

# R COURSE

Introduction to R

Daniel Vaulot

2025-01-15



# R sessions

---

- 01 - Introduction to R
- 02 - Data wrangling
- 03 - Data visualisation
- 04 - Markdown

# R - Session 01

---

- What is R and why use R ?
- Resources
- Get started
- Fundamentals of R
- Data objects
- Vectors
- Operators
- Functions
- Packages

# Introduction

---

- If you are an R guru:
  - Please refrain to answer during this session...
  - Help your neighbor
- Two special slide formatting
  - | Your turn...
  - | Warning



# R

# History of computer languages

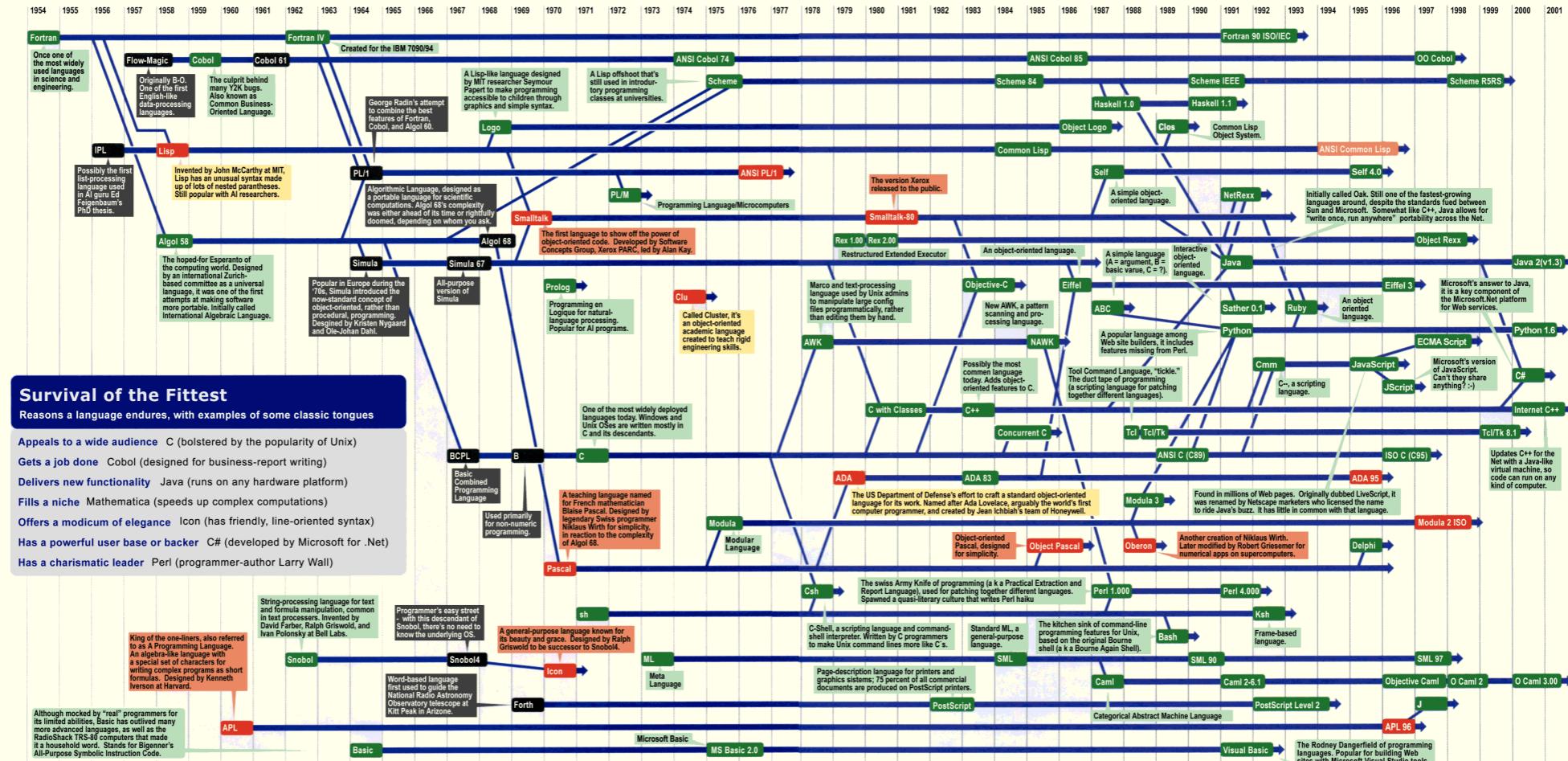
# Mother Tongues

# Tracing the roots of computer languages through the ages

Just like half of the world's spoken tongues, most of the 2,300-plus computer programming languages are either endangered or extinct. As powerhouses C/C++, Visual Basic, Cobol, Java and other modern source codes dominate our systems, hundreds of older languages are running out of life.

An ad hoc collection of engineers-electronic lexicographers, if you will—aim to save, or at least document the lingo of classic software. They're combing the globe's 9 million developers in search of coders still fluent in these nearly forgotten lingua frangas. Among the most endangered are Ada, APL, B (the predecessor of C), Lsp, Oberon, Smalltalk, and Simula.

