemented using transformation rules alone." p.503

The Mathematica Program

a general purpose computational engine for
- numerical calculations (arithmetic)
- symbolic transformations (algebra)
- graphic display (geometry)

a modern programming language with multiple styles
- procedural
- functional
- logical
- object-oriented
- rule-based

an integrated tool
- C, TeX, UNIX, Postscript

Everything is an Expression

"At a fundamental level, there are no data types in Mathematica. Every object you use is an expression, and every function can take any expression as an argument." p.496

`x+y`	`Plus[x,y]`
`120`	`Integer[120]`
`2ab`	`Times[2,a,b]`
`{a,b,c}`	`List[a,b,c]`
`i = 3`	`Set[i,3]`
`x^2+2x+1`	`Plus[Power[x,2],Times[2,x],1]`

An undefined symbol is *itself*, providing functional transparency and WYSIWYG debugging

The Meaning of Expressions

`F[x,y]` `F` is the *head*. `x,y` are the *contents*.

Apply function `F` to arguments `x` and `y`.
Do action `F` to objects `x` and `y`.
The label `F` points to elements `x` and `y`.
The object-type `F` has parts `x` and `y`.
The arguments `x,y` are of data type `F`

The head can both act on its contents (as a function) and maintain the structure of its contents (as an object), depending on context.

Lists

The List container is used for all collections and all database entries:

data record	`{John, 555-1234, j@mma.com}`
matrix	`{{11,12},{21,22}}`
set	`Union[{a,b},{a,c}] ==> {a,b,c}`
graphics spec	`Line[{{0,1},{1,1},{1,0}}]`
stream	`{1,2,3,...}`
structure template	`{_,{_,_},{{_,_},...}}`

The Fundamental Principle of Computation

Take any expression and apply transformation rules until the result no longer changes.

1. Reduce head	
2. Reduce each element	base case arithmetic
3. Standardize	
4. Apply user defined rules	inductive case algebra
5. Apply built-in rules.	
6. Reduce the result.	recursion

Patterns

A *pattern* is a class of expressions with the same structure.

`_`	"underbar" means *any* expression
`x_`	any expression locally named `x`
`x__`	any sequence of expressions (double underbar)
`x___`	any sequence, including none (triple underbar)
`x_h`	any expression with head = `h`.

Examples:

`f[n_]`	the function `f` with a parameter named `n`
`2^n_`	2 raised to any power
`a_ + b_`	the sum of two arbitrary expressions
`{a__}`	a list with at least one element

Object-oriented Organization

```
square/:    perimeter[ square[n_] ] := 4*n
square/:    area[ square[n_] ] := n^2
circle/:    area[ circle[r_] ] := Pi*r^2
```

The outer "function" transforms the inner "argument".
The inner "object" contains a private outer "message handler".
The outer "matrix" is indexed by the inner "accessors".