Introduction to Programming

Lecture 9-10: Introduction to R (cont'd)

Clément Mazet-Sonilhac
clement.mazet@sciencespo.fr

Sciences Po Paris

More functions

Working with several files (cont'd)

- Write a function called `analyze()` that :
  1. takes a `filename` as an argument
  2. displays the three graphs produced in the previous lesson (average, min and max inflammation over time).
- Hint : `analyze(".../data/inflammation-01.csv")` should produce the three graphs already shown, while `analyze(".../data/inflammation-02.csv")` should produce corresponding graphs for the second data set. Be sure to document your function with comments.

More functions

Working with several files (cont'd)

- How to save results ?

  i add `pdf("inflammation-01.pdf")` before calling the function `analyze()`

  ii add `dev.off()` after.

More functions

Functions + Loops = New functions !

- How to list all .csv files in a folder ? (in Bash : ls -a *.csv)

More functions

Functions + Loops = New functions !

- How to list all .csv files in a folder ? (in Bash : ls -a *.csv)
- Several solutions with `list.files()` :
  1. `list.files(path = "C:/.../data", pattern = "csv")`

More functions

Functions + Loops = New functions !

- How to list all .csv files in a folder ? (in Bash : ls -a *.csv)
- Several solutions with `list.files()` :
    1. `list.files(path = "C:/.../data", pattern = "csv")`
    2. `list.files(path = "C:/.../data", pattern = "inflammation")`

More functions

Functions + Loops = New functions !

- How to list all .csv files in a folder ? (in Bash : ls -a *.csv)
- Several solutions with `list.files()` :
    1. `list.files(path = "C:/.../data", pattern = "csv")`
    2. `list.files(path = "C:/.../data", pattern = "inflammation")`
    3. Bash-style ? `list.files(path = "C:/.../data", pattern = "*inf*.csv")`

More functions

Functions + Loops = New functions !

- How to list all .csv files in a folder ? (in Bash : ls -a *.csv)
- Several solutions with `list.files()` :
    1. `list.files(path = "C:/.../data", pattern = "csv")`
    2. `list.files(path = "C:/.../data", pattern = "inflammation")`
    3. Bash-style ? `list.files(path = "C:/.../data", pattern = "*inf*.csv")`

Solution 1 : use `pattern = glob2rx("*inf*.csv")`
Solution 2 : use `pattern = "inflammation-[0-9]2.csv"`

More functions

Functions + Loops = New functions !

- We want to run our function `analyze()` on all files :

More functions

Functions + Loops = New functions !

- We want to run our function `analyze()` on all files :
- **Exercice 1** : write a function `analyseall()` using `analyze()` in a loop.
  - Hint 1 : use the function `list.files()` to create a list of files and store it in `filenames`
  - Hint 2 : use a loop (`for f in filenames`)

Some help !

More functions

Functions + Loops = New functions !

- We want to run our function `analyze()` on all files :
- **Exercice 1** : write a function `analyseall()` using `analyze()` in a loop.
  - Hint 1 : use the function `list.files()` to create a list of files and store it in `filenames`
  - Hint 2 : use a loop (`for f in filenames`)
- Try to save every graph in a `pdf` file !

More functions

Conditions

- Our previous lessons have shown us how to :

  1. manipulate data
  2. define our own functions
  3. repeat things.

- However, the programs we have written so far always do the same things, regardless of what data they're given. We want programs to make choices based on the values they are manipulating.

More functions

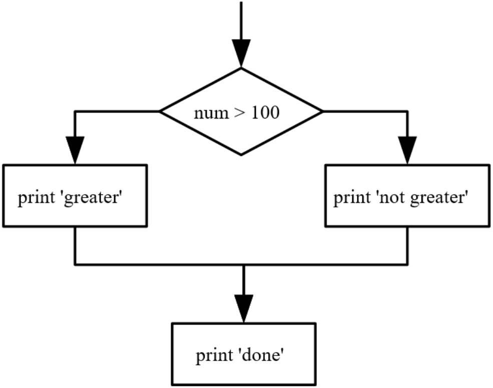
Conditions

- **Exercise 2** : What does this code do ?

```
num <- 37

if (num > 100) {
print("greater")
}

else {
print("not greater")
}

print("done")
```

More functions

Conditions



In the example above, the test `num > 100` returns the value FALSE, which is why the code inside the if block was skipped and the code inside the else statement was run instead.

