Part 1 - R introduction

In this introduction to R, we will present the main R concepts, by introducing R variables, vectors, data frames, lists, functions and packages.

We have just 2h - in this time no one learns how to speak a new language! This time is meant to give you just a glimpse in R with live coding and introduce some basic concepts. There are many fantastic sources to learn R online, I strongly recommend: https://www.adventures-in-r.com and Software Carpentry course http://swcarpentry.github.io/r-novice-inflammation/ (all Software/Data Carpentries are generally highly recommended! https://software-carpentry.org/ https://datacarpentry.org/)

To see examples of data reshaping, processing and plotting: https://paulc91.github.io/intro_to_r/#1 https://evamaerey.github.io/tidyverse_in_action/tidyverse_in_action.html#1

We will present basic R and then some useful features of the fantastic world of tidyverse https://www.tidyverse.org/.

#R and RStudio installation These are preinstalled on the course Virtual Machine. To install them on your personal computer, get R for your operating system from https://cran.r-project.org/ and RStudio (the free version) from https://www.rstudio.com/.

RStudio

RStudio is not a part of R, but an environment to work with R. If you are reading it in RStudio, this text is in a file is opened in the 'source' panel. You can either type your commands here and send them to the console for execution, or type them directly in the console. In another panel there are tabs for displaying your plots (Plots), tree of directory structure (Files), functions help (Help). The Last panel has tabs Environment - when you'll see all objects you create and History - with history of all executed commands.

Now click in the console after the ">" sign. ">" indicates a start of new input line in R. You type a command and click ENTER.

Using R as a calculator

The simplest thing you could do with R is do arithmetic:

```
1 + 100
```

And R will print out the answer, with a preceding "[1]". Don't worry about this for now, we'll explain that later. For now think of it as indicating output.

Like bash, if you type in an incomplete command, R will wait for you to complete it:

```
> 1 +
```

```
+
```

Any time you hit return and the R session shows a "+" instead of a ">", it means it's waiting for you to complete the command. If you want to cancel a command you can simply hit "Esc" and RStudio will give you back the ">" prompt.

When using R as a calculator, the order of operations is the same as you would have learned back in school.

From highest to lowest precedence:

- Parentheses: `(`, `)`
- Exponents (raising to a power): `^` or `**`
- Divide, Multiply: `/`,`*`
- Add,Subtract: `+`,`-`

```
3 + 5 * 2
```

Use parentheses to group operations in order to force the order of evaluation if it differs from the default, or to make clear what you intend.

```
(3 + 5) * 2
```

This can get unwieldy when not needed, but clarifies your intentions. Remember that others may later read your code.

```
(3 + (5 * (2 ^ 2))) # hard to read
3 + 5 * 2 ^ 2       # clear, if you remember the rules
3 + 5 * (2 ^ 2)     # if you forget some rules, this might help
```

The text after each line of code is called a "comment". Anything that follows after the hash (or octothorpe) symbol `#` is ignored by R when it executes code.

Really small or large numbers get a scientific notation:

```
2/10000
```

Which is shorthand for "multiplied by `10^XX`". So `2e-4` is shorthand for `2 * 10^(-4)`.

You can write numbers in scientific notation too:

```
5e3  # Note the lack of minus here
```

Functions

R has many built in mathematical functions. To call a function, we simply type its name, followed by open and closing parentheses. Anything we type inside the parentheses is called the function's arguments:

```
sin(1)  # trigonometry functions
```

```
log(1)  # natural logarithm
```

```
log10(10) # base-10 logarithm
```

```r
exp(0.5) # e^(1/2)
```

A functions take in one or more arguments, processes it and returns a value. It might perform some additional operations, like plotting.

Don't worry about trying to remember every function in R. You can simply look them up on Google, or if you can remember the start of the function's name, use the tab completion in RStudio.

This is one advantage that RStudio has over R on its own, it has auto-completion abilities that allow you to more easily look up functions, their arguments, and the values that they take.

Typing a `?` before the name of a command will open the help page for that command. As well as providing a detailed description of the command and how it works, scrolling to the bottom of the help page will usually show a collection of code examples which illustrate command usage. In RStudio, when you type function name, it shows little bubble with basic help about this function.

(RStudio overview)

```r
help("mean")
?mean
```

Variables

Variables persist, workspace In R, we have two main types of values: numeric values and characters. Type each of these in the console and click ENTER.

```r
1
"text"
```

Characters are indicated by `"` quotes

We can save these values in variables, so we can easily access them later.

```r
a <- 1
a
a = 1
```

They will persist in the workspace untill the R session is finished or they are removed (see them in the "Environment" tab!) And re-use them :

```r
a + a
```

CAPITAL/lower case matters for R. Check:

```r
a
A
```

You can also type your code in a source file and execute code from file.

Exercises:

1) Assign a number of your choice to "b"
2) Show what's in b
3) Multiply a by 2
4) Add 10 to a
5) Assign your name to an object called my_new_object

Vectors

Vectors contain several values:

```r
b <- 1:10
```

You can combine these values with the "c()" function:

```r
c(b, b)
```

You can also apply mathematical formulas to them:

```r
b + b
```

To access elements in a vector, give the indexes of elements of interest in square brackets:

```r
fruit_vector <- c("apple", "pear", "banana")

fruit_vector[2]

fruit_vector[c(1,3)]
```

5) How to get a banana?