Code-raker Grady Booch, Rational Software's chief scientist, is working with the Computer History Museum in Silicon Valley to record and, in some cases, maintain languages by writing new compilers so our ever-changing hardware can grok the code. Why bother? "They tell us about the state of software practice, the minds of their inventors, and the technical, social, and economic forces that shaped history at the time," Booch explains. "They'll provide the raw material for software archeologists, historians, and developers to learn what worked, what was brilliant, and what was an utter failure." Here's a peek at the strongest branches of programming's family tree. For a nearly exhaustive rundown, check out the Language List at [HTTP://www.informatik.uni-freiburg.de/java/misc/lang\\_list.html](http://www.informatik.uni-freiburg.de/java/misc/lang_list.html). - Michael Mendenlo



Sources: Paul Boutin; Brent Hailpern, associate director of computer science at IBM Research; The Retrocomputing Museum; Todd Proebsting, senior researcher at Microsoft; Gis Wiederhold, computer scientist, Stanford University

# History of R

---

- **Mid 1970s** - S Language for Statistical Computing conceived by John Chambers, Rick Becker, Trevor Hastie, Allan Wilks and others at Bell Labs
- **Early 1990's** - R was first implemented in the early 1990's by Robert Gentleman and Ross Ihaka, both faculty members at the University of Auckland.
- **1995** - Open Source Project
- **1997** - Managed by the R Core Group
- **2000** - First release of R
- **2011** - First release of R studio
- [Historical notes - Paper from 1998](#)



# Why use R ?

---

- **Script vs. Menu driven software (e.g. Excel)**
  - Can be re-rerun with new data
  - Reproducible workflow
- **Open source**
  - Huge number of libraries
  - Tidy “universe” : tidyverse and ggplot2
    - Very easy to manipulate tables (select columns, create new variables)
    - High quality graphics
- **Work environment**
  - R studio
- **Document your data processing**
  - R markdown
  - Create HTML, pdf, presentations

# What can you do with R ?

---

- Science
  - Statistics of course...
  - Data processing
  - Graphics
  - Time series analyses
  - Maps
  - Bioinformatics
- But also
  - Teach
  - Do a presentation
  - Write your CV
  - Build a web site
  - Write a book
  - Much more...

# **What can you do with R ?**

---

**Example of web page**



# Yahoo channel - A must

- Riffomonas by Pat Schloss

The screenshot shows the YouTube channel page for "Riffomonas Project". The channel banner features a grid of orange and blue microorganisms. The channel name "Riffomonas Project" and the handle "@Riffomonas" are displayed, along with 12,3 k abonnés. A "Subscribed" button is visible. The navigation bar includes links for ACCUEIL, VIDÉOS (which is underlined), PLAYLISTS, COMMUNAUTÉ, CHAÎNES, and À PROPOS. A search icon is also present.

**CODE CLUB**  
Thank you! Riffomonas channel hits 10,000 subscribers (CC255)  
Riffomonas Project • 1,6 k vues • il y a 3 mois

I can't think each of you for watching, subscribing, and telling a friend about this channel. I never thought we could get to 10,000 subscribers, but here we are! Help spread the word and let's...

**Vidéos** ► Tout lire

**MAPS!** 18:44      **case\_when** 21:48      **Custom fonts with showtext and google fonts** 11:50      **rebase!** 18:20      **This is huge!** 28:00      **Amount of precipitation for September 17 to October 10, 2021** 23:55

How to add maps to a ggplot2 figure in R (CC264)      Incorporating logic into a visual made with R using a...      How to use a custom font in R with showtext and google...      Rewrite your commit history using git rebase to squash...      How to use GitHub actions to run a Snakemake pipeline...      Using GitHub pages to host a web page made with...

3,1 k vues • il y a 2 mois      2 k vues • il y a 2 mois      1,1 k vues • il y a 2 mois      535 vues • il y a 2 mois      709 vues • il y a 2 mois      1,2 k vues • il y a 2 mois

# Help

## Cheat sheets

- R basics
- ggplot2
- dplyr

## Forum

- Stack overflow
- Stat Blog
- R bloggers

The screenshot shows a Stack Overflow post with the following details:

**Title:** How to sort a dataframe by multiple column(s)?

**Post ID:** 1176

**Upvotes:** 1176

**Downvotes:** 514

**Owner:** smci (14.7k reputation)

**Asked:** Aug 18 '09 at 21:33 by Christopher DuBois

**Tags:** r, sorting, dataframe, order, r-faq

**Share:** share edit

**Comments:** add a comment

**Answers:** 18 Answers

**Order:** active, oldest, votes

**Content:** The post contains R code to sort a data frame by multiple columns:

```
dd <- data.frame(b = factor(c("Hi", "Med", "Hi", "Low"),
  levels = c("Low", "Med", "Hi"), ordered = TRUE),
  x = c("A", "D", "A", "C"), y = c(8, 3, 9, 9),
  z = c(1, 1, 1, 2))
dd
#> #>   b x y z
#> #> 1 Hi A 8 1
#> #> 2 Med D 3 1
#> #> 3 Hi A 9 1
#> #> 4 Low C 9 2
```

**Top Answer:** You can use the `order()` function directly without resorting to add-on tools -- see this simpler answer which uses a trick right from the top of the `example(order)` code:

**Post ID:** 1471

**Upvotes:** 1471

**Downvotes:** 2

**Owner:** smci (14.7k reputation)

**Asked:** May 1 '18 at 10:09

**Tags:** r, order, data.table, dd, order, dd[,1]

**Share:** share edit

**Comments:** add a comment

**Answers:** 18 Answers

**Order:** active, oldest, votes

**Content:** The top answer provides an alternative method using the `order()` function:

```
R> dd[with(dd, order(-z, b)), ]
#> #>   b x y z
#> #> 4 Low C 9 2
#> #> 2 Med D 3 1
#> #> 1 Hi A 8 1
#> #> 3 Hi A 9 1
```

**Text:** Edit some 2+ years later: It was just asked how to do this by column index. The answer is to simply pass the desired sorting column(s) to the `order()` function:

