An Introduction to Loops in Stata: Forvalues, Foreach of, Foreach in, and Nested Loops

A Community Resource

Created by: Ashley Weyers, University of Arizona

March 2019

Getting Started: Loops 101

Why use loops?

Loops are an extremely helpful tool that save you time when cleaning, manipulating, and visualizing your data by allowing you to run the same command for several variables at once without having to write separate lines of code.

Loops are easy to proof, and they will keep your do-file concise and clean by minimizing the space taken up by repetitive commands.They are also safer than repeating code.

Loops can be a bit complicated to learn at first, but they will save you time in the long run. So invest in learning how to write loops now because this skill will pay dividends down the road!

When to use loops:

When you go to copy and paste any commands to repeat them in your do-file, you should be asking yourself: “Should I be using a loop?”

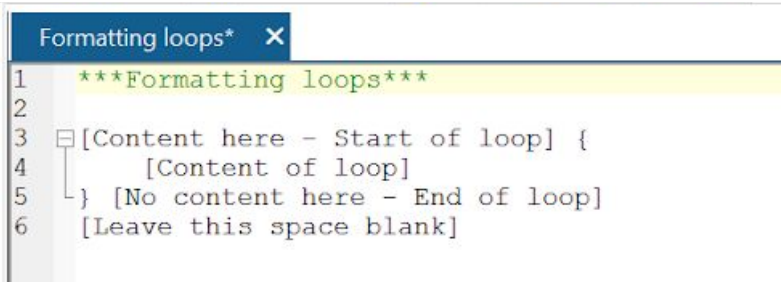
The answer is likely YES!

For example, if you want to recode variables in the same way and are copy and pasting that code over and over again and only changing the variable name each time, you should be using a loop to save you time and space in your do-file.

Formatting loops:

There are some key components of formatting loops:

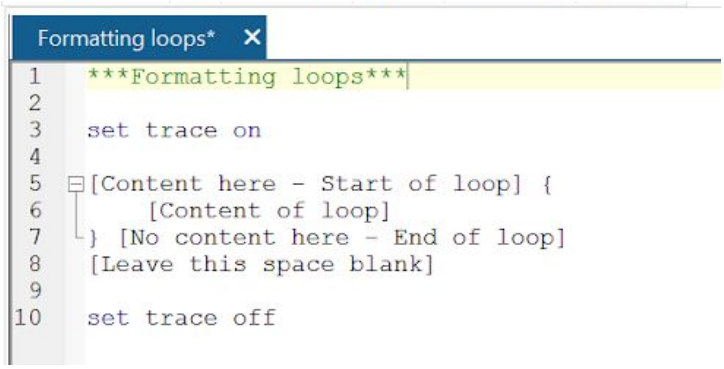
- You have to have an open bracket like this { to start your loop. Note that the open bracket is not indented. It follows the beginning text of your loop.
- Then, you will go to the second line of your loop. You will need to make an indent before you write the content of your loop. What you want to accomplish goes here. You can have multiple lines of code but each line should be indented.
- Once you have entered what you wish to accomplish with your loop, you will hit enter to go to the next line. There you will close your loop with a bracket }. This should not be indented.
- Finally, you will leave a blank line following your loop. This is something you need to do for your loop to run properly in Stata.
- Note: In the example format below, you will notice that to the left of your code there is a box with a dash in it, and a vertical line that then hooks to the end of the loop. This signals where the end bracket } should be, because there is not any indented content on that line.



Set trace on:

- Using the command *set trace on* before you start your loop will help you debug your loops.
- When your loop “breaks” (i.e., doesn’t run correctly), it will show you exactly what part of your loop is not working.

You will *set trace on* before your run your loop(s), and then *set trace off* after running your loop(s).

