Introduction to Programming in R

Ivan G. Costa
Institute for Computational Genomics
Joint Research Centre for Computational Biomedicine
RWTH Aachen University, Germany

Institute for Computational Genomics
01011011010
10100100101

RWTH AACHEN UNIVERSITY

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 350º 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 - back 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 350º 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 - back 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 350º 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 - back 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 can 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 offer
8. While “not” edges turn loose from pan or 30 minutes were pasted.
   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.

R Language

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

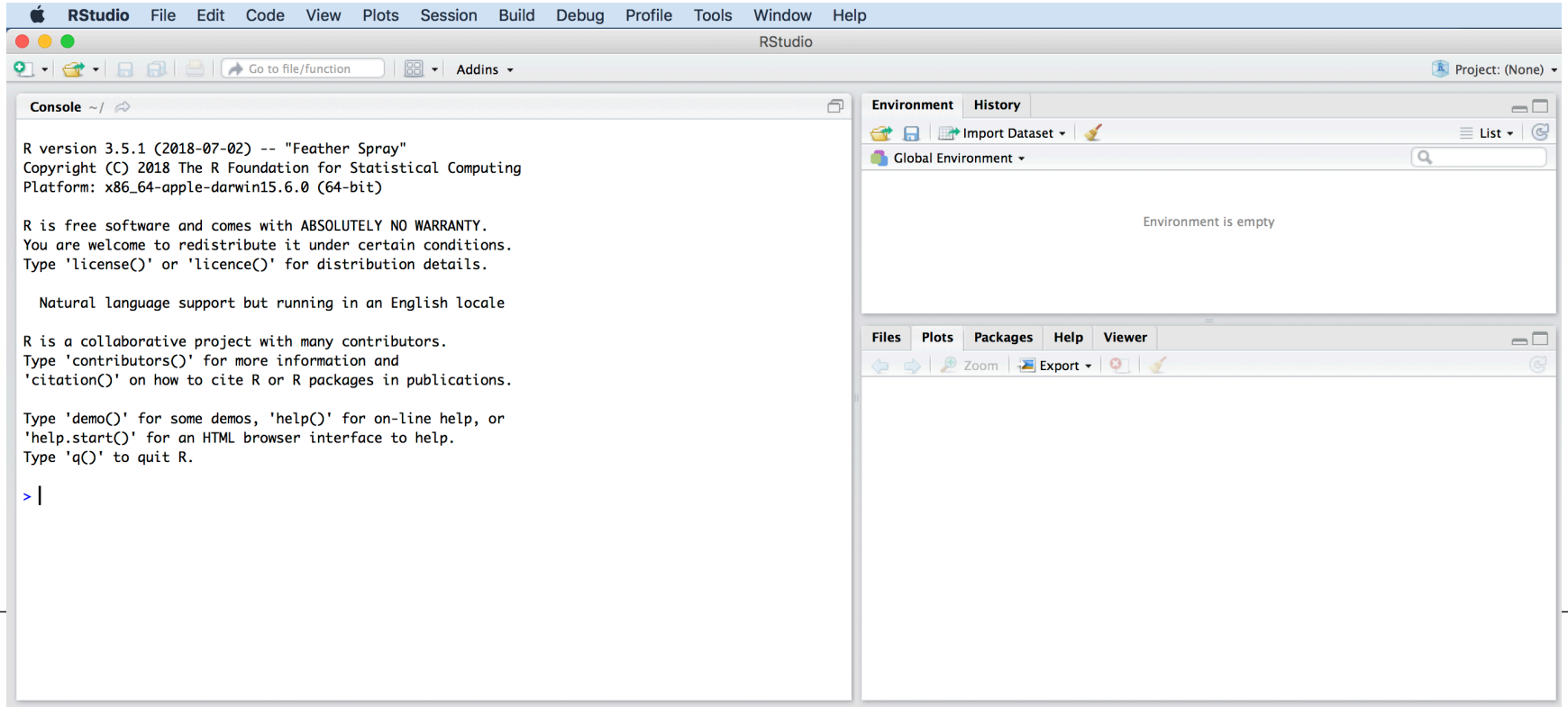
http://www.r-project.org/

RStudio - Getting Started

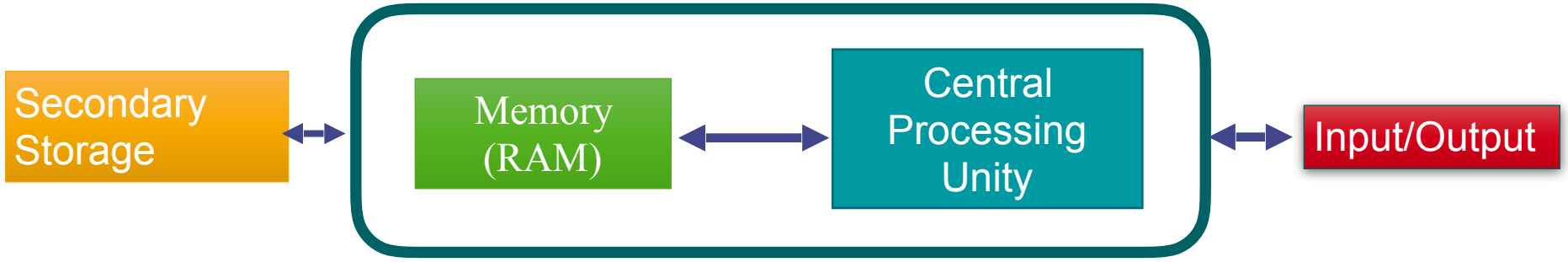
- Install RStudio

  https://www.rstudio.com

- Run RStudio

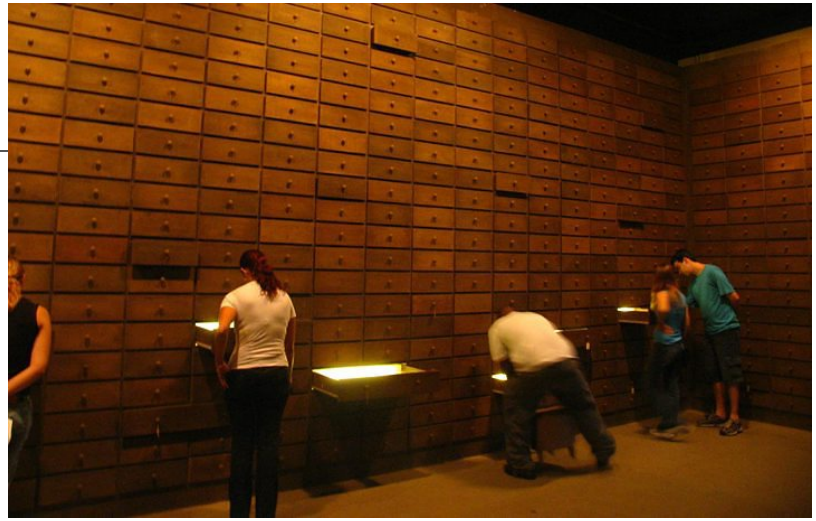


Computer Architecture



- **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

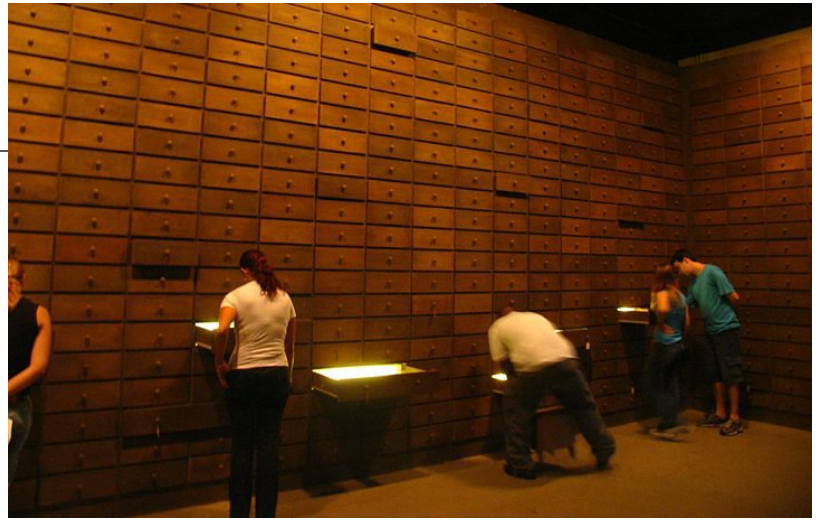
Memory (RAM)



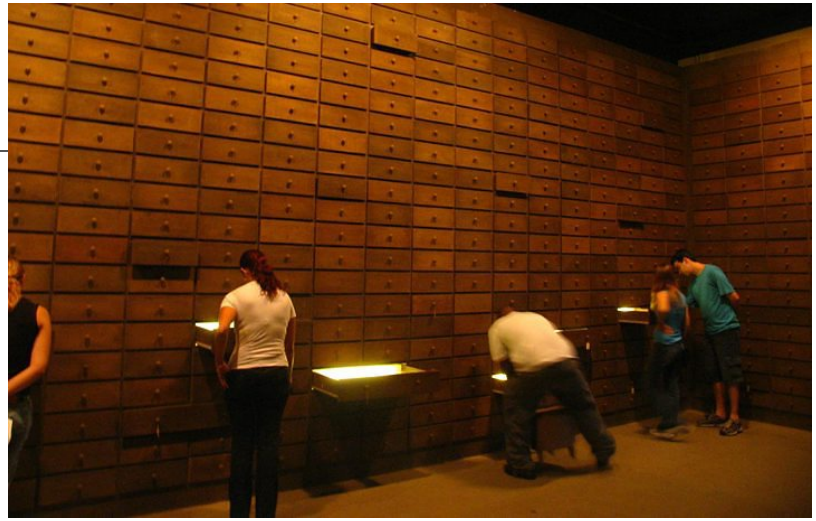
- A **computer memory** is like a large cabinet
- Each drawer can be used to keep information
  - i.e. names, telephones
- Each drawer holds a particular type of information
  - i.e. **strings**, **numbers**
- Computer knowns the location of a particular drawer

