PROGRAMS AND PASSING ARGUMENTS

A community resource
created for SOC 561
by Rina James

OVERVIEW

This worksheet is meant to give you hands-on practice writing Stata programs. Since previous resources do a thorough job defining the difference between do-files, loops, and programs, the exercises here will focus on rewriting loops that accomplish specific tasks into programs that can be used for more general data management. The goal is to provide practice into understanding how to construct programs and pass arguments, and to be able to identify when a program is useful.

PREVIOUS RESOURCES

The exercises presented build off the information and examples presented in previous community resources. If you have not already familiarized yourself with these resources, I recommend doing so before starting. You can view them using the links below:

- Programs vs. do-files: What's the difference?
- Writing Programs in Stata

INSTRUCTIONS

Below I present three loops written to accomplish different data management tasks. Each loop can be converted into a program that provides a more generalized utility. Use the previous community resources and other class materials to rewrite the loop into a program that accomplishes the task specified in the instructions.

GETTING STARTED

The problems below use variables included in the 2016 cross-section of the General Social Survey (GSS). To download the data:

1. Visit: http://gss.norc.org/get-the-data/stata
2. Scroll down to the second box titled 'Download Individual Year Data Sets'
3. Click '2016'
4. Save the .zip file
5. Extract and open the data in Stata

EXERCISE #1

Dichotomous variables in the General Social Survey are coded using 1 and 2, where 1 typically indicates a 'Yes' or affirmative response, and 2 typically indicates a 'No' or negative response. However, you know that coding variables so that 'No' is 0 makes it much easier to interpret regression results. You've written the following loop to recode four variables recording attitudes about racial differences – racdif1, racdif2, racdif3, and racdif4:

```
foreach var of varlist racdif1 racdif2 racdif3 racdif4 {
      gen `var'_recode =`var'
      recode `var'_recode (1=1)(2=0)(*=.)
      tab `var'_recode `var'
      }
```

After writing the loop, you realize it would be more useful to have a program that could complete this recoding for any related set of variables coded 1/2. Rewrite this loop into a program called -dummy_recode- that accomplishes this task and test it to make sure it works.

EXERCISE #2

You are interested in including interaction effects in your analysis, and wrote the following code to create race dummies and easily interact gender and race:

```
tab race, gen(race_)

forval i=1(1)3 {
      gen femxrace`i'=race_`i'*sex
      }
```

After generating these interactions, you realize that you will want to create interactions for other demographic variables. How can you rewrite this loop as a program that can create interactions for a specified variable with number of other

variables (for example, gender interacted with income, education, and age). Write and test the program with the race variables used in the initial loop.

EXERCISE #3

In the previous worksheet with exercises for debugging loops, you corrected the following loop aimed at verifying that the variable measuring total support for abortion accurately reflected the responses given to questions asking about support for abortion in more specific circumstances:

```
foreach var in abdefect abhlth abnomore abpoor abrape absingle {
      replace abmatch=1 if abany==1 & `var'!=1
      }
```

Rewrite this loop into a program that can be used more generally to check that the response given on a particular variable is consistent with the response(s) given for another related variable. Be sure to test it after rewriting!

Note: Depending on what you would like to compare, some variables may still require recoding before these checks can be completed.

SOLUTIONS

In most cases, there is more than one way to adequately approach troubleshooting. The point of these exercises is to help you get practice thinking about how you can generalize your code – if the solution you try is not exactly the same as that presented, simply ensure you're considering what approach is the most efficient.

EXERCISE #1

The following program will allow you to recode any dummy variable or set of dummy variables from 1/2 to 0/1.

```
capture program drop dummy_recode
program define dummy_recode
      args list
      foreach var of varlist `list' {
            gen `var'_recode=`var'
            recode `var'_recode (1=1)(2=0)(*=.)
```

```
            tab `var'_recode `var', nolabel
      }
end
```

It is important to note that because a single macro, `list' is used to insert a variable list into the code, you will need to include quotations around your variable list when running the program, like so:

```
dummy_recode "libath libcom libhomo libmil libmslm librac"
```

```
. dummy_recode "libath libcom libhomo libmil libmslm librac"
(1,010 missing values generated)
(libath_recode: 2446 changes made)

           | allow anti-religious
libath_rec |    book in library
      ode |         1          2 |     Total
----------+----------------------+----------
        0 |         0      1,436 |     1,436
        1 |       421          0 |       421
----------+----------------------+----------
    Total |       421      1,436 |     1,857
```

The Stata output shows that the program was learned and ran correctly, evidenced by the tables allowing you to verify the recoding.

