



# INTRODUZIONE AD R

**Marzo 2019**

# Organizzazione del corso

Organizzato in 2 parti:

1. Introduzione ad R
  - Presentazione del SW e delle principali funzionalità
  - Strutture dati e strutture di controllo
  - Semplici funzioni e grafici di base
2. Utilizzo di alcuni pacchetti in Rstudio integrati nell'ambiente *tidyverse*

# Organizzazione del corso

- Si parte dalle basi, vedendo alcune delle *best practices* ...
- Si alterneranno parti teoriche ad esercitazioni pratiche
- Enfasi su utilizzo dell'help e delle risorse disponibili online (tipicamente gratis)
- Esercizio finale per verificare le competenze
- Alcune slide, con uno sfondo bianco, presentano alcuni approfondimenti ai temi trattati in questa introduzione. Sono utili soprattutto in fase di rilettura del materiale.

# Organizzazione del corso

Il calendario degli incontri è il seguente:

- venerdì 1 marzo dalle 14:00 alle 18:00
- mercoledì 6 marzo dalle 14:00 alle 18:00
- mercoledì 13 marzo dalle 14:00 alle 18:00
- venerdì 15 marzo dalle 14:00 alle 18:00
- mercoledì 20 marzo dalle 14:00 alle 18:00
- **Approfondimento tematico:** venerdì 22 marzo dalle 14:00 alle 18:00

Ogni lezione sarà divisa in due o tre parti separate da un coffee break.

# R

E' un insieme integrato di risorse SW per la manipolazione dei dati, il calcolo e la visualizzazione di grafici.

## **Perché dovremmo utilizzare R?**

- efficace manipolazione e memorizzazione dei dati;
- ampia, coerente e integrata raccolta di strumenti per l'analisi intermedia dei dati;
- risorse grafiche per l'analisi dei dati;
- ben sviluppato, semplice linguaggio di programmazione;
- la community è molto ampia (#rstats su twitter), variegata ed esistono tantissimi gruppi (es: milanoR) e conferenze (useR);
- cutting edge tools: tipicamente un nuovo lavoro/metodologia viene presentato assieme ad un pacchetto R da usare per replicare le analisi.

# R

## Perché non dovremmo utilizzare R?

- Esistono numerose particolarità ed eccezioni che è necessario ricordare.
- Anche le funzioni di base hanno una struttura spesso inconsistente.
- Tipicamente, essendo utilizzato da statistici e non da informatici, le cosiddette *best practices* vengono ignorate.
- Uno script R può rivelarsi molto più lento del suo equivalente in C++, python, Julia o simili.

# R

*R* è un dialetto di *S*, sviluppato da *John Chambers* e collaboratori presso i *Bell Labs* a partire dal 1976, nel 1988 viene riscritto interamente in C.

*John Chambers* ebbe ad affermare:

*Volevamo che gli utenti fossero in grado di iniziare con un ambiente interattivo, dove non fossero consci del fatto che stavano programmando. Quando i bisogni fossero diventati chiari ed il livello di capacità aumentato si doveva fornire loro una transizione graduale verso la programmazione, proprio dove linguaggio e aspetti sistemici dovrebbero diventare più rilevanti.*

<http://www.stat.bell-labs.com/S/history.html>

# Storia

**1991:** creato in Nuova Zelanda da Ross Ihaka e Robert Gentleman.

**1993:** annuncio pubblico

**1995:** adozione di licenza GNU General Public (R software libero)

....

....

**2000:** rilascio versione 1.0.0

....

**2018:** rilascio versione 3.5.2 (20/12/2018)

**2019:** la versione 3.5.3 dovrebbe essere resa disponibile l'11 Marzo.



# Caratteristiche

- Funziona su tutte le piattaforme computazionali e sistemi operativi
- Release molto frequenti, (annuali + bug fixing release); sviluppo attivo.
- Abbastanza "pulito", da quando è stato reso disponibile; funzionalità suddivise in packages modulari
- Funzionalità grafiche molto sofisticate, migliori di molti altri software
- Utile per lavorare in modalità interattiva, rende disponibile un linguaggio di programmazione potente per sviluppare nuovi tool (es: shiny, plumber)
- Comunità di utilizzatori molto attiva

# Caratteristiche

È **gratis** ed **Open Source** per cui si ha garantita la libertà di:

- eseguire il programma, con qualsiasi finalità **(livello 0)**.
- studiare come il programma funziona, adattandolo alle proprie esigenze **(livello 1)**. Accedere al codice sorgente è condizione per questo.
- ridistribuire copie per aiutare i "vicini" **(livello 2)**.
- migliorare il programma, renderli pubblicamente disponibili in modo che l'intera comunità ne benefici **(livello 3)**. Accedere al codice sorgente è condizione per questo.

<http://www.fsf.org>

# Design del Sistema R

Suddiviso in due parti concettuali

*1. Sistema base, scaricabile da CRAN*

*2. Resto del mondo*

- Funzionalità suddivise in **packages** (insieme di funzioni, dati e codice compilato in un formato ben definito).
- Sistema base contiene i package base richiesti per far girare R e contenente le funzioni fondamentali
- Alcuni packages contenuti nel sistema base sono **utils**, **stats**, **datasets**, **graphics**, **grDevices**, **grid**, **methods**, **tools**, **parallel**, **compiler**, **splines**, **tcltk**, **stats4**.

# Design del Sistema R

Moltissimi altri package disponibili pubblicamente. Più di 13,000 packages su CRAN sviluppati da utenti e programmatori sparsi nel mondo.

Parecchi packages associati al progetto di ricerca Bioconductor: un progetto software internazionale il cui obiettivo consiste nel fornire strumenti software di alta qualità per attività professionali e di ricerca nell'ambito della bioinformatica

<http://bioconductor.org>

Un'altra piattaforma utilizzata molto spesso per condividere alcuni pacchetti R (soprattutto se al loro stato embrionale) è <https://github.com/>

# Risorse R

Scaricare il SW: <https://cran.r-project.org>

Documentation: <https://cran.r-project.org/manuals.html>

- Introduzione ad R
- R Data Import/Export
- Scrivere estensioni di R
- Installazione ed amministrazione di R

R Project

<https://www.r-project.org>

# Altre risorse

Lista di testi su CRAN

<http://www.r-project.org/doc/bib/R-books.html>

Libri:

”R in Action” - Kabacoff 2011

Hand on Programming with R: <https://rstudio-education.github.io/hopr/>

**“The R Software – Fundamentals of Programming and Statistical Analysis” –  
Lafaye de Micheaux, Pierre Drouhillet, R  my Liquet Benoit**

“The R Inferno”: [https://www.burns-stat.com/pages/Tutor/R\\_inferno.pdf](https://www.burns-stat.com/pages/Tutor/R_inferno.pdf)

## Altre risorse

Esistono numerosissimi corsi R online disponibili su piattaforme come Coursera o DataCamp. Inoltre:

- Chromebook Data Science:  
<https://jhudatascience.org/chromebookdatascience/>
- R for Data Science: <https://r4ds.had.co.nz/> (dedicato a *tidyverse*).
- Advanced R: <https://adv-r.hadley.nz/> (disponibile tra qualche mese/anno)

# Altre risorse

- Se utilizzate Rstudio esistono numerosi **Cheatsheet** che illustrano le principali funzionalità di alcuni pacchetti R:

<https://www.rstudio.com/resources/cheatsheets/>

- Se vi occupate di un particolare ambito scientifico ed utilizzate R esistono delle mailing list apposite per discutere di dubbi/problemi:

<https://www.r-project.org/mail.html>

- Esistono numerosi (non tutti introduttivi):

<https://blog.rstudio.com/2019/02/06/rstudio-conf-2019-workshops/>

- I talk dell'ultima RstudioConf 2019:

<https://resources.rstudio.com/rstudio-conf-2019>



# Come imparare ad usare R? Giocandoci!



via boredpanda, bbc, reddit

Your taste develops faster than your ability.

