

# Introduction to Programming in R

**Ivan G. Costa & Tiago Maie**

Institute for Computational Genomics

Joint Research Centre for Computational Biomedicine

RWTH Aachen University, Germany

# Organization

---

Slides and exercises are available online:

<https://www.costalab.org/bioinformatics-in-r-2024/>

## **Schedule: 11/11 to 9/12**

- 9:30 - 12:00 - Theory and practices
- 13:00 - 17:00 - **Mandatory (30% of grade)** and optional exercises available on webpage

## **Final Project:**

- 16.12 - 9:30-17:00 - Analysis of a gene expression data & presentation (70% of grade)
- groups of 2-3 people

**Important:** M.Sc. and B.Sc. students need to send scripts with solutions for all **mandatory** exercises by the end of the day to [courses@costalab.org](mailto:courses@costalab.org)

---

# Programming, Language & Algorithms

---

## **What is an algorithm?**

- finite set of well defined and unambiguous commands to solve a task.

## **Programming language**

- vocabulary and set of instructions to command a computer
-

# Algorithm Example - “Cake baking”

---



- Prepare a cake pan by spraying with baking spray or buttering and lightly flouring. Next, combine flour, baking powder, baking soda, and salt in a large bowl and set the mix aside. Add 3 eggs, one at a time, and mix just until combined. Add flour mixture and buttermilk, alternately, beginning and ending with flour. Preheat oven to 200 C. Pour the dough in a pan and bake it for 25-30 minutes until edges turn loose from pan and toothpick inserted into middle of cake comes out clean. Remove from the oven and allow to cool for about 10 minutes.
-

# Algorithm Analysis

---

## Algorithm Example - “Cake baking”



- Prepare a cake pan by spraying with baking spray or buttering and lightly flouring. Next, combine flour, baking powder, baking soda, and salt in a large bowl and set the mix aside. Add 3 eggs, one at a time, and mix just until combined. Add flour mixture and buttermilk, alternately, beginning and ending with flour. Preheat oven to 350° F, pour the dough in a pan and bake it for 25-30 minutes until edges turn loose from pan and toothpick inserted into middle of cake comes out clean. Remove from the oven and allow to cool for about 10 minutes.
- 

**Task - bake a cake**  
**Language - English**

# Algorithm Analysis

---

## Algorithm Example - “Cake baking”



- Prepare a cake pan by spraying with baking spray or buttering and lightly flouring. Next, combine flour, baking powder, baking soda, and salt in a large bowl and set the mix aside. Add 3 eggs, one at a time, and mix just until combined. Add flour mixture and buttermilk, alternately, beginning and ending with flour. Preheat oven to 350° F, pour the dough in a pan and bake it for 25-30 minutes until edges turn loose from pan and toothpick inserted into middle of cake comes out clean. Remove from the oven and allow to cool for about 10 minutes.
- 

**Task** - bake a cake  
**Language** - English  
**Exact** - ???  
**Well defined** - ???

# Algorithm Analysis

---

## Algorithm Example - “Cake baking”



- Prepare a cake pan by spraying with baking spray or buttering and lightly flouring. Next, combine flour, baking powder, baking soda, and salt in a large bowl and set the mix aside. Add 3 eggs, one at a time, and mix just until combined. Add flour mixture and buttermilk, alternately, beginning and ending with flour. Preheat oven to 350° F, pour the dough in a pan and bake it for 25-30 minutes until edges turn loose from pan and toothpick inserted into middle of cake comes out clean. Remove from the oven and allow to cool for about 10 minutes.

**Task** - bake a cake  
**Language** - English  
**Exact** - ???  
**Well defined** - ???

# Language & Algorithms

---

## Computer Language

- well defined commands.
  - tests to decide the next steps (if-else command)
  - tests for repeating commands until a condition is satisfied (while or repeat)
-



# My first algorithm- “Cake baking”

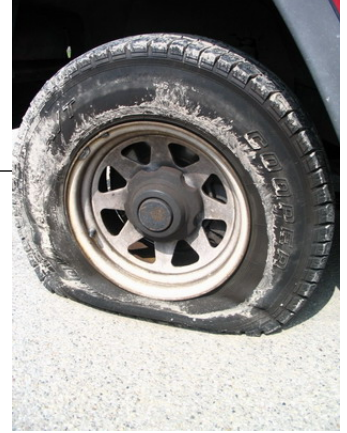
---



1. **If** baking spray is available **then**  
    prepare cake pan by spraying  
**else**  
    prepare pan by buttering and lightly flouring.
  2. **While** mixture is not creamy
    1. Combine flour, baking powder, baking soda, and salt in a large bowl
  3. **Repeat** 3 times
    1. Add an egg
    2. **While** mixture not homogeneous
      1. Mix dough.
  4. Pour the dough in a pan.
  5. Turn oven on.
  6. Wait until temperature is 200 C.
  7. Put pan into oven
  8. **While** “not” edges turn loose from pan or 30 minutes have passed.
    1. Wait 1 minute.
  9. Remove from the oven
  10. Wait for 10 minutes.
-

# Algorithms

---



## 1. Exercise:

1. Describe how to change a tire using “if” and “else” and while.

Equipment:

