Lecture 6

Programming Syntax for Maple

1. Introduction

While there is no one required programming language for the course, all of the explicit programming examples presented in class will be carried out in Maple. For those of you that more comfortable with another programming language (C, Pascal, FORTRAN, Basic, whatever) you are free to carry out your homework assignments in that language.

This lecture is devoted to outlining the basic programming constructs for Maple.

2. Basic Basics

Maple is loosely typed procedural programming. This makes it particularly straight-forward programming novices.

The simplest Maple statement might be a command like

\[ x := 2; \]

This sets equal to the integer 2 (and also declares a new variable texttxtx if the symbol x has not been previously introduced. An alternative to this statement might be

\[ x := 2: \]

Note that the semi-colon has been replaced by a colon. This is equivalent to the first example, except that the result will not be confirmed on screen. This construct is handy when one has a long sequence of repetitive statements; it avoids a lot of “screen clutter”. The statement

\[ x := [2,2]; \]

declares and defines x as a two element list. Advanced programmers might note that we did not have to declare the variable x to be a particular data type. Maple figured out a data type for x for us. In short, the assignment operation

\[ A := B; \]

assigns to (and/or creates) a variable named A that's equal to whatever the value of the right hand side is.

2.1. Simple Logic Tests.
2.1.1. Logic Operators.

- \((A \text{ and } B)\) returns 1 (true) if both condition \(A\) and condition \(B\) are true, otherwise it returns 0 (false).
- \((A \text{ or } B)\) returns 1 (true) if either
  - condition \(A\) is true,
  - condition \(B\) is true, or
  - both conditions \(A\) and \(B\) are true.
  Otherwise, it returns 0 (false).
- \((\text{not } A)\) the negation of condition \(A\). Returns 1 (true) if \(A\) is false, and 0 (false) if \(A\) is true.

2.1.2. Comparison Operators.

- \((A = B)\) returns 1 (true) if variable \(A\) equals \(B\), and returns 0 (false) if \(A\) is not equal to \(B\).
- \((A <> 0)\) returns 0 (true) if variable \(A\) equals \(B\), and returns 1 (false) if \(A\) is not equal to \(B\). Another way of writing the "not-equal" test is \((\text{not } A = B)\).
- \((A > B)\) returns 1 (true) if variable \(A\) is greater than \(B\), and returns 0 (false) if \(A\) is less than or equal to \(B\).
- \((A < 0)\) returns 1 (true) if variable \(A\) is greater than \(B\), and returns 1 (false) if \(A\) is greater than or equal to \(B\).
- \((A >= B)\) returns 1 (true) if variable \(A\) is greater than or equal to \(B\), and returns 0 (false) if \(A\) is less than \(B\). (Equivalent to \((\text{not } A < B)\)).
- \((A <= 0)\) returns 1 (true) if variable \(A\) is less than or equal to \(B\), and returns 1 (false) if \(A\) greater than \(B\). (Equivalent to \((\text{not } A > B)\)).

2.2. Flow Control.

2.2.1. If Statements. The simplest type of if statement would be a routine of the form

\[
\text{if condition}_A \text{ then statement}_B \text{ fi;}
\]

Here if, then, and fi are all essential keywords, that are used by Maple to delimit and identify the various components of this if statement. For example,

\[
\text{if } (2 = 1 + 1) \text{ then } x := 2 \text{ fi;}
\]

ends up assigning the value 2 to the variable \(x\).

More elaborate if statements are constructed as follows

\[
\begin{align*}
\text{if condition}_A \text{ then do} \\
\text{statement}_1; \\
\text{statement}_2; \\
\text{statement}_3; \\
\text{od;}
\text{elif condition}_B \text{ then do} \\
\text{statement}_4; \\
\text{statement}_5; \\
\text{od;}
\text{elif condition}_C \text{ then do} \\
\text{statement}_6; \\
\text{statement}_7; \\
\text{od;}
\end{align*}
\]
else do
    statement8;
    statement9;
    od;
fi;

2.3. Iterative Loops. An example of the most comprehensive *iterative* loop routine for Maple is
might be a sequence of statements

```maple
for i from 1 to 1000 by 2 while (x < 100000) do
    statement1;
    statement2;
    statement3;
    od;
```

It is not necessary to include all these options in an iterative loop. Here are two examples.

**Example 6.1.**

```maple
for i to 1000 do
    statement1;
    statement2;
    statement3;
    od;
```

In this example, Maple assumes that the index $i$ is ranging from 1 to 1000 and increasing by 1 at each iteration of the do loop.

**Example 6.2.**

```maple
i := 1;
while (i < 1000) do
    i := i + 1;
    od;
```

In this example, the loop continues until the condition $x < 1000$ fails. Note that this will cause all sorts of trauma if this condition is never met. In using this construct, it is important to take care that

1. an initial value of $i$ is defined
2. eventually the while test must fail.