First steps in R

Claudius Gräbner-Radkowitsch

2/12/2022

Contents

In this post we will learn about the basic *syntax* of R. The syntax basically refers to the grammatical rules you must adhere to when communicating with your computer in the language `R`: if you you do not follow the right syntax, i.e. you 'speak' gramatically incorrect, your computer will not understand you and communicate this to you by throwing up an error message.

To learn about these important basics, the post follows the following structure:

- Commands
- Objects, functions, and assignments
- Kind of objects

Issue commands to your computer

There are two ways we can communicate with our computer in R Studio: either issuing commands directly via the *console*, or by executing a script.

Lets start by using the console and use R as a simple calculator first: we first want to add the numbers `2`and `5`. To this end, simply type `2 + 5` into the console and press `Enter`. Since the expression `2 + 5` is syntactically correct R code, the computer 'understands' what we want from it and returns the result:

```
2 + 5
```

```
#> [1] 7
```

The `#>` at the beginning of the line indicates that what is written on this line is the output of an R command (but the concrete sign might be different on your computer).

The result of `2+5` is a number (more precisely: a 'scalar'). In R, scalars are always represented as a *vector* of length 1. The `[1]` here indicates that the first element on this line is the first element of the vector. If the result of our calculation was a very long vector that needs to span several lines, at the beginning of the next line R would show us the index of the first number displayed on this line.[1]

In this way we can use R as a simple calculator, because for all simple mathematical operations we can use certain symbols as operators. At this point it should be pointed out that the symbol `#` in R introduces a *comment*, that means everything in a line after `#` will be ignored by the computer and you can make notes in the code that only help *you* (or other humans) to understand what you have written.

[1] You may try this out by typing `1:100` into your console and see what happens: this returns a vector of length 100, which certainly will contain some line breaks.

```r
2 + 5 # Addition
```

```
#> [1] 7
```

```r
2/2 # Division
```

```
#> [1] 1
```

```r
4*2 # Multiplication
```

```
#> [1] 8
```

```r
3**2 # Exponentiation
```

```
#> [1] 9
```

As an alternative to typing the commands into the console and then press `Enter` to execute them, we can write down the commands in a script, and then execute this script. While the interaction via the console is useful to test the effects of certain commands, scripts are useful if we want to develop more complex operations, and save what you have written for later, or to make them accessible to other people: we can save scripts as a file on our computer, and then use them later.

The operations that we have conducted so far and not particularly exciting, to be honest. Before we proceed with more complex operations, however, we need to understand the ideas of `objects`, `functions`, and `assignments`.

Objects, functions, and assignments

> To understand computations in R, two slogans are helpful: Everything that exists is an object. Everything that happens is a function call. John Chambers

The statement 'Everything that exists is an object.' means that every number, function, letter, or whatever there is, is an object that is stored somewhere in the physical memory of your computer. For instance, in the computation `2 + 3`, the number `2` is as much an object as the number `3` and the addition-function, which we call via the operator `+`.

The statement 'Everything that happens is a function call.' means that whenever we tell our computer to do something via R, we are effectively calling a *function*.

Functions are algorithms that apply certain routines to an *input* and produce an *output*. The addition function we called in the calculation `2 + 3` took as input the two numbers `2` and `3`, applied to them the addition routine and produced the number `5` as output. The output `5` is an object in R just like the inputs `2` and `3`, as well as the addition function.

A 'problem' is that in the present case R prints the output of the calculation but we have no access to it afterwards:

```r
2 + 3
```

```
#> [1] 5
```

It is stored, for some time, on the physical memory of our computer, but we basically have no way to find it. To address this problem we can issue an *assignment*: whenever we want to keep using the output of an operation, we may give the output a *name*. This name works effectively as a kind of pointer, which points to the place on the computer memory where the output is saved. This way, we can access, and reuse it whenever we call the name. The process of giving a name to an object is called *assignment*, and it is effectuated via the function `assign`:

```r
assign("intermediate_result", 2 + 3)
```

We explain the process of calling a function in more detail below. Here we focus on the process of assignment instead. What the function `assign` does is the following: it assigns the name `intermediate_result` to the result of the operation `2 + 3`. We can now call this result by writing its name into the console and press `Enter`:

```
intermediate_result
```

```
#> [1] 5
```

