

R Functions: Writing, Using and Documenting

Robert Gentleman

October 22, 2002

R Functions

R functions are defined using the reserved word `function`. Following that must come the argument list contained in round brackets, `()`. The argument list can be empty. These arguments are called the *formal arguments* to the function.

Then comes the body. The body of the function can be any R expression (and is generally a compound expression). When the function is *called* or *evaluated* the user supplies actual values for the formal arguments which are used to evaluate the body of the function.

All R functions take arguments (the number could be zero, though) and return a value. The value can be returned either by an explicit call to the function `return` or it can be the value of the last statement in the function.

A very simple R function,

```
function() 1
```

This function has no argument. What is its value? This function is not very useful because we did not save it (it also isn't so useful because it just returns 1).

```
simplefun <- function() 1
simplefun
simplefun(1)
simplefun()
```

What is the value returned by `simplefun`?

Now, let's do something slightly more complex,

```
sf2 <- function(x) x^2
sf2(3)
sf3 <- function(x) if(x<3) return(x^2) else 4
```

What does `sf3` do? What are the formal arguments in `sf3` and what is the returned value?

Argument matching is done in a few different ways. One is positional, the arguments are matched by their positions. The first supplied argument is matched to the first formal argument and so on. A second method is by name. A named argument is matched to the formal argument with the same name. Name matching takes precedence over positional matching.

The specific rules for argument matching are a bit complicated but generally name matching happens first, then positional matching is used for any unmatched arguments. For name matching a type of partial matching is used – this makes it easy to use long names for the formal arguments when writing a function but does not force the user to type them in.

There is a special argument named `...`. This argument matches all unmatched arguments and hence it is basically a list. It provides a means of writing functions that take a variable number of arguments.

```
mypower <- function(x, power) x^power
mypower(1, 2)
mypower(p=4, 5) ##5^4 not 4^5
```

The formal arguments can have default values specified for them.

```
mypower <- function(x, power=2) x^power
mypower(4)
```

Now, if only one argument is specified then it is `x` and `power` has the default value of 2.

Partial argument matching requires that you specify enough of the name to uniquely identify the argument.

```
foo <- function(aa=1, ab=2) aa+ab
foo(a=1, 2)
```

R is among a class of languages roughly referred to as having *pass by value semantics*. That means that the arguments to a function are copied and the function works on copies rather than on the original values. Because R is a very flexible language this can (like just about everything else) be circumvented. It is a **very bad** idea to do so.

```
x<-1:10
foo <- function(x) x[x<5]<-1
foo(x)
x
```

Notice that `x` is unchanged. Notice also that the expression `foo(x)` did not *seem* to return a value.

```
y <- foo(x)
y
```

Now, we see that it did, it returned the value 1. This is probably not what we intended. What does a function return? What is the value of the statement `x[x<5]<-1`?

Recursion

Here we consider writing a function to compute the sum of the values in a vector. Note, this is purely of pedagogical value, no one in their right mind would do it this way. We will use two different approaches, the first is iterative, the second recursive.

```
sum1 <- function(x) {
  lenx <- length(x)
  sumx <- 0
  for(i in 1:lenx)
    sumx <- sumx + x[i]
  sumx
}

sum2 <- function(x) {
  if(length(x) == 1) return(x)
  x[1] + sum2(x[-1])
}
```

I have put copies of these function in `/usr1/users/biostat/rgentlem/bio271/mypack`.

Writing Documentation

Writing functions can be a lot of fun. But, if you do not write some documentation for them they will soon not be useful. You won't be able to find them or to understand what you have done.

I want to consider briefly some of the tools provided in R (but not Splus) for organizing your work and your functions.

The basic object to work with is a package. Packages are simply a collection of folders that are organized according to some conventions.

A package has a DESCRIPTION file that explains what is in the package. It will also have two folders, one name R that contains your R code and one named man that contains the documentation for the functions.

R documentation is written in a L^AT_EX like syntax called Rd. You don't need to know very much about it since you can use the R function `prompt` to create the documentation and then simply edit it.

For this phase of the lecture, create a directory on hspk called `mypack`. Then `cd` into that directory and create two more directories, one called R and one called man Change to the R directory and edit a file called `mypow.R`. In that file put the following function:

```
mypow <- function(x, power) x^power
```

Then change directory to the man directory. Start R and create the function `mypow` in your R session. Then issue the command: `prompt(mypow)`. You should see something like:

```
created file named mypow.Rd in the current directory.
```

```
Edit the file and move it to the appropriate directory, possibly  
/usr4/biostatistics/src/R-1.4.1/src/library/<pkg>/man/
```

In `~rgentlem/bio271/mypack` is a prototype that I have written. You can copy my DESCRIPTION file and edit it to replace my name with yours.

I then executed

```
R INSTALL -l Rlibs mypack
```

And then I started R and did the following:

```
> getwd()  
[1] "/usr1/users/biostat/rgentlem/bio271"
```

```
> ?library
> library(mypack, lib="/usr1/users/biostat/rgentlem/bio271/Rlibs")
> search()
[1] ".GlobalEnv"      "package:mypack" "package:ctest"  "Autoloads"
[5] "package:base"
> ls(pos=2)
[1] "mypow"
> mypow
function (x, power)
x^power
>
```

You need to edit and adjust the documentation in `mypack/man/mypow.Rd`.

The R extensions manual will give you far more information on writing packages and the Rd format than you probably want to know. But, it is the place to look.

I will also put a document that I have been working on on the web site.