Introduction to R programming for data science – day 2

Dr. Francesca Finotello
Medical University of Innsbruck, Austria

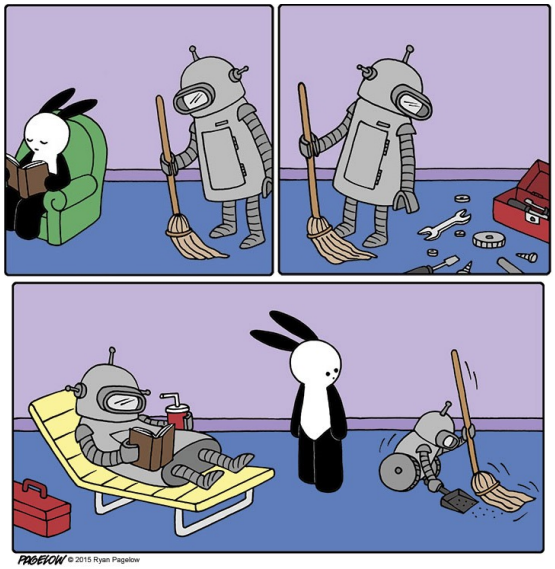
Files and paths

Types of files

- R **objects** can be saved in .RData and .Rds files; .RData files can contain multiple objects.
- Tabular data can be loaded from or saved into delimited text files (.txt, .csv, .xslx, …).
- R **functions** and **scripts** can be saved as .R files.
- R **packages** containing additional functions can be imported from local archive files or repositories like CRAN, Bioconductor, and GitHub.

File locations and paths

A **path**: indicates the unique location of a file or directory.



Absolute and relative paths

A path is expressed by a string of characters separated by a delimiting character, where each component separated by the delimiting characters represents a directory.

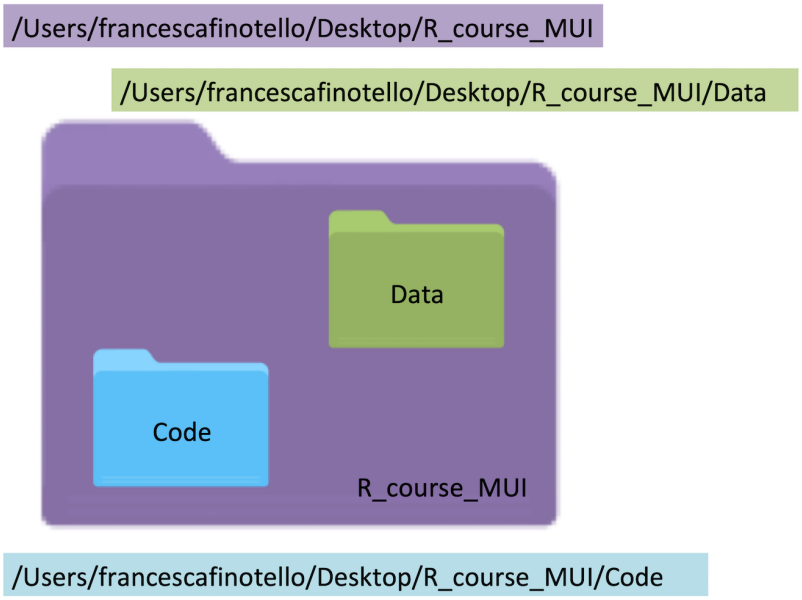
Absolute path: points to the same location in a file system, regardless of the current working directory. To do that, it must include the root directory.

Relative path: starts from some given working directory. A file name alone can be considered as a relative path based at the current working directory.

Examples:

- */home/Desktop/myscript.R*
- *../Data/myfile.txt*
- *./myplot.pdf*
- *myplot.pdf*

Absolute paths



File locations

Let's suppose that our working directory is:

/Users/francescafinotello/Desktop/R_course_MUI/Code

We can load/save whatever R script in the "Code" directory just by specifying the file name.

But if we want to load/save a file from/in the "Data" directory we need to specify its location:

- Absolute path:
  /Users/francescafinotello/Desktop/R_course_MUI/Data/myfile.txt
- Relative path:
  ../Data/myfile.txt

Working directory

The getwd function can be used to known the absolute path of your working directory:

```
getwd()
```

```
## [1] "/Users/francescafinotello/Dropbox/R_course_MUI_2021/Course_slides"
```

The setwd function can be used to set the path (absolute or relative) of the working directory:

```
setwd("/Users/francescafinotello/Desktop/")
```

To set the working directory using the toolbar: Session > Set Working Directory

Useful functions

To know which files are present in your *working directory* (or in another directory) you can use the list.files function:

```
list.files()
list.files("../")
```

This is different from asking which objects are present in the *R workspace* using the ls function:

```
ls()
```

To clean your *R workspace* you can use the rm function or the broom in the Environment panel.

```
rm( list = ls() )
```


R scripts and data

Sourcing R code

The function source can be used to

- Run a script (e.g., full analysis or exercises from day 1)

```
source("RNAseq_diff_gene_expr.R")
source("../Scripts/Day1_Ex8.R")
```

- Import one or more R functions you want to use (e.g., your BMI function)

```
source("BMI_function.R")
```


Saving and loading RData files

The function save can be used to save one ore more R objects into an .RData file. The **file** argument must always be present to specify the path to the output file

```
age <- 40
height <- 180
save(age, height,                          # List of objects to be saved
  file = "../Data/Day3_data_example.RData")  # Path to output file
```

The function load can be used to import .RData files

```
load("../Data/Day3_data_example.RData")

weight <- 80
save(age, height, weight,
  file = "../Data/Day3_data_example_updated.RData")
```

Functions

Functions

Function: set of statements organized together to perform a specific task.

