Creating functions in R

Amir Djalovski

2019-01-08

Building functions are one of the main building blocks in programming and in writing reproducible, easy to understand and readable code. In addition, it allows you to write what you need in a more efficient way and to detect errors more easily. Besides these important things, in most cases, your code will run faster. After I convinced you, I hope, we can start learning how to write functions.

Functions can do everything that you can code and ideally whenever you’re writing the same code more than twice you’ll make a function out of it.

To create a function we need to set the number of inputs the function we’ll take, what it will do, and to name the function (sometimes). Let’s start with the following example:

square <- function(x) {   
 out <- x^2  
 return(out)  
 }

We assigned the function to square, thus whenever we'd like to call the function we'll use square(). Next, the function takes only one argument (x), the input will be squared and will get assigned to out. Lastly, using the base R function return, the function will return the value of out.

Practically, we don't have to assign the function to out or use return in this example. However, getting used to using return is a good habit. Whenever we build a long function and we want to debug it, it's always nice to look for the return and see what the function suppose to return.

Now, we can test our function to see if it actually return the expected results

square(2)

## [1] 4

And we can also use our function over a vector

myVec <- c(1:4)  
print(myVec)

## [1] 1 2 3 4

square(myVec)

## [1] 1 4 9 16

We can even create more complicated function with two or three inputs (or more!). let's create a function which calculate the [Pythagorean theorem](https://en.wikipedia.org/wiki/Pythagorean_theorem).

pythagorean <- function(x, y) {   
 out <- x^2 + y^2  
 return(out)  
}

We have two inputs (x and y) and will add the square value of both of them. The result will be assigned to out and will be reutrned. We can test our function

pythagorean(2, 4)

## [1] 20

Thank you for reading!