

# Introduction to R programming for data science – day 1

Dr. Francesca Finotello

Medical University of Innsbruck, Austria

# R programming

R is:

- One of the leading programming languages for statistics and data science
- Useful for basic and advanced data analyses
- Prerequisite for job positions in life science and biomedicine

Why?

- It is free
- It supports reproducible analyses
- It provides packages for data analysis

# In this course...

We will learn the **basics of R programming** through lectures and hands-on computer exercises using Rstudio:

- variables, data types, and data structures
- data manipulation and visualization
- operations
- reading and writing files
- running and repeating tasks using functions and control statements.

Disclaimer: this is **not** a course in statistics or bioinformatics.

# Course info

Course material: XXX

Exam: written test + hand-on project

Projects description: XXX

# Installation

Install first R and then Rstudio

To install R:

<https://www.r-project.org>

See “To download R, please choose your preferred CRAN mirror”.

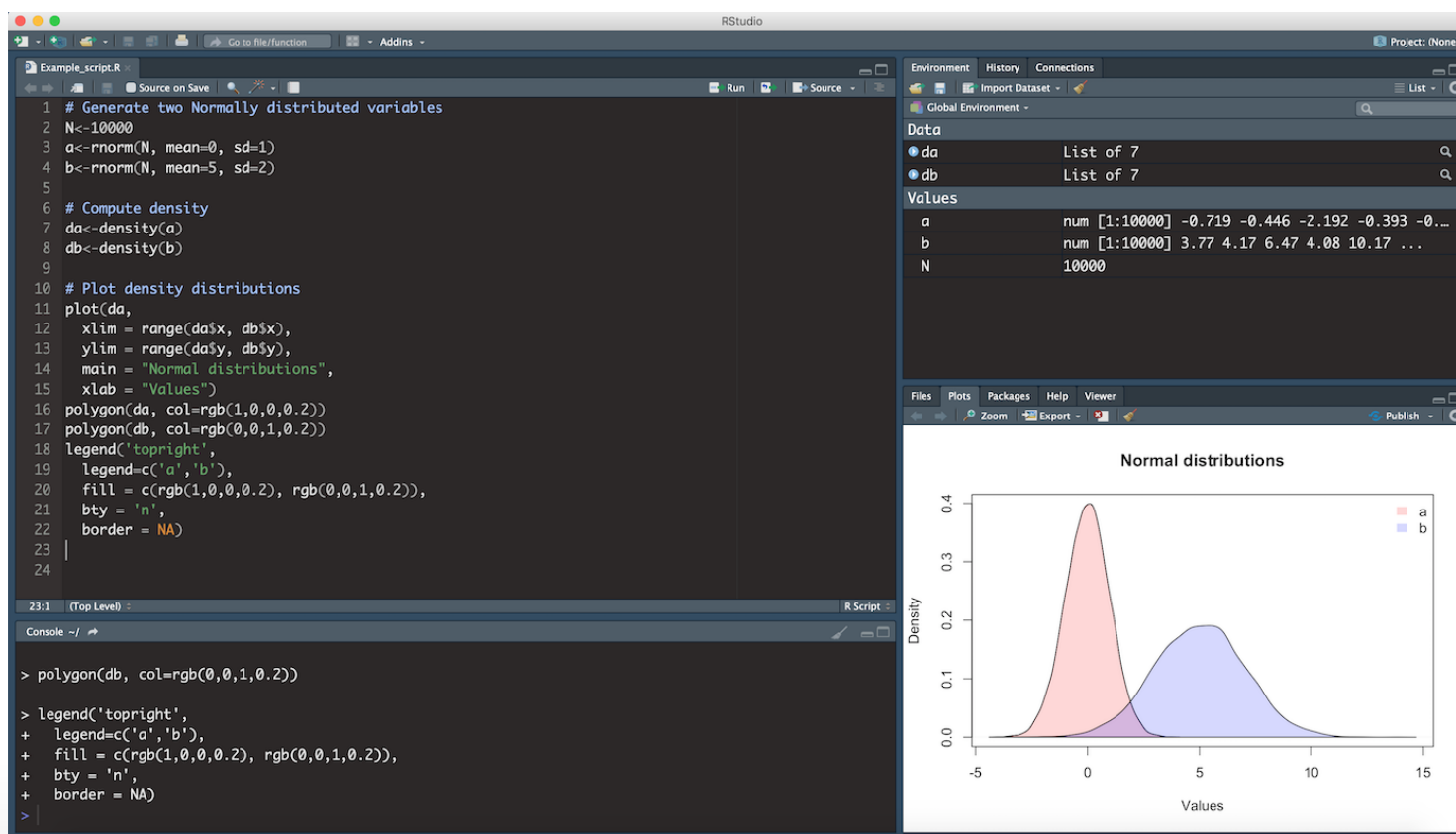
To install RStudio Desktop:

