3-minute lesson on creating your own R functions

3-minute lesson on creating your own R functions

Daniel J. Hocking

University of New Hampshire

12 November 2013

Today, we're going to learn how to write your own R functions. Functions are self-contained pieces of code with a well-defined purpose. The can take inputs, do calculations, and produce outputs (Bruan and Murdoch 2007).

Learning to write functions is useful if you have code that you are likely to use more than once and it's also a good way to share code that you've written.

Also, by learning to write your own function, you will be better at reading, using, and manipulating functions for R packages and other users. And isn't that one of the biggest benefits of using open source software.

We will start with a simple function just to raise a numeric value to a power. It's easy to do in R with the carrot operator.

```
3^3
```

```
## [1] 27
```

But having a function can be useful for using with more complicated functions and options or even just to use frequently on multiple values.

```
powerXY <- function(x, y = 2) {
    xy <- x^y
    return(xy)
}
```

Let's take a quick look at the function we created:

```
powerXY
```

```
## function(x, y = 2) {
##     xy <- x^y
##     return(xy)
## }
```

We can see that the function is stored in our R environment and looks correct. Viewing function is useful for checking what they are doing and for copying then manipulating existing functions

```
x1 <- 3
y1 <- 3

x1^y1

## [1] 27

powerXY(x1, y1)

## [1] 27

powerXY(x1)

## [1] 9

x2 <- c(1, 2, 3, 4, 5)
power(x2, y2)

## Error: unused argument (y2)
```

Now if this was a real function for general use, you would want to put in checks with warnings and error messages. For example, what happens if you try to input a vector for x with missing values (NA)? This is the type of thing to think about when writing functions, but for now this should give you a basic idea of how to write your own utility functions and give you a better understanding of the built-in function you regularly use. Remember you can always type the name of the function to see the code behind it, for example,

sd # standard deviation function

This can also allow you to take and modify existing functions, just be sure to give it a new name.

General Programming Guidelines (Bruan and Murdoch 2007)

1. Understand the problem
2. Work out a general idea of how to solve it
3. Translate your general idea into a detailed implementation
4. Check: Does it work?
    - Is it good enough?
    - If yes, you are done!
    - If not, go back to step 2