GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Loops and branching

Stata Self-Learning Course

Loops

- Loops are used to
  - repeatedly perform commands for different subgroups or variables
  - Program iterative procedures (e.g. maximum likelihood estimators)
- Useful to keep dofile or program flexible
- Types of loops
  - forvalues
  - foreach (in/of)
  - while
- Loops are time-intensive → only use when necessary

foreach

- Foreach-loops can repeatedly perform commands over a list of items
- Syntax

  foreach *localname* { in | of listtype} list {

  commands referring to `*localname*'

  }
- The list can be unstructured (in) or structured (of listtype)
- Command creates a local which equals an element of the specified list in every iteration
- Example: list country names and perform regressions separately for each country

foreach – structured lists

- Advantage: allows typical abbreviations and shortcuts
- Types of structured lists
  - local, global
  - varlist (containing existing variables)
  - newlist (containing variables to be created)
  - numlist (containing numbers)
- specify list type in loop header and then list elements that belong to the list
- Local and global are similar to unstructured list except for a minimal speed improvement

forvalues

- Similar to foreach with structured number list
- Requires numbers to have equal distance and to be ordered
- Advantage: speed improvement
- Syntax

  forvalues *localname* = *range* {

  commands referring to `*localname*'

  }
- Examples for *range*:
  - 1(3)20 = 1, 4, 7, 10, 13, 16, 19
  - 1/20 = 1,2,3,4,5,6,7,8,9,10,11,…,20
  - 1 10 to 100 = 1,10,19,28,37,46,…,100 (increment by 10-1)
  - 1 10: 100 = same as 1 10 to 100

Nesting

- Different forvalues and foreach loops can be nested
- Example: creating a list of number and letter combinations

```
forvalues n = 1/3 {
        foreach c in a b c {
                display "`n'`c'"
                }
        }
```

- Result: 1a 1b 1c 2a 2b 2c 3a 3b 3c

While

- Typically used in programming
- More flexible, easier to understand
- Define the stopping condition in the header of the loop
- Specify in the loop how to change the content of the local that is used in the condition
- Syntax example:

```
local n = 1
while `n' <= 10 {
        display `n'
        local ++n
        }
```

- While-loops are executed as long as the condition is true

Branching I

- Specify actions that should be taken in case a certain condition is fulfilled
- Example

```
local obs = r(N)
if `obs' >=100 {
        some command
        }
```

- The command within curly brackets is executed if the condition specified in the header is true
- Do not confuse with if-specifier

```
gen region= "Europe" if country== "Italy"
```

Branching II

- Specify alternative actions that are performed if condition is not fulfilled
- Example

```
local obs = r(N)
if `obs' >=100 {
        command1
        }
else                {
        command2
        }
```

→ Command 1 is executed if the number of observations is at least 100

→ Command 2 is executed if the number of observations is less than 100

Branching III – Nesting

- Example

```
local obs = r(N)
if `obs' >=100 {
        if country[10] == "Italy" {
                command1
                }
        }
```

→ Stata first checks the number of observations

→ Stata then checks whether observation number 10 is from Italy

Note: country is here a variable, not a local