Functions

Data tamers

season 1 /episode 10

What is this episode about?

One of the most crucial characteristics of programming languages is the fact that they allow us to use ready-made solutions created in the past by ourselves or by some other users. It is possible, however, only if we transform our good solutions into functions, that is, chunks of code, which can be easily reused.

In this episode you will learn:

* How to call functions?
* How to create your own functions?
* How to call functions out of functions?

As an illustration I will use the data set koty\_ptaki available in the package PogromcyDanych.

**library(PogromcyDanych) head(koty\_ptaki, 3)**

**## gatunek waga dlugosc predkosc habitat zywotnosc druzyna ## 1 Tygrys 300 2.5 60 Azja 25 Kot ## 2 Lew 200 2.0 80 Afryka 29 Kot ## 3 Jaguar 100 1.7 90 Ameryka 15 Kot**

Functions

There are many reasons for using functions. These are the three most important ones:

* Functions allow us to easily reuse the fragments of code which have already been created. We can use the solutions which we have designed in the past and thus create new ones much faster. We can also use the solutions/functions of other users and share our own functions with them.
* Functions create a logical divisionof the program; its elements become easier to grasp and document. We can much more easily understand how the program works because we get to know its elements separately.
* Functions shorten the program. They allow us to substitute similar chunks of code with functions and thus reduce the number of similar chunks. The shorter the program, the easier it is to write it, understand it and detect potential mistakes in it.

***# zamieniamy na zmienną napisową, by była dobrze wyświetlana* koty\_ptaki$gatunek <- as.character(koty\_ptaki$gatunek) *# Dla każdego wiersza w tabeli `koty\_ptaki` wyświetlamy prędkość liczbą kropek* for (i in 1:nrow(koty\_ptaki)) { n\_kropek <- koty\_ptaki[i,"predkosc"] / 5 for (j in 1:n\_kropek) { cat(".") } cat(" ", koty\_ptaki[i,"gatunek"], "\n") }**

**## ............ Tygrys ## ................ Lew ## .................. Jaguar ## .............. Puma ## ................. Leopard ## ....................... Gepard ## ............. Irbis ## .................................. Jerzyk ## .............. Strus ## ................................ Orzel przedni ## ...................... Sokol wedrowny ## .................... Sokol norweski ## ........................ Albatros**

Functions

Let us recall the code of the program from the episode 8 which drew speed of species of animals as a specific number of dots.

If we wanted to use the same manner of data presentation again, we would have to write the same code anew. You will learn in this episode how to transform a useful fragment of the code into one or more functions so that you could use it easily afterwards.

**nazwa\_funkcji <- function(argumenty\_rozdzielone\_przecinkiem) { wyrażenie }**

Functions

We use the word function to create functions. For each function we need to specify a list of its arguments and the so-called function body, which is a list of commands that a given function performs.

**rysuj\_kropki <- function(n\_kropek) { *# w pętli rysujemy n\_kropek* for (j in 1:n\_kropek) { cat(".") } }**

where argumenty\_rozdzielone\_przecinkiem is a list of arguments separated by comas (which can have one or no arguments at all as well), and wyrażenie are commands which are to be carried out when the function is called.

**rysuj\_kropki(n\_kropek = 20)**

**## ....................**

The result of the command presented above is a function which needs to be attributed to a variable so that it could be used in the future. This is why the most common declarations include attribution to function, as you can see in the declaration below.

It is also good to remember that a function does not necessarily need a name. You will see later on that we will often use anonymous, that is, unnamed functions.

Functions with arguments

Now we will create a function with dots as an example and we will attribute it to the variable rysuj\_kropki. This function will take only one argument n\_kropek and it will draw exactly as many dots as we specify.

**rysuj\_kropki <- function(n\_kropek = 20) { for (j in 1:n\_kropek) { cat(".") } }**

We will use the loop for to draw the dots.

In the example below a function drawing 20 dots is called.

If we write the arguments in the default order, we do not need to specify their names. This rule applies to both our own and original functions.

In the example below we give the value of the argument. As the name of the argument is not specified, the value will be attributed to the first (in this case the only one) argument.

**rysuj\_kropki(35)**

**## ...................................**

Functions –default arguments

If we frequently call a function selecting the same value of a parameter, we can easily spare ourselves a lot of writing by defining that value as a default value.

**rysuj\_kropki(0)**

**## ..**

When we declare a function, we write the default value after the sign =. The example below creates a new function (overwriting the previous declaration at the same time) with the default value of the parameter n\_kropek.

