R for life sciences. Chapter 2: Operations in R

2019-07-10

Cite as: Alfonso Garmendia (2019) R for life sciences. Chapter 2: Operations in R. <http://personales.upv.es/algarsal/Documentation/Garmendia-R-Tutorial-02_Operations.html>

available in [PDF](http://personales.upv.es/algarsal/Documentation/Garmendia-R-Tutorial-02_Operations.pdf) and [EPUB](http://personales.upv.es/algarsal/Documentation/Garmendia-R-Tutorial-02_Operations.epub)



This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](http://creativecommons.org/licenses/by-nc-sa/4.0/).

Written in [Rmarkdown](http://rmarkdown.rstudio.com/), using [Rstudio](https://www.rstudio.com/) and [pandoc](http://pandoc.org/).

# Operations

## Sintax and operators

We have already seen in chapter 1 several operators to address or assign data into and object. We can see a table of operators by precedence order using **?Syntax**

### Adressing operators

$ , @ :

Address to a component into an objects, by names

[ , [[ :

Indexing components into an object

? :

Help

<- , <<- , = :

Assignment, right to left. The use of = for assignment is not advisable.

~ :

As in formulae (write it with Alt-4, or Alt-Ñ in Spanish keyboard)

### Arithmetic operators

by order of precedence. If the objects are not numeric, they will be coerced into numeric, if possible.

^ :

Exponential

%/% , %% :

Divisor and remainder for a division.

\* , / :

Multiply and divide

+ , - :

Addition and subtraction

### Comparison operators

Output will be a Logical or a list of logicals (True - False)

< , > , <= , >= :

Leaser, greater, leaser or equal, greater or equal

== , != :

Equal and different

%in% :

Indicates if there is a match

### Logical operators

! :

Logical NOT

& , && :

Logical AND

| , || :

Logical OR

Let’s see some examples with operators:

#=====================================  
# Some examples with operators  
#=====================================  
x <- -1:12 # Create vector  
x # See vector  
x + 1 # sum 1  
2 \* x + 3 # same that (2\*x)+3  
2 \* (x + 3)  
x %% 2 #-- is periodic  
x %% 5 #-- is periodic  
x %/% 5  
x / 5  
###### Logical AND ("&&") has higher precedence than OR ("||"): ######  
TRUE || TRUE && FALSE # is the same as  
TRUE || (TRUE && FALSE) # and different from  
(TRUE || TRUE) && FALSE  
  
#### Special operators have higher precedence than "!" (logical NOT).  
# You can use this for %in% :  
1:10 %in% c(2, 3, 5, 7)  
!1:10 %in% c(2, 3, 5, 7) # same as !(1:10 %in% c(2, 3, 5, 7))  
!(1:10 %in% c(2, 3, 5, 7))  
# but it is strongly advise to use the "!( ... )" form in this case!

### Some Arithmetical commands

There are too many commands in R to list them all, but some of them are frequently used for calculations.

This commands return a number:

#=====================================  
# Commands that return a number  
#=====================================  
sum(x) # sum of the elements of x  
prod(x) # product of the elements of x  
max(x) # maximum of the elements of x  
min(x) # minimum of the elements of x  
which.max(x) # index of the maximum of the elements of x  
which.min(x) # index of the minimum of the elements of x  
which(x == 2) # index of the first element that fits  
length(x) # number of elements in x  
mean(x) # mean of the elements in x  
median(x) # median of the elements in x  
var(x) # variance of the elements in x  
sd(x) # standard deviation of the elements in x  
# Sometimes is useful to round the result, for example:  
round(sd(x), 2) # round(x, n) rounds the elements of x to n decimals

An these operations can modify either a number or all the numbers in a vector or a matrix:

#=====================================  
# Commands to modify vectors  
#=====================================  
log(x, 2) # logarithm in base 2 ; log(x, base)  
sqrt(x) # Square root of x. (NaN: Not a Number)  
# match (x, y) returns a vector with the elements of x which are in y  
match(x, 2)   
# na.omit(x) # supresses the observations with missing data   
na.omit(log(x, 2)) # (NA: Not Available)