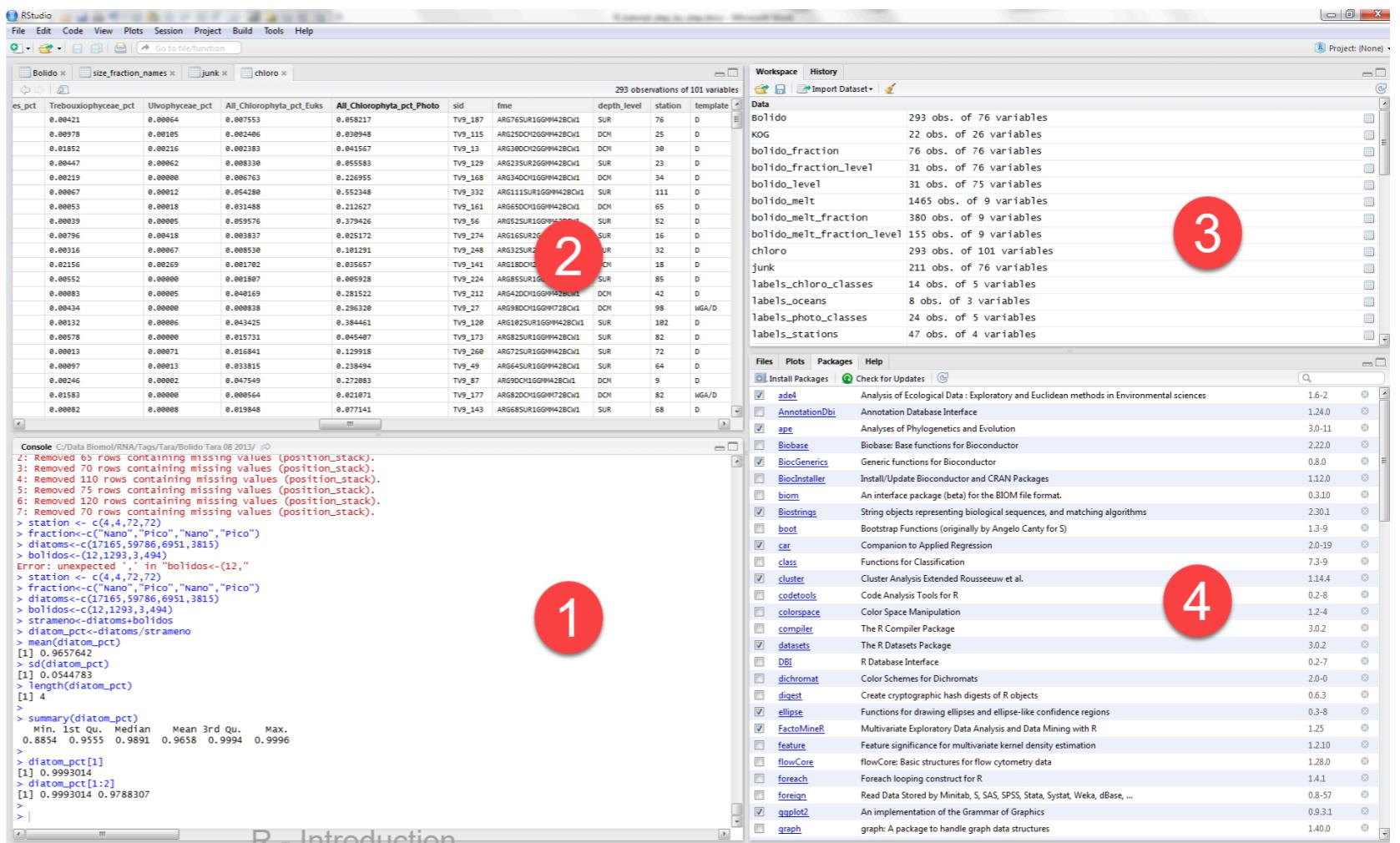
```
R> dd[order(-dd[,4], dd[,1]), ]
#> #>   b x y z
#> #> 4 Low C 9 2
```

**Page Footer:** R - Introduction

# Let's get started

# The R studio interface

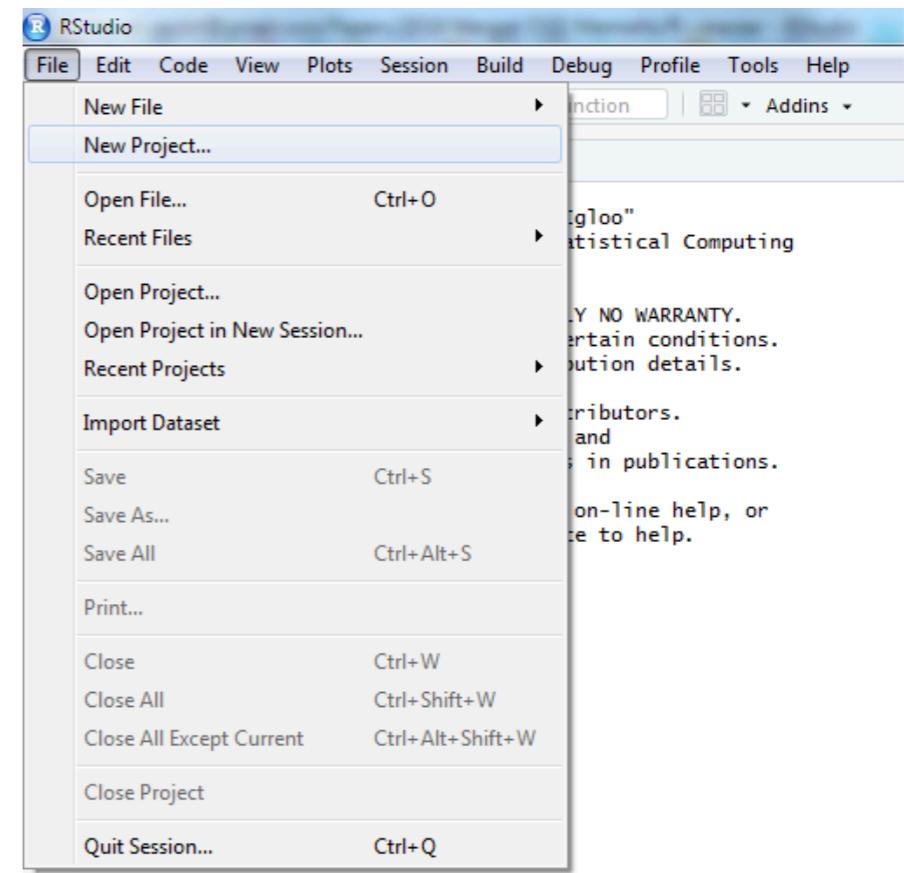
- Bottom left
  - Console
- Top left
  - File editor for .R and .Rmd files
  - Data frame visualization
- Top right
  - Environment (i.e. R objects)
  - History
- Bottom right
  - Files
  - Plots
  - Packages
  - Help



