**Description**

`return` causes the function to stop execution and return to the caller, returning nothing.

`return(exp)` causes the function to stop execution and return to the caller, returning the evaluation of `exp`.

**Syntax**

```
return

return(exp)
```

**Remarks and examples**

Remarks are presented under the following headings:

- Functions that return results
- Functions that return nothing (void functions)

**Functions that return results**

`return(exp)` specifies the value to be returned. For instance, you have written a program to return the sum of two numbers:

```
function mysum(a, b)
{
    return(a+b)
}
```

`return(exp)` may appear multiple times in the program. The following program calculates $x$ factorial; it assumes $x$ is an integer greater than 0:

```
real scalar myfactorial(real scalar x)
{
    if (x<=0) return(1)
    return(x*factorial(x-1))
}
```

If $x \leq 0$, the function returns 1; execution does not continue to the next line.

Functions that return a result always include one or more `return(exp)` statements.
Functions that return nothing (void functions)

A function is said to be void if it returns nothing. The following program changes the diagonal of a
matrix to be 1:

```plaintext
function fixdiag(matrix A)
{
    real scalar i
    for (i=1; i<=rows(A); i++) A[i,i] = 1
}
```

This function does not even include a `return` statement; execution just ends. That is fine, although
the function could just as well read

```plaintext
function fixdiag(matrix A)
{
    real scalar i
    for (i=1; i<=rows(A); i++) A[i,i] = 1
    return
}
```

The use of `return` is when the function has reason to end early:

```plaintext
void fixmatrix(matrix A, scalar how)
{
    real scalar i, j
    for (i=1; i<=rows(A); i++) A[i,i] = 1
    if (how==0) return
    for (i=1; i<=rows(A); i++) {
        for (j=1; j<i; j++) A[i,j] = 0
    }
}
```

Also see

[M-5] `exit()` — Terminate execution
[M-2] `Intro` — Language definition