Title

forvalues — Loop over consecutive values

Syntax

```
forvalues lname = range {
            Stata commands referring to `lname'
}
```

where *range* is

$\#_1$`(`$\#_d$`)`$\#_2$	meaning $\#_1$ to $\#_2$ in steps of $\#_d$
$\#_1$`/`$\#_2$	meaning $\#_1$ to $\#_2$ in steps of 1
$\#_1$ $\#_t$ `to` $\#_2$	meaning $\#_1$ to $\#_2$ in steps of $\#_t - \#_1$
$\#_1$ $\#_t$ `:` $\#_2$	meaning $\#_1$ to $\#_2$ in steps of $\#_t - \#_1$

The loop is executed as long as calculated values of '*lname*' are $\leq \#_2$, assuming that $\#_d > 0$.

Braces must be specified with `forvalues`, and

1. the open brace must appear on the same line as `forvalues`;
2. nothing may follow the open brace except, of course, comments; the first command to be executed must appear on a new line;
3. the close brace must appear on a line by itself.

Description

`forvalues` repeatedly sets local macro *lname* to each element of *range* and executes the commands enclosed in braces. The loop is executed zero or more times.

Remarks and examples

`forvalues` is the fastest way to execute a block of code for different numeric values of *lname*.

▷ Example 1

With `forvalues` *lname* `=` $\#_1$`(`$\#_d$`)`$\#_2$, the loop is executed zero or more times, once for *lname* $= \#_1$, once for *lname* $= \#_1 + \#_d$, once for *lname* $= \#_1 + \#_d + \#_d$, and so on, as long as *lname* $\leq \#_2$ (assuming $\#_d$ is positive) or as long as *lname* $\geq \#_2$ (assuming $\#_d$ is negative). Specifying $\#_d$ as 0 is an error.

```
. forvalues i = 1(1)5 {
  2.          display `i'
  3. }
1
2
3
4
5
```

lists the numbers 1–5, stepping by 1, whereas

```
      . forvalues i = 10(-2)1 {
        2.          display `i'
        3. }
      10
      8
      6
      4
      2
```

lists the numbers starting from 10, stepping down by 2 until it reaches 2. It stops at 2 instead of at 1 or 0.

```
      . forvalues i = 1(1)1 {
        2.          display `i'
        3. }
      1
```

displays 1, whereas

```
      . forvalues i = 1(1)0 {
        2.          display `i'
        3. }
```

displays nothing.

◁

`forvalues` *lname* = $\#_1$`/`$\#_2$ is the same as using `forvalues` *lname* = $\#_1$`(1)`$\#_2$. Using `/` does not allow counting backward.

▷ Example 2

```
      . forvalues i = 1/3 {
        2.          display `i'
        3. }
      1
      2
      3
```

lists the three values from 1 to 3, but

```
      . forvalues i = 3/1 {
        2.          display `i'
        3. }
```

lists nothing because using this form of the `forvalues` command allows incrementing only by 1.

◁

The `forvalues` *lname* = $\#_1$ $\#_t$ `to` $\#_2$ and `forvalues` *lname* = $\#_1$ $\#_t$ `:` $\#_2$ forms of the `forvalues` command are equivalent to computing $\#_d = \#_t - \#_1$ and then using the `forvalues` *lname* = $\#_1$`(`$\#_d$`)`$\#_2$ form of the command.

▷ Example 3

```
. forvalues i = 5 10 : 25 {
  2.          display `i'
  3. }
5
10
15
20
25
. forvalues i = 25 20 to 5 {
  2.          display `i'
  3. }
25
20
15
10
5
```

◁

❑ Technical note

It is not legal syntax to type

```
. scalar x = 3
. forvalues i = 1(1)`x' {
  2.          local x = `x' + 1
  3.          display `i'
  4. }
```

`forvalues` requires literal numbers. Using macros, as shown in the following technical note, is allowed.

❑

❑ Technical note

The values of the loop bounds are determined once and for all the first time the loop is executed. Changing the loop bounds will have no effect. For instance,

```
. local n 3
. forvalues i = 1(1)`n' {
  2.          local n = `n' + 1
  3.          display `i'
  4. }
1
2
3
```

will not create an infinite loop. With `` `n' `` originally equal to 3, the loop will be performed three times.

Similarly, modifying the loop counter will not affect `forvalues`' subsequent behavior. For instance,

```
. forvalues i = 1(1)3 {
  2.          display "Top of loop  i = `i'"
  3.          local i = `i' * 4
  4.          display "After change i = `i'"
  5. }
Top of loop  i = 1
After change i = 4
Top of loop  i = 2
After change i = 8
Top of loop  i = 3
After change i = 12
```

will still execute three times, setting `` `i' `` to 1, 2, and 3 at the beginning of each iteration.

❑

Reference

Cox, N. J. 2010. Stata tip 85: Looping over nonintegers. *Stata Journal* 10: 160–163.

Also see

[P] **continue** — Break out of loops

[P] **foreach** — Loop over items

[P] **if** — if programming command

[P] **while** — Looping

[U] **18 Programming Stata**

[U] **18.3 Macros**