An Introduction to Programming in R (Part 2)

Philipp Buschmann

Hochschule Fresenius - Market Research and Empirical Research Methods

Winter term 2024

Contents

Vectors (1/2) [cf. HB22 chap. 5.6]

- To store multiple numbers in a vector you can use the combine function:

```
c()
```

- Example:

```
sales.by.quarter <- c(0, 100, 200, 50)
```

We have a single variable here called `sales.by.quarter`: this variable is a vector that consists of 4 **elements**.

- To pull out the second element of the `sales.by.quarter`-vector you can use

```
sales.by.quarter[2]
```

You could store the outcome in a new variable, e.g.

```
sales.second.quarter <- sales.by.quarter[2]
```

Vectors (2/2) [cf. HB22 chap. 5.6]

- Altering the elements of a vector:

```
sales.by.quarter[2] <- 45
```

- Extending a vector:

```
sales.by.quarter[5] <- 600
```

- Further useful things to know about vectors:

```
test <- c(1, 2, 3)
length(test)  # number of elements in test, here 3
test * 7      # multiplies each element in test by 7
test + 7
test - 7
test / 7
```

Text data [cf. HB22 chap. 5.7]

- To create a variable that stores the word "hello", we can type this:

```
greeting <- "hello"
```

Note: The quote marks are not part of the string itself. They just make sure that R knows to treat the characters as text data, known as a **character string**.

- Of course, you can create a vector containing character strings, e.g.

```
months <- c("January", "February", "March", "April",
            "May", "June", "July", "August",
            "September", "October", "November",
            "December")
```

TRUE or FALSE data [cf. HB22 chap. 5.8]

- A key concept are so-called **logical values**: TRUE and FALSE.

```
2 + 2 == 4          # checks equality, here: TRUE
2 + 3 == 4          # FALSE
2 + 3 >  4          # 5 greater than 4? TRUE
2 + 3 >= 4          # TRUE
2 + 3 <  4          # FALSE
2 + 3 != 4          # checks inequality, here: TRUE
1 == 1 | 2 == 3   # 1 equals 1 OR 2 equals 3? TRUE
1 == 1 & 2 == 3   # 1 equals 1 AND 2 equals 3? FALSE
!(1==1 & 2 == 3) # ! negates a statement: TRUE
1 == 1 & ! 2 == 3       # TRUE
! 1 == 1 & ! 2 == 3     # FALSE
!(! 1 == 1 & ! 2 == 3) # TRUE
```

Contents

if-statements

- Sometimes we want to execute a code section in R only if a specific condition is fulfilled. We can do this by using if-statements.
- The general form of an `if`-statement is:

```
if(conditon){commands_1}else{commands_2}
```

- The `condition` is a logical value, i.e. `TRUE` or `FALSE`.
- If the condition is `TRUE`, the commands in the first curly parentheses (`commands_1`) are executed.
- If the condition is `FALSE`, the commands in the second curly parentheses (`commands_2`) are executed.
- If you do not need any commands in the `else`-block (`commands_2`), you can omit the block:

```
if(conditon){commands_1}
```

Now, if the condition is `FALSE`, no code is executed.

Task 1 (1/2)

Assume that the following codes are run in  . What is printed to console after executing code line 4?

- a)

```
x <- 1
if(10 < 3){x <- x * 5
}else{x <- x * 10}
x   # line 4
```

- b)

```
x <- 1
if( x == 1 ){x <- x * 5
}else{x <- x * 10}
x   # line 4
```

Task 1 (2/2)

- c)

```
x <- 1
y <- -1
if( -x == y | x == y ){x <- x * 5}
x   # line 4
```

- d)

```
x <- 1
y <- -1
if( !(-x == y & x == y) ){x <- x * 5}
x   # line 4
```

- e)

```
x <- 3
y <- 3
if(!x >= y ){x <- x * y}
x   # line 4
```

Contents

Loops: the for-loop

Instead of repeating commands manually, you can iterate a statement via a loop: either a specific number of times via a `for`-loop or an indefinite number of times, via a `while`-loop.

- `for`-loop:

```
for(variable in vector){commands}
```

The number of elements in the `vector` specifies how often the `commands` will be executed repeatedly. In the i-th iteration, the `variable` takes the value of the i-th element of the vector.

Example:

```
x <- 0
for(i in c(1,2,3,4,5)){x <- x + i}
x
```

... returns $1 + 2 + 3 + 4 + 5 = 15$

Loops: the while-loop

- while-loop:

```
while(condition){commands}
```

The commands will be executed repeatedly as long as the condition (a logical value) is TRUE.

Example:

```
n <- 5
i <- 1
x <- 0
while(i <= n){
x <- x + i
i <- i + 1
}
x
```

... returns $1 + 2 + 3 + 4 + 5 = 15$

Task 2

Write an R-code that computes the product $1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 7 \cdot 8 \cdot 9 \cdot 10$

a) ... using a `for`-loop,

b) ... using a `while`-loop.

Contents

Writing functions in R

You can create an own function using the R-function `function()`:

```
myfunc <- function(argument1, argument2, ...){commands}
```

Here, `myfunc` is an example for the name of the function.
`argument1`, `argument2`,... are the inputs of the function.
In the curly parentheses you can enter commands the function should execute.
You can use the command `return()` to specify the object that should be returned by your function.

Example:

```
sum_of_two_numbers <- function(x,y){
z <- x + y
return(z)
}
```

Task 3

a) Write an R-function that computes the first binomial formula for two numbers, i.e $(a + b)^2$.

b) Write an R-function that returns the first and the last element of a vector.

c) Write an R-function that returns whether a natural number is even or uneven. For an even number the word "even" should be returned, "uneven" for an uneven number.

Task 4

a) Write an R-function that returns whether a number is negative or positive. The function should return the words "positive" or "negative" respectively. If the number equals zero, "no decision" should be returned.

b) Write an R-function that uses a `for`-loop to reverse the order of the elements of a vector, e.g. the vector (10,2,7,4,5) should be transformed to (5,4,7,2,10).

c) Write a `while`-loop that successively prints all natural numbers up to the natural number N, e.g. if $N = 8$, the numbers 1,2,3,4,5,6,7,8 should be printed to the console successively. Hint: Use the function `print()` to print to the console within a `while`-loop.

d) Write an R-function that uses a `for`-loop to count how many ones a vector contains. The function should return the number of ones.