Notas de clase – Taller de Stata[1]

Clase 8 y 9 – Loops

Contenido

1. Loops
   1.1. While
   1.2. Forvalues
   1.3. Foreach
2. Tokenize
3. Levelsof

[Estas notas se diseñaron con base en la versión 16 de *Stata*, conforme el software cambie las notas deben ser actualizadas].

[1] Estas notas están basadas en la guía desarrollada previamente para este curso por Rodrigo Azuero Melo, Nicolás de Roux, Luis Roberto Martínez y Román Andrés Zárate y Santiago Echeverry.

Taller de Stata
Clases 8 y 9 – Loops
Miguel Garzón Ramírez, Cristhian Acosta Pardo
Facultad de Economía, Universidad de los Andes

1. Loops

Dos elementos esenciales en cualquier tipo de programación son los condicionales y los ciclos (loops). Los primeros tienen como objetivo limitar la ejecución de elementos del programa a situaciones que cumplen con una cierta condición lógica. En *Stata*, los comandos utilizados para los condicionales son if y else, los cuales fueron estudiados en notas de clase anteriores. Los ciclos tienen como objetivo ejecutar iterativamente un comando, unas líneas de programación e incluso un programa. Los ciclos utilizan macros de tipo *local* para el control de su ejecución, definen la cantidad de ciclos y los elementos utilizados en la ejecución del programa. En *Stata* hay tres comandos utilizados para generar ciclos. Estos son: while forvalues y foreach. A continuación, explicamos con detalle cada uno de ellos.

1.1. While

El comando while ejecuta una serie de comandos siempre que una expresión lógica sea verdadera. La sintaxis del comando es la siguiente:

```
while [expresión] {
        comandos para ejecutar siempre que la expresión sea verdadera
}
```


Por ejemplo, si queremos ejecutar el programa anterior, pero utilizando la sintaxis del comando *while*, el programa se debe especificar de la siguiente forma:

Ej. 1 ⟶

```
local i=1
while `i'<=15 {
        display `i'
        local i=`i'+1 // local ++i
}
```

A continuación, se describe el proceso que ejecutan las anteriores líneas de programación:

1. *Stata* crea el local *i* que toma, en la primera iteración del ciclo, un valor de 1. Esta es la macro local que se utiliza para controlar la ejecución del ciclo.
2. El comando while específica que lo que aparece entre los corchetes (hasta el renglón 5) se debe ejecutar siempre que el local *i* tome un valor menor o igual a 15.
3. Se muestra en la ventana de resultados el valor que toma el local en cada iteración. Aparece el número 1.

4. Se redefine el valor del local *i*. Ahora, se suma una unidad a este valor para que en la siguiente iteración del ciclo tome el valor de dos[2].

Estos cuatro pasos se repiten hasta que el local *i* toma el valor de 16. En ese momento, la condición lógica no se cumple y *Stata* sale del ciclo.

Note que, si la expresión utilizada para ejecutar el comando while es siempre verdadera, los comandos se repetirán indefinidamente hasta que se detenga o se cierre el programa. Por ejemplo, si omitimos la cuarta línea del Ejemplo 1 el comando se vería de la siguiente forma:

Ej. 2 ⟶
```
local i=1
while `i'<=15 {
        display `i'
}
```

Al ejecutar estas líneas, *Stata* crea el local *i* (que toma un valor de uno) pero no lo redefine. Este local siempre tomará el valor de uno y, por lo tanto, en la ventana de resultados aparecerá un número 1 indefinidamente o hasta que se detenga o se cierre el programa.

Igualmente, si definimos una condición que nunca se cumple (reemplazando, por ejemplo, la primera línea del programa anterior por local *i*=20) el comando nunca será ejecutado.

1.2. Forvalues

La sintaxis del comando forvalues es la siguiente:

```
forvalues [local]=rango {
        (comandos a ejecutar)
}
```

El rango especifica los diferentes valores del local para los que *Stata* ejecuta el comando. En el rango se debe incluir un valor inicial (el valor que toma el local en la primera iteración del ciclo), un valor final (donde finaliza el ciclo) y los intervalos que debe haber entre los distintos valores que toma el local. Las posibles especificaciones del rango se presentan en la Tabla 1.

[2] La línea 4 del ejemplo 1 es equivalente a emplear: local ++*i*, donde el local es usado para incrementar *i*.

Tabla 1. Posibles especificaciones de Rango

Especificación	Desde	Hasta	Intervalos	Ejemplo
k(m)n	k	n	m	1(1)10
k/n	k	n	1	1/10
k s to n	k	n	s-k	1 2 to 10
k s : n	k	n	s-k	1 2 : 10

Nota: El ejemplo inicia en 1 y termina en diez con saltos de 1 en 1

El comando forvalues es el más eficiente y el más fácil de utilizar si necesitamos generar ciclos a través de números consecutivos o que presentan un patrón fácil de identificar. El siguiente ejemplo nos ayudará a entender cómo funciona forvalues. Si queremos ver en la ventana de resultados los números del 1 hasta 15 utilizamos el siguiente programa:

Ej. 3 ⟶
```
forvalues i=1(1)15 {
        display `i'
}
```


Al ejecutar estas tres líneas, *Stata*:

1. Crea un local llamado i con el valor de 1, pues el rango inicia con el valor 1.
2. El comando display ` *i* ' hace que se muestre el valor del local llamado *i*. Dado que el primer valor que toma el local *i* es 1, debe aparecer un 1 en la ventana de resultados de *Stata*.
3. El cierre de los corchetes (}) indica que el ciclo termina y, por lo tanto, el local debe tomar el siguiente valor. Dado que en el rango se especifica que inicia en 1, con saltos de uno en uno hasta 15, el siguiente valor que toma el local *i* es 2.
4. El proceso se repite iterativamente hasta que el local *i* toma el valor de 15. En este punto el programa finaliza.

