Putting the 'FUN' in Functions

Willis Wilson

12/1/2017

An introduction to creating functions and the power of for() loops within functions

The many malleable and ever changing components of R provide an infinite amount of space for data analysis, manipulation, and visualization. To perform the aforementioned, you must use the functions within one of the various R packages, or create your own. Personally, I found the initial idea of creating function to be quite daunting. When I was introduced to the task of creating functions I wasn't confident in my computing abilities within the R software package, but in no time, I had produced over a dozen well formatted and documented functions.

Speaking of documentation, I will begin this post by discussing how to properly produce documentation for a function. It is essential for that you provide documentation for all the functions you create in R. Both for reasons of future reference, and also so that others will be able to use your functions without unnecessary difficulty. With R being an open source software, ease of sharing is at the core of its being.

Steps for creating functions

1. Decide on the function of your Function
2. Write the documentation for your function
3. Write your function using minimal repetion and lines of code
4. Test you function

Functions

```
# Here is a data frame I've created called sneakers, which is a created data frame that lists characteristic of fi
gurative person's sneaker collection. I will use and refer to this data frame throughout the post. The data frame
includes fifteen observations of 6 varaibles. The variables are brand, size, color, laces, price, and resold.

# The 'brand' variable tells you the brand of the sneaker
# The 'size' variable tells you the size of the sneaker
# The 'color' variable give you the primary color of the sneaker
# The 'laces' variable variable tell you if the sneaker has laces or not
# The 'price' variable tells you the original purchase price of the sneaker
# The 'resold' varaible tells you if the sneaker has been resould since the original purchase.

sneakers <- data.frame(
  brand = c("Jordan", "Nike", "Nike", "Jordan", "Vans", "Jordan", "Converse", "Jordan",
            "Vans", "Addidas", "Vans", "Nike", "Addidas", "Vans", "Converse"),
  size = c(9, 12, 10, 8, 7, 9, 12, 13, 11, 12, 13, 14, 10, 9, 7),
  color = c("black", "black", "blue", "green", "red", "white", "orange", "red",
            "green", "black", "blue", "white", "green", "orange", "yellow"),
  laces = c(TRUE, TRUE, TRUE, TRUE, FALSE, TRUE, FALSE, TRUE, TRUE, FALSE,
            TRUE, TRUE, TRUE, FALSE, FALSE),
  price = c(150, 100, 130, 175, NA, 120, 50, NA, 60, 100, 110, 75, NA, 90, 65),
  resold = c("yes", "no", "yes", "yes", "no", "no", "yes", "no", "yes",
             "no", "no", "no", "yes", "yes", "no"))
```

Documenting Functions

Function documentation is typically done using a particular format that is prominent throughout most R software packages. This format includes a title, description, parameters with descriptions, returning value, and an example.

In the description below, I have detailed how to create proper function documentation. The example uses all the aforementioned details. In the example, notice that when creating documentation, you use a hashtag followed by an apostrophe instead of a single hashtag as you would when creating comments in R.

```
#' @title remove_missing()
#' @description takes a vector, and
#' returns the input vector without missing values.
#' @param x numeric vector
#' @return returns the input vector without missing values
#' @example
#' remove_missing(x)
```

Creating a basic funtion

Now that our documentation for our function is complete, we can begin creating our function. The goal of the following function, as stated in our function documentation, is to remove the NA values from a vector. We will create a function that has a single argument and is comprised of a few simple arguments.

The single parameter of our function is a numeric vector. Within in the body, we include the is.na() function and the return() function. Using the body of our function we are requesting that R locates the NA values within our vector, removes them, then assigns the remaining values to a new vector. The "!" preceding the "is.na(x)" command tells R that we want the values within the vector that are not NA. The return() function returns the values of our new vector. Take a look at the two outputs below to compare the vector with and without NA values.

```
remove_missing <- function(x) {
  return(x <- x[!is.na(x)])
}
```

```
print(remove_missing(sneakers$price))
```

```
##  [1] 150 100 130 175 120  50  60 100 110  75  90  65
```

```
sneakers$price
```

```
##  [1] 150 100 130 175  NA 120  50  NA  60 100 110  75  NA  90  65
```

for() loops within functions

For() loops are used to iterate over vectors within R. We will now use the first function we created along with a for() loop to create a new function that averages the values of a vector.

