Understand the function environment & function arguments in R

Learning Objectives

After completing this tutorial, you will be able to:

- Document a function in R describing the function purpose, inputs, outputs and associated structures
- Describe what happens to intermediate variables processed during a function call

What you need

You will need a computer with internet access to complete this lesson.

In the last lesson, we learned how to create a function in `R`. We talked about how functions are efficient ways to reduce variables in your global environment. In this lesson we will explore that further. We will also explore function arguments.

The function environment is self-contained

There are many benefits to using functions in your code including it will make your code simpler and easier to read. However another significant benefit is that it will save memory and keep your coding environment cleaner.

The function environment is self-contained. What this means is that when you run a function, it does not create intermediate variables in your global environment.

For example, in the previous lessons, we created a function called `fahr_to_censius`. Within that function, we see it create two variables:

1. `temp_k`
2. `result`

Run the function below. Then call it `fahr_to_celsius(15)`. Look closely at your global environment in `R`. Do you see the variables `temp_k` or `result` in your list of variables in R Studio?

```
fahr_to_celsius <- function(temp) {
  temp_k <- fahr_to_kelvin(temp)
  result <- kelvin_to_celsius(temp_k)
  return(result)
}

fahr_to_celsius(15)
## [1] -9.444444
```

When we run the function above, it creates a distinct environment where it runs the steps required to complete the tasks specified in the function. However, the variables defined by each intermediate steps are not retained in your global environment. These variables only exist within the function environment. This is convenient for us as programmers because each variable that you create consumes memory on your computer. It also reduces the "clutter" associated with too many variables in your global environment which could conflict further down in your code. For example you may have another variable called "result" in your code code further down.

Documentation

Now that we understand the function environment, let's talk about documenting our functions. This documentation is important to both

1. Remind ourselves later what the function does how to use it and
2. To help anyone else looking at our code understand what the function does.

A common way to add documentation in software is to add comments to your function that specify

1. What the function does
2. The inputs that the function takes and the associated structure of each input
3. the outputs the that function provides and it's associated structure.

Like this:

```
# define function
fahr_to_celsius <- function(temp) {
  # function that converts temperature in fahrenheit to celsius
  # input: temperature in degrees F (must be a numeric value)
  # output: temperature in celsius (numeric)
  # return: temperature in celsius (numeric)
  temp_k <- fahr_to_kelvin(temp)
  result <- kelvin_to_celsius(temp_k)
  return(result)
}
```

Writing Documentation

Formal documentation for R functions that you see when you access the help in R is written in separate `.Rd` using a markup language similar to LaTeX. You see the result of this documentation when you look at the help file for a given function, e.g. `?read.csv`. The `roxygen2` package allows `R` coders to write documentation alongside the function code and then process it into the appropriate `.Rd` files. You should consider switching this more formal method of writing documentation when you start working on more complicated `R` projects. Or if you aspire to write packages in R!

- More on LaTeX
- More on roxygen2 {: .notice }