Variables

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



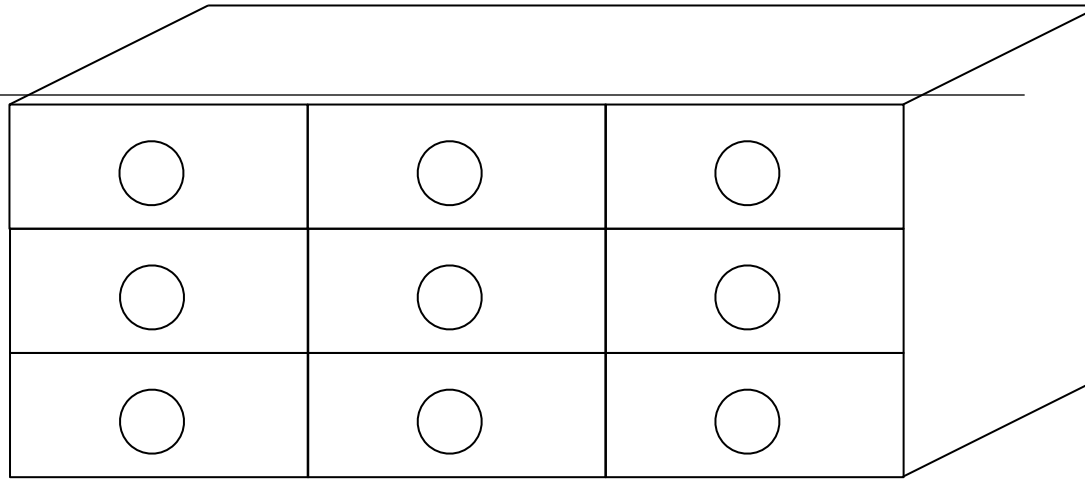
Variables



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

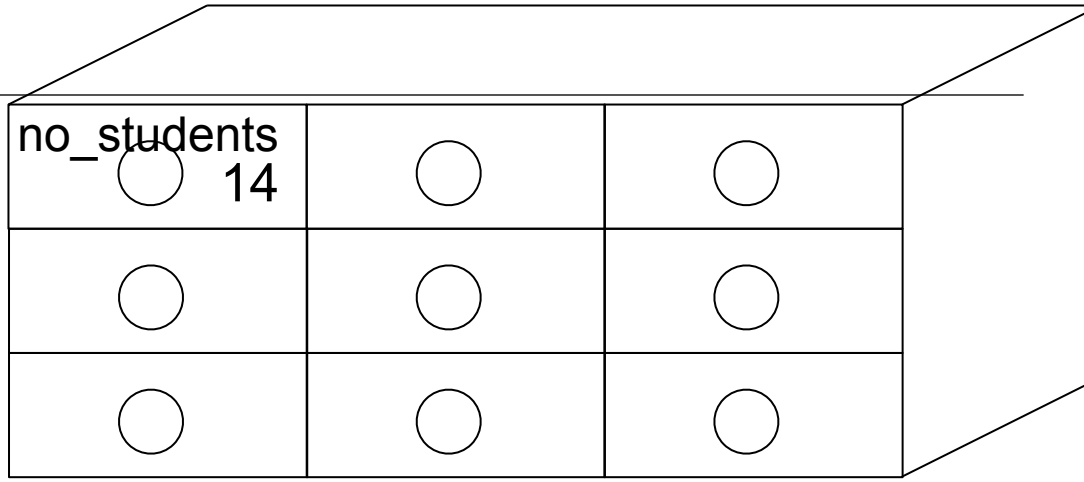
Variables

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



Variables

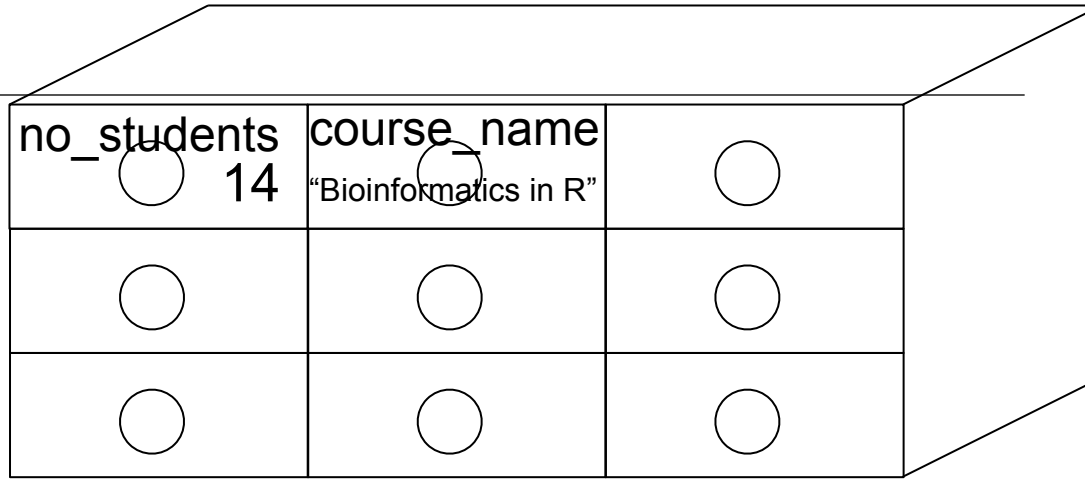
- Each drawer is called a **variable** (and we can give it a name)



  - Each drawer has a **type**
  - In R, we have the following **types:**
    - **numeric:** no_students = 14
    -

Variables

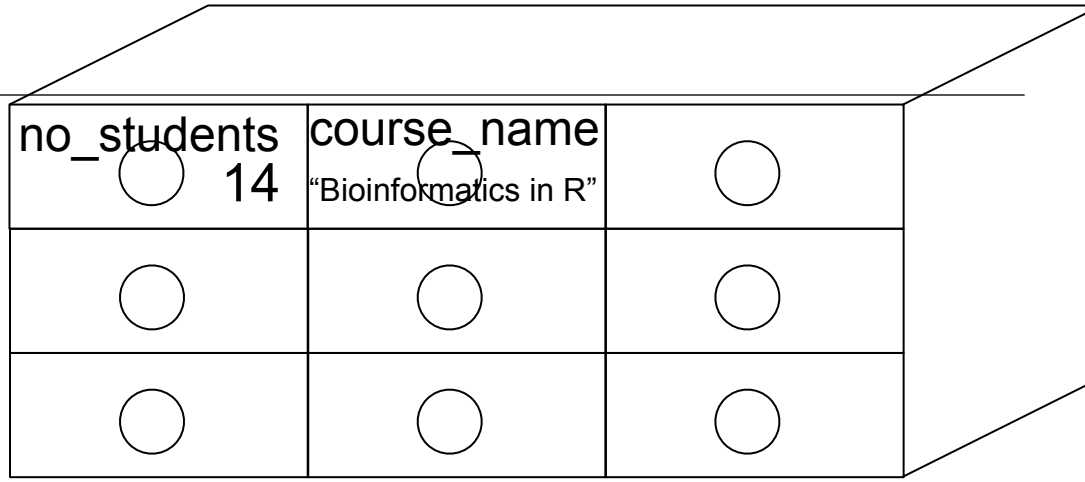
- Each drawer is called a **variable** (and we can give it a name)



  - Each drawer has a **type**
  - In R, we have the following **types:**
    - **numeric:** no_students = 14
    - **character:** course_name **=** “Bioinformatics in R”

Variables

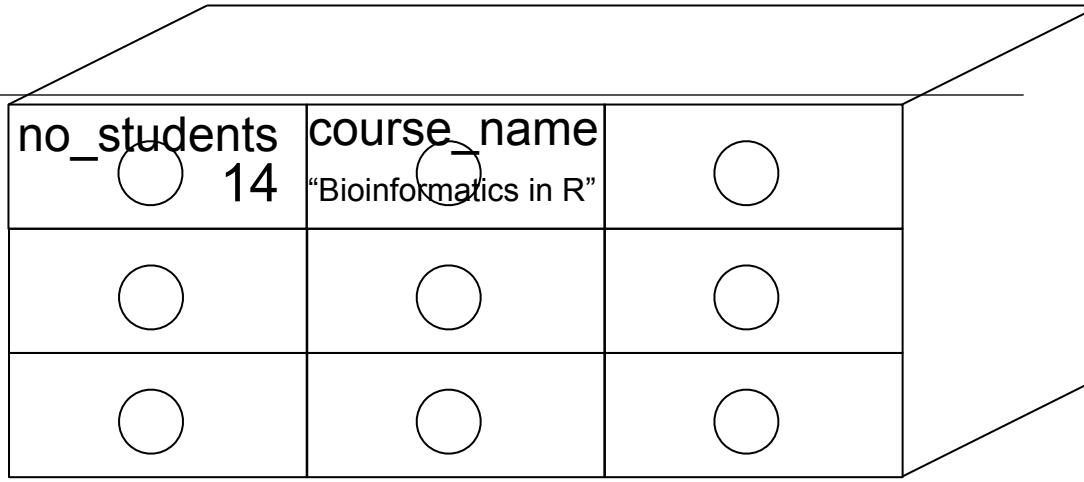
- Each drawer is called a **variable** (and we can give it a name)