- jack, bolts, tire, wrench
-

# R Language

---

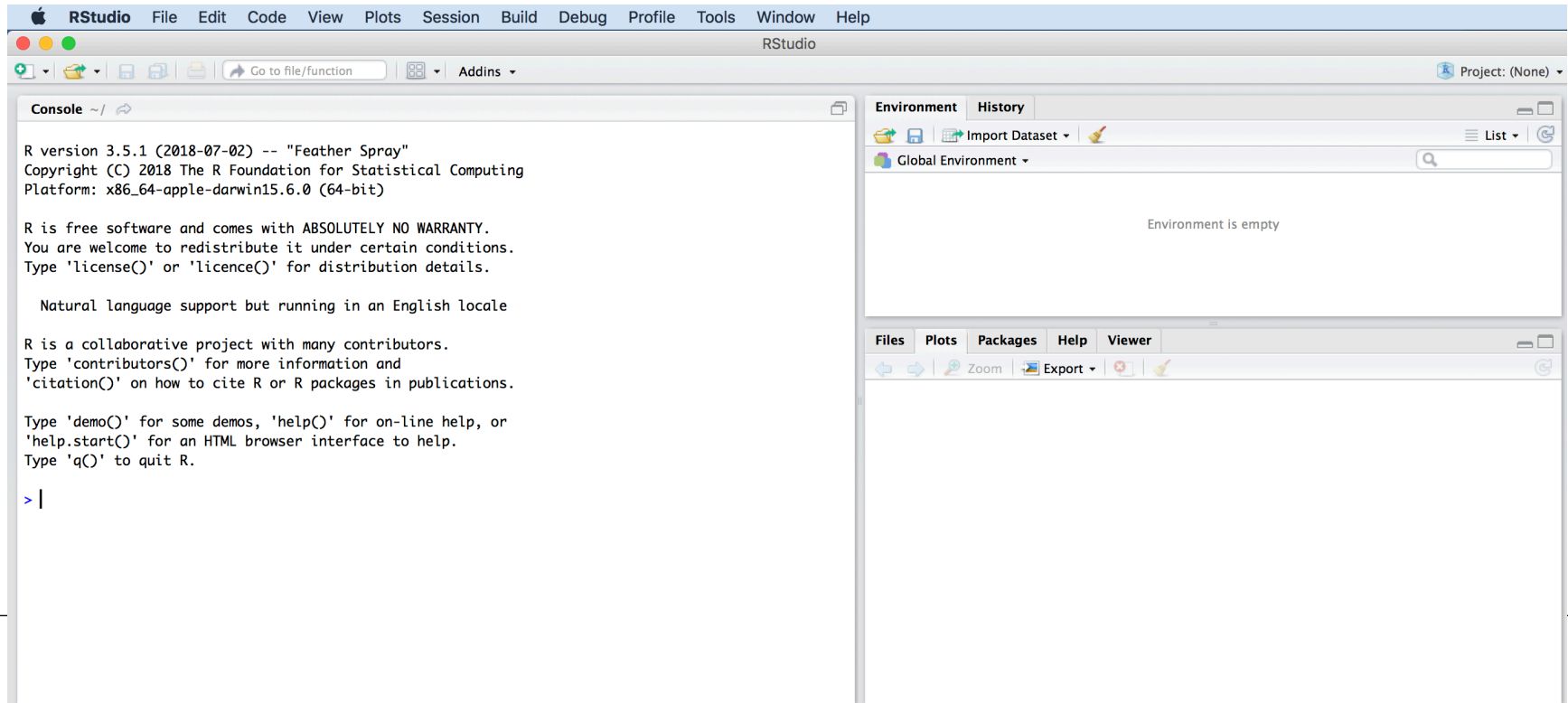


- Script based Programming language
- Focus of statistical data analysis
- Open source
- Contributing packages
  - Bioconductor (bioinformatics functions)
  - ggplot2 (plotting functions)
  - ...

# RStudio - Getting Started

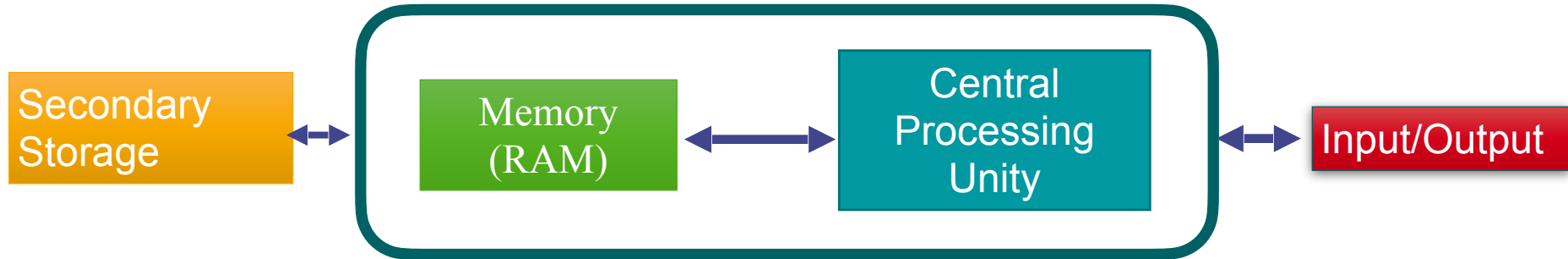


- Install RStudio  
<https://www.rstudio.com>
- Run RStudio



# Computer Architecture

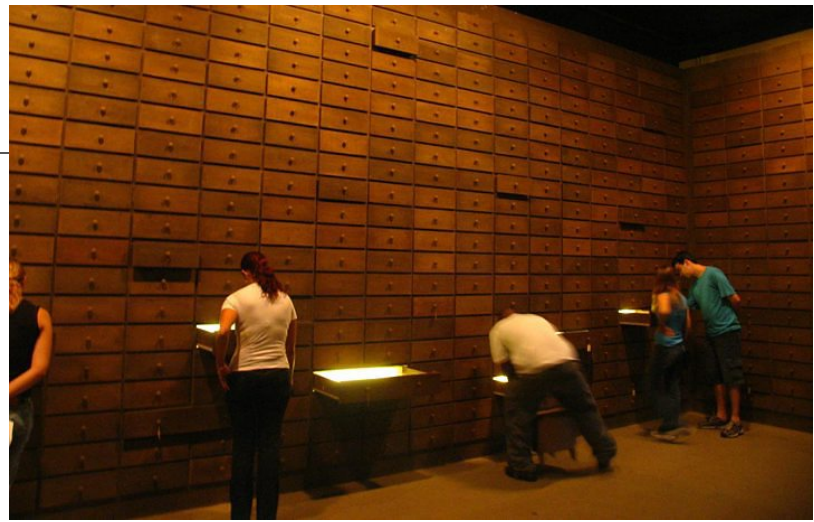
---



- **Central Processing Unity (CPU)**
  - execute mathematical operations
- **Memory (RAM)**
  - stores (limited) data for CPU (4-32 Gigabytes)
  - fast access but not permanent
- **Permanent Storage**
  - Slow access / large capacity (1.000 Gigabytes)
  - Permanent storage of files
- **Input/output**
  - monitor/keyboard/network card

# Memory (RAM)

---

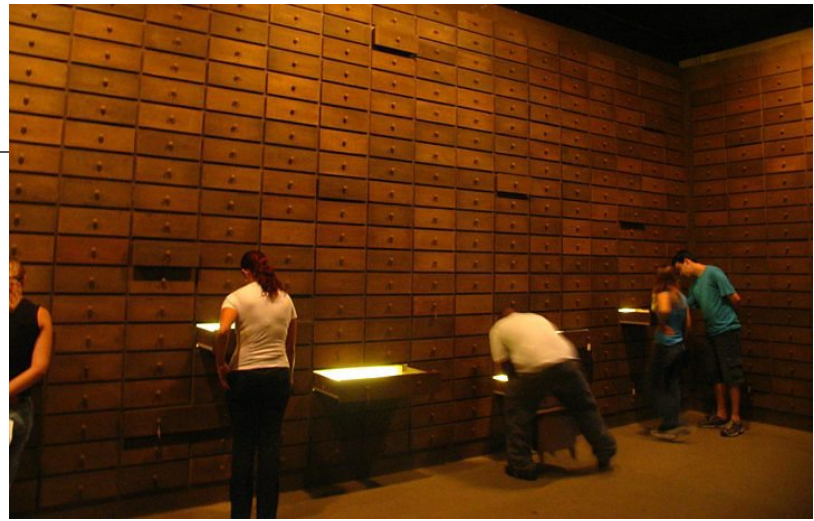


- A **computer memory** is like a large cabinet
- Each drawer can be used to keep information
  - i.e. names, telephones
- Each drawer holds a particular type of information
  - i.e. **strings, numbers**
- Computer knows the location of a particular drawer

# Variables

---

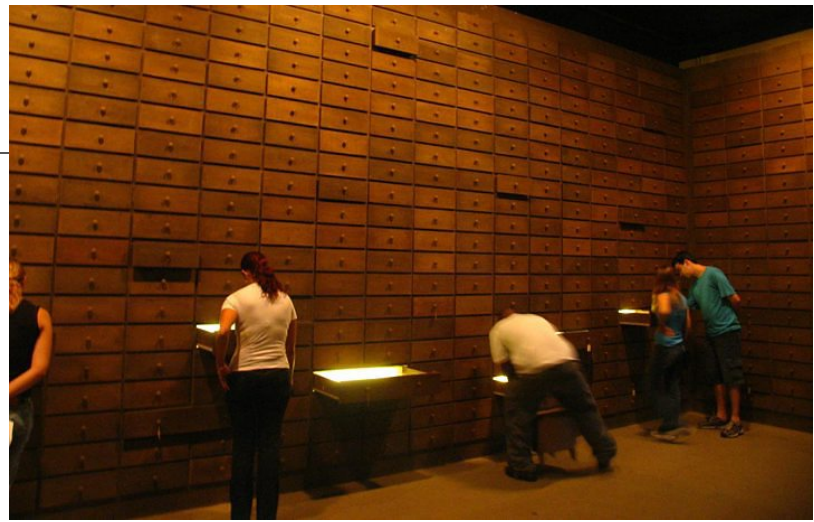
- Each drawer is called a **variable** (and we can give it a name)