More functions

Conditions

- **Exercise 3** : Write a function `sign()` that takes as argument a number and returns the value "negative", "null" or "positive"
- Hint : use `else`, `if`, `else if`

More functions

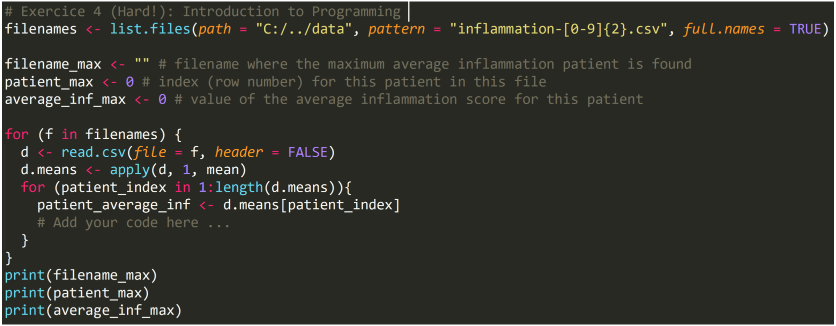
Conditions

- We can combine tests with **&** (both true) and **|** (one true or the other) :
- **Exercise 3bis** : Write a function `sign2()` that takes as argument two numbers and returns the value "both negative", "other", "both zero" or "both positive"

More functions

Conditions

- **Exercise 4** : Find the file containing the patient with the highest average inflammation score. Print the file name, the patient number (row number) and the value of the maximum average inflammation score.



```r
# Exercice 4 (Hard!): Introduction to Programming
filenames <- list.files(path = "C:/../data", pattern = "inflammation-[0-9]{2}.csv", full.names = TRUE)

filename_max <- "" # filename where the maximum average inflammation patient is found
patient_max <- 0 # index (row number) for this patient in this file
average_inf_max <- 0 # value of the average inflammation score for this patient

for (f in filenames) {
  d <- read.csv(file = f, header = FALSE)
  d.means <- apply(d, 1, mean)
  for (patient_index in 1:length(d.means)){
    patient_average_inf <- d.means[patient_index]
    # Add your code here ...
  }
}
print(filename_max)
print(patient_max)
print(average_inf_max)
```

More functions

Conditions

- **Exercise 5** : Re-write the `analyze()` function with an option to save of not, using a second argument `output` that takes the default value NULL and using `if`
- Tips : Have a look to this function `!is.null()`

More functions

Key points :

1. Save a plot in a pdf file using pdf("name.pdf") and stop writing to the pdf file with dev.off().
2. Use if (condition) to start a conditional statement, else if (condition) to provide additional tests, and else to provide a default.
3. The bodies of conditional statements must be surrounded by curly braces

More functions

Key points :

1. Save a plot in a pdf file using pdf("name.pdf") and stop writing to the pdf file with dev.off().
2. Use if (condition) to start a conditional statement, else if (condition) to provide additional tests, and else to provide a default.
3. The bodies of conditional statements must be surrounded by curly braces
4. Use == to test for equality.
5. X & Y is only true if both X and Y are true.
6. X | Y is true if either X or Y, or both, are true.

Datatype

What and why ?

In the 3-4 last sessions, we have learn how to :

- import data : `read.csv()`
- explore and plot data
- create functions : `function(arg1, arg2)`
- make choices (i.e. logicals) `if, else, &, |`
- combine everything in complex algorithms (last session...)

Datatype

What and why ?

In the 3-4 last sessions, we have learn how to :

- import data : `read.csv()`
- explore and plot data
- create functions : `function(arg1, arg2)`
- make choices (i.e. logicals) `if, else, &, |`
- combine everything in complex algorithms (last session...)

⇒ Now : let's put some **structure** into it !

Datatype

What and why ?

Everything in R is an object. R has 6 (although we will not discuss the raw class for this workshop) atomic vector types :

Datatype

What and why ?

Everything in R is an object. R has 6 (although we will not discuss the raw class for this workshop) atomic vector types :

- characte :r `"a", "word", "a sentence like this"`

Datatype

What and why ?