  - 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

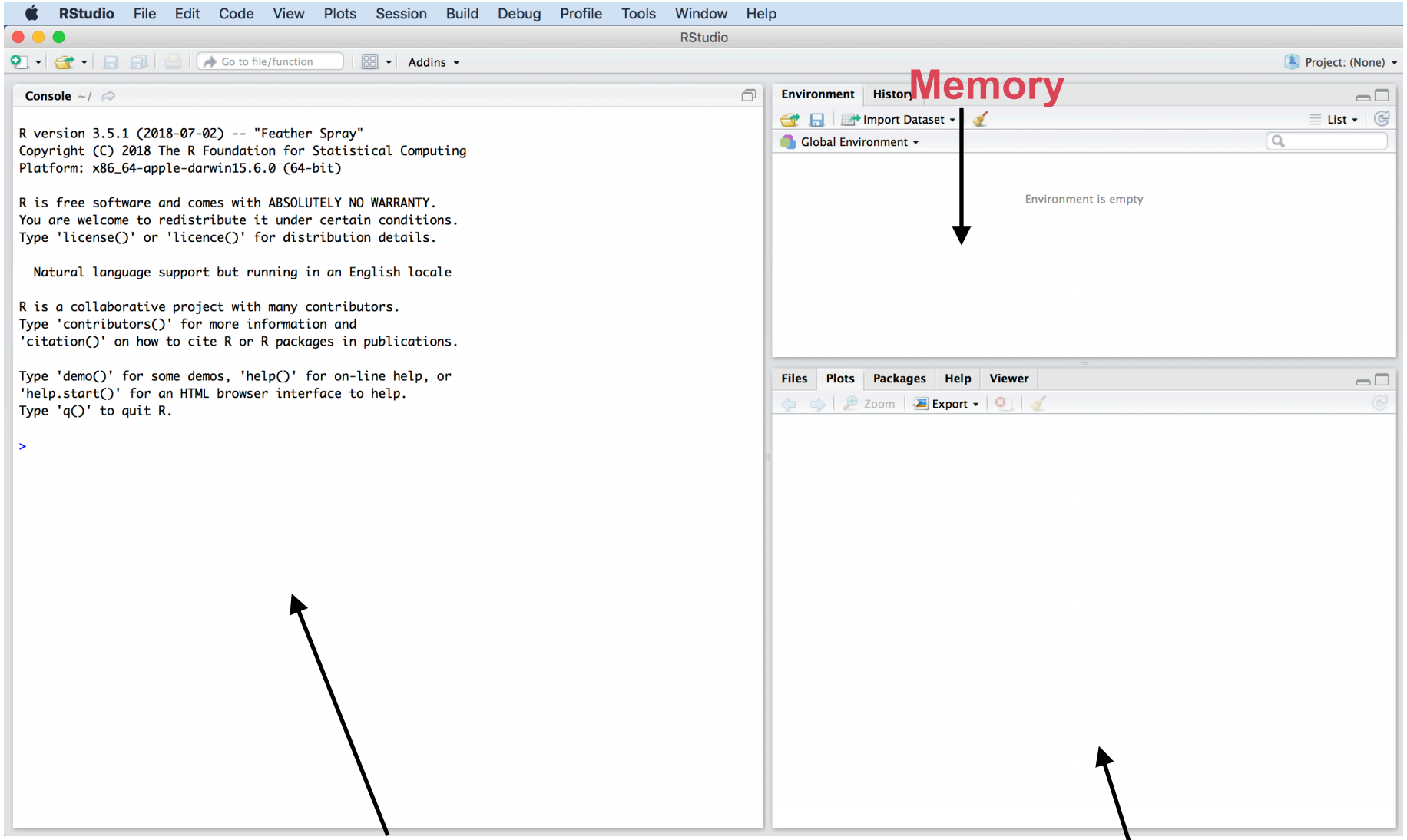
Variables

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



  - 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","Joseph","Fabio")
    - **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

