

Explicit Definition $m : n$

As defined and illustrated in Section II H, the phrase $s=: 0 : 0$ may be used to define s as a script, and the explicit definition conjunction can be further used to produce a dyadic-only verb ($4 : s$), verb ($3 : s$), conjunction ($2 : s$), adverb ($1 : s$), or noun ($0 : s$). The left arguments 14 to 10 may be used instead of 4 to 0 to produce equivalent results in tacit form if possible. The right argument 0 may be used in each case to make the corresponding definitions directly from the keyboard: $k : 0$ is equivalent to $k : (0 : 0)$. Furthermore, the boxed representation $b=: < ; _2 s$ or the table representation $t=: > b$ (or $t=: [; _2 s$) may be used in lieu of the script s in every case. Thus:

```
f=: 3 : 0
a=: 2+b=. y. ^ 2
a+a*b
:
x.*x.+y.
)
```

```
a=: b=: 19
f 3
110
```

```
a,b                                Only the globally assigned name is changed.
11 19
```

As illustrated by the foregoing:

1. The definitions of the monadic and dyadic cases produced by $3 : 0$ are separated by a colon on a line by itself; if none occurs, the domain of the dyadic case is empty.
2. The explicit result is the result of the last non-test block sentence executed. See Control Structures for the definition of a test block.
3. A name assigned by the copula $=.$ is made local; values assigned to it have no effect on the use of the same name without the entity defined or within other entities invoked by it. A name assigned by $=:$ is global.
4. A locative cannot be localized.
5. The names $x.$ and $y.$ denote the left and right arguments. In defining a conjunction it may be necessary to refer to its left and right arguments (using $u.$ and $v.$) as well as to the arguments of the resulting function ($x.$ and $y.$). The use of $m.$ instead of $u.$ restricts the corresponding argument to being a noun, as does the use of $n.$ instead of $v.$. For example:

Continued

Explicit Definition $m : n$ _ _ _

Continued

```

conj=: 2 : '(u. y.)+ (v. y.)'
mc=: 2 : 0
(u.y.)+(v.y.)
)

dc=: 2 : 0          Dyadic case
:
(u.y.)+(v.x.)
)

(!conj% 2 4 5);(!mc% 2 4 5);(1 2 3 !dc% 2 4 5)
+-----+-----+-----+
|2.5 24.25 120.2|2.5 24.25 120.2|3 24.5 120.333|
+-----+-----+-----+

```

Control Structures . The sequence of execution of an explicit definition may be determined by control words such as `if.` `do.` `else.` `end.` and `while.`. For example, a function to find the root of a function `f` from a two-element list that brackets the root may be written and executed as follows:

```

root=: 3 : 0
m=. +/%#while.~/y.do.if.~/*f b=. (m,{.)y.do.y.=.b else.y.=. (m,{:)y.end.end.m y.
)

f=: 4:-%:
b=: 1 32
root b
16

```

Such a definition may also be written on successive lines by breaking it before or after any control word, and the foregoing definition may be made more readable as follows:

```

root=: 3 : 0
m=. +/%#
while. ~/y.
do. if. ~/*f b=. (m,{.) y.
do. y.=. b
else. y.=. (m,{:) y.
end.
end. m y.
)

```

Continued

Explicit Definition `m : n` `— — —`

Continued

As illustrated by the foregoing, the word `if.` and a matching `end.` mark the beginning and end of a control structure, as `do` `while.` and a matching `end.`; such structures may be nested as is the `if.` structure within the `while.` structure.

The control words `for.`, `if.`, `select.`, `try.`, `while.`, and `whilst.` mark the beginnings of control structures that are each terminated by a matching `end.`, and therefore provide a form of punctuation. In the foregoing example, `do.` and `else.` break the `if.` structure into three simple blocks, each comprising a sentence, whereas the `do.` in the `while.` structure breaks it into two blocks, the first being a simple sentence, and the second being itself an `if.` control structure.

In general, a block comprises zero or more simple sentences and control structures. The role of blocks is summarized as follows:

```

for.      T do. B end.
for_xyz. T do. B end.
if. T do. B end.
if. T do. B else. B1 end.
if. T do. B elseif. T1 do. B1 elseif. T2 do. B2 end.
select. T case. T0 do. B0 fcase. T1 do. B1 case. T2 do. B2 end.
try. B catch. B1 end.
while. T do. B end.
whilst. T do. B end.      Like while., but Skips Test first time.

```

Words beginning with `B` or `T` denote blocks. The last sentence executed in a `T` block is tested for a non-zero value in its leading atom, and the result of the test determines the block to be executed next. (An empty `T` block result or an omitted `T` block tests true.) If an error occurs in a `try.` block, execution continues in the matching `catch.` block. The final result is the result of the last sentence executed that was not in a `T` block, and if there is no such last executed sentence, the final result is `i.0 0`.

The behaviour of the remaining control words may be summarized as follows:

```

break.      Go to end of for., while., or whilst. control structure
continue.   Go to top of for., while., or whilst. control structure
goto_name.  Go to label of same name
label_name. Target of goto_name.
return.     Exit the function

```

Additional explanations and examples can be found in the Control Structures section.

Continued

Explicit Definition $m : n$ _ _ _

Continued

The following example uses control words as well as `u.` and `n.` in modelling the Level conjunction `L:`:

```

Level=: 2 : 0
m=. 0{ 3&$&.|. n.
ly=. L. y. if. 0>m do. m=.0>.m+ly end.
if. m>:ly do. u. y. else. u. Level m&.> y. end.
:
'l r'=. 1 2{ 3&$&.|. n.
lx=. L. x. if. 0>l do. l=.0>. l + lx end.
ly L. x. if. 0>r do. r=.0>. r + ly end.
b=. (l,r)>:lx,ly
if. b-: 0 0 do. x. u. Level(l,r)&.>y.
elseif. b-: 0 1 do. x. u. Level(l,r)&.>y.
elseif. b-: 1 0 do. (<x.) u. Level(l,r)&.>y.
elseif. 1 do. x. u. y.
end.
)
] a=: (i.2 3);(<<2 3 4) ; 3

```

```

+-----+
| 0 1 2 | +-----+ | 3 | | |
| 3 4 5 | | +-----+ |
|         | | | 2 3 4 | |
|         | | +-----+ |
|         | | +-----+ |
+-----+

```

+: Level 0 a

```

+-----+
| 0 2 4 | +-----+ | 6 | | |
| 6 8 10 | | +-----+ |
|         | | | 4 6 8 | |
|         | | +-----+ |
|         | | +-----+ |
+-----+

```

+: L: 0 a

```

+-----+
| 0 2 4 | +-----+ | 6 | | |
| 6 8 10 | | +-----+ |
|         | | | 4 6 8 | |
|         | | +-----+ |
|         | | +-----+ |
+-----+

```

Monad / Dyad u : v _ _ _

The first argument specifies the monadic case and the second argument the dyadic case.

For example:

```

y=: 10 64 100
^ . y
2.30259 4.15888 4.60517
Natural logarithm

10&^ . y
1 1.80618 2
Base ten logarithm

log=: 10&^ . : ^ .
log y
1 1.80618 2

8 log y
1.10731 2 2.21462

(^1) log y
2.30259 4.15888 4.60517

LOG=: 10&$ : : ^ .
LOG y
1 1.80618 2
Use of self-reference

f=: % : : ($:@-)
(f y),: 100 f y
3.16228 8 10
9.48683 6 0

ABS=: | : [:
ABS _4
4

3 ABS _4
|valence error: ABS
| 3 ABS _4

```

The domain of the dyad ABS is empty because the domain of | : is empty.