Returned args — Function arguments used to return results

Description Syntax Remarks and examples Also see

Description

Most Mata functions leave their arguments unchanged and return a result:

```
: y = f(x, ...)
```

Some Mata functions, however, return nothing and instead return results in one or more arguments:

```
: g(x, ..., y)
```

If you use such functions interactively and the arguments that are to receive results are not already defined (*y* in the above example), you will get a variable-not-found error. The solution is to define the arguments to contain something—anything—before calling the function:

```
: y = .
: g(x, ..., y)
```

You can combine this into one statement:

```
: g(x, ..., y=.)
```

Syntax

$y = f(x, \ldots)$	(function returns result the usual way)
$g(x, \ldots, y)$	(function returns result in argument y)

Remarks and examples

`sqrt(`*a*`)`—see [M-5] **sqrt( )**—calculates the (element-by-element) square root of *a* and returns the result:

```
: x = 4
: y = sqrt(x)
: y                 // y now contains 2
  2
: x                 // x is unchanged
  4
```

Most functions work like `sqrt()`, although many take more than one argument.

On the other hand, `polydiv(`c_a`,` c_b`,` c_q`,` c_r`)`—see [M-5] **polyeval( )**—takes the polynomial stored in c_a and the polynomial stored in c_b and divides them. It returns the quotient in the third argument (c_q) and the remainder in the fourth (c_r). c_a and c_b are left unchanged. The function itself returns nothing:

```
: A = (1,2,3)
: B = (0,1)
: polydiv(A, B, Q, R)
: Q                     // Q has been redefined
       1   2
  1  |  2   3  |

: R                     // as has R
  1
: A                     // while A and B are unchanged
       1   2   3
  1  |  1   2   3  |

: B
       1   2
  1  |  0   1  |
```

As another example, `st_view(`*V*`,` *i*`,` *j*`)`—see [M-5] **st_view( )**—creates a view onto the Stata dataset. Views are like matrices but consume less memory. Arguments *i* and *j* specify the observations and variables to be selected. Rather than returning the matrix, however, the result is returned in the first argument (*V*).

```
: st_view(V, (1\5), ("mpg", "weight"))
: V
          1       2
  1  |     22    2930  |
  2  |     15    4080  |
```

If you try to use these functions interactively, you will probably get an error:

```
: polydiv(A, B, Q, R)
                <istmt>:  3499  Q not found
r(3499);
: st_view(V, (1\5), ("mpg", "weight"))
                <istmt>:  3499  V not found
r(3499);
```

Arguments must be defined before they are used, even if their only purpose is to receive a newly calculated result. In such cases, it does not matter how the argument is defined because its contents will be replaced. Easiest is to fill in a missing value:

```
: Q = .
: R = .
: polydiv(A, B, Q, R)
: V = .
: st_view(V, (1\5), ("mpg", "weight"))
```

You can also define the argument inside the function:

```
: polydiv(A, B, Q=., R=.)
: st_view(V=., (1\5), ("mpg", "weight"))
```

When you use functions like these inside a program, however, you need not worry about defining the arguments, because they are defined by virtue of appearing in your program:

```
function foo()
{
     ...
     polydiv(A, B, Q, R)
     st_view(V, (1\5), ("mpg", "weight"))
     ...
}
```

When Mata compiles your program, however, you may see warning messages:

```
: function foo()
> {
>     ...
>     polydiv(A, B, Q, R)
>     st_view(V, (1\5), ("mpg", "weight"))
>     ...
> }
note: variable Q may be used before set
note: variable R may be used before set
note: variable V may be used before set
```

If the warning messages bother you, either define the variables before they are used just as you would interactively or use `pragma` to suppress the warning messages; see [M-2] **pragma**.

Also see

[M-1] **Intro** — Introduction and advice