Title

stata.com

Programming functions

Contents Functions References Also see

Contents

`maxfloat()`	the largest value that can be stored in storage type `float`
`maxint()`	the largest value that can be stored in storage type `int`
`maxlong()`	the largest value that can be stored in storage type `long`
`mi(`x_1`,`x_2`,...,`x_n`)`	a synonym for `missing(`x_1`,`x_2`,...,`x_n`)`
`minbyte()`	the smallest value that can be stored in storage type `byte`
`mindouble()`	the smallest value that can be stored in storage type `double`
`minfloat()`	the smallest value that can be stored in storage type `float`
`minint()`	the smallest value that can be stored in storage type `int`
`minlong()`	the smallest value that can be stored in storage type `long`
`missing(`x_1`,`x_2`,...,`x_n`)`	`1` if any x_i evaluates to *missing*; otherwise, `0`
`r(`*name*`)`	the value of the stored result `r(`*name*`)`; see [U] **18.8 Accessing results calculated by other programs**
`recode(`x`,`x_1`,...,`x_n`)`	*missing* if $x_1, x_2, \ldots, x_n$ is not weakly increasing; x if x is missing; x_1 if $x \leq x_1$; x_2 if $x \leq x_2$, ...; otherwise, x_n if $x > x_1$, x_2, ..., x_{n-1}. $x_i \geq .$ is interpreted as $x_i = +\infty$
`replay()`	`1` if the first nonblank character of local macro `` `0' `` is a comma, or if `` `0' `` is empty
`return(`*name*`)`	the value of the to-be-stored result `r(`*name*`)`; see [P] **return**
`s(`*name*`)`	the value of stored result `s(`*name*`)`; see [U] **18.8 Accessing results calculated by other programs**
`scalar(`*exp*`)`	restricts name interpretation to scalars and matrices
`smallestdouble()`	the smallest double-precision number greater than zero

Functions

`autocode(`x`,`n`,`x_0`,`x_1`)`

Description: partitions the interval from x_0 to x_1 into n equal-length intervals and returns the upper bound of the interval that contains x

This function is an automated version of `recode()`. See [U] **25 Working with categorical data and factor variables** for an example.

The algorithm for `autocode()` is

if $(n \geq . \,|\, x_0 \geq . \,|\, x_1 \geq . \,|\, n \leq 0 \,|\, x_0 \geq x_1)$
  then return *missing*
  if $x \geq .$, then return x
otherwise
  for $i = 1$ to $n - 1$
    $xmap = x_0 + i * (x_1 - x_0)/n$
    if $x \leq xmap$ then return $xmap$
  end
  otherwise
    return x_1

Domain x: −8e+307 to 8e+307
Domain n: integers 1 to 8e+307
Domain x_0: −8e+307 to 8e+307
Domain x_1: x_0 to 8e+307
Range: x_0 to x_1

`byteorder()`

Description: `1` if your computer stores numbers by using a hilo byte order and evaluates to `2` if your computer stores numbers by using a lohi byte order

Consider the number 1 written as a 2-byte integer. On some computers (called hilo), it is written as "00 01", and on other computers (called lohi), it is written as "01 00" (with the least significant byte written first). There are similar issues for 4-byte integers, 4-byte floats, and 8-byte floats. Stata automatically handles byte-order differences for Stata-created files. Users need not be concerned about this issue. Programmers producing custom binary files can use `byteorder()` to determine the native byte ordering; see [P] **file**.

Range: 1 and 2

`c(`*name*`)`

Description: the value of the system or constant result `c(`*name*`)` (see [P] **creturn**)

Referencing `c(`*name*`)` will return an error if the result does not exist.

Domain: names

Range: real values, strings, or *missing*

`_caller()`

Description: `version` of the program or session that invoked the currently running program; see [P] **version**

The current version at the time of this writing is 15, so 15 is the upper end of this range. If Stata 15.1 were the current version, 15.1 would be the upper end of this range, and likewise, if Stata 16 were the current version, 16 would be the upper end of this range. This is a function for use by programmers.

Range: 1 to 15.1

`chop(`x`, `ϵ`)`

Description: `round(`x`)` if `abs(`x − `round(`x`))` $< \epsilon$; otherwise, x; or x if x is missing

Domain x: −8e+307 to 8e+307

Domain ϵ: −8e+307 to 8e+307

Range: −8e+307 to 8e+307

`clip(`x`,`a`,`b`)`

Description: x if $a < x < b$, b if $x \geq b$, a if $x \leq a$, or *missing* if x is missing or if $a > b$; x if x is missing

If a or b is missing, this is interpreted as $a = -\infty$ or $b = +\infty$, respectively.