# Variables

---

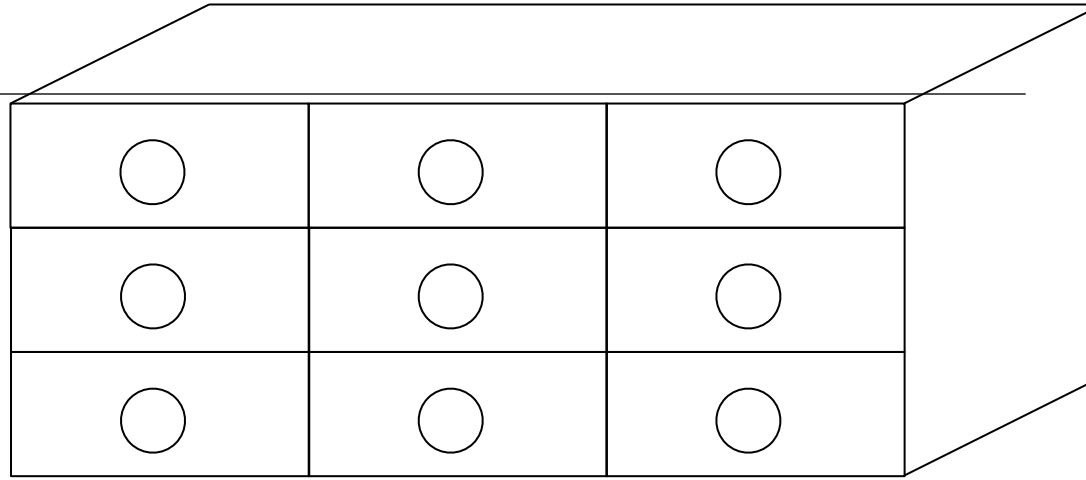
- Each drawer is called a **variable** (and we can give it a name)
- Each drawer has a **type**





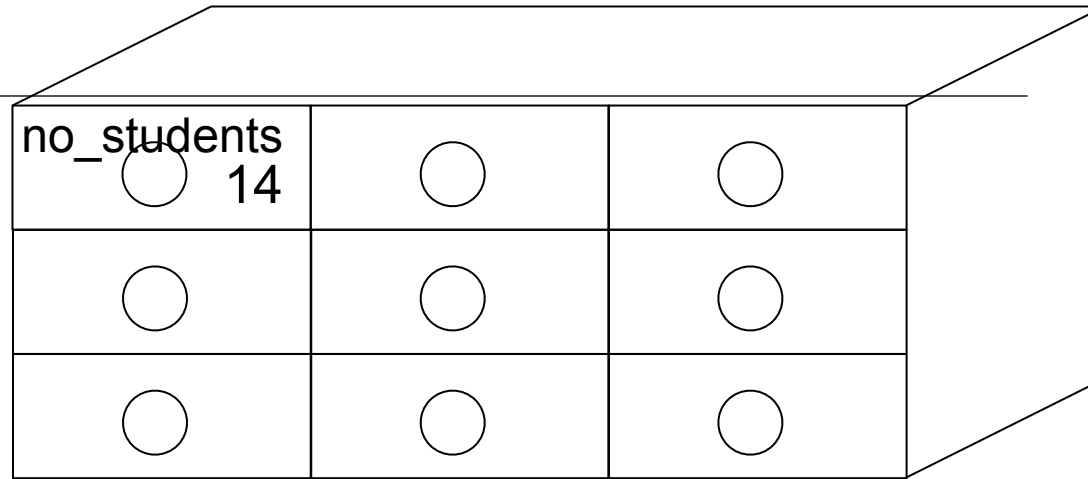
# Variables

- Each drawer is called a **variable** (and we can give it a name)
- Each drawer has a **type**



# Variables

- Each drawer is called a **variable** (and we can give it a name)



- Each drawer has a **type**
- In R, we have the following **types**:
  - **numeric**: no\_students = 14
  -

# Variables

- Each drawer is called a **variable** (and we can give it a name)

no_students ○ 14	course_name ○ "Bioinformatics in R"	○
○	○	○
○	○	○

- Each drawer has a **type**
- In R, we have the following **types**:
  - **numeric**: no\_students = 14
  - **character**: course\_name = "Bioinformatics in R"

# Variables

- Each drawer is called a **variable** (and we can give it a name)

no_students <input type="radio"/> 14	course_name <input type="radio"/> "Bioinformatics in R"	<input type="radio"/>
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

- Each drawer has a **type**
- In R, we have the following **types**:
  - **numeric**: no\_students = 14
  - **character**: course\_name = "Bioinformatics in R"
  - **boolean**: graduate\_level = TRUE

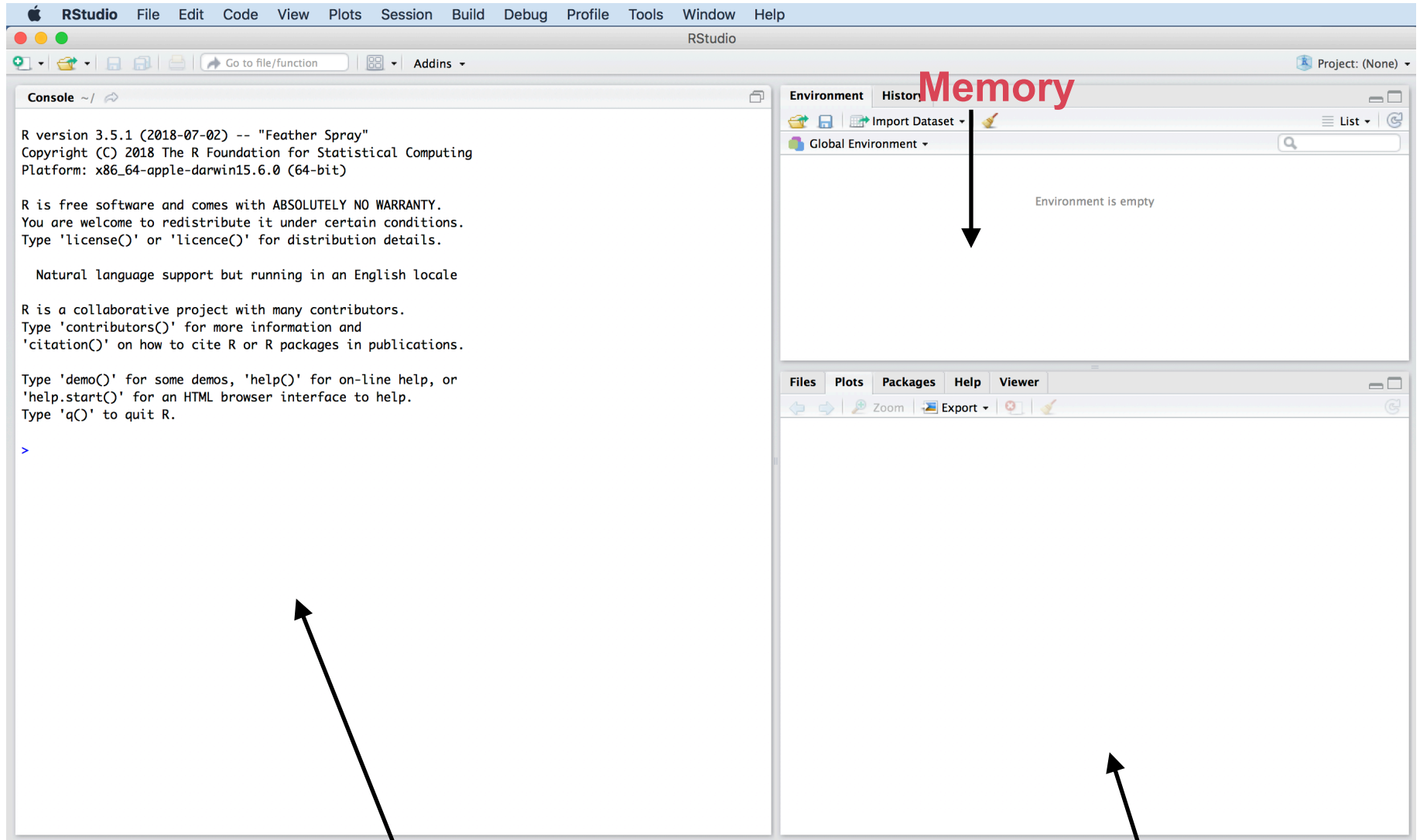
# Variables

- Each drawer is called a **variable** (and we can give it a name)

no_students ○ 14	course_name ○ "Bioinformatics in R"	○
○	○	○
○	○	○

- Each drawer has a **type**
- In R, we have the following **types**:
  - **numeric**: no\_students = 14
  - **character**: course\_name = "Bioinformatics in R"
  - **boolean**: graduate\_level = TRUE
  - **vectors**: (combination of several variables of same type): instructors = c("Ivan", "Tiago", "Johannes")
  - **Matrices**: ...

