Video 10: Conditional Statements

Stats 102A

Miles Chen, PhD

Related Reading

- Introduction to Scientific Programming and Simulation Using R
  - Chapter 3 - Basic Programming
  - Must be on UCLA Network or connected to VPN to access:
  - https://www.taylorfrancis.com/books/mono/10.1201/9781420068740/introduction-scientific-programming-simulation-using-owen-jones-robert-maillardet-andrew-robinson
- Advanced R
  - Chapter 5: Control Flow
  - https://adv-r.hadley.nz/control-flow.html

Operators that produce logical values

Vectorized Logical Operators

R has the following logical operators. Use of a logical operator will coerce non-logical types to logical type. (e.g. 0 becomes FALSE, all other numeric values become TRUE.)

- or: `x | y`
- and: `x & y`
- not: `!x`
- exclusive or: `xor(x,y)`

The above logical operators are vectorized and will perform element-wise comparison as well as recycling. The vectorized operations will return a vector of logical values.

Non-Vectorized Logical operators

Non-Vectorized operators will return only one logical value.The non-vectorized operators are used in if clauses.

- non-vector or: `x || y`
- non-vector and: `x && y`

The 'non-vector' `||` and `&&` accepts only logical vectors of length 1.

If x has length greater than 1 it will return an error.

If y has length greater than 1 and is evaluated, it will return an error. If y has length greater than 1 but is not evaluated, R will not error.

Older R versions (prior to 4.3) evaluated left to right examining only the first element of each vector. If you have an older version of R, you should update.

See `?base::Logic`

Vectorized vs Non-Vectorized

```
x <- c(TRUE,  TRUE, FALSE, FALSE)
y <- c(FALSE, TRUE, FALSE,  TRUE)
x | y
```

```
[1]  TRUE  TRUE FALSE  TRUE
```

```
x & y
```

```
[1] FALSE  TRUE FALSE FALSE
```

```
x || y  # x length greater than 1 leads to error
```

```
Error in x || y: 'length = 4' in coercion to 'logical(1)'
```

```
FALSE && y # FALSE AND blank is always FALSE. y isn't evaluated. no error
```

```
[1] FALSE
```

```
TRUE || y # TRUE OR blank is always TRUE. y isn't evaluated. no error
```

```
[1] TRUE
```

```
FALSE || y # returns error because y has to be evaluated
```

```
Error in FALSE || y: 'length = 4' in coercion to 'logical(1)'
```

Logical operators with NA

What is `TRUE | NA`?

What is `TRUE & NA`?

What is `FALSE | NA`?

What is `FALSE & NA`?

Solutions:

```
1 TRUE | NA
```

```
[1] TRUE
```

```
1 TRUE & NA
```

```
[1] NA
```

```
1 FALSE | NA
```

```
[1] NA
```

```
1 FALSE & NA
```

```
[1] FALSE
```

Exclusive Or

`xor()` indicates elementwise exclusive OR (True if x *or* y is true, but not if both are true). That is, `xor(TRUE, TRUE)` is `FALSE`

```
x <- c(TRUE,  TRUE, FALSE, FALSE,   NA,    NA,  NA)
y <- c(FALSE, TRUE, FALSE,  TRUE, TRUE, FALSE,  NA)
xor(x, y)
```

```
[1]  TRUE FALSE FALSE  TRUE    NA    NA    NA
```

any() and all()

any() and all() are generalizations of OR and AND for more than 2 values.

```
1 u <- c( TRUE,  TRUE,  TRUE)
2 c(any(u), all(u))
```

```
[1] TRUE TRUE
```

```
1 v <- c( TRUE,  TRUE, FALSE)
2 c(any(v), all(v))
```

```
[1]  TRUE FALSE
```

```
1 w <- c(FALSE, FALSE, FALSE)
2 c(any(w), all(w))
```

```
[1] FALSE FALSE
```

Question

If we add a 0-length vector (like `logical(0)` or `NULL`) to a vector of logical values `v`, it should not affect the result of `any()` or `all()` applied to `v`.

That is:

- `any(logical(0), v)` should produce the same result as `any(v)`
- `all(logical(0), v)` should produce the same result as `all(v)`

With these rules in mind:

- What does `any(logical(0))` return?
- What does `all(logical(0))` return?

`any(logical(0))` is FALSE

```
1 x <- logical(0)
2 any(x)
```

```
[1] FALSE
```

```
1 u <- c( TRUE,  TRUE,  TRUE)
2 c(any(u), any(x, u))
```

```
[1] TRUE TRUE
```

```
1 v <- c( TRUE,  TRUE, FALSE)
2 c(any(v), any(x, v))
```

```
[1] TRUE TRUE
```

```
1 w <- c(FALSE, FALSE, FALSE) # if any(logical(0)) were TRUE, this would change
2 c(any(w), any(x, w))
```

```
[1] FALSE FALSE
```

`all(logical(0))` is TRUE

```
1 x <- logical(0)
2 all(x)
```

```
[1] TRUE
```

```
1 u <- c( TRUE,  TRUE,  TRUE) # if all(logical(0)) were FALSE, this would change
2 c(all(u), all(x, u))
```

```
[1] TRUE TRUE
```

```
1 v <- c( TRUE,  TRUE, FALSE)
2 c(all(v), all(x, v))
```

```
[1] FALSE FALSE
```

```
1 w <- c(FALSE, FALSE, FALSE)
2 c(all(w), all(x, w))
```

```
[1] FALSE FALSE
```

Comparison operators

Comparison operators produce logical values. They are vectorized and perform recycling.

