# First steps in R

Paul Schmidt

2022-11-21

Data types, vectors, functions and R packages.

## Table of contents



This chapter is mostly aimed at people who are very new to R. However, people who do know R may still find useful insights from the sections where I emphasize how I use R.

> **Note**
>
> This is certainly not the best tutorial you'll ever find, so if you want other tutorials check out this curated list of R Tutorials here.

Furthermore, this tutorial teaches R the way I use it, which means you can do (and may have done) almost everything I do here with other code/functions/approaches. "The way I use it" mostly refers to me using the {tidyverse}, which *"is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures"* and has become quite popular over the last years. Here is a direct comparison of how to do things in R with *base R* and via the tidyverse.

# 1 Let's go

You can use R like a calculator. It does not matter[1] whether you put spaces in or not as shown here:

```
2+3
```

```
[1] 5
```

```
2 *   6
```

```
[1] 12
```

Similar to what you may know from Excel, you can use functions like e.g. `sqrt()` to obtain the square root of a number:

```
sqrt(9)
```

```
[1] 3
```

You can find out more about a function by running the command with a question mark in front: `?sqrt()`.

Besides built-in functions, R also knows certain things like $\pi$ or the alphabet, which are stored in the built-in constants named `pi` and `letters`:

```
pi
```

```
[1] 3.141593
```

```
letters
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
```

Importantly, you may also define your own variables via `=` or `<-`:

[1] As long as you are dealing with numbers and calculations. Spaces do matter when we are talking about *strings* (i.e. text).

```r
x = 3
x
```

```
[1] 3
```

```r
x = 20
x
```

```
[1] 20
```

```r
a_long_variable_name <- 2 + 4
a_long_variable_name
```

```
[1] 6
```

```r
x + a_long_variable_name
```

```
[1] 26
```

```r
mytext <- "This is my text"
mytext
```

```
[1] "This is my text"
```

Note that the variable `x` was overwritten - at first it was 3, but then it was 20. Also, as can be seen when the variable `a_long_variable_name` is defined, you may put more than just a simple number on the right side of the `<-` or `=`.

## 2 data types

As you just saw, R can deal with both numbers and text. We can check the data type via the `typeof()` function:

```r
typeof(x)
```

```
[1] "double"
```

```
typeof(mytext)
```

```
[1] "character"
```

Here is a simplified overview over some of R's data types you may see more often:

- Numbers
  - `integer`/`int`: whole number
    e.g. *42*, *-1504*
  - `numeric`/`num` & `double`/`dbl`: real number
    e.g. *3.14*, *0.051795*
- Text
  - `character`/`chr`: string values
    e.g. *"hello"*, *"Two words"*
- Factor
  - `factor`/`fct`: categorical variable that stores both string and integer data values as "levels"
    e.g. *Control*, *Treatment*
- TRUE/FALSE
  - `logical`/`logi`: logical value
    either *TRUE* or *FALSE*

## 3 vectors

Instead of dealing with single numbers, we obviously want to deal with entire datasets. Before we get to an entire table with multiple rows and columns, the first step is to understand what a *vector* is in R: It is *a sequence of elements that share the same data type*. I often think about them as a single column in my dataset. Above, we actually already looked at a vector: `letters` is a built-in vector with 26 elements of the data type `character`. We could check this via the `length()` or `str()` functions:

```
length(letters)
```

```
[1] 26
```

```
str(letters)
```

```
 chr [1:26] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" ...
```

Note how the built-in constant `pi` is not a vector, because it's just a single number (*a.k.a.* a scalar).

```
str(pi)
```

```
 num 3.14
```

```
length(pi)
```

```
[1] 1
```

If you want to create your own vector from scratch, you must put all the elements together in the `c()` function and separate them with commas:

```
mynumbers <- c(1, 4, 9, 12, 12, 12, 16)
mynumbers
```

```
[1]  1  4  9 12 12 12 16
```

```
mywords <- c("Hakuna", "Matata", "Simba")
mywords
```

```
[1] "Hakuna" "Matata" "Simba"
```

Interestingly, we can still apply the `sqrt()` function we used above to a vector with numbers and it will simply take the square-root of every element:

```
sqrt(mynumbers)
```

```
[1] 1.000000 2.000000 3.000000 3.464102 3.464102 3.464102 4.000000
```

However, there are also functions like `mean()` which return the mean of all numbers in a vector as a single output element:

```
mean(mynumbers)
```

```
[1] 9.428571
```

## 4 function arguments

So far, the functions we used had in common that they required only one input. The really good stuff in R happens with more complex functions which need multiple inputs. Let us use `seq()` as an example, which seems simple enough, because it generates a sequence of numbers:

```
seq(1, 10)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

As you can see, putting in `1` and `10` separated by a comma generates a numeric vector with numbers from 1 to 10. However, I would like you to fully understand what is going on here, because it will help a lot with more complex functions.

See, we could switch the numbers and the function will work as expected:

```
seq(10, 1)
```

```
 [1] 10  9  8  7  6  5  4  3  2  1
```

So this means that the first input is always the starting point and the second one is always the end point of the sequence, right? Well, yes by default, but you can have it your way if you specifically use the names of the arguments.

Looking at `?seq()` it says `seq(from = 1, to = 1, by = ...)` so this `seq(10, 1)` is more explicitly this: `seq(from = 1, to = 10, by = 1)`. Here is proof:

```
seq(from = 1, to = 10, by = 1)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

Again, if you do not write out the arguments like this, it will simply assume the default order: The first number supplied is `from =` the second is `to =` and the third is `by =`. However, if we write out the arguments, we can use any order we like:

```
seq(1, 9, 2)
```

```
[1] 1 3 5 7 9
```

```
seq(from = 1, to = 9, by = 2)
```

```
[1] 1 3 5 7 9
```

```
seq(from = 1, by = 2, to = 9)
```

```
[1] 1 3 5 7 9
```

```
seq(1, 2, 9)
```

```
[1] 1
```

In short: If you understand why the first three lines of the code above produce the same result, but the last one does not, you are good to go!

# 5 R packages

Any function is always part of an R package.

## 5.1 base R

After installing R there are many functions etc. you can use right away - which is what we did above. For example, when running `?mean`, the help page will tell you that this function is part of the {base} package. As the name suggests, this package is built-in and its functions are ready to use the moment you have installed R. You can verify this by going to the "Packages" tab in RStudio - you will find the *base* package and it will have a checked box next to it.

## 5.2 loading packages

When looking at the "Packages" tab in RStudio you may notice that some packages are listed, but do not have a check mark in the box next to them. These are packages that are installed, but not loaded. When a package is not loaded, its functions cannot be used. In order to load a package, the default command is `library(package_name)`. This command must be run once every time you open a new R session.

## 5.3 installing additional packages

R really shines because of the ability to install additional packages from external sources. Basically, anyone can create a function, put it in a package and make it available online. Some packages are very sophisticated and popular - *e.g.* the package `{ggplot2}`, which is not built-in, has been downloaded 75 million times. In order to install a package, the default command is `install.packages(package_name)`. Alternatively, you can also click on the "Install" button in the top left of the "Packages" tab and type in the *package_name* there.

> **Note**
>
> A package only needs to be installed once, but
> A package must be loaded every time you open a new R session!

Here is a curated list of R packages and tools for different areas.