Sept. 12 2005: Lecture 3:

Introduction to Mathematica II

Functions and Rules

Besides MATHEMATICA® 's large set of built-in mathematical and graphics functions, the most powerful aspects of MATHEMATICA® are its ability to recognize and replace patterns and to build functions based on patterns. Learning to *program* in MATHEMATICA® is very useful and to learn to program, the basic programmatic elements must be acquired.

The following are common to almost any programming language:

Machine State

Loops

Variable Scope

Switches

Functions

The following are common to Symbolic and Pattern languages, like MATHEMATICA® .

Pattern

Recursion

MATHEMATICA® Example: Lecture-03

Procedural Programming

Initialization

Loops

Very complex expressions and concepts can be built-up by loops, but within MATHEMATICA® the complexity can be buried so that only the interesting parts are apparent and shown to the user.

Sometimes, as complicated expressions are being built up, intermediate variables are used. Consider the value of `i` after running the program
`FindMinimum[For[a = dx; i = 1, i ≤ 4, i++, a = 2a; a = a∧a]; Log[a], {dx, 0.15, 0.25}]`, the value of `i` (in this case 5) is has no useful meaning anymore. If you had defined a symbol such as `x = 2i` previously, then now `x` would have the value of 10, which is probably not what was intended. It is much safer to localize variables—in other words, to limit the scope of their visibility to only those parts of the program that need the variable. Sometimes this is called a "Context" for the variable in a programming language; MATHEMATICA® has contexts as well, but should probably be left as an advanced topic.

MATHEMATICA® Example: Lecture-03

Procedural Programming, cont'd

Local Variables and Modules

Conditionals: If, Which, Switch

Patterns are extremely important in mathematics and in MATHEMATICA® . In MATHEMATICA® , the use of the underscore, _, means "this is a placeholder for something that will be used later." It is a bit like teaching like teaching a dog to fetch—you cock an arm as if to throw _something_, and then when something gets thrown your dog runs after the "something." The first _something_ is a place holder for an object, say anything from a stick to a ball to the morning paper. The second something is the actual object that is actually tossed, that finally becomes the "something" your dog uses as the actual object in the performance of her ritual response to the action of throwing.

Usually, one needs to name to call the pattern to make it easier to refer to later. The pattern gets named by adding a head to the underscore, such as `SomeVariableName_`, and then you can refer to what ever pattern matched it with the name `SomeVariableName`.

This is a bit abstract and probably difficult to understand without the aid of a few examples:

MATHEMATICA® Example: Lecture-03

Patterns and Replacement

The real power of patterns and replacement is obtained when defining your own functions.

MATHEMATICA® Example: Lecture-03

Functions with Patterns

It is probably a good idea to define all function with delayed assignment (`:=`) instead of immediate assignment (`=`). With delayed assignment, MATHEMATICA® does not evaluate the right-hand-side *until* you ask it to perform the function. With immediate assignment, the right-hand-side is evaluated when the function is defined making it much less flexible because your name for the pattern may get "evaluated away."

MATHEMATICA® Example: Lecture-03

Defining Functions in Mathematica

Delayed Replacement

Functions

Defining functions are essentially a way to eliminate repetitive typing and to "compactify" a concept. This "compactification" is essentially what we do when we define some function or operation (e.g., $\cos(\theta)$ or $\int f(x)dx$) in mathematics—the function or operation is a placeholder for something perhaps complicated to describe completely, but sufficiently understood that we can use a little picture to identify it.

Of course, it is desirable for the function to do the something reasonable even if asked to do something that might be unreasonable. No one would buy a calculator that would try to return a very big number when division by zero occurs—or would give a real result when the arc-cosine of 1.1 is demanded. Functions should probably be defined so that they can be reused, either by you or someone else. The conditions for which the function can work should probably be encoded into the function. In MATHEMATICA® this can be done with restricted patterns:

MATHEMATICA® Example: Lecture-03

Functions with Restricted Patterns