Since making assignments happens so frequently in practice, there is a shortcut to the use of the function `assign`, namely the operator `<-`. Thus, the following two commands to effectively the same thing:

```
assign("intermediate_result", 2 + 3)
intermediate_result <- 2 + 3
```

From now on, we will only use the `<-` operator, which also represents quite nicely the idea of assignments as pointers to certain objects.[2]

> **Digression: why `<-`?** The use of the string `<-` as an assignment operator is at first sight unintuitive, uncomfortable, and rather unique in the world of programming languages. Much more common is the use of `=`. Where does this particularity of R come from? Besides practical reasons – in contrast to `=`, the use of `<-` makes explicit the unidirectionality of an assignment – the main reason is historical: `R` originated from the programming language `S`. This in turn has taken over the `<-` from the language `APL`. And APL, in turn, was developed on a keyboard layout, where `<-` had its own key. Moreoever, since the operator `==` was not commonly used at that time, `=` was already assigned as test for equality (which, today, is basically always done by using `==`). And so one has decided to use `<-` as an assignment operator and while since 2001 you can also make assignments in R using `=`, `<-` remains strictly recommended for the sake of readability as well as some technicalities.

You are not allowed to give names to objects as you wish. All syntactically correct names in R...

- only contain letters, numbers, or the symbols `.` and `_`
- do not start with `.` or a number

Moreover, there are some reserved words that you must not (and cannot) use as names, e.g. `function`, `TRUE`, of `if`. You can have a look at the complete list of forbidden words by calling `?Reserved`.

There is, however, nothing to remember since whenever you try to give an object a name that conflicts with the rules just described, R immediately throws an error message:

```
TRUE <- 5
```

```
#> Error in TRUE <- 5: invalid (do_set) left-hand side to assignment
```

There are, however, some rules that determine what is a *good* name and that you should adhere to whenever possible:

- Names should be short and informative; `sample_mean` is a good name, `vector_2` not so much
- You should **never use special characters**, especially *Umlaute*
- R is *case sensitive*, meaning that `mean_value` is a different name than `Mean_Value`
- Even if this is possible you should never use names that are already used for function provided by R. For instance, an assignment such as `assign <- 2` is possible, but it effectively prevents you from using the function `assign` without further complications.

  Note: You can have a look at all current assignments in the `Environment` pane in R-Studio, or list them by calling `ls()`

[2] In theory we can use `<-` also the other way around: `2 + 3 -> intermediate_result`. At first sight this is more intuitive and respects the sequence of events: first, the result of `2 + 3` gets created, i.e. a new object gets defined. Then, this object gets the name `intermediate_result`. However, the code that results from such practice is usually much more difficult to read, so it is common practice to use `<-` rather than `->`.

Note: One object can have more than one name, but no name can ever point to two object. If you re-assign a name, the old assignment will be overwritten:

```
x <- 2
y <- 2 # The objekt 2 now has two names
print(x)
```

```
#> [1] 2
```

```
print(y)
```

```
#> [1] 2
```

```
x <- 4 # The name 'x' now points to '4', not to '2'
print(x)
```

```
#> [1] 4
```

Note: As you might have experienced, R does not return results after making an assignment:

```
2 + 2 # No assignment, R returns the result in the console
```

```
#> [1] 4
```

```
x <- 2 + 2 # Assignment, R does not return the results in the console
```

Lets recap what we have learned so far about issuing commands, names and assignments:

- We can issue commands to the computer in R by (a) typing R code into the console and press `Enter`, or (b) write the code into a script and then execute it
- Everything that *exists* in R is an *object*, everything that *happens* is a function call
- A function is an object, that takes an input, applies a certain routine, and returns an output
- We can assign an object a name by using `<-`. Then we can call this object by typing its name. The process of giving a name to an object is called *assignment*, and we can have a look at all names currently given to objects by calling `ls()` anzeigen lassen

Finally, I want to point your attention to the function `help()`, which can provide you with additional information about the object a name points to. For instance, if you want to get more information about the function with the name `assign`, then just type the following:

```
help(assign)
```