Introductory course on the R software

P. Lafaye de Micheaux[1]

[1]Mathematics and Statistics Departement
Université de Montréal
CANADA

New South Wales University
Sydney, Australia
July 1, 2014

https://biostatisticien.eu/springeR/courseRw7.pdf

Goals of today lecture

Describing the instructions for

- developing functions ;
- debugging functions ;
- managing sessions ;
- making R packages.

Quick start

Declaring a function is done with the following general form :

`function(<list of arguments>) <body of the function>`

where

- `<list of arguments>` is a list of named (formal) arguments ;
- `<body of the function>` represents, as the name suggests, the contents of the code to execute when the function is called.

Here is an example of function declaration :

```
> function(name) cat("Hello",name,"!")
function(name) cat("Hello",name,"!")
```

Quick start

For R, a function is a specific object. Creating a function thus corresponds to affecting the object "R function" to a variable, the name of which corresponds to the function itself. For example, to create the function `hello()`, you can proceed as follows :

```
> hello <- function(name) cat("Hello",name,"!")
> hello
function(name) cat("Hello",name,"!")
```

Quick start

For this function to be executed, the user needs to call the function, followed by the effective arguments listed in brackets. Recall that an **effective argument** is the value affected to a formal argument. We will use the terms calling argument and input argument as synonyms of effective argument.

```
> hello("Peter")
Hello Peter !
```

Body of a function

The body of a function can be a simple R instruction, or a sequence of R instructions. In the latter case, the instructions must be enclosed between the characters { and } to delimit the beginning and end of the body of the function. Several R instructions can be written on the same line as long as they are separated by the character ;. When the body of the function includes several R instructions written on the same line, do not forget to enclose them between characters { and }. Recall that on a line, any code written after the character # is not interpreted by R and is taken to be a comment.

```
> hello <- function(name) {
+   # Convert the name to upper case.
+   name <- toupper(name)
+   cat("Hello",name,"!")
+ }
> hello("Peter")
Hello PETER !
```

Arguments

When declaring a function, **all arguments are identified by a unique name**. Each argument can be associated with a default value. To specify a default value, use the character = followed by the default value, as when declaring a list object (`list`). When the function is called with no effective argument for that argument, the default value will be used. We have used this functionality many times in previous chapters, but we now know how to include it when developing new functions. Here is an example :

```
> hello <- function(name="Peter") cat("Hello",name,"!")
> hello()
Hello Peter !
```

Naming effective arguments

In R, an effective argument can be entered by adding the name of the formal argument. Of course, this is of little interest when the function only depends on a single formal argument. Let us add to our function `hello()` the possibility of choosing a language, and see a few calls of this function.

```
> hello <- function(name="Peter",language="eng") {
+   cat(switch(language,fr="Bonjour",sp="Hola",
+              eng="Hello"),name,"!")
+ }
> hello()
Hello Peter !
> hello(name="Ben")
Hello Ben !
> hello(language="fr")
Bonjour Peter !
```

Partial naming of effective arguments

In the same context, a second functionality of **R** is that it allows calling a function without typing in the complete name of a formal argument. Consider the following calls of the function `hello()` :

```
> hello(lang="eng")
Hello Peter !
> hello(l="eng")
Hello Peter !
> hello(l="e")
 Peter !
```

The rule for determining the formal argument corresponding to a partial name is : in the ordered list of formal arguments of the function, the selected formal argument is the **first** formal argument for which there is a match between the first letters of the argument name and the partial name given by the user.

List of supplementary arguments "..."

You can give a list of supplementary arguments with the syntax `...`. When calling the function, all "named" arguments which are not in the list of formal arguments are grouped in the structure `...`. In the body of the function, the user can then use the syntax `...` as if copy-pasting the list of supplementary named arguments.

```
> test.3points <- function(a="foo",...) print(list(a=a,...))
> test.3points("bar",b="foo")
$a
[1] "bar"
$b
[1] "foo"
```

Unless for very specific purpose, use `...` as the last formal argument.