Fonte: <https://www.youtube.com/watch?v=7oyiPBjLAWY>

# INTRODUZIONE AD R

**Rstudio**

# Rstudio

RStudio is an integrated development environment, or IDE, for R programming. Download and install it from <http://www.rstudio.com/download>.

It's updated once or twice a year and you get a note if it's necessary.

RStudio

File Edit Code View Plots Session Build Debug Profile Tools Help

Go to file/function Addins

Project: (None)

Environment History Connections

Import Dataset

Global Environment

Environment is empty

Files Plots Packages Help Viewer

print

R: Print Values Find in Topic

print {base} R Documentation

## Print Values

### Description

`print` prints its argument and returns it *invisibly* (via `invisible\(x\)`). It is a generic function which means that new printing methods can be easily added for new [classes](#).

### Usage

```
print(x, ...)
```

```
## S3 method for class 'factor'
print(x, quote = FALSE, max.levels = NULL,
      width = getOption("width"), ...)
```

```
## S3 method for class 'table'
print(x, digits = getOption("digits"), quote = FALSE,
      na.print = "", zero.print = "0",
      right = is.numeric(x) || is.complex(x),
      justify = "none", ...)
```

```
## S3 method for class 'function'
print(x, useSource = TRUE, ...)
```

1:1 (Top Level) R Script

Console Terminal

```
~/
```

```
R version 3.5.2 (2018-12-20) -- "Eggshell Igloo"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (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.

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.

> |
```

# INTRODUZIONE AD R

## Concetti preliminari

# R objects:

To understand computations in R, two slogans are helpful:

1. Everything that exists is an object.
2. Everything that happens is a function call.

(John Chambers)

# Help...

E' fondamentale conoscerne l'utilizzo:


- `help.start()` # formato HTML
- `help(mean)` # permette di ottenere l'help di una specifica funzione
- `?mean` # in alternativa
- `help("for")` # tra doppi o singoli apici per le parole riservate e per i caratteri speciali (es: TRUE, FALSE, NA,...)
- `help.search("mean")` # cerca la stringa mean in tutta la documentazione
- `??mean` # in alternativa al precedente comando
- `?help` # per ulteriori dettagli

# ?print



**Andrew Hall** @andrewhall\_NU · 14 Std.  
Reading R help documentation.



 Tweet übersetzen

How to draw an owl

1.



1. Draw some circles

2.



2. Draw the rest of the fucking owl



# Help me Help you

E' possibile che, anche leggendo l'help, non si capisca come utilizzare una particolare funzione o comando. Che fare?

- <https://stackoverflow.com/>
- <https://guides.github.com/features/issues/>

In questi casi è fondamentale creare un esempio semplice e riproducibile che illustri il problema ed aiuti chi ci vorrebbe rispondere.



**Approfondimento:** <https://reprex.tidyverse.org/>

# Workspace

- `getwd()` #working directory
- `setwd("mydirectory")` #cambiare la working directory      esempio in Windows ("c:\\CorsoR")
- `ls()` #lista degli oggetti presenti nel workspace
- `rm(myobject)` #cancella uno o più oggetti
- `rm(list=ls())` #per rimuovere tutti gli oggetti

# Project Oriented Workflow

If the first line of your R script is

```
setwd("C:\\Users\\jenny\\path\\that\\only\\I\\have")
```

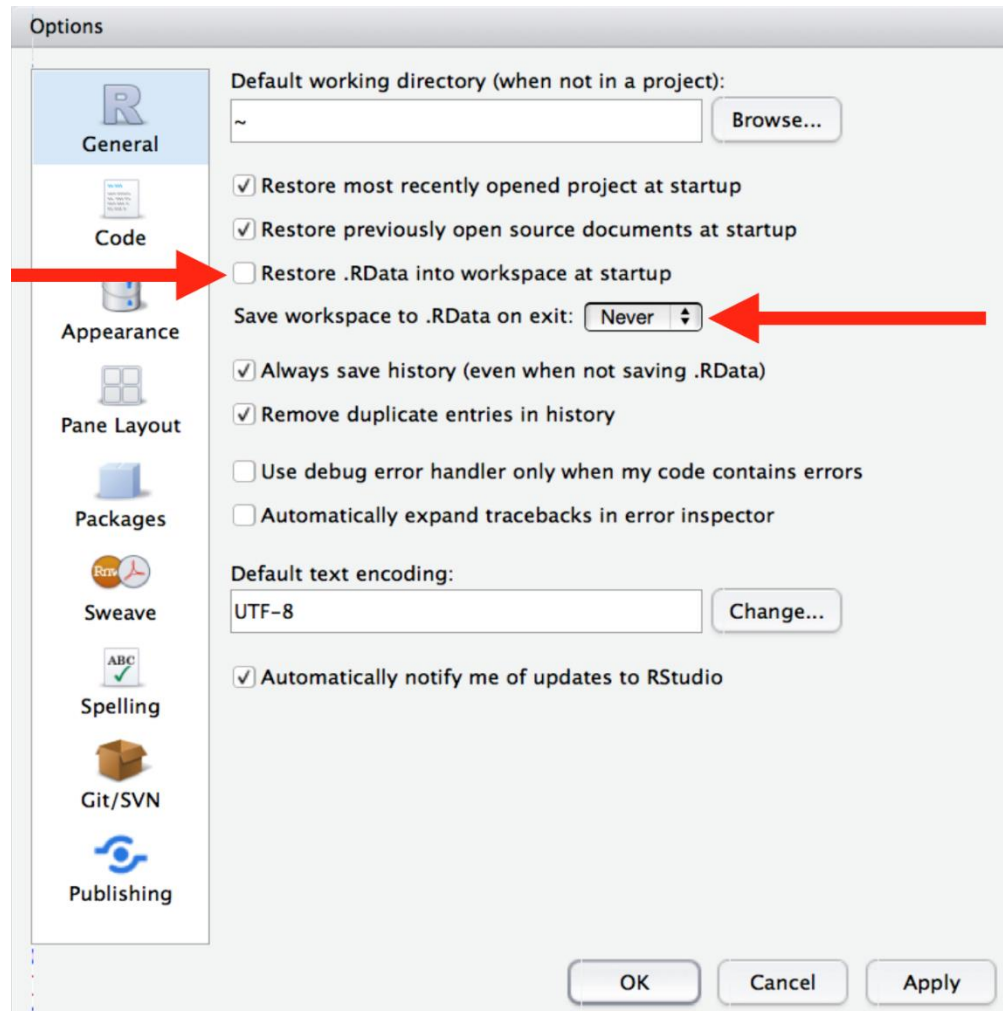
I\* will come into your office and  
SET YOUR COMPUTER ON FIRE 🔥.

\* or maybe Timothée Poisot will

**Approfondimento:**

<https://www.tidyverse.org/articles/2017/12/workflow-vs-script/>

# Project Oriented Workflow



Approfondimento: <https://r4ds.had.co.nz/workflow-projects.html>

# Packages

- `library()` #packages installati nella libreria `.libpath()`

Molti dei pacchetti R sviluppati dagli utenti vengono salvati su di un server chiamato CRAN: Comprehensive R Archive Network.

- `install.packages("nome package")` #installare un pacchetto dal CRAN

Potete anche utilizzare l'interfaccia proposta da Rstudio.

- `library(nome package)` #caricare pacchetto. Restituisce un errore se non è presente in `.libpath()`

# Packages - Esercizio

- Proviamo ad installare dal CRAN il pacchetto *dplyr*.

