# Title

stata.com

**isfleeting( ) —** Whether argument is temporary



## Description

`isfleeting(`*A*`)` returns 1 if *A* was constructed for the sole purpose of passing to your function, and returns 0 otherwise. If an argument is fleeting, then you may change its contents and be assured that the caller will not care or even know.

## Syntax

*real scalar* `isfleeting(`*polymorphic matrix A*`)`

where *A* is an argument passed to your function.

## Remarks and examples

stata.com

Let us assume that you have written function `myfunc(`*A*`)` that takes *A*: $r \times c$ and returns an $r \times c$ matrix. Just to fix ideas, we will pretend that the code for `myfunc()` reads

```
real matrix myfunc(real matrix A)
{
        real scalar     i
        real matrix     B

        B=A
        for (i=1; i<=rows(B); i++) B[i,i] = 1
        return(B)
}
```

Function `myfunc(`*A*`)` returns a matrix equal to *A*, but with ones along the diagonal. Now let's imagine `myfunc()` in use. A snippet of the code might read

```
...
C = A*myfunc(D)*C
...
```

Here `D` is passed to `myfunc()`, and the argument `D` is said not to be fleeting. Now consider another code snippet:

```
...
D = A*myfunc(D+E)*D
...
```

In this code snippet, the argument passed to `myfunc()` is `D+E` and that argument is fleeting. It is fleeting because it was constructed for the sole purpose of being passed to `myfunc()`, and once `myfunc()` concludes, the matrix containing `D+E` will be discarded.

Arguments that are fleeting can be reused to save memory.

Look carefully at the code for `myfunc()`. It makes a copy of the matrix that it was passed. It did that to avoid damaging the matrix that it was passed. Making that copy, however, is unnecessary if the argument received was fleeting, because damaging something that would have been discarded anyway does not matter. Had we not made the copy, we would have saved not only computer time but also memory. Function `myfunc()` could be recoded to read

```
real matrix myfunc(real matrix A)
{
        real scalar     i
        real matrix     B

        if (isfleeting(A)) {
                for (i=1; i<=rows(A); i++) A[i,i] = 1
                return(A)
        }
        B=A
        for (i=1; i<=rows(B); i++) B[i,i] = 1
        return(B)
}
```

Here we wrote separate code for the fleeting and nonfleeting cases. That is not always necessary. We could use a pointer here to combine the two code blocks:

```
real matrix myfunc(real matrix A)
 {
        real scalar     i
        real matrix     B
        pointer scalar  p

        if (isfleeting(A)) p = &A
        else {
                B = A
                p = &B
        }
        for (i=1; i<=rows(*p); i++) (*p)[i,i] = 1
        return(*p)
}
```

Many official library functions come in two varieties: `_foo(`*A*`, ...)`, which replaces *A* with the calculated result, and `foo(`*A*`, ...)`, which returns the result leaving *A* unmodified. Invariably, the code for `foo()` reads

```
function foo(A, ...)
{
        matrix B

        if (isfleeting(A)) {
                _foo(A, ...)
                return(A)
        }
        _foo(B=A, ...)
        return(B)
}
```

This makes function `foo()` whoppingly efficient. If `foo()` is called with a temporary argument—an argument that could be modified without the caller being aware of it—then no extra copy of the matrix is ever made.

## Conformability

`isfleeting(`*A*`)`:

| | |
|---|---|
| *A*: | $r \times c$ |
| *result*: | $1 \times 1$ |

## Diagnostics

`isfleeting(`*A*`)` returns 1 if *A* is fleeting and not a view. The value returned is indeterminate if *A* is not an argument of the function in which it appears, and therefore the value of `isfleeting()` is also indeterminate when used interactively.

## Also see

[M-4] **programming** — Programming functions