Exercise: Basic program structure and scripts

BIOS13, Lund University Pedro Rosero

The following instructions assume you use RStudio, but the exercise can also be done in another environment, if you prefer. If you find some exercises ‘too easy’, feel free to skip to the next section. These are all for your own sake, there is no need to write a report or anything like that.

All scripts for all sections and questions will be provided after the exercise ends. You can then revisit them at home and come back to us with questions on Thursday for the next three weeks between 15.00 and 17.00 at Mossen (which we strongly suggest you do if you have any doubt!)

**A. Variables and labels**

You can do these exercises in the R console window. A complete session, solving parts A-C, can be found as a single script in Canvas (script\_sections\_A\_to\_C.R).

1. Create a variable named tigers with the value 10
2. Create another variable lions with the value tigers + 12
3. Check that there now are two object called tigers and lions, using ls() or objects().
4. Make sure that lions has the value 22.
5. Change the value of tigers to 15
6. Make sure that lions still has the value 22 (and that you understand why).
7. It is possible to *name* or *tag* almost anything in R. A name attribute can be a longer, more meaningful description of a variable, which may have more meaning than the, often short, *variable name* (such as tigers or lions above). It can also be a short description of a column of a data frame. The names() function is used to access or change the name attribute of a variable. Try the following commands:

names(tigers) <- "The numbers of tigers in this jungle" tigers  
names(tigers)  
tigers2 <- 2\*tigers

tigers2 (notice that names are inherited)

unname(tigers2) (removes the name)

**B. Expressions**

Using variables and mathematical operators and functions you can use R as a calculator.

*Basic mathematical operators*: +, –, \*, /, ^ (addition, subtraction, multiplication, division, power)

Mathematical operators are carried out in order of *precedence*, which is reverse to the order above. You can override precedence with parentheses ().

8. Calculate (you find the correct answer on the right):

2. 3.714286
3. 40.28571
4. 28.20818
5. ( 45.07076

9. Create variables 𝑎 = 4 and 𝑏 = 7 and calculate

2. 1095.876
4. 7.53321

**C. Vectors**

10. Use the commands c(), seq() or rep() to create the vectors:

1. [2 2 2]
2. [2 4 6 8 10]
3. [0 3 6 9 ... 300]
4. [2 3 2 3 2 3]
5. [0.5 0.6 0.7 ... 2.0]
6. [23 evenly spaced numbers between 11 and 27]
7. [8 7 6 5 4 3 2 1 0 -1]

11.

1. Create a vector u = [5 10 15 20 ... 100]
2. Set elements 3 and 7 to 0
3. Set all elements larger than 50 to -3

*Hint: remember that you can index by using logical operators for comparison (<, <=, >, >=, ==, !=)*

1. Add 7 to all elements
2. Check that your result is:

[12 17 7 27 32 37 7 47 52 57 4 4 4 4 4 4 4 4 4 4]

**D. Writing scripts**

Writing a script means producing a text-file with all the necessary commands to do whatever task you want to be done. Running a script means using the source command, usually in the console window (but a script can also run another script).

1. First of all, decide where to save your scripts. A designated folder is recommended.
2. Make that same folder the ‘working directory’ in R. This can be done most easily by selecting that folder in the “Files” tab (see below) and next clicking page2image56954736and then “Set as Working Directory”. Another alternative is An alternative is using the setwd command at the prompt: setwd("~/...your chosen directory path here...") Graphical user interface, application

   Description automatically generated
3. Create a new, empty, script file by choosing [File | New File | R script] from the menus. It should appear in an editor window.
4. Write a script that  
   Assigns the value 3 to a variable named blue  
   Assigns the value 5 to a variable named red  
   Prints the text ‘Blue and red is ‘ followed by the sum of the two (the cat command is useful here).

My solution looks like this:

blue <- 3

red <- 5

cat('Blue plus red is', blue+red)

1. Save the file with the name blue\_and\_red.R
2. Run the file using source in the console: source('blue\_and\_red.R'). You

should get the output “blue plus red is 8”.