Everything in R is an object. R has 6 (although we will not discuss the raw class for this workshop) atomic vector types :

- characte :r `"a", "word", "a sentence like this"`
- numeric (real or decimal) : `"2", "1.75"`

Datatype

What and why ?

Everything in R is an object. R has 6 (although we will not discuss the raw class for this workshop) atomic vector types :

- characte :r `"a", "word", "a sentence like this"`
- numeric (real or decimal) : `"2", "1.75"`
- integer : `"2L"`

Datatype

What and why ?

Everything in R is an object. R has 6 (although we will not discuss the raw class for this workshop) atomic vector types :

- characte :r `"a", "word", "a sentence like this"`
- numeric (real or decimal) : `"2", "1.75"`
- integer : `"2L"`
- logical : `"TRUE", "FALSE"`

Datatype

What and why ?

Everything in R is an object. R has 6 (although we will not discuss the raw class for this workshop) atomic vector types :

- characte :r `"a", "word", "a sentence like this"`
- numeric (real or decimal) : `"2", "1.75"`
- integer : `"2L"`
- logical : `"TRUE", "FALSE"`
- complex : `"1+4i"`

Datatype

What and why ?

R provides functions to examine features of vectors and other objects, for example :

Datatype

What and why ?

R provides functions to examine features of vectors and other objects, for example :

- `class()` - what kind of object is it (high-level) ?
- `typeof()` - what is the object's data type (low-level) ?
- `length()` - how long is it ? What about two dimensional objects ?
- `attributes()` - does it have any metadata ?

Datatype

What and why ?

R provides functions to examine features of vectors and other objects, for example :

- `class()` - what kind of object is it (high-level) ?
- `typeof()` - what is the object's data type (low-level) ?
- `length()` - how long is it ? What about two dimensional objects ?
- `attributes()` - does it have any metadata ?
- **Exercise 1** : create a object `x <- "dataset"` and display its type, class, lenght and attribute. What do you obtain ? Same with `y <- 1:10`. Finally, try `z <- as.numeric(y)`

Datatype

Vectors

A vector is the most common and basic data structure in R and is pretty much the workhorse of R. Technically, vectors can be one of two types : i) atomic and ii) lists

Datatype

Vectors

A vector is the most common and basic data structure in R and is pretty much the workhorse of R. Technically, vectors can be one of two types : i) atomic and ii) lists

- A vector is a collection of elements that are most commonly of mode character, logical, integer or numeric.

Datatype

Vectors

A vector is the most common and basic data structure in R and is pretty much the workhorse of R. Technically, vectors can be one of two types : i) atomic and ii) lists

- A vector is a collection of elements that are most commonly of mode character, logical, integer or numeric.
- How to generate one ? `vector("character", length = 5)` or `character(5)`

Datatype

Vectors

A vector is the most common and basic data structure in R and is pretty much the workhorse of R. Technically, vectors can be one of two types : i) atomic and ii) lists

- A vector is a collection of elements that are most commonly of mode character, logical, integer or numeric.
- How to generate one ? `vector("character", length = 5)` or `character(5)`
- Try `logical(5)`. What do you obtain ?

Datatype

Vectors

A vector is the most common and basic data structure in R and is pretty much the workhorse of R. Technically, vectors can be one of two types : i) atomic and ii) lists

- A vector is a collection of elements that are most commonly of mode character, logical, integer or numeric.
- How to generate one ? `vector("character", length = 5)` or `character(5)`
- Try `logical(5)`. What do you obtain ?
- You can also create vectors by directly specifying their content : `x <- c(1,2,3)`.

Datatype

Vectors : adding elements

Reminder : the function `c()` (for combine) can also be used to add elements to a vector :

- Try : `z <- c("Sarah", "Tracy", "Jon")` + `z <- c(z, "Annette")`

Datatype

Vectors : adding elements

Reminder : the function `c()` (for combine) can also be used to add elements to a vector :

- Try : `z <- c("Sarah", "Tracy", "Jon")` + `z <- c(z, "Annette")`

Vectors from a Sequence of Numbers : You can create vectors as a sequence of numbers :

- Try : `series <- 1:10` or `seq(10)`

Datatype

Vectors : adding elements

Reminder : the function `c()` (for combine) can also be used to add elements to a vector :