- `x == y`
- `x != y`
- `x < y`
- `x > y`
- `x <= y`
- `x >= y`
- `x %in% y` (vectorized for `x` only)

see `help(Comparison)`

Comparison operators

For character strings, comparison operations are based on alphabetical order, with the following general arrangement:

```
symbols < "0" < .. < "9" < "a" < "A" < "b" ... < "Z"
```

```
1 "10" < "2" # with characters, 10 is alphabetized before 2
```

```
[1] TRUE
```

```
1 10 < 2 # of course, numerically 10 is not less than 2
```

```
[1] FALSE
```

Just for reference, R arranges the ASCII symbols as follows:

```
"-"  "!"  "#"  "$"  "%"  "&"  "("  ")"  "*"  ","  "."  "/"  ":"  ";"
"?"  "@"  "["  "\\" "]"  "^"  "_"  "(back-tick)"  "{"  "|"  "}"  "~"
"+"  "<"  "="  ">"
```

`is` functions

The `is.____()` family of functions return logical values that can be used in conditional statements.

- `is.na()`
- `is.null()`
- `is.atomic()`
- `is.logical()`
- `is.integer()`
- `is.double()`
- `is.numeric()`
- `is.character()`
- `is.list()`
- `is.matrix()`
- `is.array()`
- `is.data.frame()`
- `is.factor()`
- `is.function()`

Condtional statements

Conditional statement - `if`

Conditional execution of code blocks is achieved via `if()` statements.

`if(cond)` The condition in an `if()` statement must be a **length-one logical vector that is not NA**.

Conditions of length greater than one result in an error. (Versions of R prior to 4.2.0 accepted longer vectors with a warning, but only the first element is used.) Other types are coerced to logical if possible.

If the condition results in `NA`, you will get an error:

If the condition is length 0, you will get an error:

Curly braces `{ }` are used to group the expressions that will run when the condition inside the `if` statement is `TRUE`. If there is only one expression to execute, the curly braces are optional.

Using `if()`

```
x <- c(1, 3)
1 %in% x
```

```
[1] TRUE
```

```
if (1 %in% x) {
    print("Hello!")
}
```

```
[1] "Hello!"
```

`if()` produces error with logical vector length > 1

```r
x <- c(1, 3)
x <= 2
```

```
[1]  TRUE FALSE
```

```r
if (x <= 2){
  # we get an error because the length of the logical vector > 1
    print("Hello again!")
}
```

```
Error in if (x <= 2) {: the condition has length > 1
```

Using logical vector length > 1 with `any()` or `all()`

```
x <- c(1, 3)
any(x >= 2)  # if your logical vector has length > 1, you might use any/all
```

```
[1] TRUE
```

```
if (any(x >= 2)) {
    print("Can you hear me now?")
}
```

```
[1] "Can you hear me now?"
```

`if()` produces error for `NA`

```
1 x <- 1:3
2 if (x[5] >= 2){
3     print("Will this work?")
4 }
```

```
Error in if (x[5] >= 2) {: missing value where TRUE/FALSE needed
```

`missing value where TRUE/FALSE needed` is a very common error that you will run into. You'll need to go through your code and see why something inside your if statement is producing an NA value.

`if()` produces error for `logical(0)`

```
1 x <- 1:3
2 which(x == 4)
```

```
integer(0)
```

```
1 if (which(x == 4) == 4){
2     print("Will this work?")
3 }
```

```
Error in if (which(x == 4) == 4) {: argument is of length zero
```

Nesting Conditionals - `if`, `else if`, and `else`

If you want to use an `else` statements, there must be a preceding `if` statement and you must use curly braces. The conditional inside an `else if` statement will only be evaluated if the starting `if` statement is `FALSE`. Make sure you put the `else` on the same line as the closing curly brace of the `if` statement, otherwise R will believe the `if` statement is complete.

```
x <- 3
if (x < 0) {
   print("Negative")
} else if (x > 0) {
   print("Positive")
} else {
   print("Zero")
}
```

```
[1] "Positive"
```

Nested conditionals

```
x <- 0
if (x < 0) {
   print("Negative")
} else if (x > 0) {
   print("Positive")
} else {
   print("Zero")
}
```

```
[1] "Zero"
```

Nested conditionals

```
x <- 3
if (x > 0) {
   print("Positive")
} else if (x > 1) {   # will not be evaluated because the first if is TRUE
   print("Bigger than 1")
} else {
   print("Not Positive")
}
```

```
[1] "Positive"
```

`if()` is not vectorized, `ifelse()` is vectorized

`if()` requires a logical vector of length-one. It is not vectorized. Functions that use an `if()` statement cannot be given a vector of values.

The function `ifelse` is vectorized. It requires three arguments:

- a condition to test
- the result if the condition is true
- the result if the condition is false

```
if(x == 5) {
    n <- "yes"
} else {
    n <- "no"
}
```

is reduced to:

```
n <- ifelse(x == 5, "yes", "no")
```