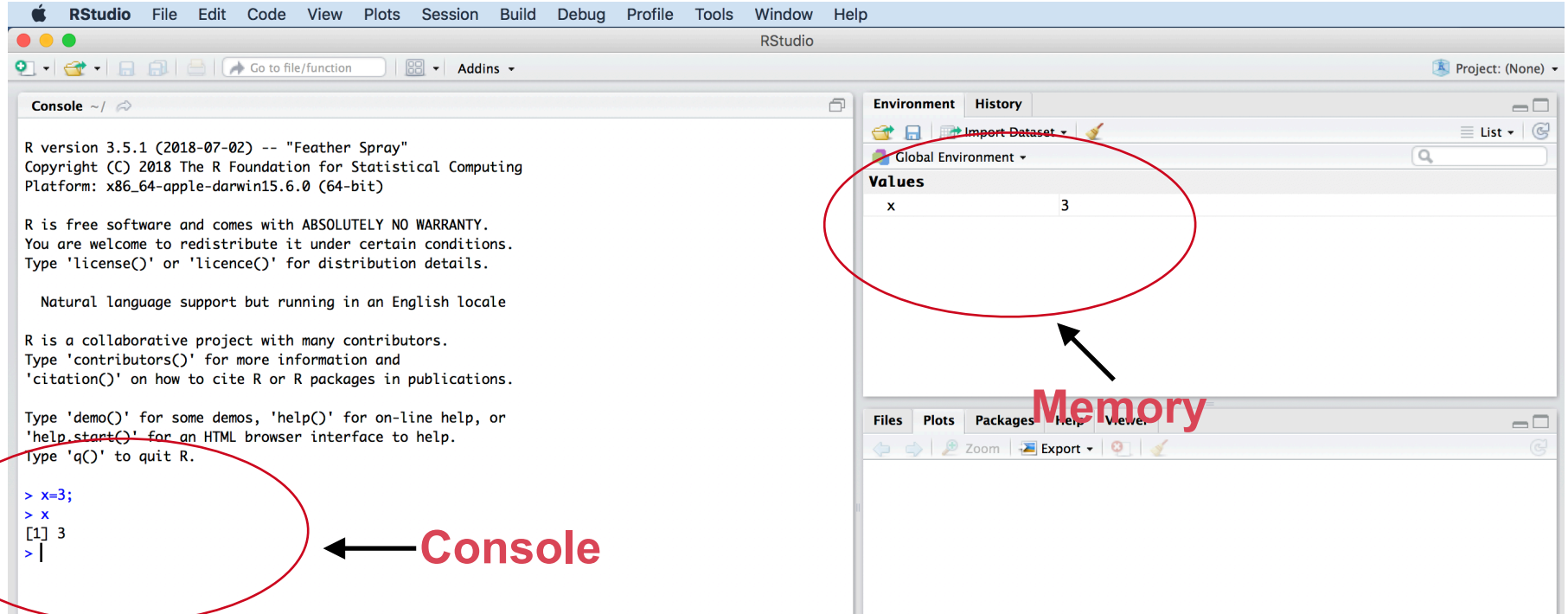
Single data can be stored in variables

- Data Types: “numeric”, ”character”, “logical”, …

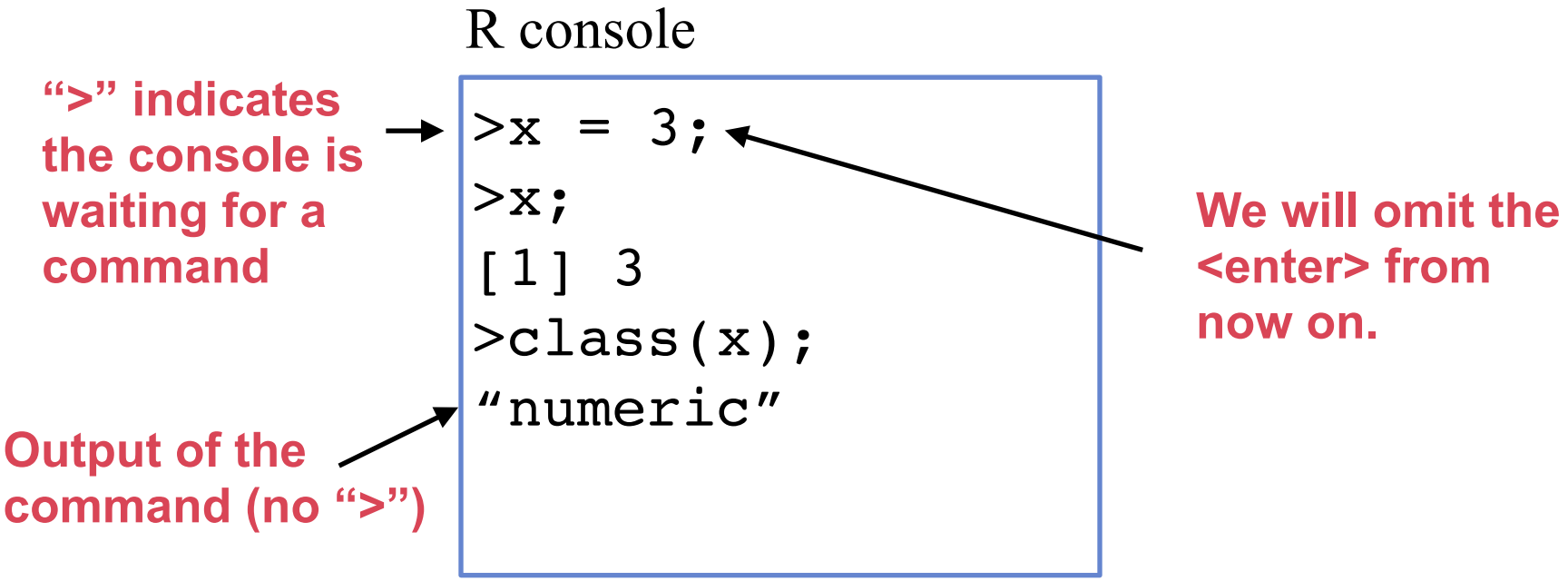
R console

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

“x = 3;” means store the number “3” at a variable named “x”



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] 8
> 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] 8
> x*y
[1] 12
> x/y
[1] 0.75
```

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

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

Complex Data Structures

- Vector – variable containing a array of items of the same type
- 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 excell sheet).
- …

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
```