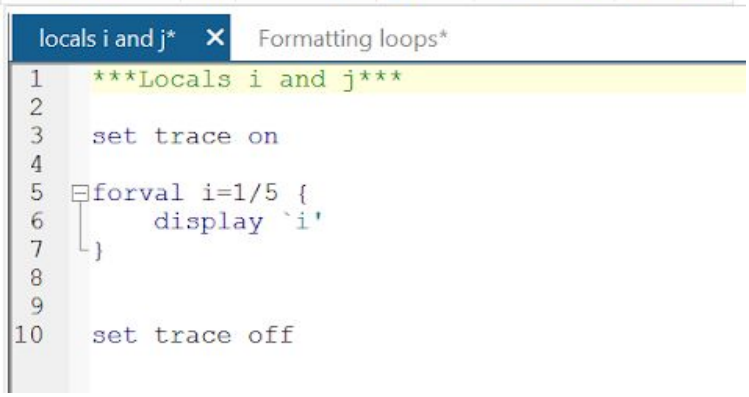


Locals i & j:

The most common programming names when writing loops are the local macros “i” and “j”. When written in the content of your loops, i and j should be written with a forward directional ` in front, and followed by an apostrophe ‘. It must be written this way or your loop will not work.

For example: `i’ or `j’

You will make your claim about “i” and/or “j” first without the forward directional and apostrophe, but include the forward directional and apostrophe around “i” and/or “j” in the content of your loop. (Reference the example below. I will explain what “forval” means in the following section.)



Types of Loops

Forvalues

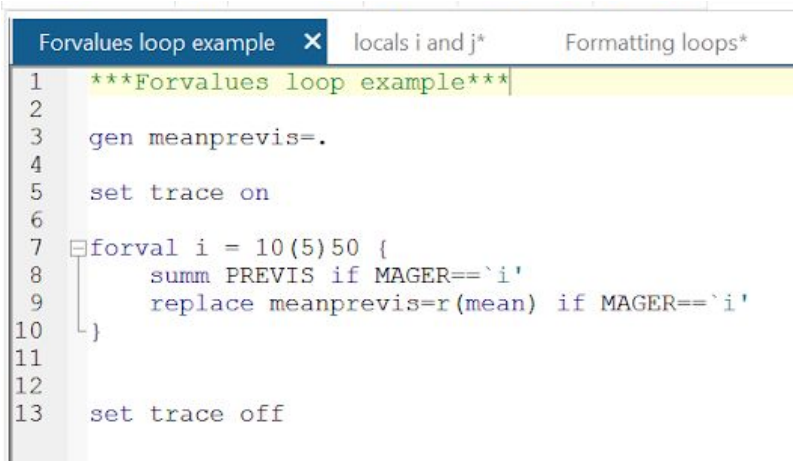
Forvalues is, arguably, the easiest loop to write. It will only count sequences of numbers, so the variable that you are calling must include numeric values.

You can write out *forvalues* or use the shortened *forval* command.

You will need to distinguish how you want to count through the number sequence. There are a few different options:

1. You can loop through a range of number by using a forward slash / between the numbers you want to loop through. For example, if you had fifteen numbers that you wanted to loop through starting at one you would write: forval i = 1/15 { to start the loop.
2. You can also loop through a pattern of numbers by denoting that pattern in parentheses ( ) between the number you want to start at and the number you want to end at. For example, if you wanted to count by 10s from 50 to 100 you would write: forval i = 50(10)100 { to start the loop.
3. You can also state the first two numbers of your pattern and use a colon : or write “to” the number you want to end the sequence on. For example, if you wanted to count down by twos from 20 to zero, you would write: forval i = 20 18 : 0 { to start the loop.

In the example that I provided (see image below), I am using forvalues to loop through mothers’ age (MAGER) categories by fives, and generating a new variable (meanprevis) that records the mean number of prenatal visits (PREVIS) during pregnancy per age category.



You can see that I decided to generate my new variable “meanprevis” outside of the loop, this is because I want one variable that includes the mean number of prenatal visits for each age category that I am requesting. I generated meanprevis equal to missing.

Then, by using forvalues I called age categories by fives, starting at 10-years-old and ending at 50-years-old. I asked Stata to put out a summary of the number of prenatal visits for each age category that I requested (10, 15, 20,..., 50). The second part of the body of the loop asks Stata to replace meanprevis with the mean number of prenatal visits for each age category.

Notice how I also remembered to *set trace on* before I started my loop, and turned it off with *set trace off* after I closed the loop.