# Create a new project

---

- Open R studio
- Create new project for the course in a new directory
  - e.g. Microbes course



# Your first script

---

## Two ways to proceed

1. Type directly in command window

```
> print("Hello world")
```

```
[1] "Hello world"
```

2. Create a new script

- | Type in script window
- | \* Select and execute (CTRL-R)
- | \* Source the script

# R objects

# Variables

---

**Variables** are abstracting your data.

Variables are **objects**

- Create a variable

```
> greeting = "Hello world"  
> print(greeting)
```

```
[1] "Hello world"
```

- Update variable

```
> greeting = "Bonjour"  
> print(greeting)
```

```
[1] "Bonjour"
```

# Assignment

---

- Assignment done with <-

```
> x <- 1  
> y <- 2  
> x + y
```

```
[1] 3
```

```
> z <- x + y  
> z
```

```
[1] 3
```

- = can be used instead of <- but refrain from it (not good style)

```
> z = x + y
```

# Visualizing objects

---

You can view the values of the objects in R-studio environment window (top-right)

The screenshot shows the RStudio interface with the 'Environment' tab selected. The top bar includes tabs for 'Environment', 'History', and 'Connections'. Below the tabs are icons for creating a new file, opening a file, importing a dataset, and a paintbrush icon. A dropdown menu labeled 'Global Environment' is open. The main area is titled 'Values' and lists three objects: 'x' with value '1', 'y' with value '2', and 'z' with value '3'. The background of the interface is blurred.

Values	
x	1
y	2
z	3

# R is case sensitive

---

```
z
```

```
[1] 3
```

```
Z
```

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

# Rules for naming objects

---

- Use
  - letters
  - numbers
  - the dot
  - the underscore (not the minus sign !)
- Start always with a letter
  - Myvariable, Myvariable1, Myvariable.1, Myvariable\_01 are OK
  - 1Myvariable, My-variable, Myvariable@ are **not** OK

# Rules for naming objects

---

- Use consistent naming: five conventions
  - alllowercase: e.g. adjustcolor
  - period.separated: e.g. plot.new
  - **underscore\_separated**: e.g. numeric\_version
  - lowerCamelCase: e.g. addTaskCallback
  - UpperCamelCase: e.g. SignatureMethod
- Prefer third one, much more easy to read
  - Use **names** for objects : **last\_name**
  - Use **verbs** for function : **build\_name**
- Think about best order
  - e.g. prefer maybe **name\_last** because then you can have name\_first, name\_full...
  - and you identify that all these objects are related to a name...

# Data types

---

- **character**: “Daniel”, “This is a course in R”, ‘Joe Biden’
- **numeric**: 2, 15.5, 10e-3
- **integer**: 2L (the L tells R to store this is an integer)
- **date**: 2018-02-25
- **logical**: TRUE, FALSE
- **complex**: 1+4i (complex numbers with real and imaginary parts)
- **No data** “NA”
- **Not a number** “NaN” (e.g. division by zero)

# Data structures

---

- Vector
- List
- Matrix
- Data frames
- Function

# Vectors

# Vectors

---

- The basic R structure is a vector (think as a column in Excel):

$$\begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix}$$

- A vector can contain only a single element

[ 10 ]

- Assign a value to a vector

```
x <- 10  
x
```

```
[1] 10
```

# Vectors

---

- Assign several elements

```
x <- c(10,20,30)
```

```
x
```

```
[1] 10 20 30
```

- Assign range

```
x <- 10:30
```

```
x
```

```
[1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

- Assign characters

```
PoTU <- c("Jo", "Biden")
```

```
PoTU
```

```
[1] "Jo"     "Biden"
```

- Assign logical

```
flags <- c(TRUE, FALSE, TRUE)
```

```
flags
```

# Access specific elements of a vector

---

- First

```
x[1]
```

```
[1] 10
```

- Range

```
x[1:5]
```

```
[1] 10 11 12 13 14
```

- Remove one element

```
x[-1]
```

```
[1] 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

# Determine object properties

---

Apply functions (we will come back to functions latter)

- **typeof()** - what is the object's data type (low-level)?
- **length()** - how long is it? What about two dimensional objects?

```
typeof(x)  
length(x)
```

```
[1] "integer"  
[1] 21
```

| What is the type and length of PoTU ?