I pacchetti R ancora in fase di sviluppo o in beta-testing vengono solitamente salvati su github. Per installare un pacchetto da github possiamo usare la funzione

*install\_github*(«nome repository/nome pacchetto»)

E' necessario avere il pacchetto *devtools* (scaricabile dal CRAN) e potrebbe essere necessario anche il software *Rtools* scaricabile da <https://cran.r-project.org/bin/windows/Rtools/>.

- Proviamo ad installare da github il pacchetto *trump* scaricabile da <https://github.com/romainfrancois/trump>

# library vs require

E' possibile utilizzare entrambe le funzioni per caricare un pacchetto R ma non fanno la stessa cosa.

In linea di principio dovrete sempre preferire *library*, utilizzando *require* solo se strettamente necessario.

## Approfondimento:

- <https://yihui.name/en/2014/07/library-vs-require/>
- <https://stackoverflow.com/questions/5595512/what-is-the-difference-between-require-and-library>

# Packages – Demo e vignette

E' possibile che, ad un dato pacchetto R, siano associate una o più vignette che illustrano come utilizzare alcune funzioni.

Con il comando *vignette(package = «nome\_pacchetto»)* è possibile ottenere una lista di tutte le vignette associate ad un pacchetto. Per visualizzarla utilizzo nuovamente la funzione:

*vignette(topic = «nome\_topic», package = «nome\_pacchetto»)*

Alcuni pacchetti possono avere anche delle demo, visualizzabili in maniera analoga alle vignette con la funzione *demo()*.



# Data

- `data(package='nome package')` #mostra la lista dei dataset presenti in un particolare pacchetto
- `data()` #lista dataset presenti nei pacchetti caricati
- `data(nome dataset)` #caricare dataset presente nella versione base di R

Proviamo a visualizzare i dataset presenti nel pacchetto *cluster* ed a caricarne uno.

NB: Esiste anche un pacchetto R chiamato *datasets* che contiene numerosi datasets da usare per esercizio.

# Esercizio

- Verificare di non avere già installato il package vcd
- Installare il package vcd (in ogni caso).
- Caricare il package vcd
- Listare i dataset presenti nel pacchetto vcd e caricarne almeno uno.
- Leggere l'help della funzione «mosaic»
- Caricare una delle possibili vignette
- Eseguire una delle demo.

# INTRODUZIONE AD R

## Tipi di Dati e Comandi di base

# Prima di cominciare ...

- R differenzia tra minuscole e maiuscole (in gergo si dice essere *case sensitive*)
- # indica un commento. Potete usare «CTRL + Shift + C» per commentare tutto un blocco di codice.

Se utilizzate Rstudio potete usare «Alt + Shift + K» per vedere tutte le scorciatoie utilizzabili.

NB: E' **MOLTO** importante che voi commentiate il codice, descrivendo cosa una certa funzione fa e perché avete scelto di usarla. Serve a tutte le altre persone che dovranno leggere il vostro codice ma soprattutto a voi stessi quando andrete a rileggerlo dopo tanti mesi.

## Prima di cominciare ...

`<-` è il simbolo di assegnazione (anche `->` ). Utilizzando Rstudio è possibile utilizzare la scorciatoia «Alt + -».

**Using = instead of `<-` for assignment**



# Prima di cominciare ... (Names style guide)

“There are only two hard things in Computer Science: cache invalidation and naming things.”

-- Phil Karlton

Quando dovete decidere il nome da assegnare ad un oggetto dovrete rispettare alcune regole di base:

1. Usare solamente lettere in minuscolo e numeri.
2. Utilizzare gli underscore, `_`, per separare diverse parole che compongono il nome di un oggetto (es: `model_one`).
3. Evitate nomi che potrebbero generare ambiguità.
4. Utilizzare una spaziatura adeguata; es: `x <- 2` è meglio che `x<-2`.

**Approfondimento:** <https://style.tidyverse.org/syntax.html>

# Prima di cominciare ... (Names style guide)

When you try to choose  
a meaningful variable name.



# Prima di cominciare ... (Reserved Words)

Non tutte le parole possono essere utilizzate come nomi di oggetti. Esistono infatti alcune parole riservate come *for*, *while* e *if* che non possono essere rinominate ed identificano precisi comandi.

E' possibile leggerne l'elenco completo digitando il comando *?Reserved*.



# Prima di cominciare ... (Mathematic)

- Operazioni semplici:  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $^$ ,  $\% \%$  (divisione intera tra due numeri),  $\% \%$  (resto della divisione intera)
- Operatori relazionali (ritornano TRUE o FALSE):  $>$ ,  $<$ ,  $>=$ ,  $<=$ ,  $==$ ,  $!=$
- Operatori logici:  $!$ ,  $\&$  (AND),  $|$  (OR)
- Il logaritmo di un numero viene calcolato con la funzione *log*, il fattoriale con la funzione *factorial*, la radice quadrata con la funzione *sqrt*, il seno, il coseno e la tangente con le funzioni *sin*, *cos* e *tan*.

# Input

E' possibile usare R anche come una calcolatrice utilizzando le funzioni appena viste:

- `3 * 2; exp(3); log(10); sin(pi); sqrt(4)`
- `x <- 1`
- `y <- 2`
- `z <- x + y; z`

# Esercizio

La formula per calcolare il BMI (Indice di Massa Corporea) per una persona è

$$\text{BMI} = \text{Peso}(\text{kg}) / \text{Altezza}(\text{m})^2$$

- Definire una variabile, chiamata *peso*, ed assegnargli il valore 95
- Definire una variabile, chiamata *altezza*, ed assegnargli il valore 1,80.
- Calcolare il BMI per quella persona.

ATTENZIONE ALL'ORDINE DELLE OPERAZIONI MATEMATICHE!

# Oggetti

E' possibile usare R anche come una calcolatrice utilizzando le funzioni appena viste. Ad esempio:

- `3 * 2; exp(3); log(10); sin(pi); sqrt(4)`
- `x <- 1`
- `y <- 2`
- `z <- x + y; z`

# Oggetti

R ha 5 classi di **oggetti base**

- *character*
  - *numeric* che si divide in *double* e *integers*
  - *complex*
  - *logical (TRUE/FALSE) or boolean*
- 
- `typeof(msg)` # mostra come un oggetto viene memorizzato

# Numeri

Generalmente trattati come oggetti di tipo numeric (i.e. double precision real numbers). Se si desidera un intero si deve specificarlo tramite il suffisso L.

- *Inf*, numero speciale che rappresenta infinito;  $1/0$ ; *Inf* utilizzato computazioni ordinarie;  $1/Inf$  è uguale a 0. Esiste anche  $-Inf$ , es:  $\log(0)$ .
- *NaN*, rappresenta un valore indefinito (Not A Number) (rappresentato come double);  $0/0$
- *NA*, valore mancante

$3 * NA$  # restituisce NA

$NA == NA$  # restituisce NA

Esistono tuttavia alcune eccezioni:  $NA \mid TRUE, NA \& FALSE$

# Warning – Finite Precision Arithmetic

Un errore molto comune è il seguente:

- `x <- sqrt(2)`
- `x * x == 2`

La precedente espressione restituirà TRUE o FALSE?

**Approfondimento:** [https://cran.r-project.org/doc/FAQ/R-FAQ.html#Why-doesn\\_t-R-think-these-numbers-are-equal\\_003f](https://cran.r-project.org/doc/FAQ/R-FAQ.html#Why-doesn_t-R-think-these-numbers-are-equal_003f)

# Warning – Finite Precision Arithmetic

Un errore molto comune è il seguente:

- `x <- exp(10 ^ 5000)`
- `x`

Cosa restituirà la precedente espressione?