1.3. Foreach

El comando foreach nos permite hacer ciclos a través de listas de variables, elementos de una macro, o listas de números que no presentan ningún tipo de patrón (por ejemplo, los números primos o los códigos DANE de los departamentos y municipios en Colombia). Este comando

opera con listas creadas al interior de su sintaxis o creadas previamente. Cuando la lista se crea al interior del comando la sintaxis es la siguiente:

```
foreach [nombre del local a generar] in [elemento 1] [elemento 2] … {
        (Comandos a ejecutar)
}
```

Los elementos de la lista pueden ser palabras, cadenas de caracteres de varias palabras delimitados con comillas (“ “) o números separados por espacios. Por ejemplo, el siguiente ciclo permite mostrar en la ventana de resultados cada cadena de caracteres y su longitud:

Ej. 4 ⟶

```
foreach y in "algo 1" "alguno 2" {
        display "`y'"
        display length("`y'")
}
```


Cuando la lista fue creada previamente, la sintaxis del comando es la siguiente:

```
foreach [nombre del local a generar] of [tipo de lista] [nombre de la lista] {
        (Comandos a ejecutar)
}
```

foreach permite hacer ciclos a través de elementos de una lista especificada en un *global* o en un *local*, de una lista de números o de un grupo de variables. Los tipos de listas posibles son:

1. Local
2. Global
3. Varlist
4. Numlist

Para contar los números del uno al diez utilizando el comando foreach la programación es la siguiente:

Ej. 5 ⟶

```
foreach x of numlist 1/10 {
        display `i'
}
```

Para hacer ciclos a través de variables, se puede, por ejemplo, definir un global que incluya a las variables y luego utilizar el comando foreach:

Ej. 6 ⟶

```
foreach var of varlist $variables {
        sum `var'
}
```

Ej. 7 ⟶

```
global variables "var1 var2 var3"
foreach x of global variables {
        sum `x'
}
```


Otras formas de hacer lo mismo, con un local y con dos formas de escribir la lista de variables:

Ej. 8 ⟶

```
local variables "var1 var2 var3"
foreach x of local variables {
        sum `x'
}
```

Ej. 9 ⟶

```
foreach var of varlist var*{
        sum `var'
}
```

Ej. 10 ⟶

```
foreach var of varlist var1-var3{
        sum `var'
}
```

Sin embargo, si utilizamos *in*, en lugar de *of*, este comando generará un error porque *Stata* no reconoce la lista var1-var3 como una lista de variables. Es necesario introducir cada una de las variables separadas por un espacio entre ellas.

2. Tokenize

El comando tokenize permite dividir una serie de palabras (es decir, un *string*) en sus componentes individuales y almacenarlos cada uno como un objeto *local* distinto. La sintaxis del comando tokenize es la siguiente:

tokenize ` "serie de palabras"'

Las palabras serán almacenadas en local de acuerdo con el número que toman en el orden de la serie de palabras en el comando. Por ejemplo, la sintaxis:

tokenize *serie de palabras*

almacena en el local `1' la palabra "serie", en el local `2' la palabra "de" y en el local `3' la palabra "palabras". Ahora, si se desea almacenar más de una palabra en un local, es necesario separarlos por comillas dobles de la siguiente forma:

Ej. 11 ⟶ tokenize ` ""Carlos Valderrama" "Faustino Asprilla" "Freddy Rincón"" '
display "`2' juega con `1' pero no con `3 ' "

Si omitiéramos el paso de las comillas, cada palabra sería almacenada como un local. El local `1' almacenaría la palabra "Carlos", el local `2' haría lo propio con "Alberto", etc.

3. Levelsof

El comando levelsof permite guardar una lista de los posibles valores que toma alguna variable en un objeto *local*. El *local* creado es una lista de todos los valores de la variable separados con un espacio entre ellos. El comando levelsof es útil cuando una variable presenta una serie de valores que no presenta ningún patrón (nuevamente, un buen ejemplo es el de los códigos municipales en Colombia) pero también puede utilizarse para variables *string*. La sintaxis del comando es:

levelsof varname. Donde *varname* hace referencia a la variable de interés. En caso de que se desee almacenar los resultados en un local, la sintaxis es la siguiente:

levelsof *varname*, local(localname)

Lo anterior permite guardar todos los valores de una variable (varname) en el *local* (localname). Esta opción no está disponible para los global. Así mismo los valores que toma dicha variable son guardados en r(levels). Es muy común utilizar este comando para crear objetos *local* con la lista de valores sobre los cuales vamos a iterar con *loops*.