Data Science with R

Part III: Functions and How to Get Help

Raphael Schleutker

March 6, 2020

There are only two industries that refer to their customers as users. — Edward Tufte

Table of contents

Function Basics

Function Basics

To understand computations in R, two slogans are helpful:

- Everything that exists is an object.
- Everything that happens is a function call.

— John Chambers

Function Basics

This has two very important consequences:

- Whenever something happens in R, may it be the computation of a value, an assignment, deletion of a variable, plotting graphics, etc., there is a function called. No exception.

Given the central role of functions it's worth studying them in some detail.

Function Basics

This has two very important consequences:

- Whenever something happens in R, may it be the computation of a value, an assignment, deletion of a variable, plotting graphics, etc., there is a function called. No exception.
- Objects and functions are not mutually exclusive. Indeed, functions are objects as well (as they exist) and can be passed around like any other object. This is not a matter of course and we will see in later presentations (functional programming) why this is useful.

Given the central role of functions it's worth studying them in some detail.

Function Basics

The essential parts of a function are

- The name, by which the function is called.

For now, we only focus on how to use functions, i.e. we can ignore the function body until later.

Function Basics

The essential parts of a function are

- The name, by which the function is called.
- The argument list that is passed to the function.

For now, we only focus on how to use functions, i.e. we can ignore the function body until later.

Function Basics

The essential parts of a function are

- The name, by which the function is called.
- The argument list that is passed to the function.
- The function body, which is a series of expressions 'that do something'.

For now, we only focus on how to use functions, i.e. we can ignore the function body until later.

Function Basics

The essential parts of a function are

- The name, by which the function is called.
- The argument list that is passed to the function.
- The function body, which is a series of expressions 'that do something'.
- The return value of the function. Every function has one (and only one) even though it doesn't seem so sometimes.

For now, we only focus on how to use functions, i.e. we can ignore the function body until later.

Function Basics

Functions in programming are quite similar to functions as we know them from math.

$$f(x) = 3 \cdot x^2$$

These functions also have a name (f), an argument list (x, but can be more of course), a function body ($3 \cdot x^2$), and a return value (e.g. 12 for $x = 2$).

Function Basics

```
mean(x = c(1, 2, 3, 4, 5))

# [1] 3
```

- `mean` is the function name.
- `x` is one argument of the function.
- 3 is the return value.

Note that the function is called (executed) by appending parentheses to the function name.

Function Basics – Functions as Objects

```
mean()

# Error in mean.default():  argument "x" is missing, with no
default

mean

# function (x, ...)
# UseMethod("mean")
# <bytecode: 0x0000000013beb3b0>
# <environment: namespace:base>
```

- In the first case, the function is called, which produces an error because there is no parameter to calculate the mean from.
- In the second case, the object stored in `mean` is returned, i.e. the function object. It's like writing the name of a variable holding some numerical values. In that case you also get back what the variable holds, i.e. the numerical values.

Function Basics – Functions as Objects

```r
a <- mean(x = c(1, 2, 3, 4, 5))
b <- mean
```

What do the variables `a` and `b` hold?

Function Basics – Functions as Objects

```
a

# [1] 3

b

# function (x, ...)
# UseMethod("mean")
# <bytecode: 0x0000000013beb3b0>
# <environment: namespace:base>
```

`a` holds the result of the function call (and only the result; there is no information stored in `a` how this value was calculated) whereas `b` holds the function object. Thus, `b` is now a different name for the same function object that is stored in `mean`.

Function Basics – Functions as Objects

The proof...

```
b(x = c(1, 2, 3, 4, 5))


# [1] 3
```

Function Basics – Arguments

`x` is one argument of `mean` but not the only one.

```
a <- c(6, 6, 5, 7, 9, 1, 3, NA, 2, 3, NA)
mean(x = a, trim = 0.1, na.rm = TRUE)


# [1] 4.666667
```

If `na.rm` is set to `TRUE` all missing values will be removed before the mean is calculated. `trim` is the fraction of values removed from either end of the vector.

Function Basics – Arguments

`x` is one argument of `mean` but not the only one.

```
a <- c(6, 6, 5, 7, 9, 1, 3, NA, 2, 3, NA)
mean(x = a, trim = 0.1, na.rm = TRUE)


# [1] 4.666667
```

If `na.rm` is set to `TRUE` all missing values will be removed before the mean is calculated. `trim` is the fraction of values removed from either end of the vector.

Some function arguments have default values. For instance, `na.rm` from `mean` is by default set to `FALSE`. `x` does not have a default value (wouldn't make any sense...).

```
mean(x = a)


# [1] NA
```

Function Basics – Arguments

It is not necessary to always write the name of all arguments.

```
mean(c(6, 6, 5, 7, 9, 1, 3, NA, 2, 3, NA), 0.1, TRUE)

# [1] 4.666667
```

The parameters are matched by position to each argument. It is convenient to not name the first one or two arguments (those that are known anyways). However, leaving away the name of each argument soon becomes confusing.