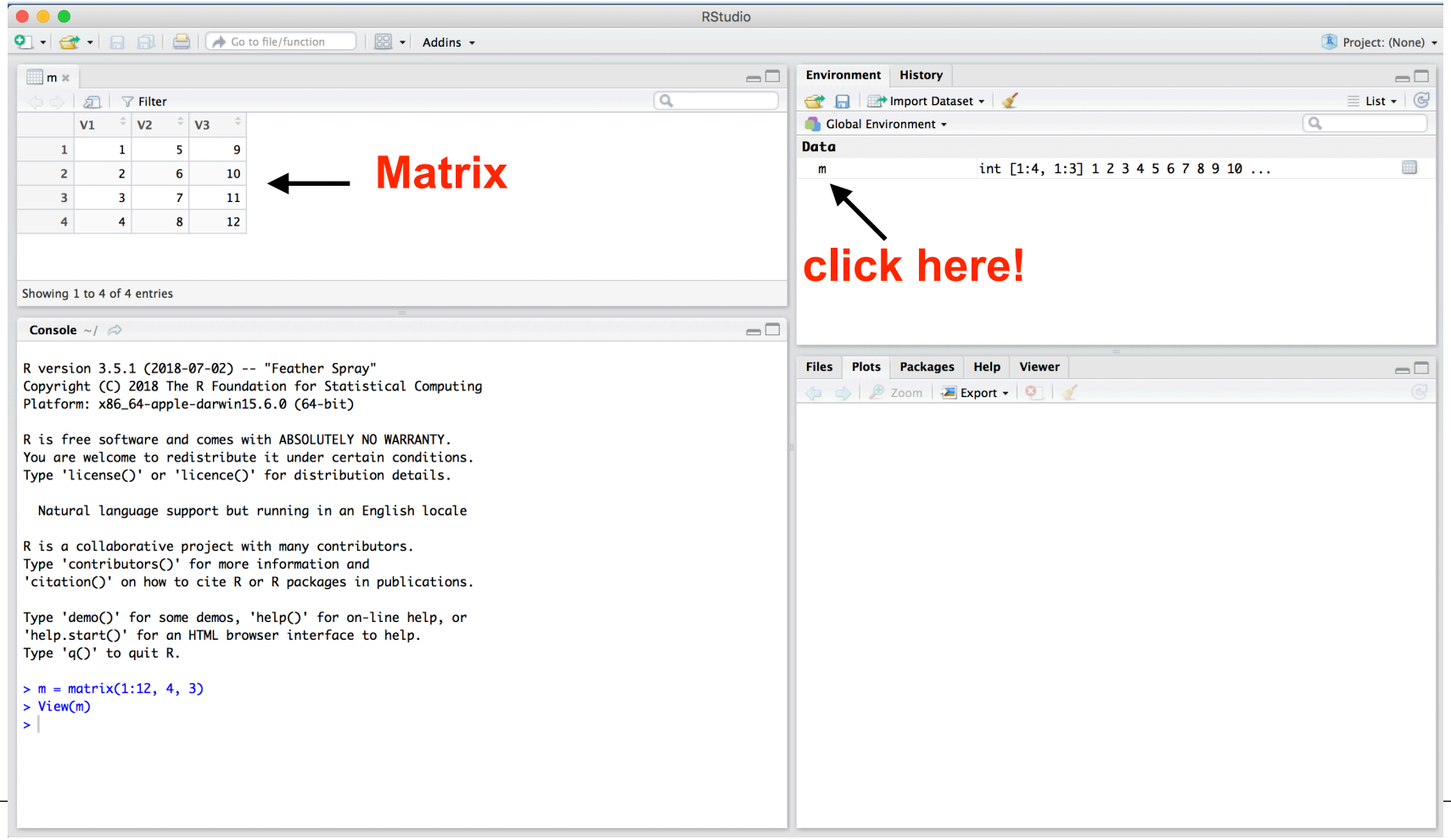
Matrix

- 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

- RStudio also helps vizualisation of a matrix



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?

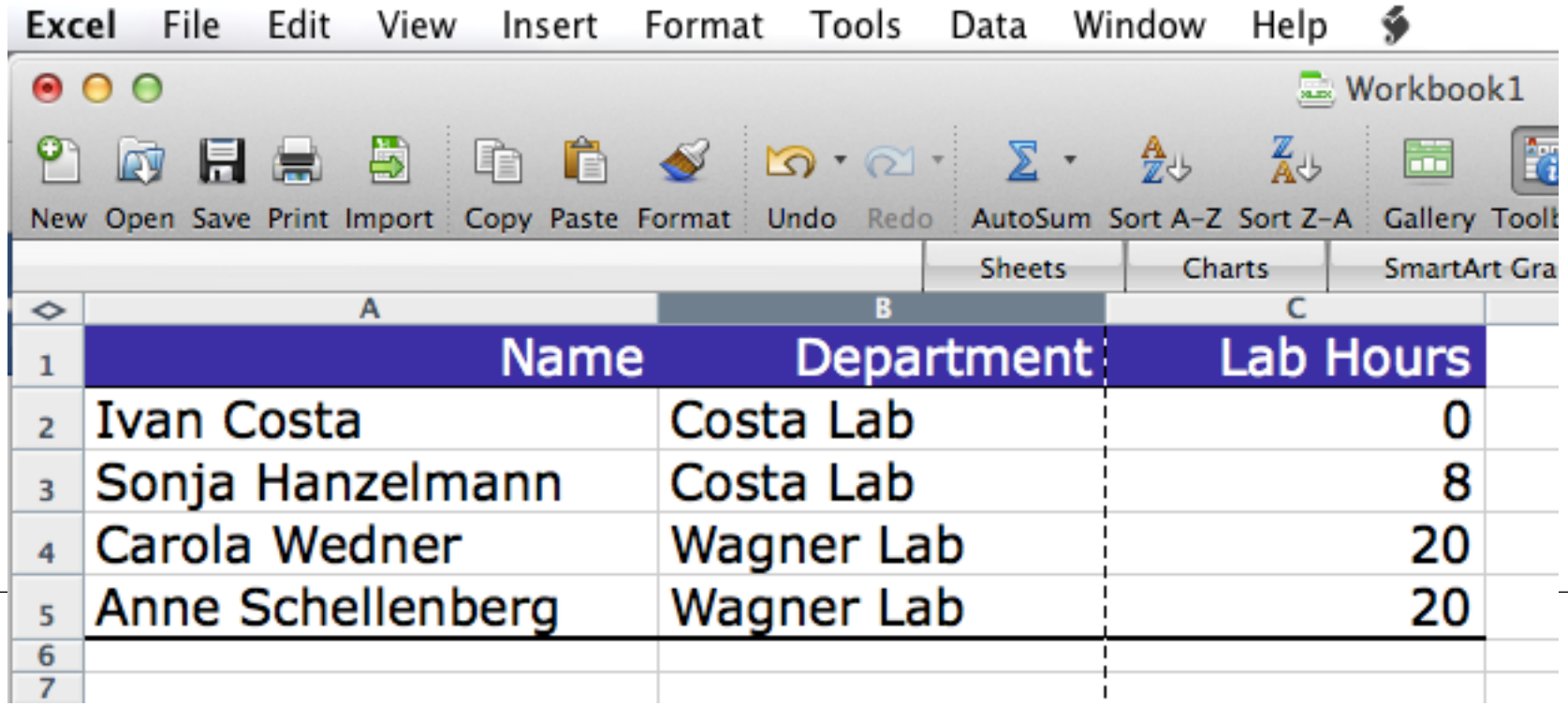
```
> m = matrix(1:12, 450000, 1000) # 4 by 3 matrix
> dim(m)     # size of matrix
[1] 450000   1000
> m[,1]      # show first column of matrix
[1] 1  2 3 4 5 6 …
```

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

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

Data Frames

- Data frames are hold a spreadsheet like table. The observations are the rows and the covariates are the columns. Columns share the same type.
- Data frames can be operated as matrices and be indexed with two subscripts.