R has a large number of built-in functions.

Additionally, users can:

- Import *R packages* enclosing a set of functions for specific tasks (e.g., differential gene expression analysis, machine learning)
- Create *new functions*.

Functions

Functions are usually invocated with their name, followed by round brakets listing the arguments to be considered (if any).

```
v1 <- c(1, 3, 56, 6)
v2 <- c(4, 5, 40, 7)
cor(x=v1, y=v2, method="pearson")
```

```
## [1] 0.999966
```

The name of the arguments can be omitted if the right order of the arguments is respected:

```
cor(v1, v2, method="pearson")
```

```
## [1] 0.999966
```


Functions

So far, we have used already several functions:

log10, sqrt, class, is.na, as.character, as.factor, as.numeric, c, length, names, levels, which, rep, seq, factor, mean.

There are other arithmetical functions:

sum, prod, max, min, which.max, which.min, range, median, var, sd, round, sign, exp.

The *help* function

If we do not know how a function works, we can invocate the help function to understand:

- Which task it performs
- Which are its arguments, their expected order, and their default values
- Which results it generates (e.g. a plot or an object with a specific format)
- If there are similar functions
- How it can be used, explained through examples

```
help(mean)

help(sample)
```


Useful functions: *length* and *nchar*

```
x <- c("Schildkroete",
  "Ohrwurm",
  "Rechtsschutzversicherungsgesellschaften")

length(x) # Length of the vector of strings
```

```
## [1] 3
```

```
nchar(x) # Length of the strings
```

```
## [1] 12  7 39
```

Useful functions: *table* and *unique*

```
x <- c("a", "b", "c", "c", "d", "e", "e", "e")

unique(x) # Unique elements of x
```

```
## [1] "a" "b" "c" "d" "e"
```

```
table(x) # Occurrences of each value in x
```

```
## x
## a b c d e
## 1 1 2 1 3
```


Useful functions: *order* and *sort*

```
x <- c(10, 2, 20, -1)

sort(x) # Sorted x
```

```
## [1] -1  2 10 20
```

```
order(x) # Indexes of sorted x elements
```

```
## [1] 4 2 1 3
```

Useful functions: *match*

```
names <- c("Maria", "Markus", "Lena")
sel <- c("Lena", "Maria")

match(sel, names) # Find 'sel' elements in 'names'
```

```
## [1] 3 1
```


Useful functions: *head* and *tail*

```
x <- seq(1,100)
head(x,3)
```

```
## [1] 1 2 3
```

```
tail(x,5)
```

```
## [1]  96  97  98  99 100
```

Useful functions for sets

```
A <- c(1, 2, 3, 4, 5, 6, 6)
B <- c(2, 4, 6, 8, 10)
union(A,B)
```

```
## [1]  1  2  3  4  5  6  8 10
```

```
intersect(A,B)
```

```
## [1] 2 4 6
```

```
setdiff(A,B)
```

```
## [1] 1 3 5
```


Function: Hello World!

```
helloWorld <- function() {

  outmessage <- "Hello World!\n"

  cat(outmessage)

}
```

Function: Hello World!

```
helloWorld <- function() {

  outmessage <- "Hello World!\n"

  cat(outmessage)

}
```

```
helloWorld()
```

```
## Hello World!
```


Function: Hello… you!

```
helloWorld2 <- function(to = "World") {

  outmessage <- paste("Hello", to, "!\n", sep = " ")

  cat(outmessage)

}
```

Function: Hello… you!

```
helloWorld2 <- function(to="World") {

  outmessage <- paste("Hello", to, "!\n", sep=" ")

  cat(outmessage)

}
```

```
helloWorld2()
```

```
## Hello World !
```

```
helloWorld2("Francesca")
```

```
## Hello Francesca !
```


Function: BMI

Let's build a function that computes the body mass index (BMI) taking as arguments the weigth in kg and the height in meters.

```
BMI <- function (weight, height) {

  bmi <- weight/height^2

  return(bmi)

}
```

Note: the return function returns the final result/object produced by the function and stops any other evaluation. Any code written after the return statement is not evaluated.

Function: BMI

We can calculate the BMI of an individual who weights 80 kg and is 1.90 m tall

```
BMI(80, 1.90)
```

```
## [1] 22.16066
```

To get meaningful results, we have to respect the argument order or specify their names

```
BMI(1.90, 80)
```

```
## [1] 0.000296875
```

```
BMI(height = 1.90, weight = 80)
```

```
## [1] 22.16066
```


R packages

Install packages from CRAN or local archive files

R packages can be installed from CRAN or local archive files using the “Packages” panel of Rstudio

Packages > Install

By selecting one of the two following options

1. Package Archive File (.tgz; .tar.gz) > [Locate the archive file]
2. Install from Repository (CRAN) > [Specify the package name]

Or, for functions available on CRAN, by using directly the install.packages function (example installation of xlsx)

```
install.packages("xlsx")
```


Install packages from Bioconductor

Bioconductor is an open source software project for the analysis of genomic data with a repository containing >1500 R packages

Bioconductor pages contain different info about a package, including:

- Installation instructions
- Package manual and vignette
- Source code as archive files

R packages can be installed from Bioconductor with the following code (example installation of edgeR)

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("edgeR", version = "3.8")
```

Install packages from GitHub

GitHub is a provider of Internet hosting for software development and version control that can be used host R packages in public repositories.

R packages can be (tenatively) installed from GitHub using the install_github function from the devtools CRAN package (example installation of xCell)

```
library(devtools)
install_github('dviraran/xCell')
```