**Approfondimento:** [https://cran.r-project.org/doc/FAQ/R-FAQ.html#Why-doesn\\_0027t-R-think-these-numbers-are-equal\\_003f](https://cran.r-project.org/doc/FAQ/R-FAQ.html#Why-doesn_0027t-R-think-these-numbers-are-equal_003f)



## Altre semplici funzioni - sequenze

- `x <- 1:20` #crea una sequenza di numeri interi
- `x`

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

```
[16] 16 17 18 19 20
```

- `seq(1, 100, length.out = 10)` #crea una sequenza di 10 numeri con valore minimo pari a 1 e valore massimo pari a 100
- `seq(1, 100, by = 5)` #crea una sequenza da 1 a 100 con incremento 5
- `rep(2, times = 10)` #numero 2 ripetuto 10 volte
- `rep(1:5, each = 5)` #ripete ogni elemento 5 volte
- `rep(1:5, length.out = 12)` # resituisce un vettore di lunghezza 12 ripetendo i valori se necessario

# Esercizio

1. Moltiplicare 130 per 4.63
2. Impostare la funzione logica seconda la quale si chiede ad R se 5 è diverso da 3
3. Creare una variabile zz contenente la stringa Word
4. Generare una sequenza di valori che va da 0 a 1 con passo 0.1
5. Generare un sequenza di valori 0.1 lunga 20
6. Scrivere il comando per generare la sequenza:  
000111222333
7. Creare una sequenza da 30 a 70 di 5 numeri
8. Valutare (a mente se possibile) la seguente espressione:  
x <- 5  
y <- 7  
!(!(x < 4) & !(y > 12))

# INTRODUZIONE AD R

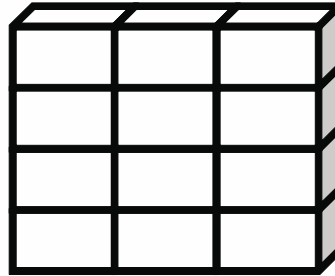
## Strutture Dati

# Strutture Dati

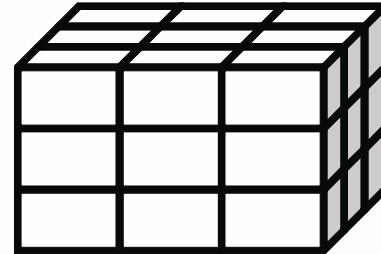
(a) Vector



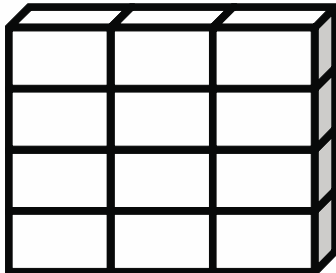
(b) Matrix



(c) Array

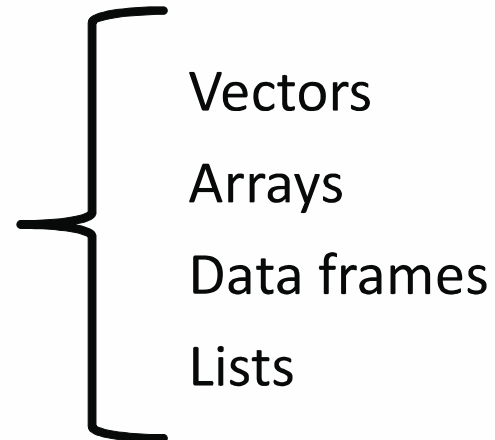


(d) Data frame



Columns can be different modes

(e) List



# Strutture Dati : Vettori

E' una "sequenza ordinata" di oggetti dello stesso tipo. La funzione **c()** (dove **c** è l'iniziale di *collection* o *concatenation*) può essere usata per creare vettori di oggetti

- `x <- c(0.5, 0.6) ## double`
- `x <- c(1L, 2L) ## integer`
- `x <- c(TRUE, FALSE) ## logical`
- `x <- c("a", "b", "c") ## character`
- `x <- 9:29 ## integer`

Per creare un vettore vuoto di lunghezza *n* posso utilizzare la funzione *numeric()* a cui devo però specificare il tipo di vettore che voglio creare

- `x <- vector("numeric", length = 10); x`

```
[1] 0 0 0 0 0 0 0 0 0 0
```

# Strutture Dati : Vettori

Operatore [ ] per fare subsetting

- `vt <- c(1,2, 3, 4.5, -6)` #costruzione di vettore di cinque elementi

**NB: R comincia a contare gli elementi di un vettore dalla posizione 1!!!**

- `vt[2]` #visualizza il secondo elemento del vettore

```
[1] 2
```

- `vt[3:5]` #visualizza dal terzo al quinto elemento

```
[1] 3.0 4.5 -6.0
```

- `vt[c(1,3)]` #visualizza il primo e il terzo

```
[1] 1 3
```

- `vt[-c(1,3)]` #non visualizza il primo e il terzo

```
[1] 2.0 4.5 -6.0
```

# Strutture Dati : Vettori

Selezionare elementi:

- `which(vt > 1)` #ritorna indice del vettore che soddisfa la condizione

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

- `vt[which(vt > 1)]` #estrapolo gli elementi

```
[1] 2.0 3.0 4.5
```

- `vt[vt>0]` #visualizza gli elementi maggiori di 0

```
[1] 1.0 2.0 3.0 4.5
```

# Strutture Dati : Vettori

Le operazioni sui vettori (o su matrici o su array) sono eseguite su ogni elemento contenuto nel vettore (o matrice o array)

- `vt * 2` #moltiplica ogni elemento per 2

```
[1]  2  4  6  9 -12
```

Qualsiasi tipologia di selezione (o qualunque tipo di operazione) che si fa su un vettore ritorna un vettore

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

- `y`

```
[1] 1.0 2.0 3.0 4.5 -6.0 1.0 1.0 2.0 3.0 4.5 -6.0
```



# Strutture Dati : Vettori (recycling)

Cosa accade quando si effettuano operazioni su vettori di lunghezza diversa?

- `r <- vt + y + 1` #genera un vettore di lunghezza pari al vettore più lungo

Warning message: In `vt + y` : longer object length is not a multiple of shorter object length

- `r`  
[1] 3.0 5.0 7.0 10.0 -11.0 3.0 4.0 6.0 8.5 -0.5 -4.0

# Strutture Dati : Vettori (Conversion)

Cosa accade se si scrive quanto sotto riportato?

- `y <- c(1.7, "a") ## character`
- `y <- c(TRUE, 2) ## numeric`
- `y <- c("a", TRUE) ## character`

Quando oggetti di tipo diverso vengono mescolati in un unico vettore, la coercizione (conversione) viene applicata, ogni elemento appartenente al vettore diviene della stessa classe. In che ordine?

*logical < integer < double < character*

# Strutture Dati : Vettori

Gli oggetti possono essere esplicitamente sottoposti a coercizione da una classe ad altra classe utilizzando la funzione **as.\*** se disponibile.

- `x <- 0:6; class(x)`

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

- `as.numeric(x)`

```
[1] 0 1 2 3 4 5 6
```

- `as.logical(x)`

```
[1] FALSE TRUE TRUE TRUE TRUE TRUE TRUE
```

- `as.character(x)`

```
[1] "0" "1" "2" "3" "4" "5" "6"
```

# Strutture Dati : Vettori

Un'operazione di coercizione che non sia opportunamente definita porta ad ottenere come risultato NAs.