<https://www.rstudio.com/products/rstudio/download/#download>

# RStudio

# RStudio console

RStudio is an integrated development environment (IDE) for R. It has four main panels with adjustable sizes:



# RStudio console

In the console you can type commands and see outputs.

```
( 12^2 - 20 * 2 + 1 ) / 5
```

```
## [1] 21
```

```
log10(100)
```

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

```
sqrt(9) / 3
```

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

Have a look at the [History](#) panel: what do you see?



# RStudio scripts

Instead of using the console, you can save your code in a .R file (e.g. *R\_script\_day1.R*)

Open a **new file**: [File > New File > New R Script](#)

**Save** as R script: [File > Save as](#)

Set working directory: [Session > Set Working Directory > To Source File Location](#)

**Open** an R script: [File > Open File](#) (e.g. *Example\_script.R*)

You can run all the in a script altogether with [Source](#) or in chunks using [Control+Enter](#).

Remember to save your code regularly!

# Variables and data types

# Variables

Variables are symbolic names used to store data that can be manipulated in a computer program

In R, variables are assigned with `<-`

```
height <- 1.90  
  
weight <- 80  
  
# Body mass index  
BMI <- weight/height^2
```

Have a look at the [Environment](#) panel: what do you see?

# Case-sensitive programming

R is case-sensitive

```
BMI
```

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

```
bmi
```

```
## Error in eval(expr, envir, enclos): object 'bmi' not found
```

# Variable names

- Short, but descriptive
- Must start with a letter
- Must not contain special characters
- Cannot be **reserved words** ( *if*, *function*, *NA*, *TRUE*, ...)

Try assign the value of 10 to a variable called:

- *myVar*
- *my\_var*
- *my.var*
- *my-var*
- *1var*
- *var1*

# Operations

Once variables are assigned, they can be used to perform operations

```
x <- 6  
y <- 2  
x + y
```

```
## [1] 8
```

```
x^y
```

```
## [1] 36
```

```
x == y
```

```
## [1] FALSE
```

# Arithmetic operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponent
%%	Modulus (Remainder from division)
%/%	Integer Division

# Relational operators

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to



# Logical operators

Operator	Description
!	Logical NOT
&	Element-wise logical AND
&&	Logical AND
	Element-wise logical OR
	Logical OR

# Atomic vector types

In R, there are several types of atomic vectors (also called *classes*):

- Character
- Numeric
- Logical
- Complex
- Integer

# Data type examples

```
Name <- "Maria"  
class(Name)
```

```
## [1] "character"
```

```
PhD <- TRUE  
class(PhD)
```

```
## [1] "logical"
```

```
yearsSincePhD <- 5  
class(yearsSincePhD)
```

```
## [1] "numeric"
```

# Missing Values

In R, missing values are usually coded with **NA**

```
Name <- "Michael"  
PhD <- FALSE  
yearsSincePhD <- NA  
  
is.na(Name)
```

```
## [1] FALSE
```

```
is.na(yearsSincePhD)
```

```
## [1] TRUE
```

Note: **NA** are not strings (no quotation marks)

# Other indefinite values

Not a number: NaN

```
( var1 <- 0/0 )
```

```
## [1] NaN
```

Infinite: Inf

```
( var2 <- 9/0 )
```

```
## [1] Inf
```

```
var3 <- -10/0
```

What is the value of *var3*?

