

Control Flow

Description

These are the basic control-flow constructs of the R language. They function in much the same way as control statements in any Algol-like language. They are all [reserved](#) words.

Usage

```
if(cond) expr
if(cond) cons.expr else alt.expr

for(var in seq) expr
while(cond) expr
repeat expr
break
next
```

Arguments

<code>cond</code>	A length-one logical vector that is not <code>NA</code> . Conditions of length greater than one are accepted with a warning, but only the first element is used. Other types are coerced to logical if possible, ignoring any class.
<code>var</code>	A syntactical name for a variable.
<code>seq</code>	An expression evaluating to a vector (including a list and an expression) or to a pairlist or <code>NULL</code> .
<code>expr</code> , <code>cons.expr</code> , <code>alt.expr</code>	An <i>expression</i> in a formal sense. This is either a simple expression or a so called <i>compound expression</i> , usually of the form <code>{ expr1 ; expr2 }</code> .

Details

`break` breaks out of a `for`, `while` or `repeat` loop; control is transferred to the first statement outside the inner-most loop. `next` halts the processing of the current iteration and advances the looping index. Both `break` and `next` apply only to the innermost of nested loops.

Note that it is a common mistake to forget to put braces (`{ . . }`) around your statements, e.g., after `if (..)` or `for(....)`. In particular, you should not have a newline between `}` and `else` to avoid a syntax error in entering a `if ... else` construct at the keyboard or via `source`. For that reason, one (somewhat extreme) attitude of defensive programming is to always use braces, e.g., for `if` clauses.

The index `seq` in a `for` loop is evaluated at the start of the loop; changing it subsequently does not affect the loop. The variable `var` has the same type as `seq`, and is read-only: assigning to it does not alter `seq`. If `seq` is a factor (which is not strictly allowed) then its internal codes are used: the effect is that of [as.integer](#) not [as.vector](#).

Value

`if` returns the value of the expression evaluated, or `NULL` if none was (which may happen if there is no else).
`for`, `while` and `repeat` return the value of the last expression evaluated (or `NULL` if none was), invisibly.
`for` sets `var` to the last used element of `seq`, or to `NULL` if it was of length zero.
`break` and `next` have value `NULL`, although it would be strange to look for a return value.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

[Syntax](#) for the basic R syntax and operators, [Paren](#) for parentheses and braces; further, [ifelse](#), [switch](#).

Examples

```
for(i in 1:5) print(1:i)
for(n in c(2,5,10,20,50)) {
  x <- stats::rnorm(n)
  cat(n, ":", sum(x^2), "\n")
}
```

[Package *base* version 2.7.1 [Index](#)]