- `x <- c("a", "b", "c")`
- `as.numeric(x)`

`[1] NA NA NA`

Warning message:

NAs introduced by coercion

- `as.logical(x)`

`[1] NA NA NA`

Nota: per determinare di che tipo è un oggetto usiamo **is.\***:

- `is.character(x)`

`[1] TRUE`

# Strutture Dati : Vettori

Attribuire un nome agli elementi del vettore

- `voti <- c(5,7,6,7)`
- `names(voti) <- c("matematica","latino","storia","inglese")`
- `voti["latino"]`
- `voti[c("latino","storia")]`

# Strutture Dati : Vettori

Alcune funzioni utili

- **length**(vt) #numero di elementi

```
[1] 5
```

- **sort**(vt) #ordinamento crescente

```
[1] -6.0  1.0  2.0  3.0  4.5
```

- **sort**(vt, decreasing = TRUE) #ordinamento decrescente

```
[1]  4.5  3.0  2.0  1.0 -6.0
```

- **sum**(vt);

```
[1] 4.5
```

- **prod**(vt)

```
[1] -162
```

# Strutture Dati : Vettori

Alcune funzioni statistiche utili

- **min**(vt) #minimo
- **max**(vt) #massimo
- **mean**(vt) #media
- **var**(vt) #varianza
- **sd**(vt) #deviazione standard
- **range**(vt) #valore minimo e valore massimo
- **summary**(vt) #principali statistiche di sintesi

# Strutture Dati : Vettori

Lavorare con gli NA

- `NA > 5`

`[1] NA`

- `10 == NA`

`[1] NA`

- `NA + 10; NA / 2`

`[1] NA; NA`

`NA == NA`

`[1] NA`



# Strutture Dati : Vettori

Alcune funzioni utili: lavorare con gli NA

- `vt <- c(NA,1,2,3,4.5,-6,NA)`
- `is.na(vt)` #indica la presenza di NA

```
[1] TRUE FALSE FALSE FALSE FALSE FALSE TRUE
```

- `sum(na.omit(vt))` #rimuove i missing e calcola la somma

```
[1] 4.5
```

- `sum(vt[!is.na(vt)])` #stesso risultato del comando precedente

# Esercizio

1. Pulire il proprio Environment riavviando R.
2. Creare un vettore  $x$  contenente i 6 valori 10, 7, 8, NA, 3, 4.
3. Verificare se nel vettore  $x$  sono presenti valori mancanti.
4. Verificare se nel vettore  $x$  è presente qualche elemento superiore a 4.
5. Calcolare la lunghezza del vettore  $x$ .
6. Calcolare la somma e la media del vettore  $x$ .
7. Ripetere l'esercizio al punto precedente escludendo il valore ad NA.
8. Cosa restituisce il comando  $x * 2$ ? Ed  $x ^ 2$ ? Ed  $X ^ 2$ ?
9. Creare un vettore  $y$  formato dai numeri interi da 1 a 6 e calcolare  $x + y$ .
10. Creare un vettore  $z$  formato dai numeri interi da 1 a 10. Cosa restituisce il comando  $x + z$ ?

# Esercizio

11. Creare un vettore  $w$  formato dalle stringhe da «A» a «F». Cosa restituisce il comando  $x + w$ ?
12. Creare un vettore  $v$  formato dalle stringhe da «1» a «6». Cosa restituisce il comando  $x + v$ ?
13. Cosa restituisce il comando  
$$\text{TRUE} + \text{FALSE} + \text{TRUE} * \text{FALSE} - \text{FALSE}^{\text{TRUE}}$$
14. Completare lo script in maniera opportuna in modo da ottenere come output: FALSE TRUE TRUE  
 $y \leftarrow \dots (\text{TRUE}, \text{FALSE}, \text{FALSE}); y$

# Strutture Dati : Matrici

Le matrici possono essere viste come una generalizzazione dei vettori in due dimensioni. Come per i vettori, tutti gli elementi di una matrice devono essere dello stesso tipo.

- `x <- matrix(1:12, nrow = 4, ncol = 3);`

- `x`

	[,1]	[,2]	[,3]
[1,]	1	5	9
[2,]	2	6	10
[3,]	3	7	11
[4,]	4	8	12

# Strutture Dati : Matrici

Come si vede dall'esempio le matrici vengono riempite per colonna. Possiamo modificare questo comportamento specificando l'opzione *byrow = TRUE* dentro la funzione *matrix()*.

- `x <- matrix(1:12, nrow = 4, ncol = 3, byrow = TRUE); x`

Possiamo ricavare le dimensioni di una matrice (cioè il numero di righe e di colonne) utilizzando la funzione *dim()*.

- `dim(x)`

```
[1] 4 3
```

Il primo numero rappresenta il numero di righe ed il secondo il numero di colonne

# Strutture Dati : Matrici

Possiamo estrarre gli elementi di una matrice in posizione  $i, j$  nel seguente modo:

- `x <- matrix(1:12, nrow = 4, ncol = 3);`

- `x`

	[,1]	[,2]	[,3]
[1,]	1	5	9
[2,]	2	6	10
[3,]	3	7	11
[4,]	4	8	12

- `x[2, 2]`

`[1] 6`

# Strutture Dati : Matrici

Se lasciamo uno dei due indici mancanti allora possiamo estrarre la  $i$  esima righe o la  $j$  esima colonna

- $x[1,]$

$[1]$  1 5 9

- $x[:,3]$

$[1]$  9 10 11 12

# Strutture Dati : Matrici

Il subsetting di una matrice restituisce un vettore. Possiamo modificare questo comportamento specificando l'opzione *drop = FALSE* dentro la funzione `[]`:

- `x[1, 2, drop = FALSE]`

`[,1]`

`[1,]`     5

- `x[1, , drop = FALSE]`

`[,1]`     `[,2]`     `[,3]`

`[1,]`     1         5         9



# Strutture Dati : Matrici

E' possibile attribuire un nome alle righe/colonne di una matrice con la funzione *dimnames()*.

- `m <- matrix(1:4, nrow = 2, ncol = 2)`
- `dimnames(m) <- list(c("a", "b"), c("c", "d"))`

- `m`

c d

a 1 3

b 2 4

# Strutture Dati : Matrici

Esistono due diverse funzioni per calcolare il prodotto tra due matrici.

- `x <- matrix(1:4, 2, 2); y <- matrix(rep(10, 4), 2, 2)`

- `x * y` ## prodotto element-wise

```
      [,1] [,2]
```

```
[1,] 10 30
```

```
[2,] 20 40
```

- `x %*% y` ## true matrix multiplication

```
      [,1] [,2]
```

```
[1,] 40 40
```

```
[2,] 60 60
```

- `x / y`

# Strutture Dati : Matrici

Alcune funzioni per il calcolo matriciale

- $x + 5$  somma 5 ad ogni elemento di  $x$ ;
- $t(x)$  calcola la trasposta di  $x$ ;
- $colSums(x)$  calcola la somma per colonna;
- $rowSums(x)$  calcola la somma per riga;
- $x[1,2] <- 5$  inserisce il valore 5 in posizione 1 e 2;
- $x[1,] <- v$  inserisce il vettore  $v$  (di dimensioni opportune) nella prima riga della matrice.
- $solve(x)$  calcola l'inversa di  $x$ ;
- $diag(n)$  crea una matrice con  $n$  righe ed  $n$  colonne avente tutti 1 sulla diagonale e 0 altrove.
- $lower.tri(x)$  restituisce una matrice di TRUE/FALSE avente TRUE solo per gli elementi sotto la diagonale principali.  $upper.tri(x)$  funziona in maniera speculare.
- $sum(diag(x))$  calcola la traccia di una matrice e  $det(x)$  il suo determinante.