Domain x: −8e+307 to 8e+307

Domain a: −8e+307 to 8e+307

Domain b: −8e+307 to 8e+307

Range: −8e+307 to 8e+307

`cond(`x`,`a`,`b`[,`c`])`

Description: a if x is *true* and nonmissing, b if x is *false*, and c if x is *missing*; a if c is not specified and x evaluates to *missing*

Note that expressions such as $x > 2$ will never evaluate to *missing*.

`cond(x>2,50,70)` returns `50` if `x` > 2 (includes `x` $\geq$ `.`)
`cond(x>2,50,70)` returns `70` if `x` ≤ 2

If you need a case for missing values in the above examples, try

`cond(missing(x), ., cond(x>2,50,70))` returns `.` if `x` is *missing*,
returns `50` if `x` > 2, and returns `70` if `x` ≤ 2

If the first argument is a scalar that may contain a missing value or a variable containing missing values, the fourth argument has an effect.

`cond(wage,1,0,.)` returns `1` if `wage` is not zero and not missing
`cond(wage,1,0,.)` returns `0` if `wage` is zero
`cond(wage,1,0,.)` returns `.` if `wage` is *missing*

Caution: If the first argument to `cond()` is a logical expression, that is, `cond(x>2,50,70,.)`, the fourth argument is never reached.

Domain x: $-$8e+307 to 8e+307 or *missing*; $0 \Rightarrow$ *false*, otherwise interpreted as *true*
Domain a: numbers and strings
Domain b: numbers if a is a number; strings if a is a string
Domain c: numbers if a is a number; strings if a is a string
Range: a, b, and c

`e(`*name*`)`

Description: the value of stored result `e(`*name*`)`; see **[U] 18.8 Accessing results calculated by other programs**

`e(`*name*`)` = scalar missing if the stored result does not exist
`e(`*name*`)` = specified matrix if the stored result is a matrix
`e(`*name*`)` = scalar numeric value if the stored result is a scalar

Domain: names
Range: strings, scalars, matrices, or *missing*

`e(sample)`

Description: `1` if the observation is in the estimation sample and `0` otherwise
Range: 0 and 1

`epsdouble()`

Description: the machine precision of a double-precision number

If $d <$ `epsdouble()` and (double) $x = 1$, then $x + d =$ (double) 1. This function takes no arguments, but the parentheses must be included.

Range: a double-precision number close to 0

`epsfloat()`

Description: the machine precision of a floating-point number

If $d <$ `epsfloat()` and (float) $x = 1$, then $x + d =$ (float) 1. This function takes no arguments, but the parentheses must be included.

Range: a floating-point number close to 0

`fileexists(`*f*`)`
Description: `1` if the file specified by *f* exists; otherwise, `0`

If the file exists but is not readable, `fileexists()` will still return `1`, because it does exist. If the "file" is a directory, `fileexists()` will return `0`.
Domain: filenames
Range: 0 and 1

`fileread(`*f*`)`
Description: the contents of the file specified by *f*

If the file does not exist or an I/O error occurs while reading the file, then "`fileread() error #`" is returned, where # is a standard Stata error return code.
Domain: filenames
Range: strings

`filereaderror(`*f*`)`
Description: `0` or positive integer, said value having the interpretation of a return code

It is used like this

```
. generate strL s = fileread(filename) if fileexists(filename)
. assert filereaderror(s)==0
```

or this

```
. generate strL s = fileread(filename) if fileexists(filename)
. generate rc = filereaderror(s)
```

That is, `filereaderror(`*s*`)` is used on the result returned by `fileread(`*filename*`)` to determine whether an I/O error occurred.

In the example, we only `fileread()` files that `fileexists()`. That is not required. If the file does not exist, that will be detected by `filereaderror()` as an error. The way we showed the example, we did not want to read missing files as errors. If we wanted to treat missing files as errors, we would have coded

```
. generate strL s = fileread(filename)
. assert filereaderror(s)==0
```

or

```
. generate strL s = fileread(filename)
. generate rc = filereaderror(s)
```

Domain: strings
Range: integers

`filewrite(`f`,`s`[,`r`])`
Description: writes the string specified by s to the file specified by f and returns the number of bytes in the resulting file

If the optional argument r is specified as 1, the file specified by f will be replaced if it exists. If r is specified as 2, the file specified by f will be appended to if it exists. Any other values of r are treated as if r were not specified; that is, f will only be written to if it does not already exist.