# RStudio & Memory



**R console: local to provide commands!**

**Graphs (not now)**

# Variables and Data Types

---

Single data can be stored in variables

- Data Types: "numeric", "character", "logical", ...

R console

```
x = 3; <enter>  
x; <enter>
```

*"x = 3;" means store the number  
"3" at a variable named "x"*

# Variables and Data Types

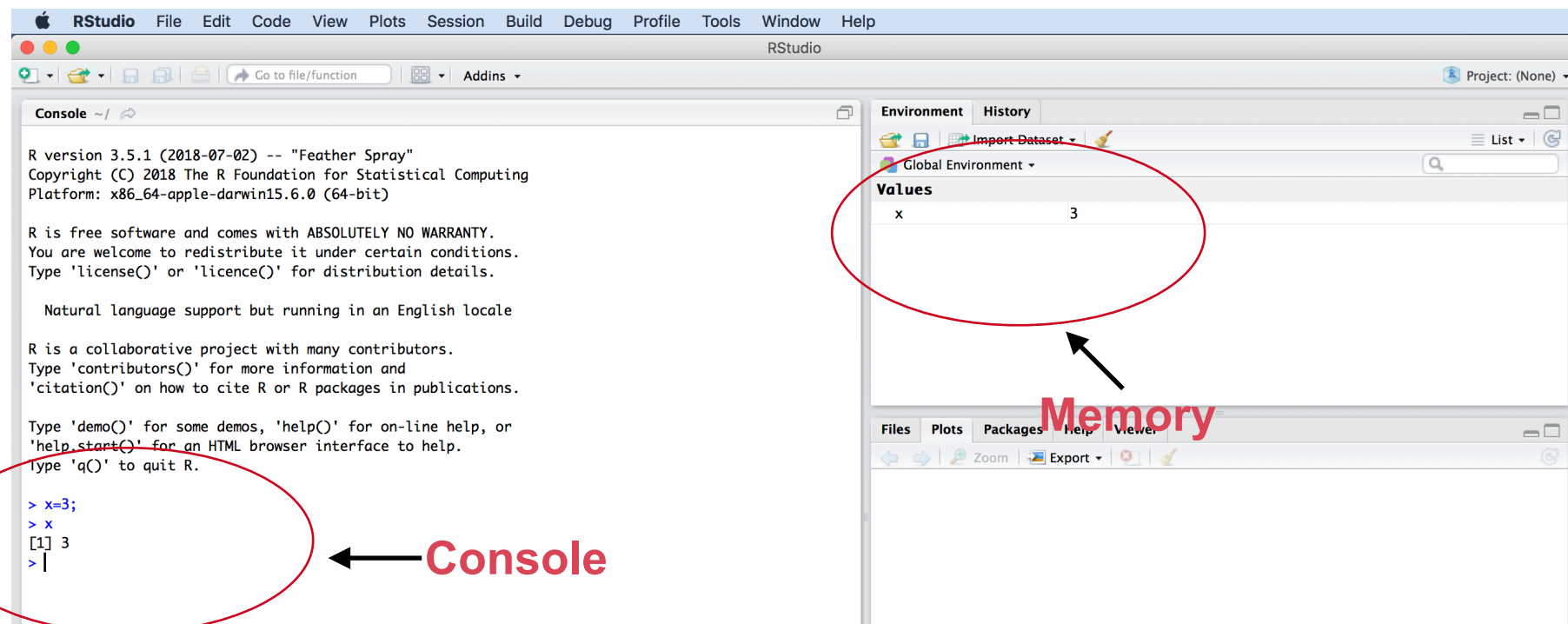
Single data can be stored in variables

- Data Types: "numeric", "character", "logical", ...

R console

```
x = 3; <enter>
x; <enter>
```

*"x = 3;" means store the number  
"3" at a variable named "x"*



The screenshot shows the RStudio interface. The console on the left displays the R version information and the execution of the commands `x = 3;` and `x;`. The Environment pane on the right shows the variable `x` with the value `3`. A red circle highlights the Environment pane, and a red arrow points to it with the word "Memory" written in red. Another red circle highlights the console output, and a red arrow points to it with the word "Console" written in red.

R version 3.5.1 (2018-07-02) -- "Feather Spray"  
Copyright (C) 2018 The R Foundation for Statistical Computing  
Platform: x86\_64-apple-darwin15.6.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.  
You are welcome to redistribute it under certain conditions.  
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.

```
> x=3;
> x
[1] 3
> |
```

Environment History  
Global Environment  
Values  
x 3

Files Plots Packages Help Viewer  
Zoom Export

Memory

Console



# R Console

---

R console

**">" indicates the console is waiting for a command**



```
>x = 3;  
>x;  
[1] 3  
>class(x);  
"numeric"
```

**Output of the command (no ">")**



**We will omit the <enter> from now on.**

# Variables and Data Types

---

Single data can be stored in variables

- Data Types: "numeric", "character", "logical", ...

R console

```
> x = 3
> x
[1] 3
> class(x)
"numeric"
> y = "Bioinformatics"
> y
"Bioinformatics"
```

```
> class(y)
"character"
> z = TRUE
> z
TRUE
> class(z)
"logical"
```

# Variables and Operations

---

We can apply arithmetic functions to variables

R console

```
> x = 3
> y = 4
> x + y
[1] 7
> x*y
[1] 12
> x/y
[1] 0.75
```

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation

# Variables and Operations

---

We can apply arithmetic functions to variables

R console

```
> x = 3
> y = 4
> x + y
[1] 7
> x*y
[1] 12
> x/y
[1] 0.75
```

```
> z = x + y
> z
[1] 7
> z = z + 2
> z
[1] 9
```

# Variables and Operations

---

We can apply logical functions to variables

& (and) and | (or)

R console

```
> x = 3
> y = 4
> x > y
[1] FALSE
> z = TRUE
> z & (x > y)
[1] FALSE
> z | (x > y)
[1] TRUE
```

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x   y	x OR y
x & y	x AND y
isTRUE(x)	test if X is TRUE

# Overview of RStudio

# Intro to RStudio

---

- RStudio is not R itself, but an **integrated development environment (IDE)**.
- It offers several panels for different purposes, such as console, help message, plots, history, scripts... etc.

```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 3 + 100 * 2
[1] 203
> █
```