# Operators

# Arithmetic Operators

---

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

We are performing vector operations !

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ \dots \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 3 \\ \dots \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 6 \\ \dots \end{bmatrix}$$

| Think about it as adding 2 columns in Excel.

# Arithmetic Operators

---

- Vector one element

```
x <- 1
y <- 2
z <- x + y
z
```

```
[1] 3
```

- Vector several elements

```
# Two instructions on the same line
x <- 1:9; y <- 1:9
z <- x + y
z
```

```
[1] 2 4 6 8 10 12 14 16 18
```

- Several instructions on same line separated by ;
- The hastag # indicate a comment -> Use heavily to document your code
- However, it is even better to use R markdown (we will see it later)

| Use the other operators

# Arithmetic Operators

---

- What happens when the vectors have different number of elements ?

```
x <- 1:9  
y <- 1  
z <- x + y  
z
```

```
[1] 2 3 4 5 6 7 8 9 10
```

Equivalent to

```
y <- c(1,1,1,1,1,1,1,1,1)
```

The recycling rule...

# Can we add logical ?

```
x <- TRUE  
y <- FALSE  
z <- x + y  
z
```

```
[1] 1
```

No error but...

The resulting variable is transformed to a **numeric**

| How you would show that ?

```
typeof(x)
```

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

```
typeof(z)
```

```
[1] "integer"
```

# Logical Operators

---

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

# Logical Operators

```
x <- TRUE  
y <- FALSE  
z1 <- x | y  
z2 <- x == y  
z1
```



```
[1] TRUE
```

```
z2
```



```
[1] FALSE
```

| Do not mix

- == which is logical operator
- = which is assignment

# Can we add characters ?

```
first <- "Jo"  
last <- "Biden"  
full <- first + last
```



```
Error in first + last: non-numeric argument to binary operator
```

Generates an error

| What can we do ?

# Functions

# Functions

Functions perform specific task on objects

- e.g. to concatatanate strings we use **paste()**

```
paste(first, last)
```

```
[1] "Jo Biden"
```

- Functions take **arguments** and return an object called **result**
- To know the arguments
  - Use “?”
  - Can also go directly to Help panel and type function name

```
? paste() # Do not forget the parenthesis
```

paste {base} ←

R Documentation

## Concatenate Strings

### Description

Concatenate vectors after converting to character.

### Usage Examples

```
## When passing a single vector, paste0 and paste work like as.character.  
paste (. paste0(1:12)  
paste0(. paste(1:12)      # same  
as.character(1:12) # same  
  
## If you pass several vectors to paste0, they are concatenated in a  
...  ## vectorized way.  
sep  (nth <- paste0(1:12, c("st", "nd", "rd", rep("th", 9))))  
collapse ## paste works the same, but separates each input with a space.  
## Notice that the recycling rules make every input as long as the longest input.  
paste(month.abb, "is the", nth, "month of the year.")  
  
Details paste(month.abb, letters)  
  
paste(collapse = " ") ## You can change the separator by passing a sep argument  
the string `## which can be multiple characters.  
result.Vec paste(month.abb, "is the", nth, "month of the year.", sep = "_*_")  
  
Note that g ## To collapse the output into a single string, pass a collapse argument.  
e.g., when paste0(nth, collapse = ", ")  
  
paste0(. ## For inputs of length 1, use the sep argument rather than collapse  
paste("1st", "2nd", "3rd", collapse = ", ") # probably not what you wanted  
If a value is paste("1st", "2nd", "3rd", sep = ", ")  
elements b  
## You can combine the sep and collapse arguments together.  
paste(month.abb, nth, sep = ": ", collapse = "; ")  
  
## Using paste() in combination with strwrap() can be useful  
## for dealing with long strings.  
(title <- paste(strwrap(  
  "Stopping distance of cars (ft) vs. speed (mph) from Ezekiel (1930)",  
  width = 30), collapse = "\n"))  
plot(dist ~ speed, cars, main = title)
```

# Getting what you want

---

Let's apply paste :

```
paste(first, last)
```

```
[1] "Jo Biden"
```

- We would like to get “Jo\_Biden”

| Can you read the help and suggest a change in the way we call the function ?

```
paste(first, last, sep="_")
```

```
[1] "Jo_Biden"
```

# Write your own function

| If you write 3 times the same piece of code, then write a function...

```
my_sum <- function(a, b) {  
  c <- a + b  
  return (c)  
}
```

- **my\_sum** : function name
- **a, b** : arguments
- instructions are enclosed by braces (**{}**)
- **return()** : the value(s) returned
- More compact way

```
my_sum <- function(a, b) {a + b}
```

# Call your function

---

```
my_sum(10, 20)
```

```
[1] 30
```

- better

```
my_sum(a = 10, b = 20)
```

```
[1] 30
```

| Write a function to compute a product

# Examples of functions

---

Most of the time you do not have to write functions because someone has already written one for what you want to do...