# Esercizio

1. Pulire il proprio Environment riavviando R.
2. Creare a matrice **A** con i valori da 10 a 24 avente 5 righe e 3 colonne e valori inseriti per colonna e stamparla a schermo.
3. Calcolarne la somma per riga, per colonna, la trasposta e la traccia.
4. Sostituire la prima colonna di **A** definendo un nuovo vettore di dimensione opportuna.
5. Creare una nuova matrice chiamata **B** (di dimensione opportuna) e calcolare **A \* B** e **A %\*% B**.
6. Cosa restituisce il comando:

*matrix(c(1, 0, 0, 1), nrow = 2) \* matrix(1:4, nrow = 2)*

# Strutture Dati : Liste

Le liste sono gli elementi più generali e flessibili di R perché possono contenere elementi di qualsiasi tipo (anche altre liste).

Ad esempio

- `x <- list(1, 2, 3)`
- `class(x)`

```
[1] «list»
```

- `x`

```
[[1]]
```

```
[1] 1
```

```
[[2]]
```

```
[1] 2
```

```
[[3]]
```

```
[1] 3
```

# Strutture Dati : Liste

E' possibile anche dare un nome agli elementi dentro una lista:

- `x <- list(a = «Andrea», b = «barca», c = «casa»)`

*Per visualizzare la struttura delle liste (dato che il printing è tipicamente scomodo) possiamo usare la funzione `str()`*

- `x <- list(a = «a», b = 5, c = 5L, TRUE, list(4, 5))`
- `x`
- `str(x)`

# Strutture Dati : Liste - Subsetting

Esistono tre modi per fare subsetting da una lista:

1. La funzione `[` che restituirà una sottolista. Il risultato sarà sempre quindi una lista
2. La funzione `[[` che restituirà un elemento della lista, togliendo quindi un *livello di complessità* all'oggetto.
3. La funzione `$` che funziona analogamente ad `[[` solamente per liste con nomi.

# Strutture Dati : Liste - Subsetting

Vediamo un esempio

- `x <- list(a = 1:3, b = «a string», c = pi, d = list(-1, -5))`
- `str(x[1])` restituisce una sottolista contenente solo il primo elemento della lista.
- `str(x[[1]])` restituisce il primo elemento della lista, cioè un vettore con i numeri da 1 a 3
- `str(x$b)` funziona analogamente al caso precedente



# Strutture Dati : Liste - Subsetting

Un esempio molto famoso (Fonte: <https://r4ds.had.co.nz/vectors.html#lists>).

Immaginatevi un contenitore (i.e. una lista) contenente delle bustine di pepe (gli elementi nella lista).



# Strutture Dati : Liste - Subsetting

Se  $x$  rappresenta questo contenitore allora  $x[1]$  restituirà un contenitore con una singola bustina di pepe.



# Strutture Dati : Liste - Subsetting

$x[2]$  sarebbe esattamente analogo e così  $x[3]$  ecc.  $x[1:2]$  rappresenta un contenitore con due bustine di pepe. Invece  $x[[1]]$  è la singola bustina di pepe.



# Strutture Dati : Liste - Subsetting

Possiamo ancora scendere un altro grado di complessità e, immaginando la bustina di pepe essa stessa come una lista, allora `x[[1]][[1]]` rappresenta il pepe dentro la bustina



# Strutture Dati : Liste - Insertion

E' possibile aggiungere elementi ad una lista utilizzando il comando `<-`. Ad esempio

- `x <- list(auto = c(«fiat», «ford»), citta = c(«Milano», «Roma»))`
- `x$pianeti <- c(«mercurio», «venere», «terra»)`
- `x$citta[3] <- «Torino»`

# Esercizio

1. Pulire il proprio Environment riavviando R.
2. Creare una lista composta da:
  - a) Una lista contenente la stringa «Prima Lista» con nome «Titolo»
  - b) Una sequenza di numeri da 13 a 25 con passo 0.33
  - c) Un vettori di valori TRUE e FALSE
  - d) Una stringa di testo
  - e) Una matrice di 5 righe contenente i primi 10 numeri pari
3. Estrarre il primo elemento della lista
4. Estrarre la prima riga della matrice
5. Cosa restituisce il comando
  - $x \leftarrow list(1, 2)$
  - $x[1,2]$

# INTRODUZIONE AD R

## Strutture di Controllo

# Strutture di controllo

Consentono di gestire il flusso di esecuzione dei programmi, dipendentemente dalle condizioni di runtime. Strutture comuni sono

- **if, else:** verifica una condizione
- **for:** esegue un loop per un dato numero di iterazioni
- **while:** esegue loop fino a che la condizione è vera
- **repeat:** esegue un loop all'infinito
- **break:** interrompe l'esecuzione di un loop
- **next:** salta un'iterazione di un loop

Molte strutture di controllo non vengono usate in modalità interattiva, ma piuttosto quando si scrivono funzioni o espressioni estese.



# Strutture di controllo - IF

Un IF statement può essere utilizzato in due modi:

1. *if (<condition>) do something*
2. *if (<condition>) {  
do something  
} else {  
do something else  
}*

In altre parole la *else* non è obbligatoria.

# Strutture di controllo - IF

E' possibile anche testare più condizioni:

```
if (<condition1>) {  
    do something  
} else if (<condition1>) {  
    do something else  
} else if (<condition3>) {  
    do something else  
} else {  
    do something else  
}
```

E' possibile fare assegnazioni sia dentro l'*if* che assegnare ad un oggetto un valore in base ad una condizione con *if*.

# Strutture di controllo - IF

La condizione dentro l'IF deve sempre risultare TRUE o FALSE (pena un errore o un warning).

- *if (c(TRUE, FALSE)) {}*

*Warning message:*

*In if (c(TRUE, FALSE)) { :*

*the condition has length > 1 and only the first element will be used*

- *if (NA) {}*

*Error in if (NA): missing values when TRUE/FALSE are needed*

# Strutture di controllo - IF

*Per valutare più condizioni si usano && e ||*

- *if (x > 10 && sqrt(x) > 5) {}*

Dentro una *IF-condition* bisognerebbe sempre usare i comandi && ed || per effettuare dei test logici. && restituisce FALSE non appena una delle condizioni che vengono valutate è FALSE mentre || restituisce TRUE non appena una delle condizioni da valutare è TRUE.

# Strutture di controllo – IF

E' possibile utilizzare le funzioni *any()* e *all()* per testare una condizione su tutti gli elementi di una struttura dati.

- `x <- 1:10`
- `any(x > 5); all(x > 5)`

Come già illustrato, bisogna fare attenzione quando si verificano condizioni di uguaglianza per valori di tipo numeric.

# Esercizio

1. Pulire il proprio Environment riavviando R.
2. Si consideri la seguente struttura:

```
if (number < 10) {  
    if (number < 5) {  
        result <- «extra small»  
    } else {  
        result <- «small»  
    }  
} else if (number < 100) {  
    result <- «medium»  
} else {  
    result <- «large»  
}
```

Cosa succede se *number* è pari a 0? Se è pari a 10? Se è pari a 50? Se è pari a 1000? Se è pari a Inf? Se è pari a NA?

# Esercizio

Completare il codice riportato di seguito (dove sono presenti i puntini) seguendo le istruzioni:

Se li e fb sono maggiori o uguali a 15, set sms uguale al doppio della somma di li and fb. Se sia li che fb are strettamente minori10, set sms uguale alla metà della somma di li e fb. In tutti gli altri casi, set sms uguale a li + fb.

Stampa a video il risultato di sms.

```
li <- 15
```

```
fb <- 9
```

```
if (.....) {  
  sms <- 2 * (li + fb)  
} ..... {  
  sms <- 0.5 * (li + fb)  
} else {  
  sms <- .....  
}  
.....
```

# Esercizio

1. Pulire il proprio Environment riavviando R.
2. Si implementi una celebre *fizzbuzz* condition con la stessa struttura del codice presentato nel primo esercizio. Se `number` è divisibile per 3 allora il codice deve restituire la stringa «fizz»; se `number` è divisibile per 5 allora deve restituire la stringa «buzz»; se `number` è divisibile per 3 e per 5 allora deve restituire la stringa «fizzbuzz», altrimenti non deve restituire nulla.