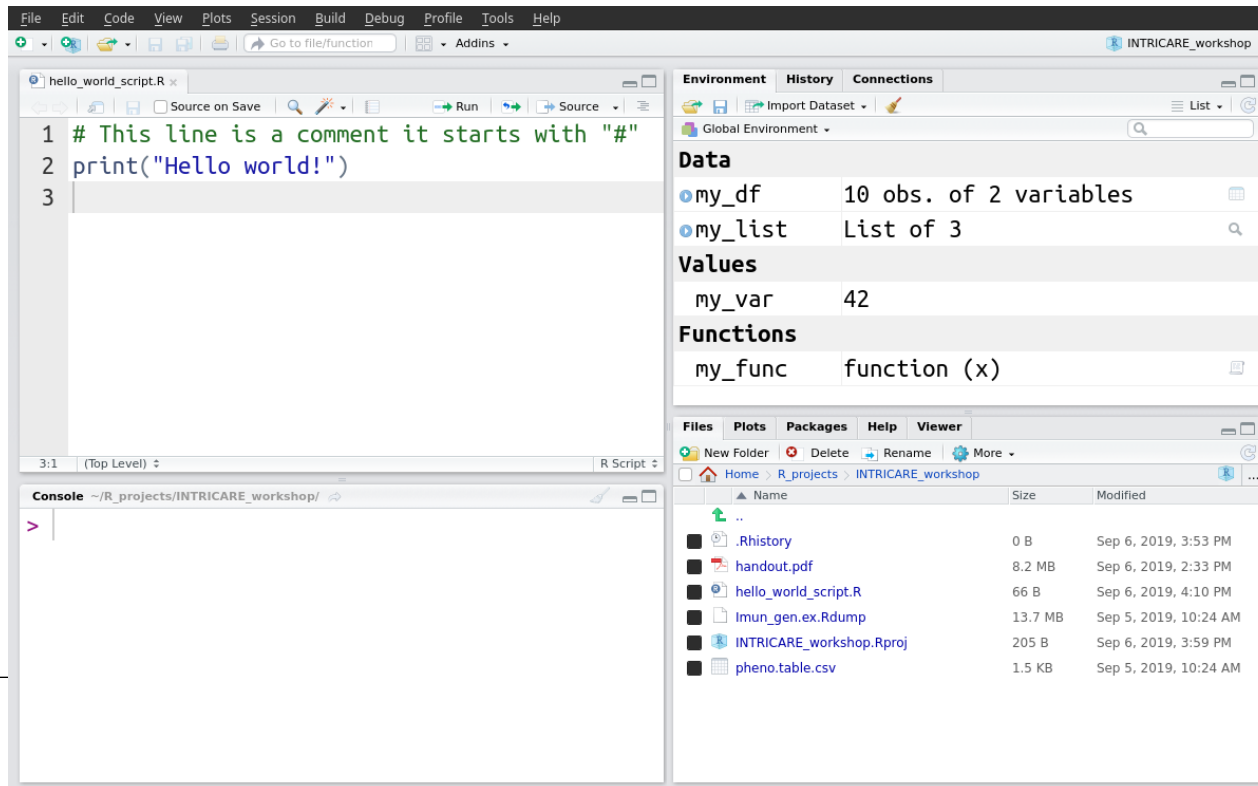




# RStudio - Getting Started



- Install RStudio  
<https://www.rstudio.com>
- Run RStudio



# RStudio - Organisation

The image shows the RStudio interface with four main components labeled in red text:

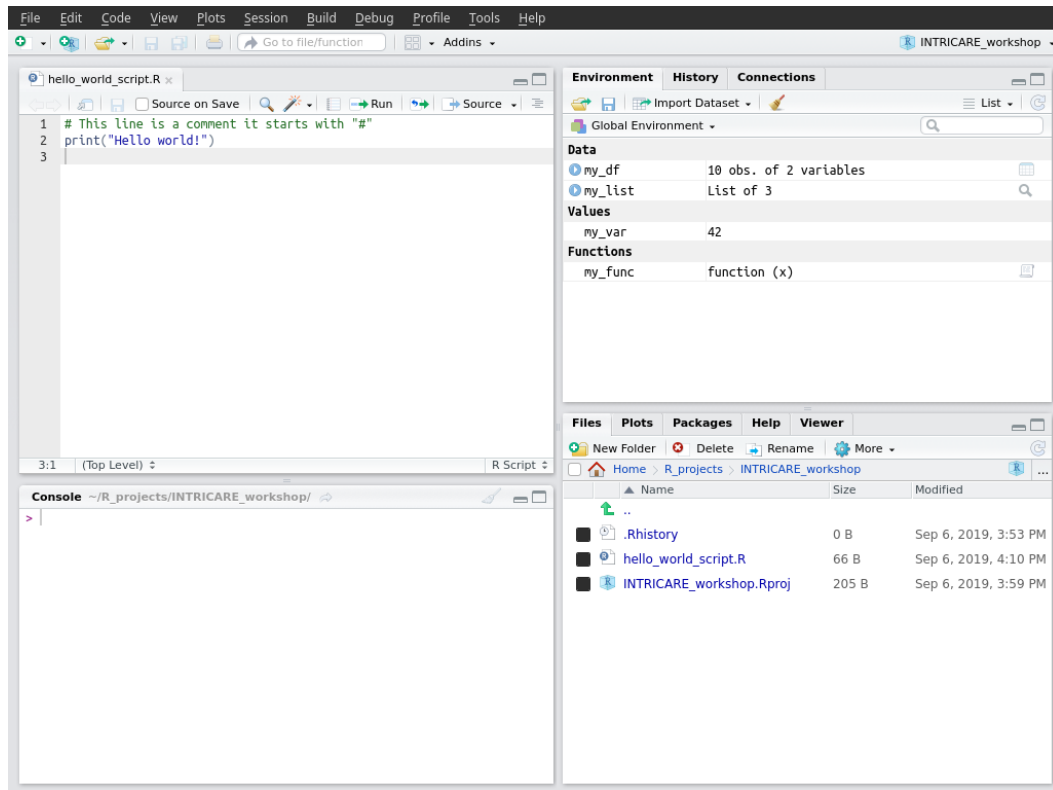
- Scripts:** The top-left pane shows a script named `hello_world_script.R` with the following code:

```
1 # This line is a comment it starts with "#"  
2 print("Hello world!")  
3
```
- Console:** The bottom-left pane shows the R console with the prompt `> |`.
- Environment:** The top-right pane shows the Environment window with the following content:

Global Environment	
<b>Data</b>	
my_df	10 obs. of 2 variables
my_list	List of 3
<b>Values</b>	
my_var	42
<b>Functions</b>	
my_func	function (*)
- Project folder:** The bottom-right pane shows the Files window with the following table:

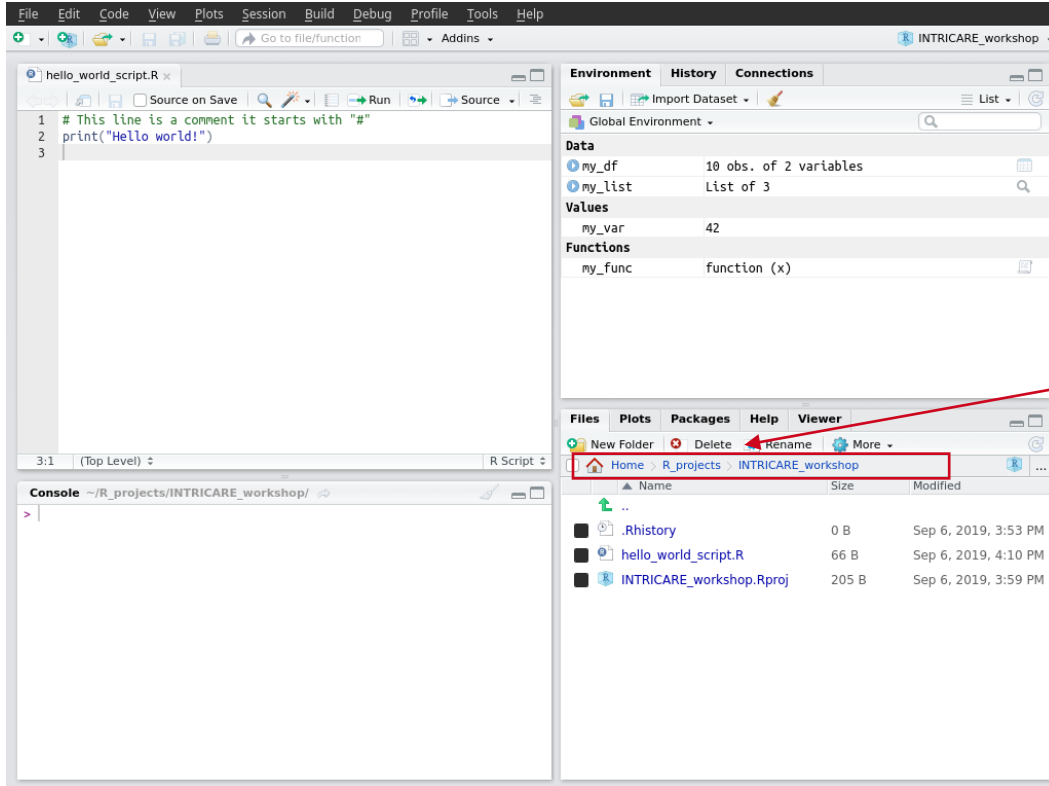
Name	Size	Modified
..		
.Rhistory	0 B	Sep 6, 2019, 3:53 PM
hello_world_script.R	66 B	Sep 6, 2019, 4:10 PM
INTRICARE_workshop.Rproj	205 B	Sep 6, 2019, 3:59 PM