If elements of a vector are named, names might be used for accession

```r
animal_vector <- c(marsupial="platypus", real_mammal="aardvark",fish="shark")

names(animal_vector)

animal_vector["fish"]

#
```

6) What would be animal_vector[2]?
7) Combine a and b
8) Combine your name and a. Is the output character or numeric?

Elements of a vector have to be of the same type. If types are mixed, R decides for more general format (in this case character)

9) Add a to b

Functions again

Many functions exist in R, and greatly simplify your life:

```r
b <- 1:10

(1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10)/10
mean(b)
```

The head() function shows the 6 first elements of any object.
The median() function computes the median of a vector.
The length() function returns the length of a vector.

```r
head(b)
head(b, n=3)
median(b)
length(b)
```

Many functions have more than one argument, function help will describe them and also specify the default values if you would not provide your own argument

```r
#compute logarithm with base 5 from 125

log(125)

log(x = 125, base = 5)

?log

log(0)
```

You can define your own functions and then use them:

```r
my_first_function <- function(x){
  x/3
}

my_first_function(45)
```

Lists

Lists in R are the most versatile objects. A list can contain any type of object - even another list - and elements can be of different size, type and class.

```r
random_numbers <- c(1, 2, 3, 7)
random_letters <- c("A", "B", "C", "Z")
```

```r
my_list <- list(my_numbers = random_numbers,
                my_letters = random_letters)
my_list
```

The different items of a list can then be accessed by using the "$" and typing their name directly:

```
my_list$my_numbers
```

Or by providing element number or its name in double square brackets:

```
my_list[[1]]
my_list[["my_letters"]]
```

10) Create a list containing your name, and job:

###Data frames

Data frame is like a table - has rows and columns. It can contain mixed types of data, but each column has to be of one type (for example all entries in a column characters or all entries numbers).

```
exampleDataFrame <- data.frame(patientID= c("A231","B452","C123"),
                               age =c(12,45,37.5),
                               disease =c(TRUE, FALSE, TRUE))
exampleDataFrame
```

Access data frame by column and/or row names or indices

data_frame[ROWS, COLUMNS] - empty field means 'all of them'. As ROWS and COLUMNS you may use vectors

```
exampleDataFrame[,"disease"]
exampleDataFrame[3,]
exampleDataFrame[1:2,"age"]
exampleDataFrame[,"patientID"]

#Now take patientID and age information only - fill the code
exampleDataFrame[,]
```

A shortcut, access one column with $

```
exampleDataFrame$age
```

Add a column

```
exampleDataFrame <- cbind(exampleDataFrame,season=c("winter", "winter", "summer"))
exampleDataFrame
```

To see an object just type its name. For a complicated/big object it might be difficult to see it - glimpse() displays structure of an object, head() the beginning of it

```
exampleDataFrame
glimpse(exampleDataFrame)
```

Matrices

Matrices are similar in shape to data.frames - they are also "tables" - but are more constrained: all the data in a matrix has to be of one type. They are very useful for fast computation, for example gene x sample gene expression matrix. Matrices can have row names and column names. They can be subset in a similar way as data.frames - by [,].

Exports/imports

Objects that were created in R can be exported in many formats to be shared. You can save them in a compact R-specific format and load later:

```
surprise <- read_rds("data/surprise.RDS")

surprise
```

Most often we will read and save data in tabular format: csv, tsv or Excel file

```
write_csv(exampleDataFrame, file = "results/testing_dataframe.csv")

#Now check in the 'Files' pane that indeed it was created. You can now remove exampleDataFrame from you

rm(exampleDataFrame)

#It is no longer present - check the Environment tab and try to print it out:

exampleDataFrame

#But you can read it back:

exampleDataFrame <- read_csv(file = "results/testing_dataframe.csv")

exampleDataFrame
```

Packages

You have access to all functions from the libraries installed for your R, however -besides of basic R - you have at some point load an installed package to your R session. Have a look on the tab “Packages” - it lists all available packages, ones which are loaded are ticked.

Packages contain sets of useful functions, which can be installed, loaded, and then used in R. You can find a package for almost anything. A package needs to be installed only once on your computer:

```
install.packages("emojifont")
```

But needs to be loaded every time you wish to use it:

```
library(emojifont)
```

Look for help/ideas

```
?emojifont
??emojifont
```

Examples of what you can do with emojis:

```
search_emoji("smile")

emoji(search_emoji("smile"))

ggplot() + geom_emoji("heart", color='steelblue') + theme_void()  ##ggplot is a graphic framework - we'
```

Export plots

Plots can be exported from R, for example by opening a pdf environement. The pdf function opens a new pdf file. All the following plotting commands will automatically be plotted in the pdf file. The dev.off() function closes the pdf.

```
pdf("results/heart_plot.pdf")

 ggplot() + geom_emoji("heart", color='steelblue') + theme_void()

dev.off()
```

You can also directly export from the "Plots" tab.

In ggplot you can keep a plot as an object. It makes tweaking it easier, as you can add parameters with "+"

```
my_plot <- ggplot() + geom_emoji("heart", color='steelblue')
my_plot
my_plot + theme_void()
my_plot + theme_dark()
my_plot + theme_bw()
```

11) You can add a title to your plot with ggtitle("TITLE"). Make now a plot with red heart and a title (hint: you can't have a title withe theme_void) and save it to a .pdf file