Object returned by a function

The function `hello()` does not return any object.

```
> res <- hello()
Hello Peter !
> res
NULL
```

A general rule to return an object is to use the function `return()`. This instruction halts the execution of the code of the body of the function and returns the object between brackets.

```
> hello <- function(name="Peter") {
+   return(paste("Hello",name,"!",collapse=" "))}
> hello()
[1] "Hello Peter !"
> message <- hello()
> message
[1] "Hello Peter !"
```

Variable scope in the body of a function

The notion of variable scope is very important for a language which allows to develop functions. The main point is that variables defined inside the body of a function have a local scope during function execution. This means that a variable inside the body of a function is physically different from another variable with the same name, but defined in the workspace of your R session. Generally speaking, local scope means that a variable only exists inside the body of the function. After the execution of the function, the variable is thus automatically deleted from the memory of the computer.

Scope of variables

```
> message <- "hello Pierre !"
> message # Workspace initialization.
[1] "hello Pierre !"
> hello <- function(name="Peter",message="hello") {
+   print(message)
+   message <- paste(message,name,"!",collapse=" ")
+   print(message)
+   invisible(message)
+ }
> hello()
[1] "hello"
[1] "hello Peter !"
> message # Workspace has not been modified!
[1] "hello Pierre !"
> message <- hello()
[1] "hello"
[1] "hello Peter !"
> message # Workspace has been modified!
[1] "hello Peter !"
> message <- hello(message="Welcome")
[1] "Welcome"
[1] "Welcome Peter !"
> message # Workspace has been modified again!
[1] "Welcome Peter !"
```

The function `browser()`

A useful debugging function in **R** is the function `browser()`. If you insert the instruction `browser()` in the source of your function, the program will stop at the place where it was inserted.

By typing the letter `n` (for *next*), you can inspect the code and the contents of variables sequentially. To leave the inspection mode, type `Q`.

The function debug()

Another interesting function is `debug()` which is equivalent to putting the instruction `browser()` at the top of a function. Thus `debug(var)` marks the functions `var` as debuggable. Any subsequent call of this function will launch the online debugger.

```
debug(var)
var(1:3)
```

To get rid of this mark, use the function `undebug()`.

```
undebug(var)
```

Listing and deleting objects

After you have created R objects, you can get the list of all objects with the function `ls()` or the synonymous function `objects()`.

To delete objects, use the function `rm()`.

It is worth noting that the command `getwd()` returns the current working directory. The command `setwd()` is used to change working directory.

Workspace : .RData files

When working with **R**, objects are created : vectors, matrices, functions, etc. These objects are physically saved in a file on the hard disk called **workspace**. The file name extension must be `.RData`.
It is possible to create several `.RData` files : one for each project you are working on. You should create these `.RData` files in different appropriate folders. For example, suppose you are working on two different projects : one related to cars and one related to climate events. You could then create a folder called `Cars` containing a file `cars.RData`, and another folder called `Climate` containing a file called `climevt.RData` ; these files will contain the **R** objects corresponding to the two studies.
The function `save.image()` is used to save a workspace ; you can use the function `load()` to load an existing workspace.

.RData files

Perform the “Do it yourself” on page 286.

`http://biostatisticien.eu/springeR/Rbook-chap9.pdf`

Command history

R includes a mechanism to recall and reexecute old commands. The `up` and `down` arrows on the keyboard can be used to go back and forward in the command history. Once you have located a command using this method, you can move the cursor using the `right` and `left` arrows, delete characters with the `DEL` key, and add or modify characters with the keyboard.

To save the command history of the current session, use the command `savehistory()`. To load the command history from a previous session, use the command `loadhistory()`. The commands are saved in a file which must have the extension `.Rhistory` (or `.rhi` in old versions of R).

Managing packages

A package is a collection of data and functions belonging to a same theme. When you install **R**, some basic functionalities come out of the box. But you can extend the functionalities of **R** by adding libraries, also called packages. First, install the package on the computer's hard disk, then load (activate) it in the memory of **R** only when needed (see Appendix A for further details).

The function `search()` gives the list of databases (collections of **R** packages) attached to the system. The function `searchpaths()` returns the same list, but adds the path to the corresponding file.

The function `library()` returns the list of all packages installed on disk.

.RData files

Perform the “Do it yourself” on page 291.

`http://biostatisticien.eu/springeR/Rbook-chap9.pdf`

Your turn to work !

Do the Exercises and the Worksheet on pages 306–309.

`http://biostatisticien.eu/springeR/Rbook-chap9.pdf`