

MATHEMATICS 201-BNJ-05

Topics in Mathematics

Martin Huard

Winter 2009

Introduction to Maple Programming

Procedures

A procedure in Maple is a prearranged group of statements processed together. The basic format is:

```
Name := proc(...)  
    Statements  
end proc:
```

Use “SHIFT –ENTER” instead of ENTER until the end of the procedure, so that it will be in the same block. The procedure will return the execution of the last statement.

Example

Suppose we want a procedure that doubles any value.

```
Double:=proc(x)  
    2*x  
end proc:
```

Try the following:

```
Double(5);  
Double(Pi);  
Double(1.3);
```

If the procedure needs to make use of a variable, then we declare it as a local variable, which Maple will forget as soon as it completes the procedure.

```
Name := proc(...)  
    local variables  
    Statements  
end proc:
```

Example

Using the same example as before, but defining 2 as a local variable.

```
Double:=proc(x)  
    local i;  
    i:=2;  
    i*x  
end proc:
```

Try the following:

```
Double(7);  
i;
```

Note that in the second command, **i** is just returned, meaning that it is not equal to the value of 2 anymore.

Basic programming tools

Here are a few basic tools of programming.

Conditional statements - if

The basic form of the **if** statement is:

```
if expression then
    statement
end if
```

We can add alternative

```
if expression then
    statement1
else
    Statement2
end if
```

If there are many scenarios, then you can add any number of **elif**

```
if expression then
    statement1
elif expression2 then
    statement2
else
    statement3
end if
```

Example

The following Sign procedure returns “negative”, “zero” or “positive” for a given number.

```
Sign:=proc(x)
    if x<0 then
        print(“negative”)
    elif x=0 then
        print(“zero”)
    else
        print(“positive”)
    end proc:
```

Repetitions – the for and while loops

The **for** loop is used to execute a sequence of statements a counted number of times. Here is the loop in its basic form

```
for name from expression by expression to expression do
    statement
end do;
```

Example

Adding the numbers from 1 to n.

```
Add1toN:=proc(n)
    local s, i;
    s:=0;
    for i from 1 to n do
```

```

        s=s+i;
    end do
end proc:

```

Note that here the *i* goes up by 1. Suppose we want to add the first *n* odd numbers, that is, the odd numbers from 1 to $2n-1$.

```

AddOdd:=proc(n)
    local s, i;
    s:=0;
    for i from 1 to 2*n-1 by 2 do
        s=s+i;
    end do
end proc:

```

The **while** loop is used to execute a sequence of statements until a condition is satisfied.

```

while expression do
    Statements
end do

```

Example

The procedure Div4 divides a number by 4 until it is less than 1.

```

Div4:=proc(x)
    local t;
    t:=x;
    while t > 1 do
        t:=t/4;
    end do
end proc:

```

Note that if the condition is never reached, then the program can go on forever! So be careful when using the while command to make sure that it will stop at some point.

The more general **for** command has a **while** in it.

```

for name from expression by expression to expression
while expression do
    statements
end do;

```

Example

Find the first prime number after 5000

```

PR5000:=proc()
    local i;
    for i from 5000
    while not isprime(i) do
    end do;
    i;
end proc;

```

In the above procedure, by default, the *i* will go up by one. The command not **isprime(i)** just means that the loop continues while we are not in the presence of a prime number.

More on Procedures

Global variables

Procedures allow the use of global variables, which will be recognized both inside and outside the procedure, as opposed to local variables, which are recognized only inside the procedure. For example, suppose we want a procedure that lists the first n terms of the dynamical system $a_{n+1} = 3a_n$, $a_1 = 2$, and remembers them.

```

restart:
DS:=proc(n)
  local i;
  global a;
  a[1]:=2;
  for i from 1 to n-1 do
    a[i+1]:=3*a[i];
  end do;
  seq(a[j],j=1..n);
end proc:

```

Try

```

DS(5);
a[3];
a[12];

```

Note that it does not give a_{12} since we ran DS for $n = 5$ only.

Recursive Procedures

When writing a procedure, you can make it call another procedure. In particular, you can make it call itself.

Example

Suppose we want a procedure that gives the n^{th} term of the dynamical system $a_{n+1} = 3a_n$, $a_1 = 2$.

```

DA:=proc(n)
  if n=1 then 2
  else 3*DA(n-1)
  end if
end proc:

```

Try

```

DA(1);
DA(4);
DA(1.3);

```

Note that in the last instance, Maple will display an error. There are ways to force n to be a nonnegative integer by using **n::nonnegint**, which means DA will only accept a nonnegative integer.

```

DA:=proc(n::nonnegint)
  if n=1 then 2
  else 3*DA(n-1)
  end if
end proc:

```