Function Basics – Arguments

Positional matching only works if you hand over a parameter for each argument. If you want to hand over the third argument by position you also have to hand over the first and second argument. Thus, the following does not work.

```
mean(c(6, 6, 5, 7, 9, 1, 3, NA, 2, 3, NA), TRUE)

# Error in mean.default(c(6, 6, 5, 7, 9, 1, 3, NA, 2, 3, NA),
TRUE): 'trim' must be numeric of length one
```

In this case, the argument `trim` takes the parameter `TRUE`, which doesn't make sense.

Function Basics – Arguments

Using the argument names allows us to vary the order of the arguments.

```
mean(na.rm = TRUE, x = c(6, 6, 5, 7, 9, 1, 3, NA, 2, 3, NA))

# [1] 4.666667
```

Function Basics – Arguments

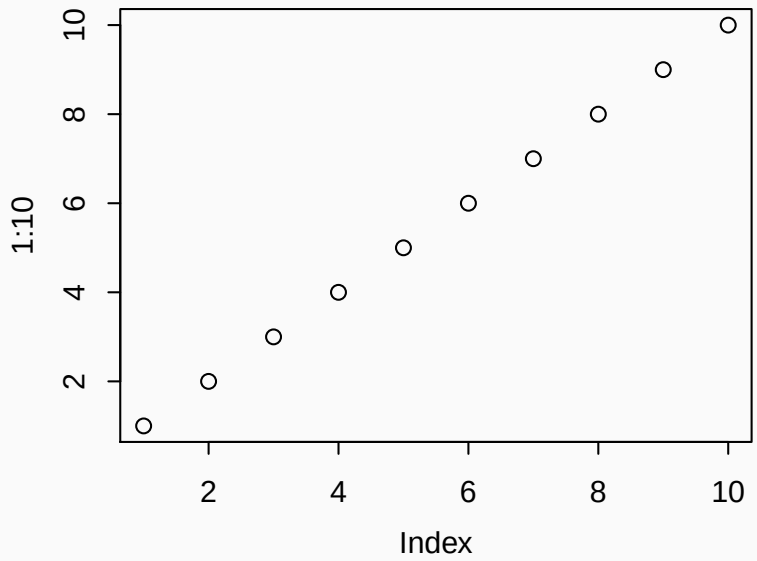
A very special argument that you might see from time to time is `...`. If a function has a `...` argument, all arguments that you hand over to the function and that do not match any existing argument name go into `...`. This is useful for passing through arguments to downstream functions

This seems very abstract for the moment but as soon as we will start to write own functions you will see how this is useful.

Function Basics – Return Values

For the case of `mean` the return value is obviously a numeric vector of length `1`. This return value could be assigned to a variable as we have seen. Sometimes, it seems as if there is no return value.

```
a <- plot(1:10)
```



The graphic produced is not the return value of `plot`. Otherwise, we would not see the graphic when we assign the result to a variable (it is a so called side effect of the function). But what is the return value then?

Function Basics – Return Values

Let's see!

```
a


# NULL
```

The return value is `NULL` which is a very special object that represents a non-existing value. Do not confuse this with a missing value `NA`. Missing values are existent. We just don't know them. `NULL` objects do not exist. There is no storage space reserved for a variable holding `NULL` and thus there is no address for `NULL`.

Function Basics – Infix Functions

We have learned above that a function is called by adding parentheses to the function name. So what about the assignment operator or math operators?

```
a <- c(1, 2, 3)
a + 2


# [1] 3 4 5
```

Function Basics – Infix Functions

We have learned above that a function is called by adding parentheses to the function name. So what about the assignment operator or math operators?

```
a <- c(1, 2, 3)
a + 2


# [1] 3 4 5
```

These are special functions called infix function (as they are in between their arguments) in comparison to prefix functions.

Function Basics – Infix Functions

Every infix function is also a prefix function and can be called as such using backticks or quotation-marks.

```
`<-`(b, c(9, 8, 7))
b

# [1] 9 8 7

`+`(b, 4)

# [1] 13 12 11
```

Infix functions written by third parties are enclosed by % (as you may know from the magrittr package: %>%). Only the R Core Team is able to create infix functions without %.

Getting Help

Getting Help

Even in base R are hundreds of functions. Obviously, we can not remember each function and what arguments they have. So how could we get informations about functions?

Getting Help

Some general tipps and tricks.

- The functions usually have common names like `mean`, `median`, `strsplit`, etc.

Getting Help

Some general tipps and tricks.

- The functions usually have common names like `mean`, `median`, `strsplit`, etc.
- Functions usually take arguments in the order you would expect them. The first argument of `mean` is of course the vector of which you'd like to know the mean.

