CHAPTER 12

Flow Control

Flow control statements are used to repeat a series of tasks a number of times or to direct flow based on a logical object. For persons who came into programming in the age of FORTRAN and BASIC, using loops is very comfortable. In R, the better choice, if possible, is to use arrays and index selection instead of looping. Using indices is much faster than looping.

That said, the control statements are **if**, **if/else**, **while**, **for**, and **repeat**. They are sometimes necessary and often useful. In this chapter, we give syntax for the flow control statements. We give examples of the use of flow control in Chapter 13.

Brackets “{}” and the Semicolon “;”

Curly brackets are used to enclose sections of code. Brackets can be used with **if**, **else**, **while**, **for**, and **repeat** flow control statements to delineate the section of code on which the control statement is to operate, both within functions and at the R console.

Brackets can also be used without an accompanying flow control statement, directly at the R console. Starting with an opening bracket, code statements can be entered one line at a time. The statements do not execute until the closing bracket is entered.

The semicolon is used to include more than one statement on one line. A statement is not evaluated until the statement before it has finished executing. If the first statement is a flow control statement followed

M. Tollefson, *R Quick Syntax Reference*, https://doi.org/10.1007/978-1-4842-4405-0_12

by a single statement of code, the control flow must finish before the second statement executes. However, if the two—or more—statements are enclosed in an opening and a closing bracket after a flow control statement, all of the statements within the brackets are executed together based on the flow control statement.

The "if" and "if/else" Control Statements

The **if** control statement takes a logical object and executes code if the object is true. If the object is not true, then, optionally, different code given by an **else** statement executes.

The logical object must be an object that can be coerced to logical. If the logical object is of length greater than one, only the first element of the object is used.

The **if** statement can take the following forms:

```
if ('logical object') 'single code statement'
```

```
if ('logical object') 'single code statement';'single code
statement'
```

```
if ('logical object') {'more than one code statement separated
by semicolons'}
```

```
if ('logical object') {
'lines of code statements'
}
```

These four forms are not exhaustive of the possible forms. In the second form, the second statement will execute even if the logical object is false since the two statements are not enclosed in brackets.

If the **logical object** is false, then the option exists to have R execute different code by using an **else** statement. For the two control statements **if** and **else**, two examples of form follow:

```
if ('logical object') 'single code statement' else 'single code
statement'
```

```
if ('logical object') {
'lines of the code statements'
}
else {
'lines of the code statements'
}
```

Again, the two forms are not exhaustive. If no **else** control statement is present and **logical object** is false, then the code statements associated with the **if** statement are skipped.

The “while” Control Statement

The **while** control statement executes a block of code while a logical condition is true. Again, the logical object must be an object that can be coerced to logical. If the logical object is of length greater than one, only the first element of the object is used.

The control statement can take the following forms:

```
while ('logical object') 'single code statement'
```

```
while ('logical object') 'single code statement'; 'single code
statement'
```

```
while ('logical object') {'multiple code statements separated
by semicolons'}
```

```
while ('logical object') {
'lines of code statements'
}
```

Again, the forms shown are not exhaustive of the possible forms. Note that for the second form, the second statement does not execute until the while loop is ended since the two statements are not in brackets.

The "for" Control Statement

The **for** control statement instructs R to loop through a section of code for a set number of times. There are a number of ways that the looping can be done based on the looping criteria.

The looping criteria can be quite flexible. The simplest form is

```
for (i in 1:n)
```

where **i** is an object that indexes from **1** to **n** and where **n** is an integer.

In general, the syntax of the flow control statement for **for** loops is

```
for ('indexing variable' in 'vector object')
```

where **indexing variable** is a variable whose value changes at each iteration of the loop and **vector object** contains the values that **indexing value** takes. The vector object can be any object that can be coerced to a vector, including objects of mode `list` and `expression`.

The object `indexing variable` will take on the values of **vector object** sequentially. Usually, the indexing variable is used in the code statements executed by the **for** loop.

Note that if the vector object is created using the function `seq()` within the **for** statement and the `seq()` argument **along.with**—which can be abbreviated **along**—is used, `seq()` gives the indices of the elements of **along.with** rather than the values of the object.

Some forms of a **for** loop are the following:

```
for ('looping criteria') 'single code statement'
```

```
for ('looping criteria') 'single code statement'; 'single code
statement'
```

```
for ('looping criteria') {'multiple code statements separated
by semicolons'}
```

```
for ('looping criteria') {
'lines of code statements'
}
```

Again, the four forms are not exhaustive of the possible forms. In the second form, the code after the semicolon does not execute until after the **for** loop is finished since the two statements are not in brackets.

According to the CRAN help page for flow control, the value of the indexing variable can be changed in the code statements referenced by **for** but, at the start of the next loop, reverts to the next indexed value of the variable. At the end of the looping, the value of **indexing variable** is the final value of the indexing variable in the loop.

The “repeat” Control Statement

The **repeat** flow control statement repeats a section of code until a stopping point is reached. The stopping point must be programmed into the section of code. Unlike **while**, **repeat** does not have a logical object as part of the control statement, and, unlike **for**, no looping index is part of the control statement. Following are two forms for repeat:

```
repeat {'some code statements separated by semicolons'}
```

```
repeat {
'lines of code statements'
}
```

Again, the two are not exhaustive. Infinite loops are possible with **repeat**, so use caution.

The Statements "break" and "next"

The statements **break** and **next** are used for flow control within those sections of code controlled by one of the flow controllers.

The statement **break** tells R to leave a **for**, **while**, or **repeat** loop or an **if** section and go to the first statement after the loop or section.

The statement **next** tells R to stop executing the code statements in a **for**, **while**, or **repeat** loop and start again at the beginning of the loop—with the value of the indexing variable, if there is one, taking on the next value of the variable.

Nesting

Any of the flow control statements can be nested within other flow control sections of code. For the sake of clarity and to prevent subtle bugs, use brackets at all levels when nesting flow control sections within other flow control sections.

Most of the information presented here on flow control is from the CRAN help page on controlling flow, which can be found by entering **?"if"** at the R prompt or by using the "Help" tab in R Studio.