# Data conversion/coercion

Class conversion is possible, but it must be handled with care

```
var1 <- 1  
( var2 <- as.character(var1) )
```

```
## [1] "1"
```

```
class(var2)
```

```
## [1] "character"
```

```
( var3 <- as.logical(var1) )
```

```
## [1] TRUE
```

# Data type conversion

Conversion to	Function	Rules
numeric	<code>as.numeric</code>	FALSE $\rightarrow$ 0 TRUE $\rightarrow$ 1 "1", "2", ... $\rightarrow$ 1, 2, ... "A", ... $\rightarrow$ NA
logical	<code>as.logical</code>	0 $\rightarrow$ FALSE other numbers $\rightarrow$ TRUE "FALSE", "F" $\rightarrow$ FALSE "TRUE", "T" $\rightarrow$ TRUE other characters $\rightarrow$ NA
character	<code>as.character</code>	1, 2, ... $\rightarrow$ "1", "2", ... FALSE $\rightarrow$ "FALSE" TRUE $\rightarrow$ "TRUE"

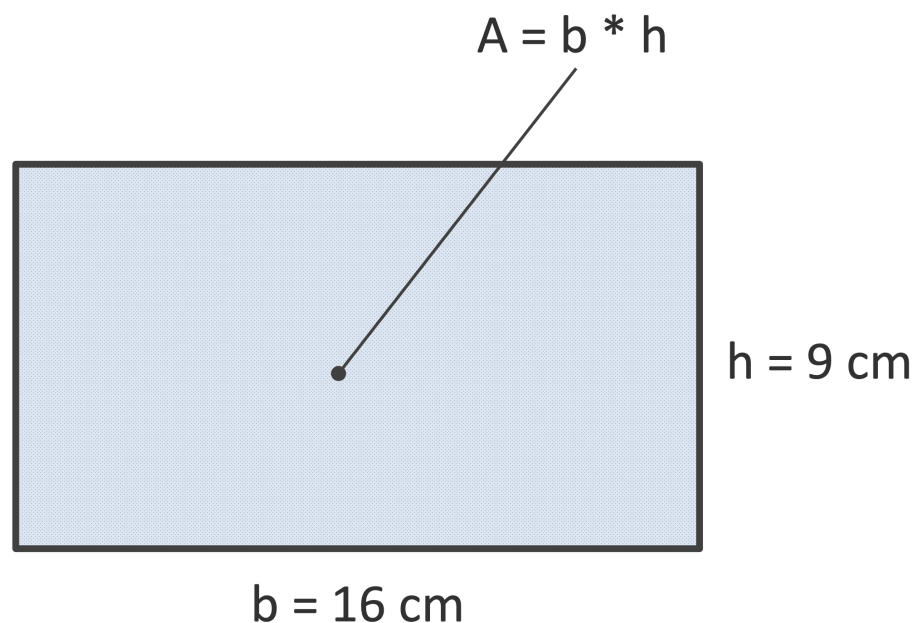
Source: [https://cran.r-project.org/doc/contrib/Paradis-rdebuts\\_en.pdf](https://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf)

23/51

# Exercises



## Ex. 1.1



- Initialize two *numeric variables* called  $b$  and  $h$  with the length of the base and height of the rectangle depicted above, respectively.
- Initialize a *numeric variable* called  $A$  with the value of the area of the rectangle computed on the fly.
- Check if the area is bigger than  $100 \text{ cm}^2$ .

## Ex. 1.2

```
Name <- "Maria"  
Age <- "20"  
PhD <- "TRUE"
```

What is the class of the variables above?

1. *Name*: character; *Age*: numeric; *PhD*: character
2. *Name*: character; *Age*: character; *PhD*: character
3. *Name*: character; *Age*: numeric; *PhD*: logical
4. None of the above

## Ex. 1.3

```
x <- 5  
y <- 0  
z <- x/y
```

What is the value of  $z$ ?

1. -Inf
2. NA
3. NaN
4. None of the above

# Data structures

# Data structures

In R, there are different data structures, including:

Data structure	Can contain different data types?
Vector	No
Matrix	No
Factor	No
List	Yes
Data.frame	Yes

# Data structure: vector

Vectors can be build with `c` and their length can be assessed with the function *length*

```
( x <- c(1, 2, 3, 4) )
```

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

```
( y <- c("a", "b", "c") )
```

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

```
length(y)
```

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

# Data structure: vector

Vectors can contain a single data type only, so beware of conversions

```
(z <- c("a", "b", 1, 2))
```

```
## [1] "a" "b" "1" "2"
```

```
(v <- c(x, y))
```

```
## [1] "1" "2" "3" "4" "a" "b" "c"
```

What are the classes of *z* and *v*?