1. Another way to run the file is to press the “Source” button in the editor window. Note that it creates an actual source command in the console. The resulting output should be the same. A shortcut command for this is ctrl + shift + s (windows) or cmd + shit + s (mac)
2. Another quick way to re-run a program is to press ‘arrow up’ in the console window. That will recall previous commands. Press arrow-up until you find the right command and then hit return.

(The following exercises, in increasing order of difficulty, are mostly modified from https://adriann.github.io/programming\_problems.html, and https://www.codeabbey.com/index/task\_list )

1. Write a script that asks the user for their name and greets them with their name.

*Hint: you can use the* readline() *command for this followed by the* cat() *command*

Write your name: Pedro

Hello Pedro!

1. Modify the script above such that only users with names beginning with P are greeted.

*Hint: the* substr() *command allows you to extract characters from a string. For example, the command* substr("Pedro", 1, 1) *will extract the first letter from "Pedro". Also, an* if *statement may come handy for this question*

1. Write a script that converts Fahrenheit (input) to Celcius (output). (Celsius = (Fahrenheit-32)/1.8)

*Hint: you can still use the same* readline() *command to get the printing the console to print* Type temperature in Fahrenheit: *then use the* cat() *command so the script prints when you source it*

Type temperature in Fahrenheit: 100

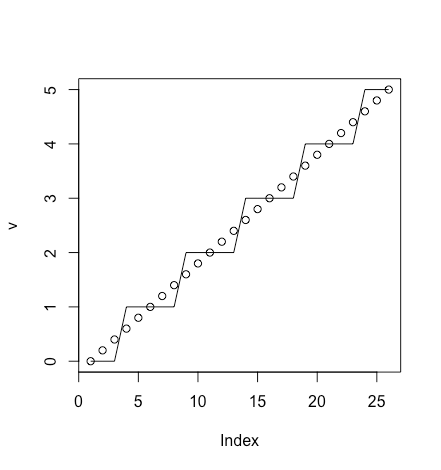
100 Fahrenheit equals 37.77778 Celsius

1. Write a script that inputs 5 numbers and prints the largest and smallest one.  
   *Hint (Try:* numbers <- scan(nmax=5*), other useful functions are* min(numbers) *and* max(numbers))
2. Write a script that
3. creates a vector v = [0, 0.2, 0.4, 0.6, ... 5]
4. rounds off the values of v to nearest integer

*Hint: you can create a new vector* v2 *for this*

1. plots both the original vector and the rounded values in the same graph

*Hint: you can use the* plot() *function for this. We will see plotting in tomorrow’s lecture but try it out by writing* plot(v) *and then to add vector* v2 *try the command* lines(v2)



1. Do the exercise on the last slide from the lecture, the random walk problem.

*Hint: You can start your script with the first command line in the slide. Then, here you need to use a* for *loop that includes the second command line in the slide. Don’t hesitate to ask for assistance! The multi random walk is trickier, perhaps do this later if you don’t feel comfortable with* for *loops so you don’t get stuck here.*

1. Write a script that inputs two vectors of length 3 and prints a new vector that alternatingly takes elements from the first two, e.g. [5,6,10], [1,2,3] → [5,1,6,2,10,3].

*Hint: here thinking about how you index when using a for loop becomes super important, odd and even numbers could be the clue…*

1. Modify the script above to first asks for the length of the vectors (which can be any integer>0 but ≤10).
2. Write a script that inputs a line of text (in English) and prints all vowels in the text in the order they appear.  
   *Hint*: The %in% *operator is useful here. The command*  
   a %in% v

*tests whether the value* a *occurs in the vector* v (and then returns a logical vector with TRUE or FALSE values). Example:  
if( a %in% v ) {

# do something

}

1. Write a guessing game where the user has to guess a secret number (between 1 and 20, use sample.int(20,1)). After every guess the program tells the user whether their number was too large or too small. At the end the number of tries needed should be printed.