

6. Evaluation and Names of Variables

6.1 Evaluation

We will see how Maple offers a flexible evaluation mechanism. With right quotes we can prevent evaluation, as we will see in two examples.

The eval command

In an assignment, the right hand side gets fully evaluated first. If the expression at the right hand side does not yet have a value, then the left hand side of the assignment is just another name for that expression. Here is an illustration :

```
[> restart;                               # ensure all variables are free
[> a := b; b := c; c := 3;                 # see Figure 1 for effect
[> a;                                       # value of a is obtained by full evaluation
[> c := 5; a;                               # there is sharing of data
[> eval(a,1); eval(a,2); eval(a,3); # trace the links
```

In Figure 1 we show the internal representation after the first assignment. After `c := 5;`, the full evaluation of `a` and `b` also yields 5.

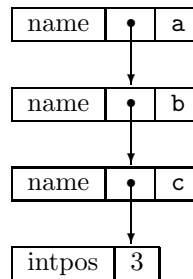


Figure 1: Internal representation of links between variables.

The following commands undo the links:

```
[> a := a;                               # destroys the link
[> c := 6; a;                             # but a still exists
[> a := 'a'; c; a;                         # release the variable a
```

Use of right quotes

As above right quotes can be used to undo an assignment, or to place links to other variables. In general, right quotes are there to prevent evaluation.

We need quotes with output variables of Maple procedures. If we supply the name of a variable without right quotes, then Maple evaluates that name and takes the value of that name. With right quotes, we can let the procedure assign a value to that name.

To illustrate this use, consider the polynomial division. Just as with integers, when we divide, we have a remainder and a quotient. The `rem` command returns the remainder of the division and returns the quotient as extra parameter.

```
[> p := x^2 - x + 3; q := x - 1; quotient := 0;
[> rest := rem(p,q,x,quotient); # this evaluates the quotient to zero
```

This results in an error, the correct way is

```
[> rest := rem(p,q,x,'quotient'); # similar to call-by-reference in C
[> p = q*quotient + rest; # check the expression
[> expand(rhs(%));
```

Another similar use appears in summations, where the running index should be a local variable.

```
[> i := 4;
[> sum(i,i=1..7); # aim to sum the first seven numbers
```

Again, this results in an error, the correct way is

```
[> sum('i','i'=1..7); # i is just a local variable, as it should be
```

6.2 Names of Variables

Be aware that Maple is case sensitive :

```
[> Pi, pi; # same symbol
[> evalf(Pi), evalf(pi); # but only one of them is "the" pi
```

Some variables like Pi are protected, so is the Greek letter γ (gamma) – for Euler's constant.

```
[> gamma := 3; # does not work
[> evalf(gamma); # approximate Euler's constant
[> unprotect(gamma); # unprotect this variable
[> gamma := 3; # works
```

To undo this unprotect, we unassign the variable gamma:

```
[> gamma := 'gamma'; # unassign with right quotes
[> evalf(gamma,30); # 30-digit approximation
```

Use of left quotes

Left quotes are used to work with names of variables that have special symbols in them, like spaces:

```
[> 'today's day is' := "Monday";
```

Concatenation of symbols

Sometimes we want to make a whole sequence of symbols, x1, x2, etc... The concatenation of symbols is done since Maple 7 with the || notation (in previous versions of Maple this was the dot) :

```
[> x||(1..4); # concatenation since Maple 7
[> a||(1..3)||(1..3); # symbols for a matrix
```

To make a1, a2, b1, b2, c1, c2, we could use

```
[> (a,b,c)||(1..2); # Maple does not like this
```

We can get around this by concatenation to the empty symbol:

```
[> '||(a,b,c)||(1..2);
```

Note that the first name does not get evaluated when concatenating:

```
[> i := 4; x||i, i||x;
```

6.3 Assignments

1. Explain the difference between the variables `Zeta` and `zeta`. Illustrate the difference with an example of a Maple session.
2. Describe in detail the difference between the two Maple sessions below by drawing picture of the internal data structures:

(a) `[> a := b;`
`[> b := 3;`
`[> a;`
`[> b := 4;`
`[> a;`

(b) `[> b := 3;`
`[> a := b;`
`[> a;`
`[> b := 4;`
`[> a;`

3. Do `[> restart; a := b; b := c; c := 17;`
and

- (a) draw a picture of the internal data structures to represent the dependencies between `a`, `b`, and `c`;
- (b) give the Maple commands to check those dependencies;
- (c) complete the right hand sides in the assignments of the sequence

`[> restart; c := 17; b := ___; a := ___;`

to obtain the same dependencies as with the other sequence above;

- (d) verify your answer to (c) with the appropriate Maple commands.
4. Explain the difference between `product(x||i, i=1..8)` and `product('x||i', 'i'=1..8)`.
Do `restart;` and show what Maple returns as output of these two commands.
Explain how Maple interprets these two commands.