- Try : `z <- c("Sarah", "Tracy", "Jon")` + `z <- c(z, "Annette")`

Vectors from a Sequence of Numbers : You can create vectors as a sequence of numbers :

- Try : `series <- 1:10` or `seq(10)`
- **Exercise 2** : What will be the result of : `seq(from = 1, to = 10, by = 0.1)`

Datatype

Vectors : missing data

R supports missing data in vectors. They are represented as NA (Not Available) and can be used for all the vector types covered in this lesson :

- create two vectors : `x <- c("a", NA, "c", "d", NA)` and `y <- c("a", "b", "c", "d", "e")`

Datatype

Vectors : missing data

R supports missing data in vectors. They are represented as NA (Not Available) and can be used for all the vector types covered in this lesson :

- create two vectors : `x <- c("a", NA, "c", "d", NA)` and `y <- c("a", "b", "c", "d", "e")`
- **Exercise 3** : test for the presence of NA with : `is.na()` and `anyNA()`. What do you obtain ?

Datatype

Vectors : mixing types

What Happens When You Mix Types Inside a Vector ?

⇒ R will create a resulting vector with a mode that can most easily accommodate all the elements it contains. This conversion between modes of storage is called “coercion”.

Exercise 3 : What does R will do ?

- `xx <- c(1.7, "a")`

Datatype

Vectors : mixing types

What Happens When You Mix Types Inside a Vector ?

⇒ R will create a resulting vector with a mode that can most easily accommodate all the elements it contains. This conversion between modes of storage is called “coercion”.

Exercise 3 : What does R will do ?

- `xx <- c(1.7, "a")`
- `xx <- c(TRUE, 2)`

Datatype

Vectors : mixing types

What Happens When You Mix Types Inside a Vector ?

⇒ R will create a resulting vector with a mode that can most easily accommodate all the elements it contains. This conversion between modes of storage is called “coercion”.

Exercise 3 : What does R will do ?

- `xx <- c(1.7, "a")`
- `xx <- c(TRUE, 2)`
- `xx <- c("a", TRUE)`

You can also control how vectors are coerced explicitly using the `as.<classname>()` functions : try `as.numeric("1")`, `as.character(1:10)` ...

Datatype

Matrix

In R matrices are an extension of the numeric or character vectors. They are not a separate type of object but simply an atomic vector with dimensions ; the number of rows and columns.

- create a matrix with `m <- matrix(nrow = 2, ncol = 2)`. What do you obtain ?

Datatype

Matrix

In R matrices are an extension of the numeric or character vectors. They are not a separate type of object but simply an atomic vector with dimensions ; the number of rows and columns.

- create a matrix with `m <- matrix(nrow = 2, ncol = 2)`. What do you obtain ?
- What is the matrix dimension ? Use `dim()`

Datatype

Matrix

In R matrices are an extension of the numeric or character vectors. They are not a separate type of object but simply an atomic vector with dimensions ; the number of rows and columns.

- create a matrix with `m <- matrix(nrow = 2, ncol = 2)`. What do you obtain ?
- What is the matrix dimension ? Use `dim()`
- `m2 <- matrix(10, 50)`. How many columns and rows ?

Datatype

Matrix

In R matrices are an extension of the numeric or character vectors. They are not a separate type of object but simply an atomic vector with dimensions ; the number of rows and columns.

- create a matrix with `m <- matrix(nrow = 2, ncol = 2)`. What do you obtain ?
- What is the matrix dimension ? Use `dim()`
- `m2 <- matrix(10, 50)`. How many columns and rows ?
- `m2 <- matrix(10, 50, 50)`. How many columns and rows ?

Datatype

Matrix

Different ways to create a matrix :

- Empty matrix : `m <- matrix(nrow = 2, ncol = 2)`

Datatype

Matrix

Different ways to create a matrix :

- Empty matrix : `m <- matrix(nrow = 2, ncol = 2)`
- Zeroes matrix : `m <- matrix(0, nrow = 2, ncol = 2)`

Datatype

Matrix

Different ways to create a matrix :

- Empty matrix : `m <- matrix(nrow = 2, ncol = 2)`
- Zeroes matrix : `m <- matrix(0, nrow = 2, ncol = 2)`
- `m <- 1:4` + `dim(m) = c(2,2)`. What do you obtain ?