When the default values are specified and we do not write the value of the argument, the function will automatically select the default value. This option is very convenient when our function has got many arguments.

**1:0**

**## [1] 1 0**

However, when we specify value of the argument, it will overwrite the default declaration.

Functions –dealing with various scenarios

We would like our functions to work just as we have planed. However, in reality functions are sometimes faulty. We also have to face unexpected situations from time to time.

**rysuj\_kropki <- function(n\_kropek = 20) { *# czy liczba kropek do wyświetlenia jest większa niż zero?* if (n\_kropek > 0) { for (j in 1:n\_kropek) { cat(".") } } }**

For example, when we call the function rysuj\_kropki() with the argument 0we would expect 0 dots, but...

Why is that so?

It is so because the vector 1:0 in the loop 1:n\_kropek which is located in the declaration of the function in fact contains two elements.

So, if we want our function to work properly for n\_kropek = 0, we need to deal with this situation differently.

**rysuj\_kropki(30)**

**## ..............................**

How to solve this problem?

Functions –dealing with various scenarios

We can deal with this problem in several ways. We will employ a solution using the command if().

We check whether n\_kropek is equal to/less than 0 (if yes, do not draw anything) or more than 0 (if yes, draw the dots).

**rysuj\_kropki <- function(n\_kropek = 20, znak = ".") { if (n\_kropek > 0) { for (j in 1:n\_kropek) { cat(znak) } } }**

It works well for the argument 0.

It works well for the default arguments.

**rysuj\_kropki(znak="X", n\_kropek = 30)**

**## XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX**

It works well for other values.

Functions –more arguments

Functions may have more arguments and each of the arguments can have (or not) a default value.

If we give arguments in the order different than the defaut order while calling a function, we have to also specify which arguments we are determining.

**rysuj\_wykres\_kropkowy<- function(nazwy, wartosci) { *# zakładamy, że oba wektory są tej samej długości* *# funkcja seq\_along(nazwy) tworzy sekwencję od 1 do długości wektora nazwy* for (i in seq\_along(nazwy)) { rysuj\_kropki(wartosci[i]) cat(" ", nazwy[i],"\n") } }**

As an illustration let us now add an argument describing a sign to be drawn by the function.

And a series of calls. We specify now the second argument while the first one assumes its default value. We must remember to specify the name of the argument.

**rysuj\_wykres\_kropkowy(koty\_ptaki$gatunek, koty\_ptaki$predkosc/5)**

**## ............ Tygrys ## ................ Lew ## .................. Jaguar ## .............. Puma ## ................. Leopard ## ....................... Gepard ## ............. Irbis ## .................................. Jerzyk ## .............. Strus ## ................................ Orzel przedni ## ...................... Sokol wedrowny ## .................... Sokol norweski ## ........................ Albatros**

We can give arguments in any order provided that we give them along with their names.

Functions –a function within a function

When we create new functions we can also use other already defined functions.

As an example I will draw a function which draws a dot chart for a vector of characters and a vector of values.

And an exemplary call.

It is best when our functions are not too long. We can easily describe them and memorize the range of their operations if they are relatively short. For the same reason it is also good to give the functions names and to document them.

Functions reporting errors

The assumption of the function from the previous slide was that both vectors are of the same length.

However, what if they are not? Maybe the user does not even know that they should be?

**rysuj\_wykres\_kropkowy <- function(nazwy, wartosci) { *# czy oba argumenty mają równą długość?* if (length(nazwy) != length(wartosci)) { *# funkcja stop() przerywa działanie funkcji* stop("Argumenty mają różną długość! ", length(nazwy), " oraz ", length(wartosci)) } *# jeżeli wszystkie warunki są spełnione to można kontynuować wykonanie funkcji* for (i in seq\_along(nazwy)) { rysuj\_kropki(wartosci[i]) cat(" ", nazwy[i],"\n") } }**

We should check such assumptions e.g. with the command if(). If the assumption is valid and the function cannot be performed if the assumption is not satisfied, we can stop the function and identify the error with the command stop().

What will happen if we call the function with wrong arguments?

**rysuj\_wykres\_kropkowy(koty\_ptaki$gatunek, 5)**

**Error in rysuj\_wykres\_kropkowy(koty\_ptaki$gatunek, 5) : Argumenty mają różną długość! 13 oraz 1**

Passing the arguments along

List of function’s arguments contains also a special argument ... . It allows us to pass along all the other arguments to the internal functions.