# Fast ways to build vectors

```
( j <- 1:4 ) # Seq. of integers
```

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

```
( x <- seq(0, 1, 0.1) ) # Seq. of numbers (from, to, step)
```

```
## [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```
( y <- rep("A", 5) ) # Seq. of repeated numbers or characters
```

```
## [1] "A" "A" "A" "A" "A"
```

```
( z <- c(rep("A", 3), rep("B", 5)) ) # Vector combination
```

```
## [1] "A" "A" "A" "B" "B" "B" "B" "B"
```



# Assign and query vector names

```
( age <- c(19, 30, 20) )
```

```
## [1] 19 30 20
```

```
names(age) <- c("Mary", "John", "Lisa") # Assign
```

```
age
```

```
## Mary John Lisa
```

```
##    19    30    20
```

```
names(age) # Query
```

```
## [1] "Mary" "John" "Lisa"
```

# Data structure: matrix

2-dimensional variable that can be build with the function *matrix*

```
( M <- matrix(1:6, ncol=3, byrow=FALSE) ) # Fill by columns (def.)
```

```
##      [,1] [,2] [,3]  
## [1,]    1    3    5  
## [2,]    2    4    6
```

```
( N <- matrix(1:6, ncol=3, byrow=TRUE) ) # Fill by rows
```

```
##      [,1] [,2] [,3]  
## [1,]    1    2    3  
## [2,]    4    5    6
```

# Matrix rows and columns

```
M <- matrix(1:6, nrow=2, byrow=FALSE)  
dim(M)
```

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

```
ncol(M)
```

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

```
nrow(M)
```

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

# Matrix row and column names

```
colnames(M) <- c("Sample_A", "Sample_B", "Sample_C") # Assign  
rownames(M) <- c("Gene1", "Gene2") # Assign  
M
```

```
##      Sample_A Sample_B Sample_C  
## Gene1      1      3      5  
## Gene2      2      4      6
```

```
colnames(M) # Query
```

```
## [1] "Sample_A" "Sample_B" "Sample_C"
```

```
rownames(M) # Query
```

```
## [1] "Gene1" "Gene2"
```

# Data structure: factor

Categorical variable that can be built with the function *factor*

```
strain <- factor(c("WildType", "WildType", "Mutant", "Mutant"))  
strain
```

```
## [1] WildType WildType Mutant   Mutant  
## Levels: Mutant WildType
```

```
length(strain)
```

```
## [1] 4
```

```
levels(strain)
```

```
## [1] "Mutant" "WildType"
```

# Factor conversion

```
( x <- as.factor(c(10, 2, 2, 3)))
```

```
## [1] 10 2 2 3
```

```
## Levels: 2 3 10
```

```
( y <- as.numeric(x) ) # This does NOT work
```

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

```
( z <- as.numeric(as.character(x)) ) # This works
```

```
## [1] 10 2 2 3
```

# Data structure: list

A list is a structure that can contain different objects (including lists!)

It can be created with the function *list*

```
v <- 1:3  
M <- matrix(1:4, ncol=2)  
( myList <- list(myVec=v, myMat=M) )
```

```
## $myVec  
## [1] 1 2 3  
##  
## $myMat  
##      [,1] [,2]  
## [1,]    1    3  
## [2,]    2    4
```

# List names and length

```
length(myList)
```

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

```
names(myList)
```

```
## [1] "myVec" "myMat"
```

```
nrow(myList)
```

```
## NULL
```

```
ncol(myList)
```

```
## NULL
```



# Data structure: data.frame

A data.frame is a 2-dimensional object (“matrix-like”) that can contain different types of data on its columns.

It can be created with the function *data.frame*

```
( DF <- data.frame(name=c("Mary", "John", "Lisa"),  
  age=c(19, 30, 20),  
  city=c("New York", "Seattle", "New York"),  
  stringsAsFactors=FALSE) )
```

```
##   name age   city  
## 1 Mary  19 New York  
## 2 John  30  Seattle  
## 3 Lisa  20 New York
```

# Data.frame rows and columns

```
dim(DF)
```

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

```
nrow(DF)
```

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

```
ncol(DF)
```

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

```
colnames(DF)
```

```
## [1] "name" "age" "city"
```