	A	B	C
1	Name	Department	Lab Hours
2	Ivan Costa	Costa Lab	0
3	Sonja Hanzelmann	Costa Lab	8
4	Carola Wedner	Wagner Lab	20
5	Anne Schellenberg	Wagner Lab	20
6			
7			

Data Frames

- Creation and manipulation

```
> data = data.frame(name = c("Ivan","Sonja","Carola","Anne"),
          department = c("Costa","Costa","Wagner","Wagner"),
          labhour = c(0, 8, 20, 20))
> data
    name department labhour
1   Ivan      Costa       0
2  Sonja      Costa       8
3 Carola     Wagner      20
4   Anne     Wagner      20
> data$department      # access department column of frame
[1] Costa  Costa  Wagner Wagner
Levels: Costa Wagner
> data[,"department"]  # access department column of frame
> data[,2]             # second column of data frame
```

Data Frames

- Creation and manipulation

```
> data[1,]                  # first line of the data frame
name department labhour
1 Ivan       Costa         0
> data$labhour > 8          # lab hours exceeding 8
[1] FALSE FALSE  TRUE  TRUE
> data[data$labhour > 8,]
# data from members with more than 8 hours
    name department labhour
3 Carola      Wagner       20
4   Anne      Wagner       20
> data[data$department=="Costa",]
# data from members of Costa dept.
   name department labhour
1  Ivan       Costa         0
2 Sonja       Costa         8
```

Factor

- A list of categorical nature
  - i.e. gender (male,female), department (wagner, costa) , tumour type (…)
  - Important for statistical tests and plots

```
> data$department
[1] Costa  Costa  Wagner Wagner
Levels: Costa Wagner
> levels(data$department)
[1] "Costa"  "Wagner"
> levels(data$department)=c("AG Costa","AG Wagner")
> data$department
> summary(data$department)
  AG Costa  AG Wagner
         2          2
```

List

- An ordered collection of variables of distinct types under one variable (similar a data frame for a single observation).

```
# example of a list with 3 components
> w = list(name="Fred", age=5.3, sex="male")
> w[[1]]      # access the first variable of the list
[1] "Fred"
> w$name      # access the variable "name" of the list
[1] "Fred"
> w[["age"]]  # access the variable age of the list
[1] 5.3
```

Exercises

1. Create a data frame with all members of your lab (or Class colleagues). Include information as age, gender, height (you can of course imagine this).

2. Create operations to list the name of all colleagues with age higher than 30. Improve your method to list only male members with age higher than 30.

Functions

Functions

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

Example of Functions

```
> log2(4)
[1] 2
> dim(data)          # size of the data frame
[1] 4 3
> summary(data)    # statistics of a data frame columns
      name   department        labhour
 Anne   :1     Costa :2     Min.    : 0
 Carola:1     Wagner:2     1st Qu.: 6
 Ivan   :1                 Median :14
 Sonja  :1                 Mean   :12
                           3rd Qu.:20
                           Max.   :20
> write.table(data,"mydata.txt")
# write data 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

`x`	the object to be written, preferably a matrix or data frame. If not, it is attempted to coerce `x` to a data frame.
`file`	either a character string naming a file or a connection open for writing. `""` indicates output to the console.
`append`	logical. Only relevant if `file` is a character string. If `TRUE`, the output is appended to the file. If `FALSE`, any existing file of the name is destroyed.
`quote`	a logical value (`TRUE` or `FALSE`) or a numeric vector. If `TRUE`, 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 `FALSE`, nothing is quoted.
`sep`	the field separator string. Values within each row of `x` 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.

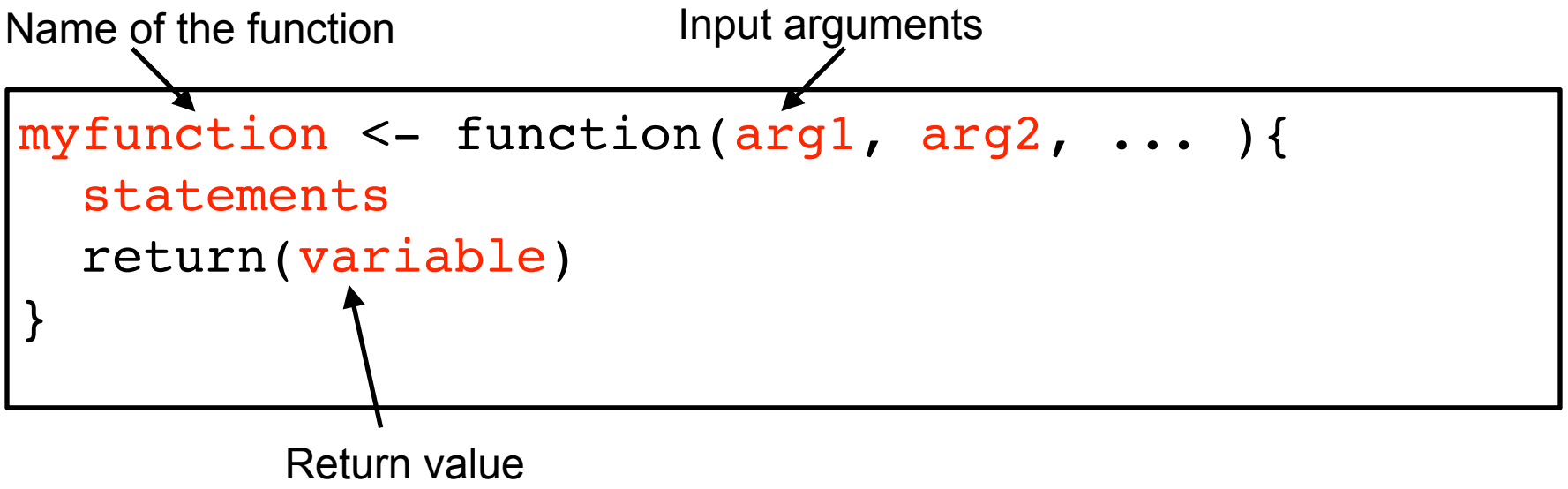
```
> source("http://bioconductor.org/biocLite.R")
> biocLite("packagename")
```

- Before using functions of a library they need to be opened.

