

# PARTE XI: Introduzione all'ambiente R - Panoramica

# Getting help

Most R functions have online documentation.

**help(topic)** documentation on `topic`

**?topic** id.

**help.search("topic")** search the help system

**apropos("topic")** the names of all objects in the search list matching the regular expression "topic"

**help.start()** start the HTML version of help

**str(a)** display the internal \*str\*ucture of an R object

**summary(a)** gives a "summary" of `a`, usually a statistical summary but it is *generic* meaning it has different operations for different classes of `a`

**ls()** show objects in the search path; specify `pat="pat"` to search on a pattern

**ls.str()** `str()` for each variable in the search path

**dir()** show files in the current directory

**methods(a)** shows S3 methods of `a`

**methods(class=class(a))** lists all the methods to handle objects of class `a`

## Input and output

**load()** load the datasets written with `save`

**data(x)** loads specified data sets

**library(x)** load add-on packages

**read.table(file)** reads a file in table format and creates a data frame from it; the default separator `sep=""` is any whitespace; use `header=TRUE` to read the first line as a header of column names; use `as.is=TRUE` to prevent character vectors from being converted to factors; use `comment.char=""` to prevent `"#"` from being interpreted as a comment; use `skip=n` to skip `n` lines before reading data; see the help for options on row naming, NA treatment, and others

**read.csv("filename", header=TRUE)** id. but with defaults set for reading comma-delimited files

**read.delim("filename", header=TRUE)** id. but with defaults set for reading tab-delimited files

**read.fwf(file, widths, header=FALSE, sep=" ", as.is=FALSE)** read a table of *fixed width formatted* data into a 'data.frame'; `widths` is an integer vector, giving the widths of the fixed-width fields

**save(file, ...)** saves the specified objects (...) in the XDR platform-independent binary format

**save.image(file)** saves all objects

**cat(..., file="", sep=" ")** prints the arguments after coercing to character; `sep` is the character separator between arguments

**print(a, ...)** prints its arguments; generic, meaning it can have different methods for different objects

**format(x, ...)** format an R object for pretty printing

**write.table(x, file="", row.names=TRUE, col.names=TRUE, sep=" ")** prints `x` after converting to a data frame; if `quote` is `TRUE`,

character or factor columns are surrounded by quotes ("); `sep` is the field separator; `eol` is the end-of-line separator; `na` is the string for missing values; use `col.names=NA` to add a blank column header to get the column headers aligned correctly for spreadsheet input

**sink(file)** output to `file`, until `sink()`

Most of the I/O functions have a `file` argument. This can often be a character string naming a file or a connection. `file=""` means the standard input or output. Connections can include files, pipes, zipped files, and R variables.

On windows, the file connection can also be used with `description = "clipboard"`. To read a table copied from Excel, use

```
x <- read.delim("clipboard")
```

To write a table to the clipboard for Excel, use

```
write.table(x, "clipboard", sep="\t", col.names=NA)
```

For database interaction, see packages `RODBC`, `DBI`, `RMySQL`, `RPgSQL`, and `ROracle`. See packages `XML`, `hdf5`, `netCDF` for reading other file formats.

## Data creation

**c(...)** generic function to combine arguments with the default forming a vector; with `recursive=TRUE` descends through lists combining all elements into one vector

**from:to** generates a sequence; “:” has operator priority; `1:4 + 1` is “2,3,4,5”

**seq(from,to)** generates a sequence `by=` specifies increment; `length=` specifies desired length

**seq(along=x)** generates `1, 2, ..., length(along)`; useful for for loops

**rep(x,times)** replicate `x` times; use `each=` to repeat “each” element of `x` each `times`; `rep(c(1,2,3),2)` is `1 2 3 1 2 3`; `rep(c(1,2,3),each=2)` is `1 1 2 2 3 3`

**data.frame(...)** create a data frame of the named or unnamed arguments; `data.frame(v=1:4, ch=c("a", "B", "c", "d"), n=10)`; shorter vectors are recycled to the length of the longest