# Exercises

## Ex. 1.4

```
x <- c("a", "b", "c")  
y <- seq(1,10)  
z <- c(x,y)
```

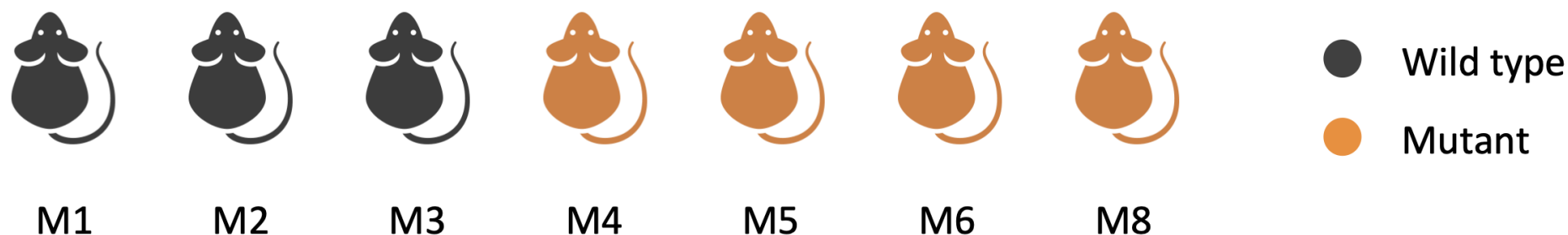
What is the *class* of *z*?

1. Character and numeric
2. Numeric
3. Character
4. NA

## Ex. 1.5

Build a *numeric vector* containing all even numbers greater than 0 and that do not exceed 100.

## Ex. 1.6



- Build a *vector of factors* storing the information about which mice is wild type ("WT") and or mutant ("MU") in the figure above (try to use *rep*).
- Assign mice identifiers (M1, M2, ...) to the same vector (use *names*).

## Ex. 1.7

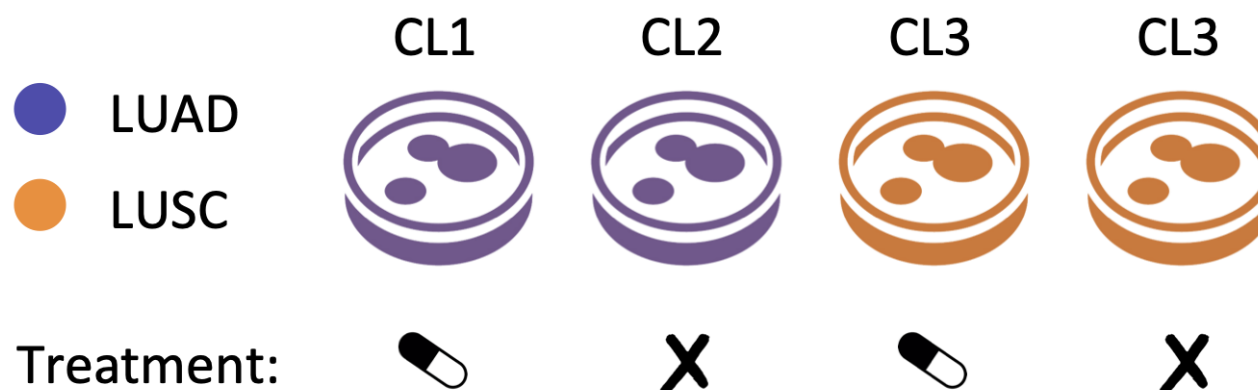
- Initialize a *matrix* with two rows and 3 columns that has all “1” on the first row and all “2” on the second row.
- Give it column and row names as you wish.
- Check its dimensions.

## Ex. 1.8

- Initialize a *numeric vector* to store the year in which you got your driving licence and the year you got your first own car (NA values are possible). Set appropriate names.
- Initialize a *character vector* to store the names of your favorite cities in Europe (as many as you like).
- Save both vectors in a *list* using meaningful names (lists can contain different objects with different sizes).



## Ex. 1.9



- Imagine you have an experiment with 4 lung cancer cell lines: 2 from adenocarcinomas (LUAD) and 2 from squamous cell carcinoma (LUSC).
- One cell line from each is treated with a drug, the others are untreated, as shown in the figure above
- Save in a *data.frame* the info about the experiment: cell line identifier, lung-cancer subtype, and treatment.
- Check how many rows and columns the *data.frame* has.

# Useful resources

R for beginners: [https://cran.r-project.org/doc/contrib/Paradis-rdebuts\\_en.pdf](https://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf)

Stackoverflow (Ask a Question): <https://stackoverflow.com>

# Solutions

Ex. 1.1

Ex. 1.2

Ex. 1.3

Ex. 1.4

Ex. 1.5