- Sum

```
x <- 1:100  
sum(x)
```

```
[1] 5050
```

- Sampling a normal distribution

```
y <- rnorm(10, mean = 0, sd = 1)  
y
```

```
[1] -1.2082061 -0.1915338 -1.4461329 -1.5664386 -2.5408725 -1.4697788  
[7] 0.5294263 0.1391368 1.7930026 -0.8428267
```

# Statistics

---

```
mean(y)  
sd(y)
```



```
[1] -0.6804224  
[1] 1.258433
```

Sample more points... 10,000 instead of 100

```
y <- rnorm(10000, mean = 0, sd = 1)  
mean(y)  
sd(y)
```



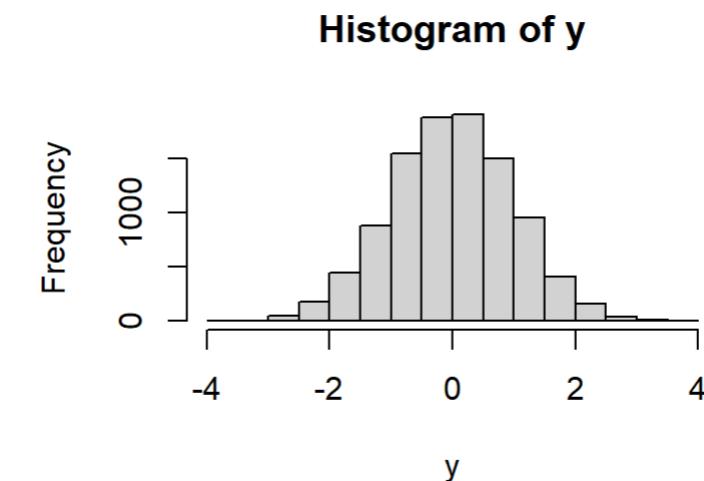
```
[1] -0.003926506  
[1] 0.9976069
```

# Plot

---

- Histogram

```
library(graphics)  
hist(y)
```



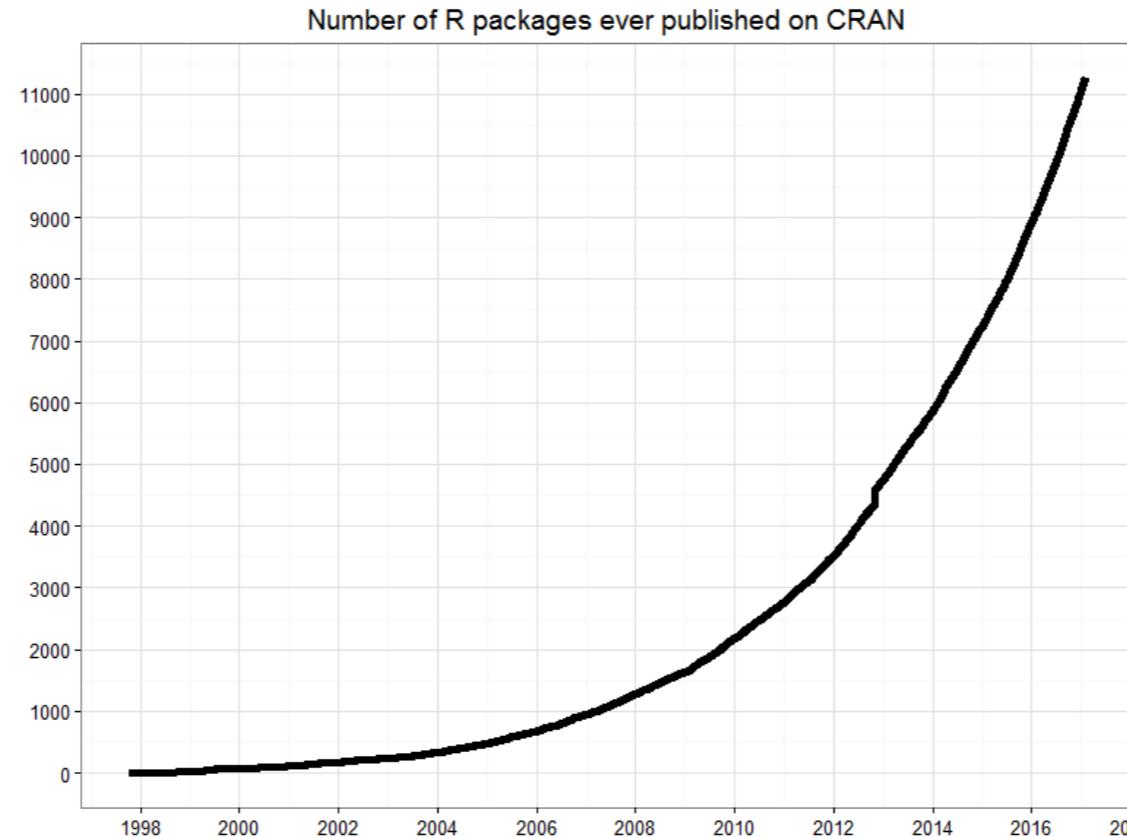
| What is this "library()"

# Packages

# Packages

---

- Packages are set of functions that have a common goal.
- They are really the strength of R