When the file f is freshly created or is replaced, the value returned by `filewrite()` is the number of bytes written to the file, `strlen(`s`)`. If r is specified as 2, and thus `filewrite()` is appending to an existing file, the value returned is the total number of bytes in the resulting file; that is, the value is the sum of the number of the bytes in the file as it existed before `filewrite()` was called and the number of bytes newly written to it, `strlen(`s`)`.

If the file exists and r is not specified as 1 or 2, or an error occurs while writing to the file, then a negative number (#) is returned, where `abs(#)` is a standard Stata error return code.
Domain f: filenames
Domain s: strings
Domain r: integers 1 or 2
Range: integers

`float(`x`)`
Description: the value of x rounded to `float` precision

Although you may store your numeric variables as `byte`, `int`, `long`, `float`, or `double`, Stata converts all numbers to `double` before performing any calculations. Consequently, difficulties can arise in comparing numbers that have no finite binary representation.

For example, if the variable `x` is stored as a `float` and contains the value `1.1` (a repeating "decimal" in binary), the expression `x==1.1` will evaluate to *false* because the literal `1.1` is the `double` representation of 1.1, which is different from the `float` representation stored in `x`. (They differ by 2.384×10^{-8}.) The expression `x==float(1.1)` will evaluate to *true* because the `float()` function converts the literal `1.1` to its `float` representation before it is compared with `x`. (See [U] **13.12 Precision and problems therein** for more information.)
Domain: −1e+38 to 1e+38
Range: −1e+38 to 1e+38

`fmtwidth(`*fmtstr*`)`
Description: the output length of the *%fmt* contained in *fmtstr*; *missing* if *fmtstr* does not contain a valid *%fmt*

For example, `fmtwidth("%9.2f")` returns `9` and `fmtwidth("%tc")` returns `18`.
Range: strings

`has_eprop(`*name*`)`
Description: `1` if *name* appears as a word in `e(properties)`; otherwise, `0`
Domain: names
Range: 0 or 1

`inlist(`z`,`a`,`b`,...)`
Description: `1` if z is a member of the remaining arguments; otherwise, `0`

All arguments must be reals or all must be strings. The number of arguments is between 2 and 255 for reals and between 2 and 10 for strings.
Domain: all reals or all strings
Range: 0 or 1

`inrange(`z`,`a`,`b`)`
Description: `1` if it is known that $a \leq z \leq b$; otherwise, `0`

The following ordered rules apply:
$z \geq$ `.` returns `0`.
$a \geq$ `.` and $b =$ `.` returns `1`.
$a \geq$ `.` returns `1` if $z \leq b$; otherwise, it returns `0`.
$b \geq$ `.` returns `1` if $a \leq z$; otherwise, it returns `0`.
Otherwise, `1` is returned if $a \leq z \leq b$.
If the arguments are strings, "`.`" is interpreted as `""`.
Domain: all reals or all strings
Range: 0 or 1

`irecode(`x`,`x_1`,`x_2`,`x_3`,...,`x_n`)`
Description: *missing* if x is missing or $x_1, \ldots, x_n$ is not weakly increasing; `0` if $x \leq x_1$; `1` if $x_1 < x \leq x_2$; `2` if $x_2 < x \leq x_3$; ...; n if $x > x_n$

Also see `autocode()` and `recode()` for other styles of recode functions.

`irecode(3, -10, -5, -3, -3, 0, 15, .)` $= 5$
Domain x: −8e+307 to 8e+307
Domain x_i: −8e+307 to 8e+307
Range: nonnegative integers

`matrix(`*exp*`)`
Description: restricts name interpretation to scalars and matrices; see `scalar()`
Domain: any valid expression
Range: evaluation of *exp*

`maxbyte()`
Description: the largest value that can be stored in storage type `byte`

This function takes no arguments, but the parentheses must be included.
Range: one integer number

`maxdouble()`
Description: the largest value that can be stored in storage type `double`

This function takes no arguments, but the parentheses must be included.
Range: one double-precision number

`maxfloat()`
Description: the largest value that can be stored in storage type `float`

This function takes no arguments, but the parentheses must be included.
Range: one floating-point number

`maxint()`
Description: the largest value that can be stored in storage type `int`

This function takes no arguments, but the parentheses must be included.
Range: one integer number

`maxlong()`
Description: the largest value that can be stored in storage type `long`

This function takes no arguments, but the parentheses must be included.
Range: one integer number

`mi(`x_1`,`x_2`,...,`x_n`)`
Description: a synonym for `missing(`x_1`,`x_2`,...,`x_n`)`

`minbyte()`
Description: the smallest value that can be stored in storage type `byte`

This function takes no arguments, but the parentheses must be included.
Range: one integer number

