**optargs** — Optional arguments

Description Syntax Remarks and examples Also see

# Description

Mata functions may have various numbers of arguments. How you write programs that allow these optional arguments is described below.

# Syntax

```
function functionname(|arg [, arg [, ...]]) {...}

function functionname(arg, |arg [, ...]) {...}

function functionname(arg, arg , |...) {...}
```

The vertical (or) bar separates required arguments from optional arguments in function declarations. The bar may appear at most once.

# Remarks and examples

Remarks are presented under the following headings:



## What are optional arguments?

### ▷ Example 1

You write a function named `ditty()`. Function `ditty()` allows the caller to specify two or three arguments:

```
real matrix ditty(real matrix A, real matrix B, real scalar scale)
real matrix ditty(real matrix A, real matrix B)
```

If the caller specifies only two arguments, results are as if the caller had specified the third argument equal to missing; that is, `ditty(A, B)` is equivalent to `ditty(A, B, .)`

◁

### ▷ Example 2

You write function `gash()`. Function `gash()` allows the caller to specify one or two arguments:

*real matrix* `gash(`*real matrix A*`,` *real matrix B*`)`
*real matrix* `gash(`*real matrix A*`)`

If the caller specifies only one argument, results are as if `J(0,0,.)` were specified for the second.

◁

### ▷ Example 3

You write function `easygoing()`. Function `easygoing()` takes three arguments but allows the caller to specify three, two, one, or even no arguments:

*real scalar* `easygoing(`*real matrix A*`,` *real matrix B*`,` *real scalar scale*`)`
*real scalar* `easygoing(`*real matrix A*`,` *real matrix B*`)`
*real scalar* `easygoing(`*real matrix A*`)`
*real scalar* `easygoing()`

If *scale* is not specified, results are as if $scale = 1$ were specified. If $B$ is not specified, results are as if $B = A$ were specified. If $A$ is not specified, results are as if $A =$ `I(2)` were specified.

◁

### ▷ Example 4

You write function `midsection()`. `midsection()` takes three arguments, but users may specify only two—the first and last—if they wish.

*real matrix* `midsection(`*real matrix A*`,` *real vector w*`,` *real matrix B*`)`
*real matrix* `midsection(`*real matrix A*`,` *real matrix B*`)`

If $w$ is not specified, results are as if $w =$ `J(1,cols(`$A$`),1)` was specified.

◁

## How to code optional arguments

When you code

```
function nebulous(a, b, c)
{
     ...
}
```

you are stating that function `nebulous()` requires three arguments. If the caller specifies fewer or more, execution will abort.

If you code

```
function nebulous(a, b, |c)
{
    ...
}
```

you are stating that the last argument is optional. Note the vertical or bar in front of *c*.

If you code

```
function nebulous(a, |b, c)
{
    ...
}
```

you are stating that the last two arguments are optional; the user may specify one, two, or three arguments.

If you code

```
function nebulous(|a, b, c)
{
    ...
}
```

you are stating that all arguments are optional; the user may specify zero, one, two, or three arguments.

The arguments that the user does not specify will be filled in according to the arguments' type,

| If the argument type is . . . | The default value will be . . . |
|---|---|
| undeclared | `J(0, 0, .)` |
| transmorphic matrix | `J(0, 0, .)` |
| real matrix | `J(0, 0, .)` |
| complex matrix | `J(0, 0, 1i)` |
| string matrix | `J(0, 0, "")` |
| pointer matrix | `J(0, 0, NULL)` |
| transmorphic rowvector | `J(1, 0, .)` |
| real rowvector | `J(1, 0, .)` |
| complex rowvector | `J(1, 0, 1i)` |
| string rowvector | `J(1, 0, "")` |
| pointer rowvector | `J(1, 0, NULL)` |
| transmorphic colvector | `J(0, 1, .)` |
| real colvector | `J(0, 1, .)` |
| complex colvector | `J(0, 1, 1i)` |
| string colvector | `J(0, 1, "")` |
| pointer colvector | `J(0, 1, NULL)` |
| transmorphic vector | `J(1, 0, .)` |
| real vector | `J(1, 0, .)` |
| complex vector | `J(1, 0, 1i)` |
| string vector | `J(1, 0, "")` |
| pointer vector | `J(1, 0, NULL)` |
| transmorphic scalar | `J(1, 1, .)` |
| real scalar | `J(1, 1, .)` |
| complex scalar | `J(1, 1, C(.))` |
| string scalar | `J(1, 1, "")` |
| pointer scalar | `J(1, 1, NULL)` |

Also, the function `args()` (see [M-5] **args( )**) will return the number of arguments that the user specified.

The vertical bar can be specified only once. That is sufficient, as we will show.

## Examples revisited

### ▷ Example 1

In this example, real matrix function `ditty(`*A*`, `*B*`, `*scale*`)` allowed real scalar *scale* to be optional. If *scale* was not specified, results were as if *scale*`=.` had been specified. This can be coded

```
real matrix ditty(real matrix A, real matrix B, |real scalar scale)
{
     ...
}
```

The body of the code is written just as if *scale* were not optional because, if the caller does not specify the argument, the missing argument is automatically filled in with missing, per the table above.

◁

## ▷ Example 2

Real matrix function `gash(`*A*`,` *B*`)` allowed real matrix *B* to be optional, and if not specified, *B* = `J(0,0,.)` was assumed. Hence, this is coded just as example 1 was coded:

```
real matrix gash(real matrix A, |real matrix B)
{
     ...
}
```

◁

## ▷ Example 3

Real scalar function `easygoing(`*A*`,` *B*`,` *scale*`)` allowed all arguments to be optional. *scale* = 1 was assumed, *B* = *A*, and if necessary, *A* = `I(2)`.

```
real scalar easygoing(|real matrix A, real matrix B,
                       real scalar scale)
{
     ...
     if (args()==2) scale = 1
     else if (args==1)  {
          B = A
          scale = 1
     }
     else if (args()==0) {
          A = B = I(2)
          scale = 1 ;
     }
     ...
}
```

◁

## ▷ Example 4

Real matrix function `midsection(`*A*`,` *w*`,` *B*`)` allowed *w*—its middle argument—to be omitted. If *w* was not specified, `J(1, cols(`*A*`), 1)` was assumed. Here is one solution:

```
real matrix midsection(a1, a2, |a3)
{
     if (args()==3) return(midsection_u(a1,       a2,        a3))
     else           return(midsection_u(a1, J(1,cols(a1),1), a2))
}

real matrix midsection_u(real matrix A, real vector w, real matrix B)
{
     ...
}
```

We will never tell callers about the existence of `midsection_u()` even though `midsection_u()` is our real program.

What we did above was write `midsection()` to take two or three arguments, and then we called `midsection_u()` with the arguments in the correct position.

◁

## Also see

[M-2] **Intro** — Language definition