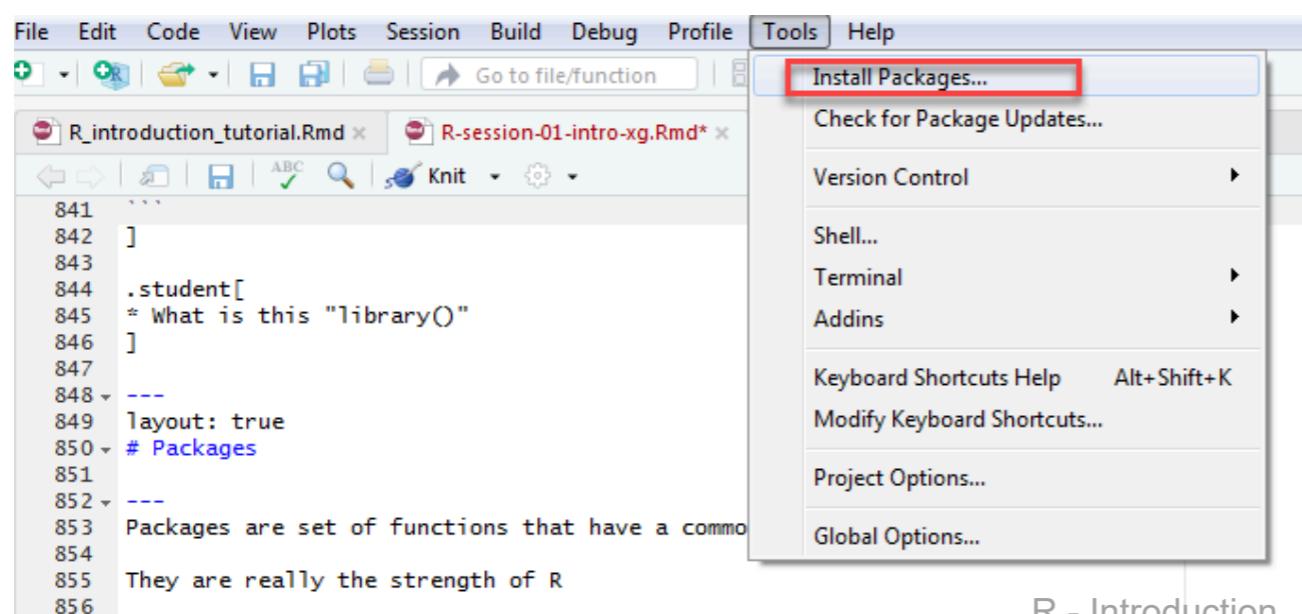


- And these are only the “official” packages. You can find more on GitHub

# Installing a package

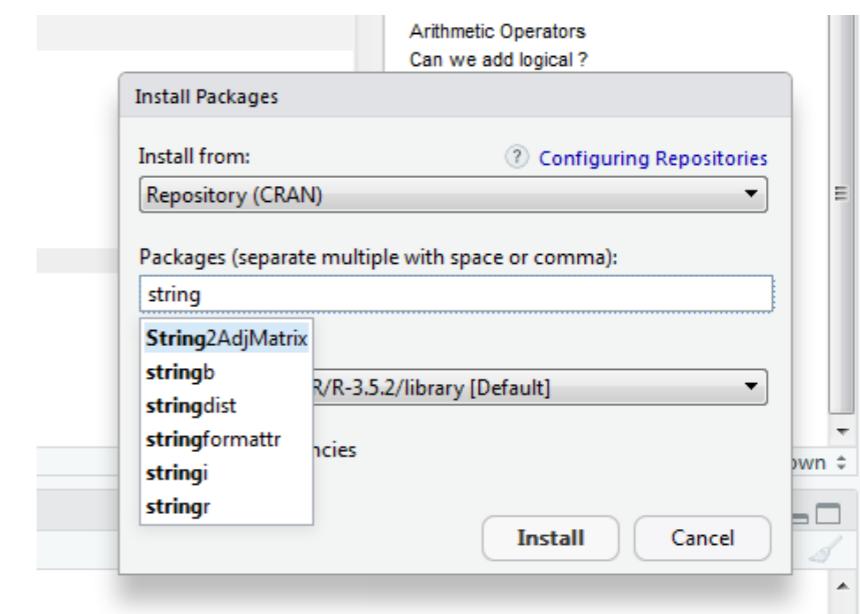
Download on your computer the package you need

| Install package **stringr** (to manipulate strings of characters)



The screenshot shows the RStudio interface. On the left is a code editor with two files open: "R\_introduction\_tutorial.Rmd" and "R-session-01-intro-xg.Rmd\*". The code in "R\_introduction\_tutorial.Rmd" includes R code and comments about packages. The "Tools" menu is open, and the "Install Packages..." option is highlighted with a red box.

```
841
842 ]
843 .student[
844 # What is this "library()"
845 ]
846
847
848 ---
849 layout: true
850 # Packages
851
852 ---
853 Packages are set of functions that have a common
854 They are really the strength of R
855
```



# Using a package

---

To use functions from the package

- use the syntax `package::function`

```
stringr::str_c(first, last, sep= " ")
```

```
[1] "Jo Biden"
```