This program could be written even more easily using local 0:

```
capture program drop dummy_recode
program define dummy_recode
      foreach var of local 0 {
            gen `var'_recode=`var'
            recode `var'_recode (1=1)(2=0)(*=.)
            tab `var'_recode `var', nolabel
      }
end
```

Note in the output below that the variable list no longer requires quotation marks in order to pass the arguments.

```
. dummy_recode libath libcom libhomo libmil libmslm librac
(1,010 missing values generated)
(libath_recode: 2446 changes made)
```

libath_recode	allow anti-religious book in library 1	2	Total
0	0	1,436	1,436
1	421	0	421
Total	421	1,436	1,857

If you already tested the program, great! If not, use the code and examples above to finish and check the program.

You could also rewrite this to be used on any set of dummy variables that share the same coding, and that you would like to recode in an identical way:

```
capture program drop dummy_recode
program define dummy_recode
      args list val1 val2 val3 val4
      foreach var of varlist `list' {
            gen `var'_recode=`var'
            recode `var'_recode
(`val1'=`val2')(`val3'=`val4')(*=.)
            tab `var'_recode `var'
      }
end
```

This approach trades simplicity for flexibility. The previous two examples only require that you specify a variable list - when local 0 is used, the list does not even need to be included in quotes. However, it is only usable in cases where a dummy variable is coded exactly as specified by the program; you could not, for example, apply it to creating a dummy variable indicating a respondent is female, if the initial variable for sex where male=1 and female=2. This is true for any case where the category you want to be ‘on’ in your dummy variable is coded as 2 in the original data.

This program overcomes that rigidity by allowing you to specify both the initial and new coding schemes. However, this added complexity makes it more important to pay attention to your coding, and be sure to include quotations

around your variable list but not the numbers of the coding, as shown in the Stata output below:

```
. dummy_recode "libath libcom libhomo libmil libmslm librac" 1 1 2 0
(1,010 missing values generated)
(libath_recode: 2446 changes made)
```

libath_recode	allow anti-religious book in library		
	1	2	Total
0	**0**	**1,436**	**1,436**
1	**421**	**0**	**421**
Total	**421**	**1,436**	**1,857**

Here, you are specifying that the val1=1 and val2=1 (that is to say, this coding is remaining the same), while you will be recoding an initial value of 2 (val3 in the program) to 0 (val4 in the program).

EXERCISE #2

The following program will create interaction variables for any single specified variable with any number of other variables.

```
capture program drop interact
program define interact
      gettoken first 0 : 0
      foreach rest of local 0 {
      gen `first'_`rest'=`first'*`rest'
      bysort `first': tab `first'_`rest' `rest', missing
      }
end
```

To create the interactions between gender and race, you can then simply specify:

```
interact female race_1 race_2 race_3
```

This will generate variables for all interactions, as show below, where the interaction term female_race_1 is 0 for all respondents who are not female, and 1 for respondents who are female and white.

```
. interact female race_1 race_2 race_3
```

-> female = 0

female_race_1	race==white 0	1	Total
0	325	951	1,276
Total	325	951	1,276

-> female = male

female_race_1	race==white 0	1	Total
0	442	0	442
1	0	1,149	1,149
Total	442	1,149	1,591

EXERCISE #3

You can rewrite the loop into the following program:

```
capture program drop varcheck
program define varcheck
      gettoken first 0 : 0
      foreach rest of local 0 {
            gen `first'_`rest'_check=0
            replace `first'_`rest'_check=1 if `first'!=`rest' ///
            & `first'!=. & `rest'!=.
            bysort `first'_`rest'_check: tab `first' `rest',
missing
      }
      end
```

You may have selected entirely new variables to run this program, or you might have initially checked it using the same abortion attitude variables, like so:

```
varcheck abany abdefect abhlth abnomore abpoor abrape absingle
```

The output Stata generates indicates the program was successful, and allows you to verify the recoding is correct.

```
. varcheck abany abdefect abhlth abnomore abpoor abrape absingle
(660 real changes made)
```

-> abany_abdefect_check = 0

abortion if woman wants for any reason	strong chance of serious defect: yes	no	DK	IAP	NA	Total
yes	**794**	**0**	**0**	**0**	**0**	**794**
no	**0**	**405**	**0**	**0**	**0**	**405**
DK	**0**	**0**	**12**	**0**	**0**	**12**
IAP	**0**	**0**	**0**	**977**	**0**	**977**
NA	**0**	**0**	**0**	**0**	**19**	**19**
Total	**794**	**405**	**12**	**977**	**19**	**2,207**