# Strutture di controllo - FOR

Il for - loop del for accetta una variabile iterazione ed assegna ad essa valori successivi prelevati da una sequenza o da un vettore.

Comunemente utilizzato per iterare sugli elementi di un oggetto (lista, vettore, etc...)

```
for(i in 1:10) {  
    print(i)  
}
```

Questo loop prende la variabile "i" e ad ogni iterazione del loop assegna a tale variabile i valori 1, 2, 3, ..., 10, e poi esce dal loop.

# Strutture di controllo - FOR

I loop di seguito hanno lo stesso comportamento.

- `x <- c("a", "b", "c", "d")`
- `for(i in 1:4) {`
- `print(x[i])`
- `}`
- `for(i in seq_along(x)) {`
- `print(x[i])`
- `}`

**NB:** `seq_along(x)` è come `1 : length(x)`

- `for(letter in x) {`
- `print(letter)`
- `}`
- `for(i in 1:4) print(x[i])`

# Strutture di controllo - FOR

for loops possono essere innestati.

```
x <- matrix(1:6, 2, 3)

for(i in seq_len(nrow(x))) {

    for(j in seq_len(ncol(x))) {

        print(x[i, j])

    }

}
```

Nidificare oltre 2-3 livelli rende spesso il codice molto difficile da leggere ed analizzare.

# Strutture di controllo - WHILE

Inizia controllando una condizione. Se la condizione è vera, il loop viene eseguito (corpo del loop). Eseguito il corpo del loop, la condizione viene nuovamente controllata, e via di seguito.

```
count <- 0

while(count < 10) {

  print(count)

  count <- count + 1

}
```

ATTENZIONE: potenziale loop infinito. Utilizzare con cura!

# Strutture di controllo - REPEAT

Inizia un loop infinito; non utilizzato comunemente in applicazioni statistiche. L'unico modo di uscire è invocare break.

```
i <- 0
```

```
repeat {
```

```
    i <- i+1
```

```
    if(i == 2)
```

```
        print(i+i^10)
```

```
    else {
```

```
        print(i)
```

```
        if(i == 10)
```

```
            break
```

```
    }
```

```
}
```

# Strutture di controllo – NEXT

**Next** viene usato per saltare un'iterazione di un loop (va all'indice successivo)

```
for(i in 1:100) {  
  if(i <= 20) {  
    next      #salta le prime 20 iterazioni (ossia non stampa i primi 20 numeri)  
  }  
  print(i)  
  ## Do something here  
}
```

# Esercizio

1. Pulire il proprio Environment riavviando R.
2. Si utilizzi un ciclo per scrivere su R le lyrics della famosa canzone «99 Bottles of beer on the wall».

Dettagli: <http://www.99-bottles-of-beer.net/>

La canzone: <https://www.youtube.com/watch?v=FITjBet3dio>

My favourite version: <https://www.youtube.com/watch?v=Z7bmyjxJuVY>

E' sicuramente utile consultare l'help per le funzioni *print()* e *paste()*.

# INTRODUZIONE AD R

## Funzioni



# Funzioni - Definizione

Su R è possibile creare delle nuove funzioni per automatizzare diverse operazioni sempre uguali. Il vantaggio di questo approccio è che il nostro codice sarà più semplice da leggere e meno pronò ad errori.

Per creare una nuova funzione si utilizzano i seguenti comandi:

```
nome_nuova_funzione <- function(argomenti_funzione){  
    corpo della funzione  
}
```

# Funzioni - Definizione

Gli argomenti della funzione rappresentano tutti gli ingredienti di cui la funzione ha bisogno per eseguirsi. Il corpo della funzione sono tutte le istruzioni che essa deve svolgere.

Il valore restituito da una funzione è l'ultima espressione che viene valutata. Possiamo modificare questo comportamento utilizzando la funzione ***return()***.

# Funzioni - Esempio

Ad esempio: proviamo a scrivere una funzione che, dato un numero  $n$ , restituisce TRUE o FALSE se il numero è pari o dispari.

```
is_even <- function(n) {  
    if (n %% 2 == 0) TRUE  
    else FALSE  
}
```

# Funzioni - Esempio

Cosa succede se alla funzione passiamo  $n = 2.5$  ?

```
is_even <- function(n) {  
  if (n %% 1 != 0) stop("Non puoi calcolare pari/dispari per un valore non intero")  
  
  if (n %% 2 == 0) TRUE  
  
  else FALSE  
  
}
```

La funzione **stop** interrompe l'esecuzione della funzione restituendo un errore. E cosa succede se gli passiamo una stringa di testo?

# Funzioni - Style

Ci sono alcune cose da tenere a mente:

- Se possibile (*io non ci riesco quasi mai...*) il nome della funzione dovrebbe essere un verbo che descrive cosa quella funzione fa (es: `riscalda_valori`, `imputa_NA...`)
- Evitate nella maniera più assoluta di sovrascrivere le funzioni di R già esistenti chiamandole allo stesso modo.

# Funzioni – Esempio: La congettura di Collatz

Fonte: <http://blog.ephorie.de/learning-r-the-collatz-conjecture>

Approfondimento: <https://www.youtube.com/watch?v=5mFpVDpKX70>

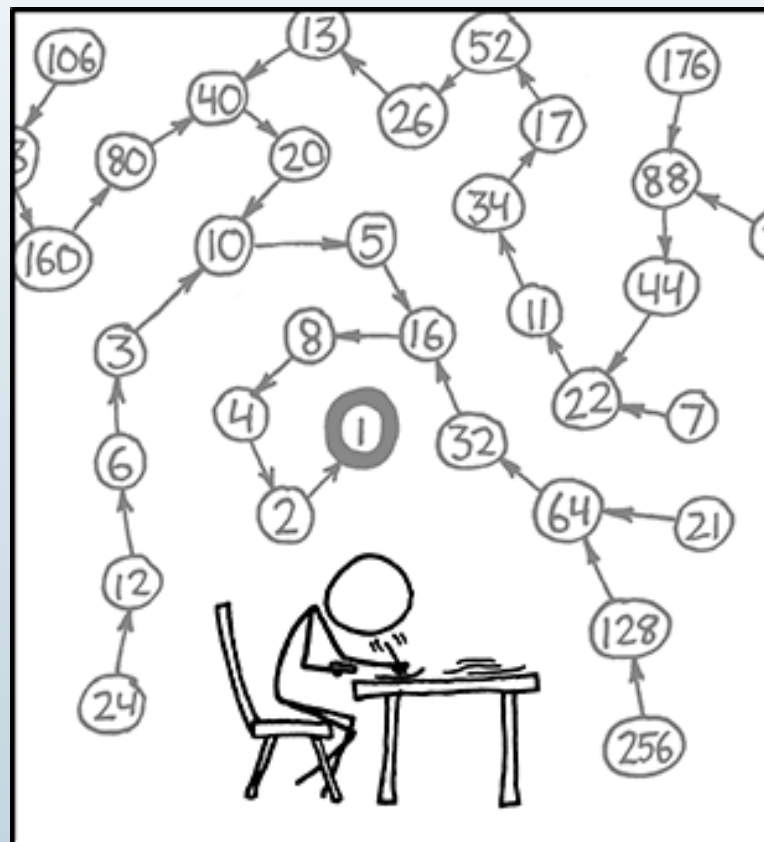
La congettura di Collatz afferma che, dato un qualsiasi numero naturale  $n$ , applicando il seguente algoritmo:

1. Se  $n$  è pari allora lo divido per 2
  2. Se  $n$  è dispari allora lo moltiplico per 3 e sommo 1
- allora la successione di elementi arriverà sempre ad 1.