```
> library("packagename")
```

Own Function

- Programming languages allow to define own functions. This is useful when you want to create a code describing a task that need to be repeated (write a table as file, complex arithmetic calculation).



Own Function - Examples

```
myfunction <- function(arg1, arg2, ... ){
  statements
  return(variable)
}
```

- Example of function for summing up 3 numbers

```
> sum3 <- function(a,b,c){
     # creates a function and stores in memory
     result=a+b+c;
     return (result)
   }
> sum3(3,4,5)
[1] 12
> sum3(1,2,3)
[1] 6
```

Exercise 1

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 2

- Create an integer vector i that can be used to subset v such that it will output the elements of v in decreasing order. For the general case, read the help pages for order and sort.

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

Exercise 3

- Create a function that converts Celsius to Fahrenheit degrees. Estimate the Fahrenheit for 40 or 0 degrees (Celsius).
- This is the conversion formula.

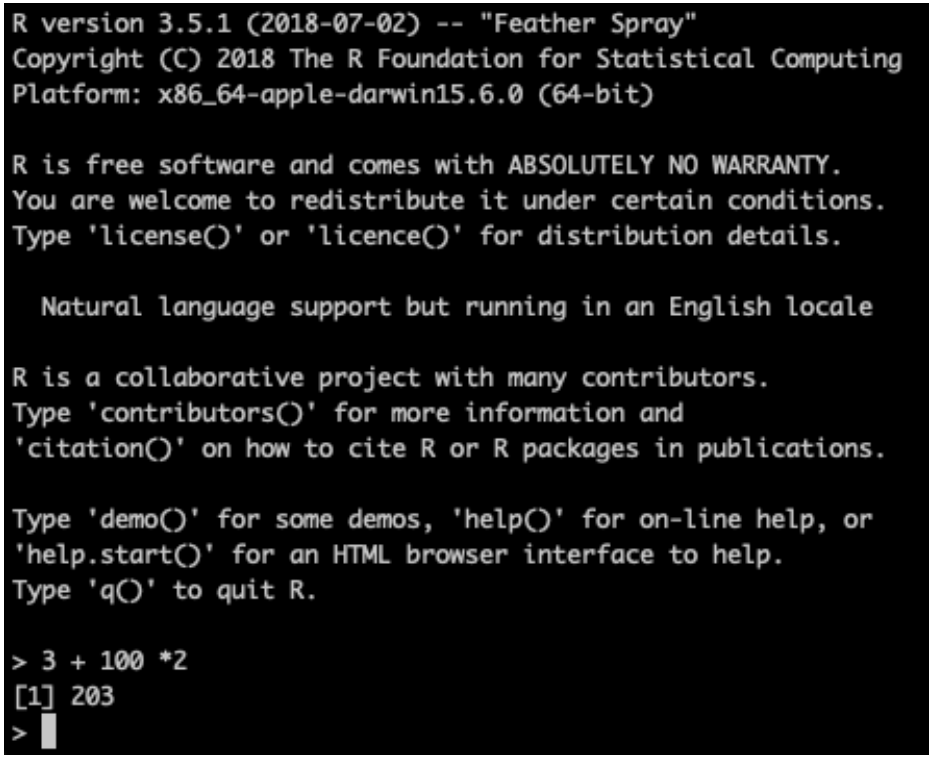
$$T_{(°\mathrm{F})} = T_{(°\mathrm{C})} \times 9/5 + 32$$

Overview of RStudio

Intro to RStudio

- RStudtio 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 Studio®

Control Commands

Control Commands

- Algorithms are usually not sequential.
- Control commands
  - test to decide the next steps
    - *if-else* command
  - Repeating commands until a condition is satisfied
    - *for* and *while*

if command

- only executed if condition is true

```
if (<logical test>){
  statements     # executed only if test is true
}
```

```
> grade = 6
> if (grade >= 6){
     print("fail")
  }
[1] "fail"
> grade = 4
> if (grade >= 6){
     print("fail")
  }
```

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 350º 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 - back a cake

Language - English

Exact - ???

Well defined - ???

if-else command

```
if (<logical test>){
  statements  # executed only if test is true
}else{
  statements  # executed only if test is true
}
```

```
> grade = 4
> if (grade >= 6){
     print("fail")
  }else{
     print("pass")
  }
[1] "pass"
```

if-else function

- an if else function variant that evaluate a vector of conditions

```
ifelse( vector conditions , expression 1, expression 2)
```

```
> ifelse(data$labhour > 8,"Biologist","Bioinformatician")
  [1] "Bioinformatician" "Bioinformatician"
  [3] "Biologist"        "Biologist"
> ifelse(data$labhour>8, c(1,2,3,4), c(5,6,7,8))
# expressions can also be lists
[1] 5 6 3 4
```

For command

- Repeats statement while interacting in a list

```
For (value in sequence){
  statements # executed for every value in sequence
}
```

```
> name=c("Ivan","Sonja","Carola","Anne")
> for (n in name) {
     print(n)
  }
[1] "Ivan"
[1] "Sonja"
[1] "Carola"
[1] "Anne"
```

For command examples

```
> range = 1:6     # command that creates a list from 1 to 6
> for (i in range) {
     print(i)
  }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
> value = 1
> for (i in range){
     value = value * i  # computes the factorial of 6
  }
> value
[1] 720
```

For command examples

```
> data =data.frame(name=c("Ivan","Sonja","Carola","Anne"),
   department=c("Costa","Costa","Wagner","Wagner"),
   labhour=c(0,8,20,20))
> row.names(data)=data$name # makes name the identifier for
each row.
> data["Ivan",]
> for (n in data$name){
     if (data[n,]$labhour > 8){
       print(data[n,]);
     }
  }
```

For command examples

```
> data =data.frame(name=c("Ivan","Sonja","Carola","Anne"),
   department=c("Costa","Costa","Wagner","Wagner"),
   labhour=c(0,8,20,20))
> row.names(data)=data$name # makes name the identifier for
each row.
> data["Ivan",]
> for (n in data$name){
     if (data[n,]$labhour > 8){
       print(data[n,]);
     }
  }
```

- What about previous example?

```
> data[data$labhour > 8,]
```

While command

- Repeats statement while condition is true

```
while (<logical test>){
  statements     # executed while test is true
}
```

```
> i = 1
> while (i < 6) {
      print(i)
      i = i+1
  }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Exercises

1. Create a function in R to provide the factorial value of a number (using loops).

```
factorial(x) = factorial(x-1)*x
```

2. Write a loop that counts 1 to 10 and this is repeated 3 times.

3. Write a loop that writes all numbers from 1 to 35 but skips the numbers 3,9,13,19,23,29. Tips: you can use the operator %in% to check if a value is in a list and you need a loop and a if for this problem.

Want more?

- More exercises at ...

http://www.bioconductor.org/help/course-materials/2010/BioC2010/First_Steps_With_R_SOLUTIONS.pdf

- Further tutorials at ...
  - https://www.datamentor.io/r-programming/#tutorial

Inst. for Computational Genomics

- Ivan G. Costa
- Joseph Kuo

Institute for Computational Genomics
01011011010
10100100101

RWTH AACHEN UNIVERSITY