For loops are comprised of three parts, the value, the sequence, and the statement, but before we begin we must again create the documentation for our function; this is a crucial step. Like before, this includes a title, description, parameters with descriptions, returning value, and an example.

```
#' @title get_average()
#' @description Takes a numeric vector, and an optional logical na.rm argument,
#' to compute the average (i.e. mean) the input vector
#' @param x numeric vector
#' @param na.rm whether to remove missing values
#' @return Returns the median of the input vector
#' @example
#' get_average(x, na.rm = TRUE)
```

As mentioned before, the goal of this function is to compute the average (mean), of a vector. The function will include two parameters; one is the vector to be averaged, and the other is na.rm argument that removes NA values when true. For the na.rm argument we will use the remove missing function we created above, in conjunction with an if statement. The na.rm value is set to be TRUE by default, which means that it will automatically remove missing values.

```
get_average <- function(x, na.rm = TRUE){

  avg<- 0   #initializing the value
  average <- 0  #initializing the value

  # na.rm is a conditional statement

  if(na.rm){
    x <- remove_missing(x)

  #writing a for loop

    for(i in 1:length(x)){
        average <- x[i] / length(x)
        avg <- avg + average
    }
   return(avg)

  }else{
    for(i in 1:length(x)){
      average <- x[i] / length(x)
      avg <- avg + average
    }
    return(avg)
  }
}

get_average(sneakers$price)
```

```
## [1] 102.0833
```

Breaking down the for() loops within the function.

For() loops consist of three parts; the value, the sequence, and the statement. The value is used to identify the element with the vector or data frame that you want to interact with. The sequence gives the starting and ending range for iteration, and the statement provided the instructions for the action that you would like to take place over the iteration.

As mentioned before, the value is used to identify the element that you would like to interact with within a vector or data frame. In the example above, “i” is the value; this is not to be confused with the function value, which is “x”. The for() loop value “i” must be included in the for() loop’s statement. In the example I will illustrate the way values are used within for() loops. The rules of defining values within a for loop are identical to the rules for defining values throughout the R software package.

The above for() loop interacts with the specified ‘i’, beginning at 1 and proceeding over the length of the “x” vector specified in the function. what we are attempting to achieve is an average. We do this by dividing each value within the vector by the length of the vector and then summing

these values.

Testing Functions

Once you have successfully created your function, it is essential that you test your function, this can be done using the “testthat” package within R. If you have not already downloaded the “testthat” package within R, you must download the package before proceeding. Instructions on how to use “testthat” to test your created functions are included in the example below.

```
library(testthat)

context("get average function")

test_that("get_average", {

  a <- c(1, 4, 7, NA, 10)
  c <- c(NA, NA, NA, NA, NA)
  d <- c("A", "B", "C", "D")

  expect_equal(get_average(a), 5.5)
  expect_error(get_average(d))
  expect_that(get_average(a), is_a("numeric"))

})
```

Whenever you’re conducting a function test using “testthat”, always begin by stating the context of the test. The context for our example is the “get average function.” Once the context is stated, you proceed to call the test_that() function. The first parameter is the title of the test; traditionally, this is the title of the function followed by a comma. After the title and comma, you open a set of curly braces and write your test function.

Within these curly braces, you should create any values you intend to use in your test; this is very important for reproducibility of you test. Once this is done, you write the specific test you intend to run. In the above example, I use expect_equal(), expect_error(), and expect_that(). The first arguments within each of these functions is the function you are testing with a specified value followed by an expectation. The only exception to this is the expect_error() command because you simply expect an error. Once this is all written and mapped out, you conclude your test with a closing curly brace enclosed inside the closing parenthesis. To complete the testing process, you must now run the test. If no error messages are returned, then your test was successful. Congratulations on creating your first R function.

Conclusion

In conclusion, R provides a variety of functions within an extensive array of packages, but there may be instances when you feel the need to create your own functions. In the above article, I have included the steps to both create your very own functions within R and test your functions to make sure it performs the desired task effectively and efficiently. Within the world of defining functions in R there are infinite possibilities and most these can be executed using around fifteen lines of code or less. There is countless information within the R community of how to develop functions. Hopefully my article will serve as an introduction to function creation and make your transition from simply using created functions within R to creating your own functions both seamless and enjoyable.

References

1. R for Data Science
2. DataCamp.com “Writing functions in R”
3. How to Write Your First for() loop in R
4. A Tutorial on loops in R
5. R in a Nutshell, 2nd Edition
6. R Markdown CheatSheet
7. R for Loop
8. Advanced R