We can see from the tabulation (see image below) that the mean number of prenatal appointments increases by age category. There were no observations for the first age category (10-years-old), but we can tell from the output that 15-year-olds averaged about 12 prenatal appointments, whereas 50-year-olds averaged about 17 prenatal appointments.

```
. tab meanprevis

 meanprevis |      Freq.     Percent        Cum.
------------+-----------------------------------
   12.29451 |      8,098        1.01        1.01
   13.21252 |    143,454       17.83       18.84
    13.5369 |    212,490       26.42       45.26
   13.94893 |    241,753       30.05       75.31
    14.1298 |    151,747       18.86       94.17
   14.27792 |     42,394        5.27       99.44
   14.91925 |      3,719        0.46       99.91
   17.01391 |        756        0.09      100.00
------------+-----------------------------------
      Total |    804,411      100.00
```

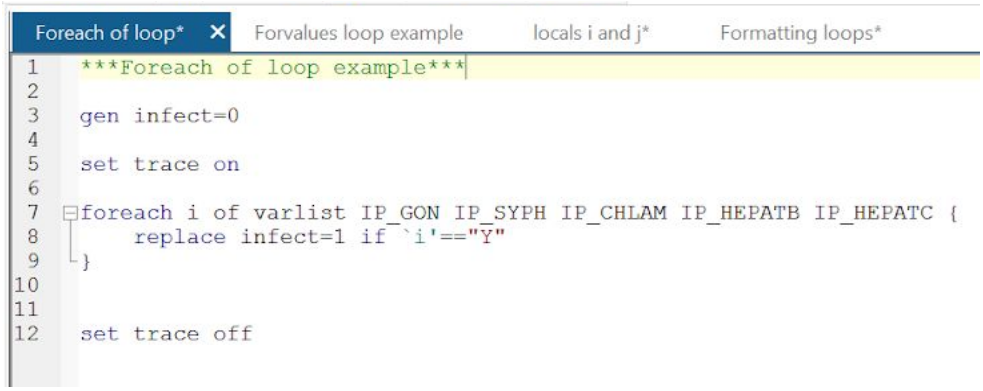
Foreach of

You can use *foreach of* to create a loop that will pull the list of things you wish to call from an indicated place.

You can use a loop with *foreach of* with a:

- local
- global
- varlist
- newlist
- numlist

In the example that I provided (see image below), I am using *foreach of* generate a dummy variable indicating whether or not a pregnant person had an STI or other infection during pregnancy. I call five variables (IP_* variables) of a varlist and use a loop to replace the value of infect=1 if a person was coded Y (i.e., yes) for having an STI or other infection.



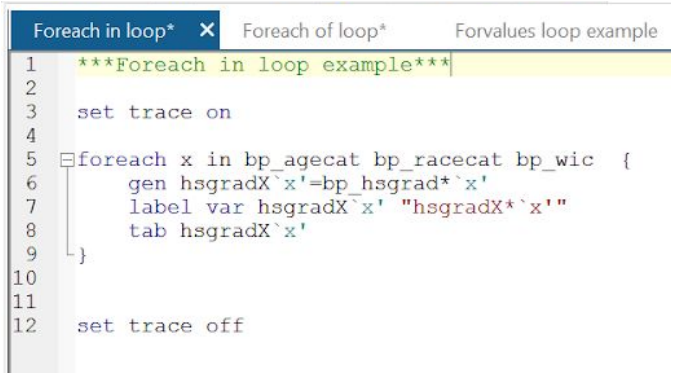
I chose to use “i” to represent each variable in the variable list, but you could also use something more intuitive like “var”: foreach var of varlist … {. (But remember that you would still need to use the forward directional and apostrophe when identifying var in the body of your loop (e.g., `var’).

Note that you want to be careful when using *foreach of* because it has a built in assert, meaning that your code in Stata will break if what you delivered it was not what you said it was. In many cases, it is safer to use *foreach in*.

Foreach in

Foreach in is similar to *foreach of*, except that you can call a list without specifying a location. *Foreach in* is nice because it does not have a built in assert, which lets you loop over things without Stata second guessing what you are trying to accomplish.

In the example I provided (see image below), I am using foreach in to generate interaction variables. I choose to interact whether a birthing person graduated high school (bp_hsgrad) with birthing persons’ age category (bp_agecat), race (bp_racecat), and whether or not they were a WIC recipient (bp_wic).



You may have noticed that I did not have to specify a location before listing the variables I was interested in interacting with bp_hsgrad. Following *foreach in*, you simply need to list the variables. You do not write any of the locations that follow *foreach of* (i.e., you do not state whether it is a local, global, varlist, newlist, or numlist).

I also used this loop as an opportunity to label my new interaction variables. See, there are so many ways that loops can save you time!