`mindouble()`
Description: the smallest value that can be stored in storage type `double`

This function takes no arguments, but the parentheses must be included.
Range: one double-precision number

`minfloat()`
Description: the smallest value that can be stored in storage type `float`

This function takes no arguments, but the parentheses must be included.
Range: one floating-point number

`minint()`
Description: the smallest value that can be stored in storage type `int`

This function takes no arguments, but the parentheses must be included.
Range: one integer number

`minlong()`
Description: the smallest value that can be stored in storage type `long`

This function takes no arguments, but the parentheses must be included.
Range: one integer number

`missing(`x_1`,`x_2`,...,`x_n`)`
Description: `1` if any x_i evaluates to *missing*; otherwise, `0`

Stata has two concepts of missing values: a numeric missing value (`.`, `.a`, `.b`, ..., `.z`) and a string missing value (`""`). `missing()` returns `1` (meaning *true*) if any expression x_i evaluates to *missing*. If x is numeric, `missing(`x`)` is equivalent to $x \geq$ `.`. If x is string, `missing(`x`)` is equivalent to x`==""`.
Domain x_i: any string or numeric expression
Range: 0 and 1

`r(`*name*`)`

Description: the value of the stored result `r(`*name*`)`; see [U] **18.8 Accessing results calculated by other programs**

`r(`*name*`)` = scalar missing if the stored result does not exist
`r(`*name*`)` = specified matrix if the stored result is a matrix
`r(`*name*`)` = scalar numeric value if the stored result is a scalar that can be interpreted as a number

Domain: names
Range: strings, scalars, matrices, or *missing*

`recode(`x`,`x_1`,`x_2`,...,`x_n`)`

Description: *missing* if $x_1, x_2, \ldots, x_n$ is not weakly increasing; x if x is missing; x_1 if $x \leq x_1$; x_2 if $x \leq x_2$, ...; otherwise, x_n if $x > x_1$, x_2, ..., x_{n-1}. $x_i \geq .$ is interpreted as $x_i = +\infty$

Also see `autocode()` and `irecode()` for other styles of recode functions.

Domain x: −8e+307 to 8e+307 or *missing*
Domain x_1: −8e+307 to 8e+307
Domain x_2: x_1 to 8e+307
...
Domain x_n: x_{n-1} to 8e+307
Range: x_1, x_2, ..., x_n or *missing*

`replay()`

Description: `1` if the first nonblank character of local macro `` `0' `` is a comma, or if `` `0' `` is empty

This is a function for use by programmers writing estimation commands; see [P] **ereturn**.

Range: integers 0 and 1, meaning *false* and *true*, respectively

`return(`*name*`)`

Description: the value of the to-be-stored result `r(`*name*`)`; see [P] **return**

`return(`*name*`)` = scalar missing if the stored result does not exist
`return(`*name*`)` = specified matrix if the stored result is a matrix
`return(`*name*`)` = scalar numeric value if the stored result is a scalar

Domain: names
Range: strings, scalars, matrices, or *missing*

`s(`*name*`)`

Description: the value of stored result `s(`*name*`)`; see [U] **18.8 Accessing results calculated by other programs**

`s(`*name*`)` = . if the stored result does not exist

Domain: names
Range: strings or *missing*

`scalar(`*exp*`)`

Description: restricts name interpretation to scalars and matrices

Names in expressions can refer to names of variables in the dataset, names of matrices, or names of scalars. Matrices and scalars can have the same names as variables in the dataset. If names conflict, Stata assumes that you are referring to the name of the variable in the dataset.

`matrix()` and `scalar()` explicitly state that you are referring to matrices and scalars. `matrix()` and `scalar()` are the same function; scalars and matrices may not have the same names and so cannot be confused. Typing `scalar(x)` makes it clear that you are referring to the scalar or matrix named `x` and not the variable named `x`, should there happen to be a variable of that name.

Domain: any valid expression

Range: evaluation of *exp*

`smallestdouble()`

Description: the smallest double-precision number greater than zero

If $0 < d <$ `smallestdouble()`, then d does not have full double precision; these are called the denormalized numbers. This function takes no arguments, but the parentheses must be included.

Range: a double-precision number close to 0

References

Kantor, D., and N. J. Cox. 2005. Depending on conditions: A tutorial on the cond() function. *Stata Journal* 5: 413–420.

Rising, W. R. 2010. Stata tip 86: The missing() function. *Stata Journal* 10: 303–304.

Also see

[FN] **Functions by category**

[D] **egen** — Extensions to generate

[D] **generate** — Create or change contents of variable

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

[U] **13.3 Functions**