OR

- load the package with the `library` function

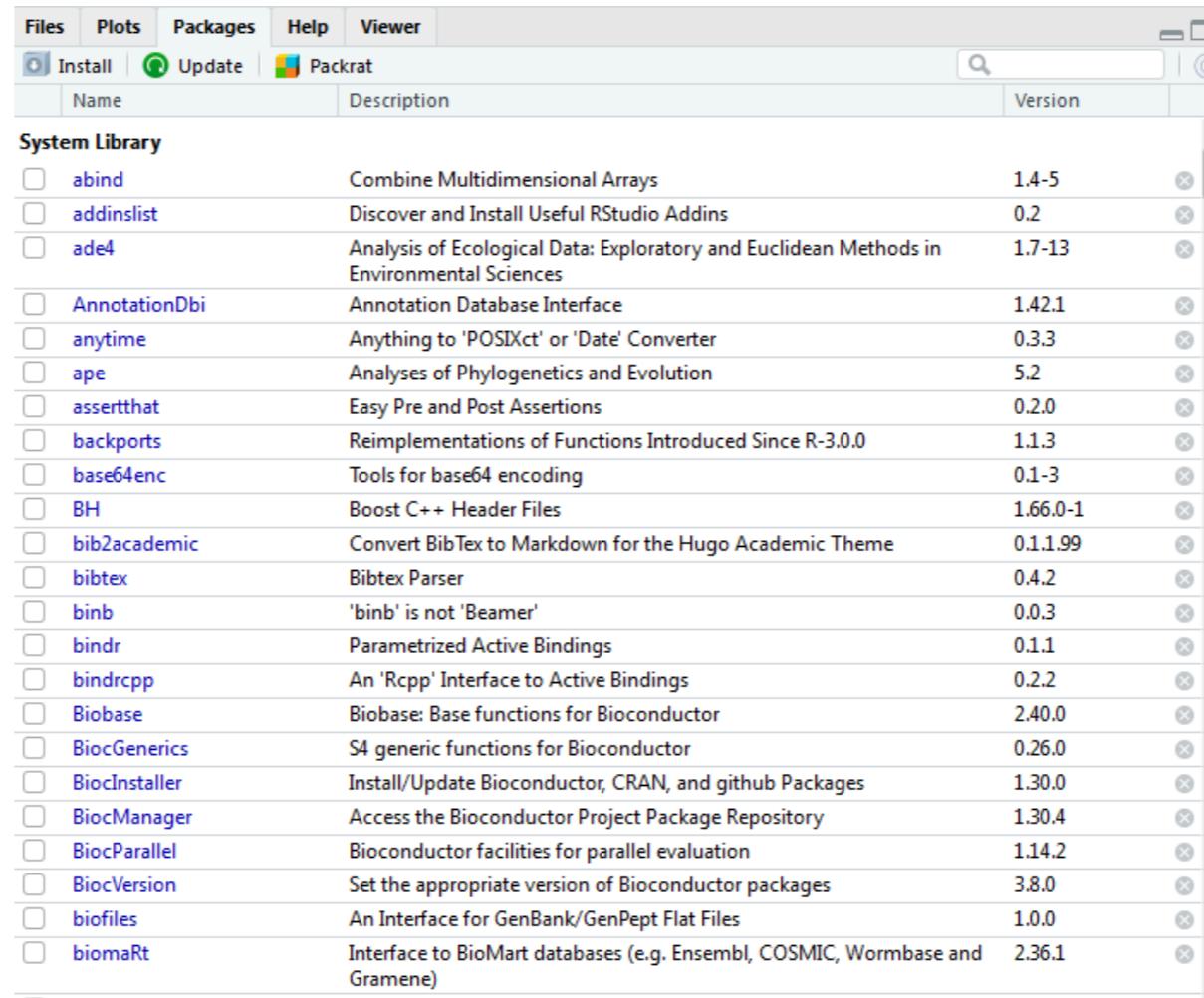
```
library(stringr)
str_c(first, last, sep= " ")
```

```
[1] "Jo Biden"
```

| Sometimes functions from different libraries have similar names

# List installed packages

---



The screenshot shows the RStudio interface with the 'Packages' tab selected. The window title is 'Packrat'. The table lists various R packages, categorized under 'System Library'. Each row includes a checkbox, the package name, its description, and its version number.

	Name	Description	Version
<b>System Library</b>			
<input type="checkbox"/>	abind	Combine Multidimensional Arrays	1.4-5
<input type="checkbox"/>	addinslist	Discover and Install Useful RStudio Addins	0.2
<input type="checkbox"/>	ade4	Analysis of Ecological Data: Exploratory and Euclidean Methods in Environmental Sciences	1.7-13
<input type="checkbox"/>	AnnotationDbi	Annotation Database Interface	1.42.1
<input type="checkbox"/>	anytime	Anything to 'POSIXct' or 'Date' Converter	0.3.3
<input type="checkbox"/>	ape	Analyses of Phylogenetics and Evolution	5.2
<input type="checkbox"/>	assertthat	Easy Pre and Post Assertions	0.2.0
<input type="checkbox"/>	backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.3
<input type="checkbox"/>	base64enc	Tools for base64 encoding	0.1-3
<input type="checkbox"/>	BH	Boost C++ Header Files	1.66.0-1
<input type="checkbox"/>	bib2academic	Convert BibTex to Markdown for the Hugo Academic Theme	0.1.1.99
<input type="checkbox"/>	bibtex	Bibtex Parser	0.4.2
<input type="checkbox"/>	binb	'binb' is not 'Beamer'	0.0.3
<input type="checkbox"/>	bindr	Parametrized Active Bindings	0.1.1
<input type="checkbox"/>	bindrcpp	An 'Rcpp' Interface to Active Bindings	0.2.2
<input type="checkbox"/>	Biobase	Biobase: Base functions for Bioconductor	2.40.0
<input type="checkbox"/>	BiocGenerics	S4 generic functions for Bioconductor	0.26.0
<input type="checkbox"/>	BiocInstaller	Install/Update Bioconductor, CRAN, and github Packages	1.30.0
<input type="checkbox"/>	BiocManager	Access the Bioconductor Project Package Repository	1.30.4
<input type="checkbox"/>	BiocParallel	Bioconductor facilities for parallel evaluation	1.14.2
<input type="checkbox"/>	BiocVersion	Set the appropriate version of Bioconductor packages	3.8.0
<input type="checkbox"/>	biofiles	An Interface for GenBank/GenPept Flat Files	1.0.0
<input type="checkbox"/>	biomaRt	Interface to BioMart databases (e.g. Ensembl, COSMIC, Wormbase and Gramene)	2.36.1

# Recap

---

- R is case sensitive:  $Z \neq z$
- Objects: data types vs data structures
- Vectors: think in vector operations
- Operators: arithmetic vs. logical
- Functions: try to practice

## Next: 02 - Data wrangling

- Data frames
- Concept of tidy data
- Reading data
- Manipulating data
- Selecting columns
- Selecting rows

