STATS 266 Handout - Functions

Qi Wang

2025-03-01

Contents

1 Introduction

Welcome to **STATS 266: Introduction to R**. This handout provides an introduction about functions in R. By the end of this document, you should be able to:

- Define a function that takes arguments.
- Return a value from a function.
- Check argument conditions with stopifnot() in functions.
- Test a function.
- Set default values for function arguments.
- Explain why we should divide programs into small, single-purpose functions.

For this part, valuable materials to refer to include https://www.dataquest.io/blog/write-functions-in-r/ and https://swcarpentry.github.io/r-novice-gapminder/10-functions.html.

2 Definition of Functions

Functions gather a sequence of operations into a whole, preserving it for ongoing use. Functions provide:

- a name we can remember and invoke it by
- relief from the need to remember the individual operations
- a defined set of inputs and expected outputs
- rich connections to the larger programming environment

As the basic building block of most programming languages, user-defined functions constitute "programming" as much as any single abstraction can. If you have written a function, you are a computer programmer. The basic format of a function looks like this:

```
my_function <- function(parameters) {
  # perform action
  # return value
}
```

For example, if we want to create a function that transform the Fahrenheit to Kelvin, in a mathematical function, we use:

$$K = \frac{(F - 32) \times 5}{9} + 273.15$$

. In an R function:

```r
fahr_to_kelvin <- function(temp) {
  kelvin <- ((temp - 32) * (5 / 9)) + 273.15
  return(kelvin)
}
fahr_to_kelvin(95)
```

```
## [1] 308.15
```

In this function, `temp` is the input argument, the part inside {} is the main body of the function. One of the most important thing is the `return()` here, which tells the computer which argument will it return to you, i.e., which variable is the output of the function. # Set Default Values

As we see from the previous function, if we don't specify a value for the argument, the function will return an error. However, we can set the default values for the argument of the function.

```r
fahr_to_kelvin <- function(temp = 95) {
  kelvin <- ((temp - 32) * (5 / 9)) + 273.15
  return(kelvin)
}
fahr_to_kelvin()
```

```
## [1] 308.15
```

For the previous example, we set the default value of the `temp` to be 95. Therefore, although we didn't specify the value of `temp` when we use the function, it takes 95 as its default value.

3 Loop and Function

We use function typically to reduce repetitive coding chunks. Therefore, there are some cases that we need to put loop inside a function. For example, let's write a function to calculate the row sums of a matrix:

```r
row_sum <- function(mat){
  d1 <- nrow(mat)
  d2 <- ncol(mat)
  result <- rep(0, d1)
  for (i in 1:d1) {
    for(j in 1:d2){
      result[i] <- result[i] + mat[i,j]
    }
  }
  return(result)
}
M <- matrix(1:16,4,4)
row_sum(M)
```

```
## [1] 28 32 36 40
```

```r
rowSums(M)
```

```
## [1] 28 32 36 40
```

Note: Please be always careful even when you don't use a function, why do I define d1 and d2? Since if we change the input, the dimensions will also change. These types of code will not be sensitive to the input, and further will be more adaptive. Take this example:

```r
M <- matrix(1:16,4,4)
result <- rep(0, 4)
  for (i in 1:4) {
    for(j in 1:4){
      result[i] <- result[i] + M[i,j]
    }
  }
result
```

```
## [1] 28 32 36 40
```

The previous code is correct, but not so adaptive:

```r
M <- matrix(1:25,5,5)
result <- rep(0, 4)
  for (i in 1:4) {
    for(j in 1:4){
      result[i] <- result[i] + M[i,j]
    }
  }
result
```

```
## [1] 34 38 42 46
```

```r
rowSums(M)
```

```
## [1] 55 60 65 70 75
```

The return of the previous code is definitely not what we want, because when we change the dimension of the matrix, we also need to change the loop index and the result length. That would be too complicated when we have a long program. Therefore, avoid using detailed information about data, instead, define some argument to bring in the information of the data.

4 Composite Functions

We can take the output of one function as the input of another function. In math, it's like $f(g(x))$. Let's make an example:

```
fahr_to_kelvin <- function(temp) {
  kelvin <- ((temp - 32) * (5 / 9)) + 273.15
  return(kelvin)
}

kelvin_to_celsius <- function(temp) {
  celsius <- temp - 273.15
  return(celsius)
}
```

What if we want to transform the temperature from Fahrenheit to Celsius? We first transform the temperature from Fahrenheit to Kelvin and then from Kelvin to Celsius:

```
fahr_to_kelvin(95)
```

```
## [1] 308.15
```

```
kelvin_to_celsius(fahr_to_kelvin(95))
```

```
## [1] 35
```

For more technical details about a function, refer to: https://drive.google.com/file/d/1fA1d_G0-w2UsODAbRB6EIJ6zzapJKXod/view. This course is an introduction level of basic R skills, and will move to data visualization quickly.

5 Ackowledgement

This teaching material is adapted from the previous material of this course made by Marcela Alfaro-Córdoba and Sheng Jiang.