# RStudio - Configure Project Directory



We need to configure the project directory:

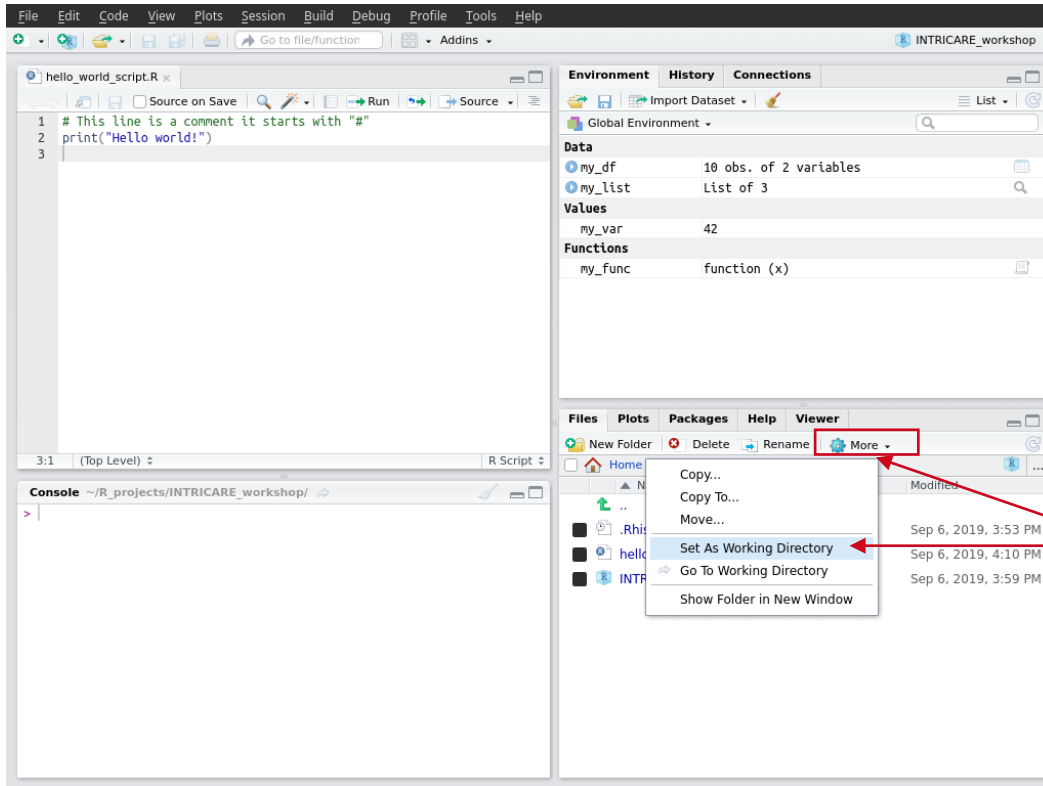
# RStudio - Configure Project Directory



We need to configure the project directory:

1 - navigate until folder with course files

# RStudio - Configure Project Directory

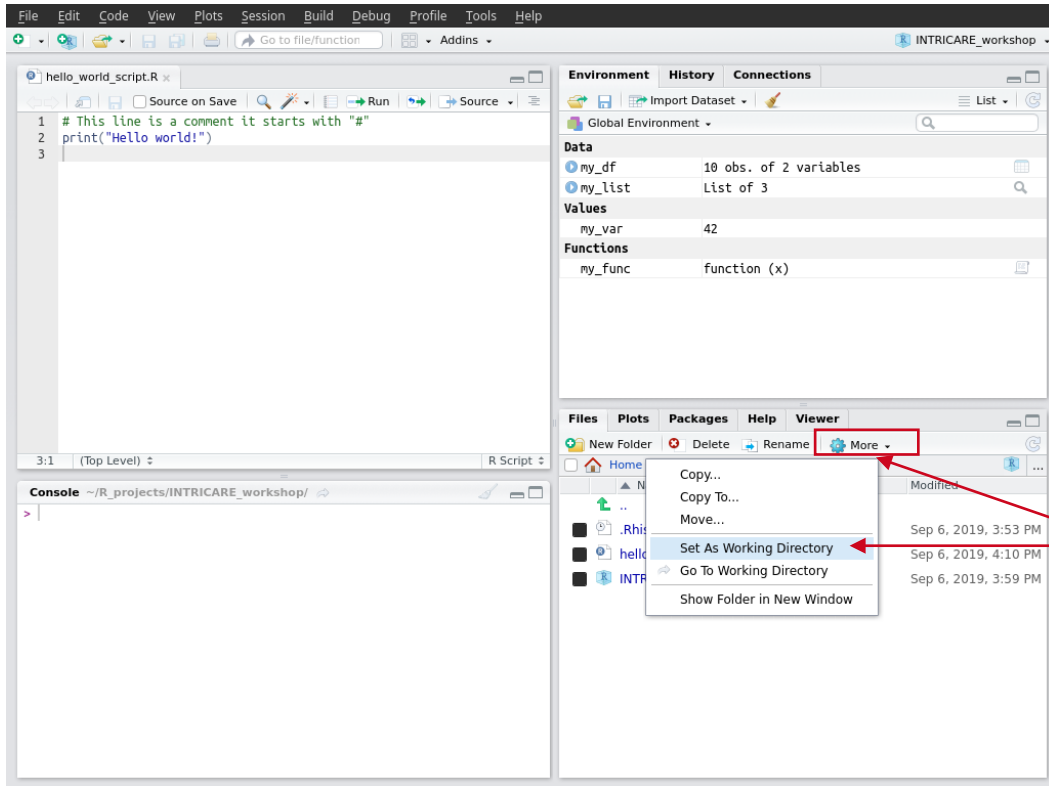


We need to configure the project directory:

**1** - navigate until folder with course files

**2** - select the "More" option and "Set as Working Directory"

# RStudio - Configure Project Directory



We need to configure the project directory:

**1** - navigate until folder with course files

**2** - select the "More" option and "Set as Working Directory"

**Now R Studio knows where to find files !**

# Exercise 1

---

- Use arithmetic operations to perform the following calculations

- 1 plus 3
- 3 minus 1
- 2 multiplied by 2
- 4 divided by 2
- 3 to the power of 2

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation

- Repeat the exercise but this time "save" the results of the operations (using variables)
-

## Exercise 2

---

- Use variables to store the amount of fruits in a shop. We have 5 green apples, 4 red apples, 10 bananas and 4 melons.
  - Write a code using variables to answer the following questions:
    - How many fruits are there is total?
    - How many apples?
-