And of course, it is possible to make combinations of different commands. For example to calculate the **standard error (SE)** of x, which is:

being the standard deviation of x.

sd(x) / sqrt( length(x) ) # Standard error of x  
# or even  
round( sd(x) / sqrt( length(x) ), 2) # Two decimals rounded SE

## Condicionals and recursive commands

The most used ones are **if()** and **for()**. Other control flow commands are while() and repeat(). if() can be used either with of without else. They function in much the same way as control statements in any Algol-like language. Also important the expressions ***break*** and ***next*** to control the flow.

Braces are not necessary in the same line, but is advisable to use them always because is a frequent source of errors.

Examples:

x <- -1:12 # Create vector  
for (i in 1:5) print(1:i) # Print numbers

## [1] 1  
## [1] 1 2  
## [1] 1 2 3  
## [1] 1 2 3 4  
## [1] 1 2 3 4 5

for (i in 1:5) { print(1:i) } # Same than before

## [1] 1  
## [1] 1 2  
## [1] 1 2 3  
## [1] 1 2 3 4  
## [1] 1 2 3 4 5

############## example of for with 2^n and print() and paste()  
for (i in x) {   
 y <- 2^i  
 z <- paste(i, ": ", y, sep = "") # Design the output  
 print(z) # Output  
}

## [1] "-1: 0.5"  
## [1] "0: 1"  
## [1] "1: 2"  
## [1] "2: 4"  
## [1] "3: 8"  
## [1] "4: 16"  
## [1] "5: 32"  
## [1] "6: 64"  
## [1] "7: 128"  
## [1] "8: 256"  
## [1] "9: 512"  
## [1] "10: 1024"  
## [1] "11: 2048"  
## [1] "12: 4096"

############## Same example with if() to alineate variables  
for (i in x) {   
 y <- 2^i # Result  
 if (i >= 10) {  
 z <- paste(i, ": ", y, sep = "") # Output  
 } else {  
 z <- paste(" ", i, ": ", y, sep = "")   
 } # add and space  
 print(z) # Output  
}

## [1] " -1: 0.5"  
## [1] " 0: 1"  
## [1] " 1: 2"  
## [1] " 2: 4"  
## [1] " 3: 8"  
## [1] " 4: 16"  
## [1] " 5: 32"  
## [1] " 6: 64"  
## [1] " 7: 128"  
## [1] " 8: 256"  
## [1] " 9: 512"  
## [1] "10: 1024"  
## [1] "11: 2048"  
## [1] "12: 4096"

############## Same example with several if()  
for (i in x) {   
 y <- 2^i # Result  
 pa0 <- paste(i, ": ", y, sep = "") # Design output  
 pa1 <- paste(" ", i, ": ", y, sep = "") # add and space  
 if (i < 0) print(pa0) # Output  
 if (0 <= i && i < 10) print(pa1)  
 if (i >= 10) print(pa0)   
}

## [1] "-1: 0.5"  
## [1] " 0: 1"  
## [1] " 1: 2"  
## [1] " 2: 4"  
## [1] " 3: 8"  
## [1] " 4: 16"  
## [1] " 5: 32"  
## [1] " 6: 64"  
## [1] " 7: 128"  
## [1] " 8: 256"  
## [1] " 9: 512"  
## [1] "10: 1024"  
## [1] "11: 2048"  
## [1] "12: 4096"

# Exercises

1. Open the data frame in **iris {datasets}**. ¿How many variables and observations are there in **iris**?
2. Make a vector with the species names
3. Make a vector with the name of all quantitative variables
4. Make a data frame with the combination of the two previous vectors like this:
5. Make a data frame with 7 variables and 12 rows including the two variables from exercise 4, and also the following statistics:

* the following variables: Species, Variable, Mean, Standard\_error, Median, Minimum and Maximum.
* Use the commands seen in this and previous chapter to do the code the shortest and neatest possible. Also comment each step to know what are you doing.