Title

stata.com

if — if programming command

Description Syntax Remarks and examples Reference Also see

Description

The `if` command (not to be confused with the `if` qualifier; see [U] **11.1.3 if exp**) evaluates *exp*. If the result is *true* (nonzero), the commands inside the braces are executed. If the result is *false* (zero), those statements are ignored, and the statement (or statements if enclosed in braces) following the `else` is executed.

Syntax

`if` *exp* `{`
    multiple_commands
`}`

or `if` *exp single_command*

which, in either case, may be followed by

`else {`
    multiple_commands
`}`

or `else` *single_command*

If you put braces following the `if` or `else`,

1. the open brace must appear on the same line as the `if` or `else`;
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.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Introduction

The `if` command is intended for use inside programs and do-files; see [U] **18.3.4 Macros and expressions** for examples of its use.

▷ Example 1

Do not confuse the `if` command with the `if` qualifier. Typing `if (age>21) summarize age` will summarize *all* the observations on `age` if the first observation on `age` is greater than 21. Otherwise, it will do nothing. Typing `summarize age if age>21`, on the other hand, summarizes all the observations on `age` that are greater than 21.

◁

▷ Example 2

`if` is typically used in do-files and programs. For instance, let's write a program to calculate the Tukey (1977, 90–91) "power" function of a variable, x:

```
. program power
        if `2'>0 {
             generate z=`1'^`2'
             label variable z "`1'^`2'"
        }
        else if `2'==0 {
             generate z=log(`1')
             label variable z "log(`1')"
        }
        else {
              generate z=-(`1'^(`2'))
              label variable z "-`1'^(`2')"
        }
        end
```

This program takes two arguments. The first argument is the name of an existing variable, x. The second argument is a number, which we will call n. The program creates the new variable `z`. If $n > 0$, `z` is x^n; if $n = 0$, `z` is $\log x$; and if $n < 0$, `z` is $-x^n$. No matter which path the program follows through the code, it labels the variable appropriately:

```
. power age 2
. describe z
              storage   display    value
variable name   type    format     label      variable label
------------------------------------------------------------------------------
z               float   %9.0g                 age^2
```

◁

❑ Technical note

If the expression refers to any variables, their values in the first observation are used unless explicit subscripts are specified.

❑

Avoid single-line if and else with ++ and -- macro expansion

Do not use the single-line forms of `if` and `else`—do not omit the braces—when the action includes the `'++'` or `'--'` macro-expansion operators. For instance, do not code

```
if (...) somecommand `++i'
```

Code instead,

```
if (...) {
        somecommand `++i'
}
```

In the first example, `i` will be incremented regardless of whether the condition is true or false because macro expansion occurs before the line is interpreted. In the second example, if the condition is false, the line inside the braces will not be macro expanded and so `i` will not be incremented.

The same applies to the `else` statement; do not code

```
else somecommand `++i'
```

Code instead,

```
else {
        somecommand `++i'
}
```

❑ Technical note

What was just said also applies to macro-induced execution of class programs that have side effects. Consider

```
if (...) somecommand `.clspgm.getnext'
```

Class-member program `.getnext` would execute regardless of whether the condition were true or false. Here code

```
if (...) {
        somecommand `.clspgm.getnext'
}
```

Understand that the problem arises only when macro substitution causes the invocation of the class program. There would be nothing wrong with coding

```
if (...) `.clspgm.getnext'
```

❑

Reference

Tukey, J. W. 1977. *Exploratory Data Analysis.* Reading, MA: Addison–Wesley.

Also see

[P] **continue** — Break out of loops

[P] **foreach** — Loop over items

[P] **forvalues** — Loop over consecutive values

[P] **while** — Looping

[U] **18 Programming Stata**