## Exercise 3

---

- An apple costs 0.5 cents, a banana 1.0 euro, a melon 3 euros (use variables to store these!).
    - How much does it cost to buy all the apples in the shop?
    - How much does it cost to buy all the fruits in the shop?
-

## Exercise 4

---

- Use logical variables to answer the following questions.
    - Is buying all bananas cheaper than buying all apples?
    - You have 20 euros. Can you buy all apples?
-

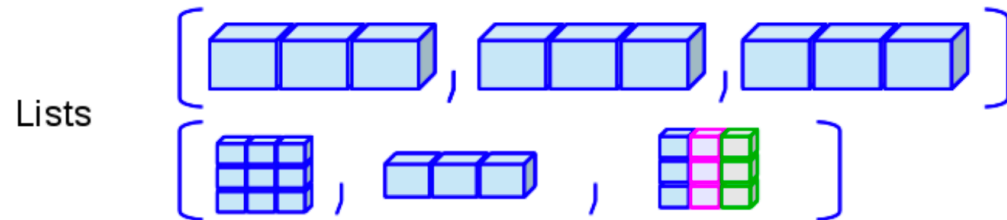
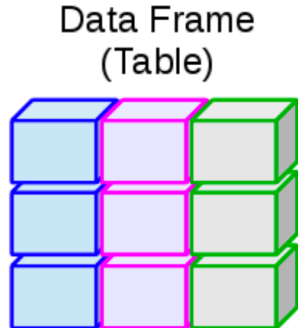
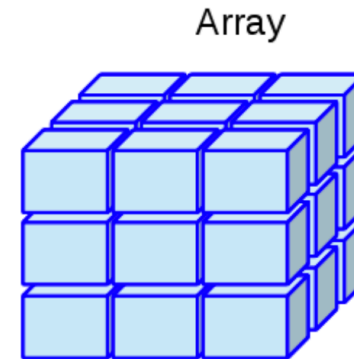
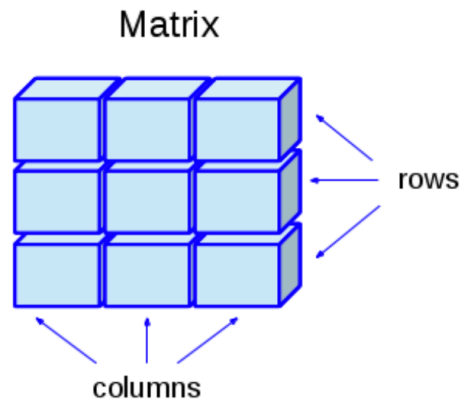
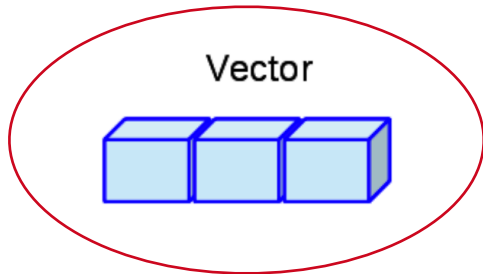
# Complex Data Structures

---

- Vector – variable containing a array of items of the same type
  - Lists - a vector where items can have distinct types (next class!)
  - Matrix – two dimensional vector with items of the same type
  - Data Frame – complex data structure for two dimensional data where columns can be of distinct type (as an excel sheet) (next class!)
-

# Complex Data Structures

---



# Vector

---

- Creating, accessing and updating vector

```
> v = c(3.2, 4.1, 1.9)
> v
[1] 3.2 4.1 1.9
> v[2]          # access 2nd position of vector
[1] 4.1
> v[3] = 10.4   #update 3rd position of vector
> v
3.2  4.1 10.4

> u = c(1,2,3)
> z = u + v     #sum 2 vectors (if size is the same)
> z
[1]  4.2  6.1 13.4
```

# Vector

---

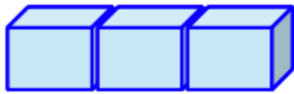
- Operations, functions and access

```
> length(z)      # function indicating size of vector
[1] 3
> 1:2            # vector with 1 and 2.
[1] 1 2
> z[1:2]        #subsetting vector (1st and 2rd pos.)
[1] 4.2 6.1
> z > 6         #logical operator
[1] FALSE  TRUE  TRUE
> z[z > 6]     # return all values greater than 6
[1] 6.1 13.4
```

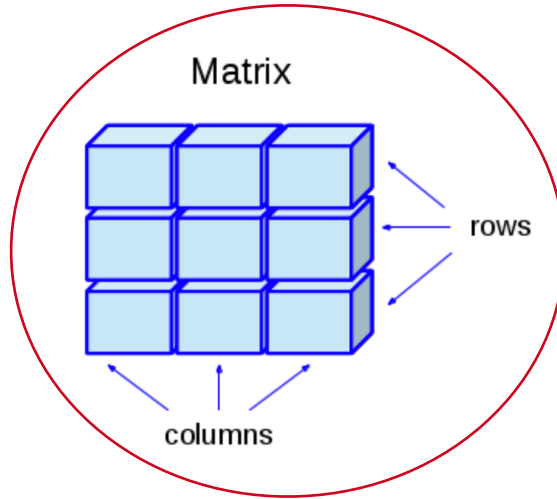
# Complex Data Structures

---

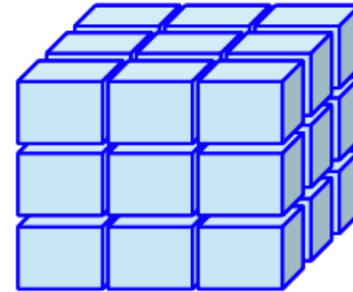
Vector



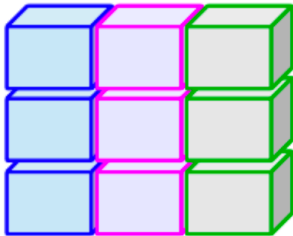
Matrix



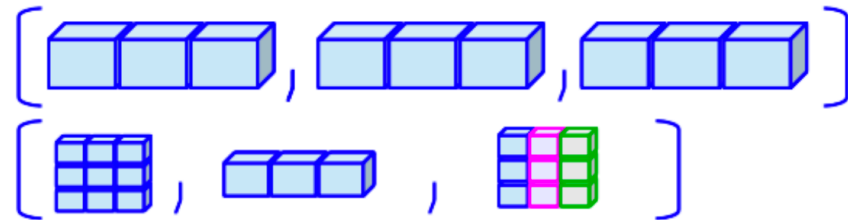
Array



Data Frame  
(Table)



Lists



# Matrix

---

- Matrix – two dimensional vector / same type

```
> m = matrix(1:12, 4, 3) # 4 by 3 matrix
> dim(m)                 # size of matrix
4 3
> m[1,]                  # show first row of matrix
[1] 1 5 9
> m[3,1]                 #show element at 3rd row / 1st column
[3]
> m
      [,1] [,2] [,3]
[1,] 1    5    9
[2,] 2    6   10
[3,] 3    7   11
[4,] 4    8   12
```



# Matrix

---

- Matrix – two dimensional vector / same type

```
> v1 = c(10,4,10) # a vector with 3 entries
> v2 = c(4,10,2)  # another 3 entry vector
> mat = rbind(v1,v2) # join two vectors as a matrix
> mat
      [,1] [,2] [,3]
v1    10    4   10
v2     4   10    2
```

# Matrix

- RStudio also helps visualisation of a matrix

The screenshot shows the RStudio interface. In the top-left pane, a matrix 'm' is displayed as a table with 4 rows and 3 columns. A red arrow points to this table with the word 'Matrix' in red text. The Environment pane on the right shows the variable 'm' with its dimensions and values. A red arrow points to the 'm' entry with the text 'click here!' in red. The Console pane at the bottom shows the R code used to create the matrix.