**rysuj\_wykres\_kropkowy <- function(nazwy, wartosci, ...) { for (i in seq\_along(nazwy)) { rysuj\_kropki(wartosci[i], ...) cat(" ", nazwy[i],"\n") } }**

In the example below the operator ... can be seen in two places –in the list of arguments of the function rysuj\_wykres\_kropkowy() and in the list of arguments of the function rysuj\_kropki().

All the arguments of the function rysuj\_wykres\_kropkowy() which have names different from nazwy and wartosci will be passed along to the function rysuj\_kropki().

**rysuj\_wykres\_kropkowy(nazwy = LETTERS[1:5], wartosci = 1:5, znak = "X")**

**## X A ## XX B ## XXX C ## XXXX D ## XXXXX E**

Let us illustrate this case with an example. We call rysuj\_wykres\_kropkowy() with three arguments. The third argument, znak = “x” will be passed along to the function rysuj\_kropki() and as result there will be the x signs on the chart.

Functions returning values

The name of these functions refers to the fact that they return their values.

**suma\_n\_liczb\_losowych <- function(n = 10) { if (n < 1) { return(0) } sum(runif(n)) }**

But the functions which we have created previously also displayed their results on the screen.

The default result of a function is the result of the last expression in the function. Another way to achieve a result is to use the function return() which stops the function and return the value of the function return() as a result.

Look at the following example. The function suma\_liczb\_losowych() draws n numbers and returns their sum as a result.

However, if the argument is a value smaller than 1, the function is stopped and we get the value 0 as a result.

Let us call that function. If its results is not attributed to any variable, it will be displayed on the screen.

And the result when the argument is less than 1.

Summary of R commands

In this episode we have discusses the functions with and without arguments and the functions with and without results.

Below you can see a list of all the commands used in this episode.

Summary of R commands

Below you can see a list of all the commands used in this episode.

***# Tworzymy funkcję wyświetlającą n\_kropek kropek* rysuj\_kropki <- function(n\_kropek) { for (j in 1:n\_kropek) { cat(".") } } *# Funkcje można wywołać podając nazwę argumentu lub nie* rysuj\_kropki(n\_kropek = 20) rysuj\_kropki(20) *# Tworzymy funkcję z domyślną wartością argumentu n\_kropek* rysuj\_kropki <- function(n\_kropek = 20) { for (j in 1:n\_kropek) { cat(".") } } *# Dodajemy obsługę argumentów mniejszych niż 1. Wystarczy instrukcja if()* rysuj\_kropki <- function(n\_kropek = 20) { if (n\_kropek > 0) { for (j in 1:n\_kropek) { cat(".") } } }**

Summary of R commands

Below you can see a list of all the commands used in this episode.

***# Dodajemy drugi argument, określający jakie znaki mają być rysowane. Domyślnie kropki* rysuj\_kropki <- function(n\_kropek = 20, znak = ".") { if (n\_kropek > 0) { for (j in 1:n\_kropek) { cat(znak) } } } *# Możemy podawać argumenty w dowolnej kolejności o ile podamy ich nazwy* rysuj\_kropki(znak="X", n\_kropek = 30) *# Wewnątrz jednej funkcji wywołujemy inną* rysuj\_wykres\_kropkowy<- function(nazwy, wartosci) { for (i in seq\_along(nazwy)) { rysuj\_kropki(wartosci[i]) cat(" ", nazwy[i],"\n") } } *# W przypadku gdy argumenty są złe, zatrzymujemy wykonanie funkcją stop()* rysuj\_wykres\_kropkowy <- function(nazwy, wartosci) { *# czy oba argumenty mają równą długość?* if (length(nazwy) != length(wartosci)) { stop("Argumenty mają różną długość! ", length(nazwy), " oraz ", length(wartosci)) } for (i in seq\_along(nazwy)) { rysuj\_kropki(wartosci[i]) cat(" ", nazwy[i],"\n") } }**

Exercises

* Write a function which receives a numerical argument and then displays on the screen the values from the argument to one.
* Write a function which receives a numerical argument n and then draws a square with a side n full of x signs.
* Write a function which assumes numerical argument n and then draws a square with a side n with x sign around its sides but not in the middle.

You can find sample answers at [https://rawgit.com/pbiecek/MOOC/master/0\_dane/9\_zadania.html](https://rawgit.com/pbiecek/MOOC/master/0_dane/9_zadania.html" \t "_blank)