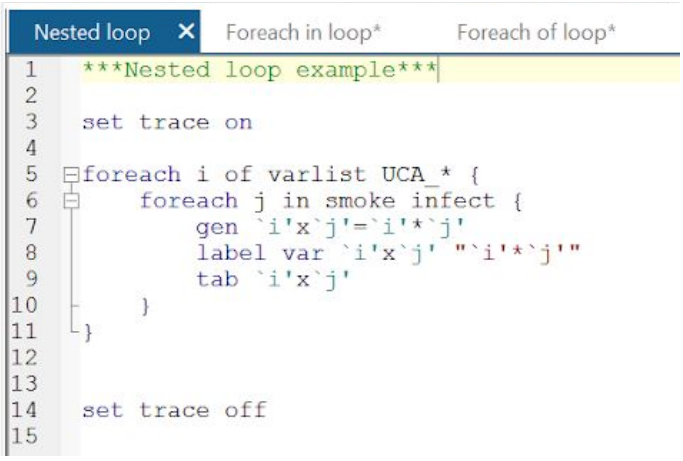
Nested loops

You can also use loops within a loop to make your work process more efficient. Nested loops can be as simple or as complex as you want to make them. However, it is recommended that you don't go more than three loops deep.

To keep things simple, my example (see image below), shows what it would look like to have one loop nested within a loop.

I used a nested loop to create interaction variables between each of the infant disability variables in my dataset (this is denoted by UCA_*--the asterisk * signals that I want each of the six variables that starts with UCA_ in the dataset) and the dummy variables smoke (whether the birthing person smoked during pregnancy) and infect (whether the birthing person had an STI or other infection during pregnancy). This loops will create 12 interaction variables. Each of the UCA_ variables will be interacted with smoke and infect, so 6 x 2=12.

You will notice that I used a combination of *foreach of* and *foreach in*. You can use any combination of *forvalues, foreach of*, and *foreach in* with your nested loops. Also, you can see in the example that the loop within the loop is indented twice, and that loop has to be closed after indenting, while the outer loop is closed on the following line without indenting.



Now that you've learned about different types of loops in Stata, I encourage you to test out writing some loops for yourself with the practice exercises in the following section.

Practice Exercises

In these exercises, use the auto dataset included in Stata to practice writing loops. The solutions are in the following section, but try to write and run the loops on your own before peeking!

Forvalues

Create a loop using forvalues to tabulate whether a car is foreign or domestic (foreign) for cars that get between 18 to 28 miles per gallon (mpg).

Foreach of/Foreach in

Create a loop that converts the variables measured in inches (length and headroom) to feet by generating two new variables measured in feet (length_ft and headroom_ft).

Nested loops

Use a nested loop to make cross tabulations of the variables you just created (length_ft and headroom_ft) with foreign and rep78.

Solutions to Practice Exercises

Forvalues

Forvalues exercise solution | Nested loop | Foreach in loo

```
1  ***Forvalues exercise solutions***
2
3  set trace on
4
5  forval i = 18/28 {
6      tab foreign if mpg==`i'
7  }
8
9
10 set trace off
11
12 ***or***
13
14 set trace on
15
16 forval i = 18(1)28 {
17     tab foreign if mpg==`i'
18 }
19
20
21 set trace off
22
23 ***or***
24
25 set trace on
26
27 forval i = 18 19 : 28 {
28     tab foreign if mpg==`i'
29 }
30
31
32 set trace off
```

Foreach of/Foreach in

Foreach exercise solution | Forvalues exercise solution | Nested loop | Foreach in loop* | For

```
***Foreach exercise solution***

set trace on

foreach i in length headroom {
    gen `i'_ft = `i'/12
}


set trace off

***or***

set trace on

foreach var of varlist length headroom {
    gen `var'_ft = `var'/12
}


set trace off

/*Notice that you can use whatever term you choose following "foreach".
I used "i" in the first example and "var" in the second example.*/
```

Nested loops

Nested loop solution | Foreach exercise solution | Forvalues exercise solution | Nested loop | Foreach ir

```
***Nested loop solution***

set trace on

foreach i of varlist length_ft headroom_ft {
    foreach j in foreign rep78 {
        tab `i' `j'
    }
}


set trace off

/* There are many different ways you can write this loop using foreach of and
foreach in. This is just one way you can write it. */
```