	V1	V2	V3
1	1	5	9
2	2	6	10
3	3	7	11
4	4	8	12

Showing 1 to 4 of 4 entries

```
R version 3.5.1 (2018-07-02) -- "Feather Spray"  
Copyright (C) 2018 The R Foundation for Statistical Computing  
Platform: x86_64-apple-darwin15.6.0 (64-bit)  
  
R is free software and comes with ABSOLUTELY NO WARRANTY.  
You are welcome to redistribute it under certain conditions.  
Type 'license()' or 'licence()' for distribution details.  
  
Natural language support but running in an English locale  
  
R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to cite R or R packages in publications.  
  
Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.  
  
> m = matrix(1:12, 4, 3)  
> View(m)  
> |
```

Environment History  
Global Environment  
Data  
m int [1:4, 1:3] 1 2 3 4 5 6 7 8 9 10 ...  
click here!

Files Plots Packages Help Viewer  
Zoom Export

# Matrix

---

- What happens if we have a large matrix?  
450.000 lines by 1000 samples?

```
> m = matrix(1:12, 450000, 1000) # 4 by 3 matrix
> dim(m) # size of matrix
[1] 450000 1000
> m[,1] # show first column of matrix
[1] 1 2 3 4 5 6 ...
```

# Matrix

---

- What happens if we have a large matrix?  
450.000 lines by 1000 samples?

```
> m = matrix(1:12, 450000, 1000) # 4 by 3 matrix
> dim(m) # size of matrix
[1] 450000 1000
> m[,1] # show first column of matrix
[1] 1 2 3 4 5 6 ...
```

- Large matrices use a lot of memory (1.7 GB)!

```
> remove(m) # remove m from memory
```

---

# Functions

# Functions

---

- A section of a program that perform a specific task
    - Takes values as input parameter and returns some new value (or performs an operation)
  - R defines several types of functions
    - math: log, exp, abs, sqrt, min, max, ...
    - array/matrix manipulation: length, dim, array, rep, ...
    - Read/write files: read.table, write.table, ...
  - Can be created by user or defined in contributing packages (tomorrow!)
-

# Example of Functions

```
> log2(4)
[1] 2
> m = matrix(1:12, 4, 3) # create a matrix
> dim(m)                # size of the data frame
[1] 4 3
> summary(m)           # statistics of the matrix columns
      V1                V2                V3
Min.   :1.00          Min.   :5.00          Min.   : 9.00
1st Qu.:1.75          1st Qu.:5.75          1st Qu.: 9.75
Median :2.50          Median :6.50          Median :10.50
Mean   :2.50          Mean   :6.50          Mean   :10.50
3rd Qu.:3.25          3rd Qu.:7.25          3rd Qu.:11.25
Max.   :4.00          Max.   :8.00          Max.   :12.00
> write.table(m, "mydata.txt")
# write matrix in a .txt file
> getwd()              # current working directory
```

# Functions and help

```
> help.start() #opens a page with manual, tutorials and
help search
> help("write.table") #show options for write.table
```

write.table {utils}

R Documentation

## Data Output

### Description

`write.table` prints its required argument `x` (after converting it to a data frame if it is not one nor a matrix) to a file or [connection](#).

### Usage

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",
            eol = "\n", na = "NA", dec = ".", row.names = TRUE,
            col.names = TRUE, qmethod = c("escape", "double"),
            fileEncoding = "")
```

```
write.csv(...)
write.csv2(...)
```

### Arguments

<code>x</code>	the object to be written, preferably a matrix or data frame. If not, it is attempted to coerce <code>x</code> to a data frame.
<code>file</code>	either a character string naming a file or a <a href="#">connection</a> open for writing. "" indicates output to the console.
<code>append</code>	logical. Only relevant if <code>file</code> is a character string. If <code>TRUE</code> , the output is appended to the file. If <code>FALSE</code> , any existing file of the name is destroyed.
<code>quote</code>	a logical value ( <code>TRUE</code> or <code>FALSE</code> ) or a numeric vector. If <code>TRUE</code> , any character or factor columns will be surrounded by double quotes. If a numeric vector, its elements are taken as the indices of columns to quote. In both cases, row and column names are quoted if they are written. If <code>FALSE</code> , nothing is quoted.
<code>sep</code>	the field separator string. Values within each row of <code>x</code> are separated by this string.



# Functions / Multiple Parameters

```
>help.start()      #opens a page with manual, tutorials and  
help search  
>help("write.table") #show options for write.table
```

write.table {utils}

R Documentation

## Data Output

### Description

`write.table` prints its required argument `x` (after converting it to a data frame if it is not one nor a matrix) to a file or [connection](#).

### Usage

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",  
           eol = "\n", na = "NA", dec = ".", row.names = TRUE,  
           col.names = TRUE, qmethod = c("escape", "double"),  
           fileEncoding = "")
```

```
write.csv(...)  
write.csv2(...)
```

```
> write.table(data, "mydata.txt", quote=FALSE, sep="-")
```

data.frame to be saved

file name

use quotes between names

separators between values

# Libraries

---

- In R the primary mechanism for distributing software (functions) is via packages
  - CRAN is the major repository for packages.
    - > `install.packages("packagename")` # install a new package
  - Bioinformatic packages are available at Bioconductor package.
    - > `install.packages("BiocManager")`
    - > `BiocManager::install(c("packagename"))`
  - Before using functions of a library they need to be opened.
    - > `library("packagename")`
-

# Example of library / saving excel table

```
> install.packages("openxlsx") # installing package
> library("openxlsx") # loading package in memory
> help("openxlsx") # description of package
> m = matrix(1:12, 4, 3) # create a matrix
> write.xlsx(as.data.frame(m), "mydata.xlsx")
# write matrix in a .xlsx file
> mydata = read.xlsx("mydata.xlsx") # read the file
and saves in another variable my data
> mydata
  V1 V2 V3
1  1  5  9
2  2  6 10
3  3  7 11
4  4  8 12
```

**Try using openxlsx to load an excel table from yourself!**

# Exercise 1

---

- Define a vector to store the amount of fruits and another one to store their prices.
    - There are 5 green apples, 14 red apples, 30 bananas and 4 melons
    - An apple costs 0.5 cents, a banana 1.0 euro, a melon 3 euros
  - Use vector operations/functions to calculate what is:
    - The total amount of fruits?
    - The total number of fruit types?
    - The total price of all fruits?
-

## Exercise 2

---

- Use functions or logical operators to answer the following questions:
    - Which fruit types have more than 5 units?
    - Which fruit types you can buy all items with 10 euros?
    - Which fruit type has the least amount of units?
-

## Exercise 3

---

Creating regular numeric sequences is a common task in statistical computing. You can use the `seq` function to create sequences.

1. Read the help page for `seq` by entering `help(seq)`.
  2. Generate a decreasing sequence from 50 to 1, then another sequence from 1 to 50.
  3. Use `seq` to generate a sequence of the even integers between one and ten.
-

## Exercise 4

---

- Create an integer vector that can be used to subset a vector named **vec** (see below) such that it will output the elements of **vec** in decreasing order. For the general case, read the help pages for **order** and **sort**.

```
> vec = c(1.1, 2, 100, 50, 60)
```

---

# Afternoon Exercise

---

- Check exercise in <https://www.costalab.org/bioinformatics-in-r-2024/>
  - See you all after lunch!
-



# Extra material

---

- More exercises at ...

[http://www.bioconductor.org/help/course-materials/2010/BioC2010/First\\_Steps\\_With\\_R\\_SOLUTIONS.pdf](http://www.bioconductor.org/help/course-materials/2010/BioC2010/First_Steps_With_R_SOLUTIONS.pdf)

(pages 1-17)

---

## Inst. for Computational Genomics

- Ivan G. Costa
- Tiago Maie
- Johannes Schoeneich

