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 the life sciences

Why?

- Its is free
- It supports reproducible analysis
- 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 and info: https://github.com/FFinotello/Rcourse

Note: to download a text file, right-click on "Raw" and select "Save link as".

Evaluation: based on a data analysis project to be described in a short report and sent by e-mail.

How to ask questions

- Chat
- Collaborative Google Doc
- E-mail

Installation

Install first R and then Rstudio

To install **R** (version ≥ 4):

https://www.r-project.org

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

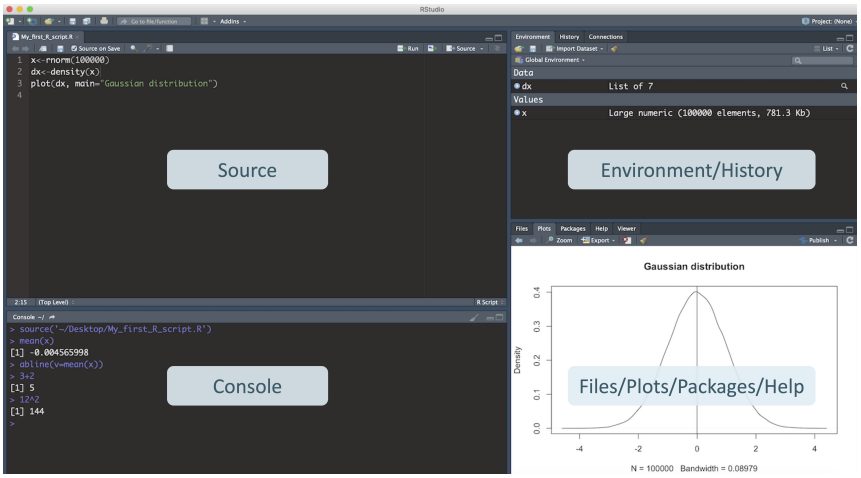
To install **RStudio Desktop**:

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:



From the toolbar: View > Panes > Show All Panes

The RStudio console

In the console you can type commands and see outputs.

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

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

```
log10(100) - sqrt(9) / 3
```

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

```
# Any text preceded by a hashtag is not evaluated (= comment!)
```

Variables and data types

Variables

Variables: 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
```

Variable names

- *Should* be short but descriptive
- *Must* start with a letter
- *Must not* contain special characters
- *Cannot* be reserved words

Reserved words: words with special meaning in R that cannot be used as identifiers. Examples: if, break, Inf, TRUE

👍 *cellFrac*, *cell_frac*, *cell.frac*, *cellfrac_1*

👎 *cell-frac*, *1cellfrac*, *TRUE*

```
(T <- 5 * 2 )
```

```
## [1] 10
```


Case-sensitive programming

R is case-sensitive

```
# Body mass index (BMI)
# defined as: weight/height^2
BMI <- 80/1.90^2

BMI
```

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

```
bmi
```

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

Operations

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

```
height <- 1.90
weight <- 80
( BMI <- weight/height^2 )
```

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

```
x <- 6
y <- 2
x <- 4
x + y
```

```
## [1] 6
```


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
var1
```

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

Infinite: Inf

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

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

```
( var3 <- -10/0 )
```

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

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	`as.numeric`	`FALSE` → 0
		`TRUE` → 1
		`"1"`, `"2"`, ... → 1, 2, ...
		`"A"`, ... → `NA`
logical	`as.logical`	0 → `FALSE`
		other numbers → `TRUE`
		`"FALSE"`, `"F"` → `FALSE`
		`"TRUE"`, `"T"` → `TRUE`
		other characters → `NA`
character	`as.character`	1, 2, ... → `"1"`, `"2"`, ...
		`FALSE` → `"FALSE"`
		`TRUE` → `"TRUE"`

Source: https://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf

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 the *c* function and their length can be assessed with the *length* function

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

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

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

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

```
length(y)
```

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


Vectors and data types

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*?

Creating numeric vectors

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

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

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

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


Creating vectors with *rep*

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

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

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

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

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"
```


Accessing vectors

You can access the elements of a vector by using *logical* or *numeric* indexes (positive and negative) between brackets:

```
days <- c("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun")

days[c(1,6,7)]
```

```
## [1] "Mon" "Sat" "Sun"
```

```
weekend <- days[c(FALSE, FALSE, FALSE, FALSE, FALSE, TRUE, TRUE)]

weekend <- days[-seq(1,5)] # Negative ind. of elements to be removed
```

Accessing vectors

To access vector elements, you must use square brackets.
Round brackets are for functions.

```
age <- c(11, 12, 18, 20, 45, 2, 33)

age[1]
```

```
## [1] 11
```

```
age(1)
```

```
## Error in age(1): could not find function "age"
```

```
mean(age)
```

```
## [1] 20.14286
```


Accessing vectors

Vector elements can be also accessed using their names, when initialized

```
age <- c(11, 12, 18, 20)
names(age) <-c("John", "Lisa", "Maria", "Markus")

age
```

```
##   John   Lisa  Maria Markus
##     11     12     18     20
```

```
age[c("John", "Maria")]
```

```
##  John Maria
##    11    18
```

Subsetting vectors

You can extract the elements of a vector that satisfy a certain condition:

```
age <- c(11, 12, 18, 20, 45, 2, 33)

age[age>=18] # Logical index
```

```
## [1] 18 20 45 33
```

```
age[which(age>=18)] # Numerical index
```

```
## [1] 18 20 45 33
```


Manipulating vectors

You can change the elements of a vector:

```
x <- c(11, 12, -18, 20, -45, -2, 33)

x[1] <- 35 # Subsitute the first element

x
```

```
## [1]  35  12 -18  20 -45  -2  33
```

```
x[x<0] <- 0 # Set to 0 all negative elements

x
```

```
## [1] 35 12  0 20  0  0 33
```

Data structure: factor

A **factor** is 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 <- 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
```