10-minute lesson on creating your own R functions

10-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.

This 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 is'nt that one of the biggest benefits of using open source software.

This lesson assumes that you're familiar with the basics of the R programming language including for loops, ifelse statements, and the use of built-in functions.

We're going to write a funciton to calculate the mean of a vector. This can already be done with the built-in mean function in R, but it's a good way to start because the results are easy to check.

So we don't have problems with the built-in 'mean' function, we are going to call our funtion 'meanVector'

```
meanVector <- function(x) {
    # We use the function command and give argument (input) x, which will be a
    # vector
    sum.vector <- 0  # Now we set the starting cummulative sum at 0
    for (i in 1:length(x)) {
        # We will use a for loop which might not be the fastest in R but it's easier
        # to understand when starting off than other options
        sum.vector <- sum.vector + x[i]  # Now we add each value in the vector list to the cummulative s
    }  # Close the bracket of the for loop
    mean.v <- sum.vector/length(x)  # We now divide by the number of values in the vector and save it as
    return(mean.v)  # Finally, we specify what will be returned by the function
}
```

We run our function and now it's saved just like any other function. To see it as you would another function just type the name of the function and run that line

```
meanVector
```

```
## function(x) {
##     # We use the function command and give argument (input) x, which will be a
##     # vector
```

```
##     sum.vector <- 0  # Now we set the starting cummulative sum at 0
##     for (i in 1:length(x)) {
##         # We will use a for loop which might not be the fastest in R but it's easier
##         # to understand when starting off than other options
##         sum.vector <- sum.vector + x[i]  # Now we add each value in the vector list to the cummulati
##     }  # Close the bracket of the for loop
##     mean.v <- sum.vector/length(x)  # We now divide by the number of values in the vector and save i
##     return(mean.v)  # Finally, we specify what will be returned by the function
## }
```

Now let's test out our function to make sure it works. We'll start by making a few vectors to try:

```
v1 <- c(-1, -1, 0, 1, 1)
meanVector(v1)
```

```
## [1] 0
```

```
v2 <- c(1, 2, 3, 4, 5)
meanVector(v2)
```

```
## [1] 3
```

```
v3 <- c(1, 2, NA, 4, 5)
meanVector(v3)
```

```
## [1] NA
```

Our function performs well when the vector is made up of numeric values but when there is a missing value as an NA, the function returns an NA. That might be okay, but what if we want to get a mean of the values not including the NA? We need to add an additional argument to the function giving an option of how to handle the NA.

```
meanVector2 <- function(x, skip.na = FALSE) {
    sum.vector <- 0
    if (skip.na == FALSE) {
        for (i in 1:length(x)) {
            sum.vector <- sum.vector + x[i]
        }
        mean.v <- sum.vector/length(x)
    } else {
        x.new <- x[!is.na(x)]
        for (i in 1:length(x.new)) {
            sum.vector <- sum.vector + x.new[i]
        }
        mean.v <- sum.vector/length(x.new)
    }
    return(mean.v)
}
```

Now let's try with the vectors again:

```
meanVector2(v2)
```

```
## [1] 3
```

```
meanVector2(v3)
```

```
## [1] NA
```

```
meanVector2(v3, skip.na = TRUE)
```

```
## [1] 3
```

This time it works even when NA are present

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 matrix for x? 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