**list(...)** create a list of the named or unnamed arguments; `list(a=c(1,2), b="hi", c=3i)`;

**array(x,dim=)** array with data `x`; specify dimensions like `dim=c(3,4,2)`; elements of `x` recycle if `x` is not long enough

**matrix(x,nrow=,ncol=)** matrix; elements of `x` recycle

**factor(x,levels=)** encodes a vector `x` as a factor

**gl(n,k,length=n\*k,labels=1:n)** generate levels (factors) by specifying the pattern of their levels; `k` is the number of levels, and `n` is the number of replications

**expand.grid()** a data frame from all combinations of the supplied vectors or factors

**rbind(...)** combine arguments by rows for matrices, data frames, and others

**cbind(...)** id. by columns

# Slicing and extracting data

## Indexing vectors

`x[n]`

$n^{th}$  element

`x[-n]`

all *but* the  $n^{th}$  element

`x[1:n]`

first  $n$  elements

`x[-(1:n)]`

elements from  $n+1$  to the end

`x[c(1, 4, 2)]`

specific elements

`x["name"]`

element named "name"

`x[x > 3]`

all elements greater than 3

`x[x > 3 & x < 5]`

all elements between 3 and 5

`x[x %in% c("a", "and", "the")]`

elements in the given set

## Indexing lists

`x[n]` list with elements `n`  
`x[[n]]`  $n^{\text{th}}$  element of the list  
`x[["name"]]` element of the list named "name"  
`x$name` id.

## Indexing matrices

`x[i, j]` element at row `i`, column `j`  
`x[i, ]` row `i`  
`x[, j]` column `j`  
`x[, c(1, 3)]` columns 1 and 3  
`x["name", ]` row named "name"

## Indexing data frames (matrix indexing plus the following)

`x[["name"]]` column named "name"  
`x$name` id.

## Variable conversion

`as.array(x)`, `as.data.frame(x)`, `as.numeric(x)`,  
`as.logical(x)`, `as.complex(x)`, `as.character(x)`,  
... convert type; for a complete list, use `methods(as)`

## Variable information

`is.na(x)`, `is.null(x)`, `is.array(x)`, `is.data.frame(x)`,  
`is.numeric(x)`, `is.complex(x)`, `is.character(x)`,  
... test for type; for a complete list, use `methods(is)`

`length(x)` number of elements in `x`

`dim(x)` Retrieve or set the dimension of an object; `dim(x) <- c(3,2)`

`dimnames(x)` Retrieve or set the dimension names of an object

`nrow(x)` number of rows; `NROW(x)` is the same but treats a vector as a one-row matrix

`ncol(x)` and `NCOL(x)` id. for columns

`class(x)` get or set the class of `x`; `class(x) <- "myclass"`

`unclass(x)` remove the class attribute of `x`

`attr(x,which)` get or set the attribute `which` of `x`

`attributes(obj)` get or set the list of attributes of `obj`

## Data selection and manipulation

**which.max(x)** returns the index of the greatest element of **x**

**which.min(x)** returns the index of the smallest element of **x**

**rev(x)** reverses the elements of **x**

**sort(x)** sorts the elements of **x** in increasing order; to sort in decreasing order: `rev(sort(x))`

**cut(x, breaks)** divides **x** into intervals (factors); **breaks** is the number of cut intervals or a vector of cut points

**match(x, y)** returns a vector of the same length than **x** with the elements of **x** which are in **y** (NA otherwise)

**which(x == a)** returns a vector of the indices of **x** if the comparison operation is true (TRUE), in this example the values of **i** for which `x[i] == a` (the argument of this function must be a variable of mode logical)

**choose(n, k)** computes the combinations of **k** events among **n** repetitions  
 $= n! / [(n - k)!k!]$

**na.omit(x)** suppresses the observations with missing data (NA) (suppresses the corresponding line if **x** is a matrix or a data frame)

**na.fail(x)** returns an error message if **x** contains at least one NA