Datatype

Matrix

Exercise 4 :

- create an empty matrix I x J (I = 10, J = 10)
- ... and use a loop to fill it. (reminder `for (i in ...){ code }`)
- Each case should contain the product of i x j !

Datatype

Matrix

Exercise 4bis :

- create an empty matrix I x J (I = 10, J = 10)
- ... and use a loop to fill it. (reminder `for (i in ...){ code }`)
- Each case should contain 0 except if in diagonal, then equals 1 (i.e. identity matrix)

Datatype

List

In R lists act as containers. Unlike atomic vectors, the contents of a list are not restricted to a single mode and can encompass any mixture of data types.

- Exemple : `x <- list(1, "a", TRUE, 1+4i)`

Datatype

List

In R lists act as containers. Unlike atomic vectors, the contents of a list are not restricted to a single mode and can encompass any mixture of data types.

- Exemple : `x <- list(1, "a", TRUE, 1+4i)`
- Again : many ways to create list (you can try `x <- 1:10` and then `x <- as.list(x)`)

Datatype

List

In R lists act as containers. Unlike atomic vectors, the contents of a list are not restricted to a single mode and can encompass any mixture of data types.

- Exemple : `x <- list(1, "a", TRUE, 1+4i)`
- Again : many ways to create list (you can try `x <- 1:10` and then `x <- as.list(x)`)
- What is the class of x[1] ? What about x[[1]] ?

Datatype

List

Elements of a list can be named (i.e. lists can have the names attribute)

- Try `mylist <- list(a = "Karthik Ram", b = 1:10, data = head(iris))`

Datatype

List

Elements of a list can be named (i.e. lists can have the names attribute)

- Try `mylist <- list(a = "Karthik Ram", b = 1:10, data = head(iris))`
- you can call each element with $: what will be the result of mylist$a ?

Datatype

List

Elements of a list can be named (i.e. lists can have the names attribute)

- Try `mylist <- list(a = "Karthik Ram", b = 1:10, data = head(iris))`
- you can call each element with $: what will be the result of mylist$a ?
- you can display the name of each element with `names(mylist)`

Datatype

List

Elements of a list can be named (i.e. lists can have the names attribute)

- Try `mylist <- list(a = "Karthik Ram", b = 1:10, data = head(iris))`
- you can call each element with $: what will be the result of mylist$a ?
- you can display the name of each element with `names(mylist)`
- What is the class of `data` ?

Datatype

Data frame

A data frame is a very important data type in R. It's pretty much the de facto data structure for most tabular data and what we use for statistics.

- To create data frames by hand : `dt <- data.frame(id = letters[1:10], x = 1:10, y = 11:20)`

Datatype

Data frame

A data frame is a very important data type in R. It's pretty much the de facto data structure for most tabular data and what we use for statistics.

- To create data frames by hand : `dt <- data.frame(id = letters[1:10], x = 1:10, y = 11:20)`

Useful functions :

- head() - shows first 6 rows
- tail() - shows last 6 rows
- dim() - returns the dimensions of data frame (i.e. number of rows and number of columns)
- nrow() / ncol()- number of rows/columns
- str() - structure of data frame
- sapply(dataframe, class)

Datatype

Data frame

Data frame are actually a 2-D version of lists : (see `is.list(dt)`)

- you can use bracket to "call" them. Ex : `dt[["y"]]`
- you can use $ to display one element. Ex : `dt$y`

Datatype

Data types

Summary :

- Dimensions : 1-D. If homogeneous = atomic vector, if heterogeneous = list

Datatype

Data types

Summary :

- Dimensions : 1-D. If homogeneous = atomic vector, if heterogeneous = list
- Dimensions : 2 or more-D. If homogeneous = matrix, if heterogeneous = data frame

Datatype

Data types

Exercise 5 :

- create an empty dataframe `dt <- data.frame(id = letters[1:10], x = "", y = "")`
- ... and use a loop to fill it. (reminder `for (i in ...){ code }`)
- In x, each case should contain 0 except if id == e, in y each case should contain i.
- finally, the loop should generate a new element z that is the sulm of element x and y

Appendix

More functions

Functions + Loops = New functions !

```
analyzeall <- function(pattern) {

filenames <- list.files(path = "C :/Users/.../data", pattern = XXX,
full.names = TRUE)

for (f in filenames) {

XXX }

}
```

back