Getting Help

Some general tipps and tricks.

- The functions usually have common names like `mean`, `median`, `strsplit`, etc.
- Functions usually take arguments in the order you would expect them. The first argument of `mean` is of course the vector of which you'd like to know the mean.
- Arguments that actually do the same in different functions have the same name. For instance, `mean`, `median`, `var` all have an argument `na.rm` that removes any missing values prior to any calculations.

Getting Help

Some general tipps and tricks.

- The functions usually have common names like `mean`, `median`, `strsplit`, etc.
- Functions usually take arguments in the order you would expect them. The first argument of `mean` is of course the vector of which you'd like to know the mean.
- Arguments that actually do the same in different functions have the same name. For instance, `mean`, `median`, `var` all have an argument `na.rm` that removes any missing values prior to any calculations.
- If you use an IDE you usually only need to remember the first letters as the IDE will auto-complete your input.

Getting Help

Some general tipps and tricks.

- The functions usually have common names like `mean`, `median`, `strsplit`, etc.
- Functions usually take arguments in the order you would expect them. The first argument of `mean` is of course the vector of which you'd like to know the mean.
- Arguments that actually do the same in different functions have the same name. For instance, `mean`, `median`, `var` all have an argument `na.rm` that removes any missing values prior to any calculations.
- If you use an IDE you usually only need to remember the first letters as the IDE will auto-complete your input.
- Google is a very good source for help but be cautious. Not every answer you find on the internet is a good one (even though it technically works).

Getting Help

Some general tipps and tricks.

- The functions usually have common names like `mean`, `median`, `strsplit`, etc.
- Functions usually take arguments in the order you would expect them. The first argument of `mean` is of course the vector of which you'd like to know the mean.
- Arguments that actually do the same in different functions have the same name. For instance, `mean`, `median`, `var` all have an argument `na.rm` that removes any missing values prior to any calculations.
- If you use an IDE you usually only need to remember the first letters as the IDE will auto-complete your input.
- Google is a very good source for help but be cautious. Not every answer you find on the internet is a good one (even though it technically works).
- You always can use the internal help system of R.

Getting Help

For getting help in R to a certain function, type ? followed by the name of the function.

```
?mean
```

If you are using pure R, i.e. not an IDE, your default web browser will open with the help site to this function (it is an HTML file on your computer so you can use the help system even without internet).

If you are using RStudio, the help site is opened in RStudio instead.

Getting Help

The help site always has the same structure.

- First comes a header with a brief description what the functions does.

Getting Help

The help site always has the same structure.

- First comes a header with a brief description what the functions does.
- (Usage) How do you call the function. Here you also see the default values for arguments.

Getting Help

The help site always has the same structure.

- First comes a header with a brief description what the functions does.
- (Usage) How do you call the function. Here you also see the default values for arguments.
- (Arguments) A description about each argument, i.e. what values does an argument take and what does it do.

Getting Help

The help site always has the same structure.

- First comes a header with a brief description what the functions does.
- (Usage) How do you call the function. Here you also see the default values for arguments.
- (Arguments) A description about each argument, i.e. what values does an argument take and what does it do.
- (Details) A more detailed description of what the function does, sometimes how it does it, and more detailed explanations about the arguments if necessary.

Getting Help

The help site always has the same structure.

- First comes a header with a brief description what the functions does.
- (Usage) How do you call the function. Here you also see the default values for arguments.
- (Arguments) A description about each argument, i.e. what values does an argument take and what does it do.
- (Details) A more detailed description of what the function does, sometimes how it does it, and more detailed explanations about the arguments if necessary.
- (Value) The return value of the functions. This is very important cause often you will use the return value of one function as input for another function. This way you can check that both have the same structure.

Getting Help

The help site always has the same structure.

- First comes a header with a brief description what the functions does.
- (Usage) How do you call the function. Here you also see the default values for arguments.
- (Arguments) A description about each argument, i.e. what values does an argument take and what does it do.
- (Details) A more detailed description of what the function does, sometimes how it does it, and more detailed explanations about the arguments if necessary.
- (Value) The return value of the functions. This is very important cause often you will use the return value of one function as input for another function. This way you can check that both have the same structure.
- References and related functions.

Getting Help

The help site always has the same structure.

- First comes a header with a brief description what the functions does.
- (Usage) How do you call the function. Here you also see the default values for arguments.
- (Arguments) A description about each argument, i.e. what values does an argument take and what does it do.
- (Details) A more detailed description of what the function does, sometimes how it does it, and more detailed explanations about the arguments if necessary.
- (Value) The return value of the functions. This is very important cause often you will use the return value of one function as input for another function. This way you can check that both have the same structure.
- References and related functions.
- (Examples) Some use cases and examples.

Getting Help

As you are becoming a better programmer this help system often is everything you need.