Ad esempio, se  $n = 3$ , la successione risulta essere

$$3 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$$

# Funzioni – Esempio: La congettura di Collatz



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

# Funzioni – Esempio: La congettura di Collatz

Proviamo ad implementare tramite R una funzione che, dato un intero  $n$ , calcola la successione ottenuta applicando l'algoritmo della congettura di Collatz. Possiamo sfruttare la funzione *is\_even* appena definita.

```
collatz_algorithm <- function(n) {  
  if (n <= 0) stop("Il numero deve essere strettamente positivo")  
  if (is_even(n)) n / 2  
  else 3 * n + 1  
}
```



# Funzioni – Esempio: La congettura di Collatz

Naturalmente noi non vogliamo semplicemente il generico elemento della successione, vogliamo tutta la successione. Possiamo risolvere il problema utilizzando un ciclo **while**:

```
collatz_conjecture <- function(n) {  
  successione <- c(n) # creo un vettore vuoto dove salvare la successione  
  # Il primo elemento della successione è proprio n  
  while(n != 1) {  
    n <- collatz_algorithm(n)  
    successione <- c(successione, n)  
  }  
  successione  
}
```

# Esercizio

1. Pulire il proprio Environment riavviando R.
2. Si implementi una funzione che, dato un vettore  $x$  di valori numerici tutti non-missing, restituisca un vettore contenente la media aritmetica e la varianza di  $x$ .
3. Si consideri il caso in cui il vettore  $x$  può contenere valori mancanti e si risolva il problema.
4. Si implementi una funzione che, dato un vettore  $x$  di valori numerici tutti maggiori di 0, ne calcoli la media geometrica

SUGGERIMENTO: [https://it.wikipedia.org/wiki/Media\\_\(statistica\)#Media\\_geometrica](https://it.wikipedia.org/wiki/Media_(statistica)#Media_geometrica)

5. Si implementi una funzione con due parametri (chiamati *peso* ed *altezza*) che calcola il BMI di una persona.
6. Si ripeta l'esercizio sulle 99 Bottles of Beer on the wall creando una funzione che ripeta la canzone per un numero arbitrario di bottiglie

# INTRODUZIONE AD R

## Base Plot

# R – Base plot

La prima funzione che si utilizza per creare grafici su R è sicuramente **plot()**. Leggiamo l'help per capire come funziona e proviamo con il primo e più banale esempio:

```
plot(x = 1:10, y = 1:10)
```

## R – Collatz plot

Proviamo ora un esempio un po' più corposo. Innanzitutto utilizziamo la funzione `collatz_conjecture` appena definita partendo da  $n = 27$  e salvando il risultato in un vettore chiamato `my_result`. Rappresentiamo poi questo vettore specificando `type = "l"`.

**Esercizio:** Rileggere l'help associato alla funzione `plot` in modo da abbellire ulteriormente questo grafico.

## R – Collatz plot

Proviamo ora un esempio un po' più corposo. Innanzitutto utilizziamo la funzione `collatz_conjecture` appena definita partendo da  $n = 27$  e salvando il risultato in un vettore chiamato `my_result`. Rappresentiamo poi questo vettore specificando `type = "l"`.

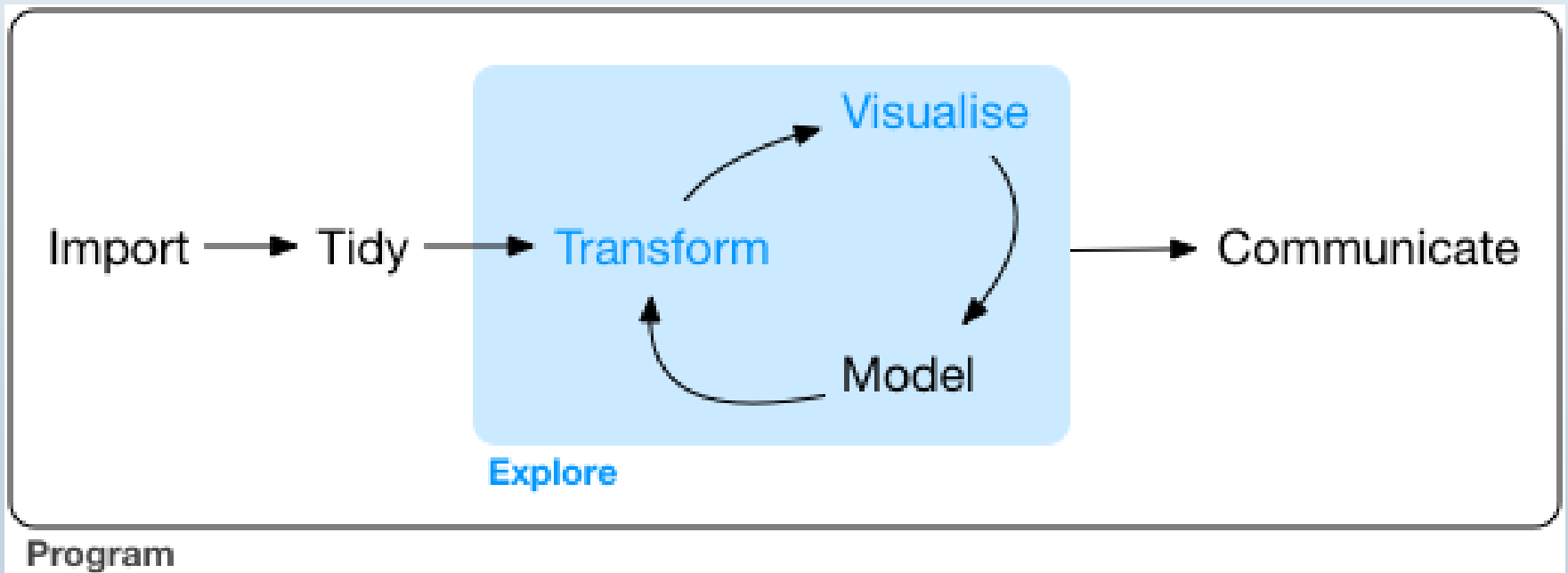
**Esercizio:** Rileggere l'help associato alla funzione `plot` in modo da migliorare questo grafico.

# INTRODUZIONE AD R

Welcome to the Tidyverse

# Tidyverse - Definizione

*Tidyverse* rappresenta un insieme di pacchetti R creati con uno stile omogeneo ed utilizzati per fare Data Analysis.





# Tidyverse - Definizione

Noi arriveremo  
qua



**Using Base R for your analysis  
and copy pasting your results  
into tables in Word**



**learning how to use dplyr  
visualize data with ggplot2  
and report your analysis in  
rmarkdown documents**



**wielding the entire might of the tidyverse**



# Tidyverse – Riferimenti

- La ‘Bibbia’ in questo caso è disponibile a : <https://r4ds.had.co.nz/>

Esistono, naturalmente, tantissimi altri tutorial:

- <https://www.tidyverse.org/>
- <https://moderndive.com/>
- <https://github.com/AmeliaMN/data-science-in-tidyverse>
- [https://www.datacamp.com/courses/introduction-to-the-tidyverse?tap\\_a=5644-dce66f&tap\\_s=213362-c9f98c](https://www.datacamp.com/courses/introduction-to-the-tidyverse?tap_a=5644-dce66f&tap_s=213362-c9f98c)

# Tidyverse – `install.packages`

Proviamo ora ad installare il pacchetto *tidyverse*:

```
install.packages("tidyverse")
```

*Questo comando installa anche numerosi altri pacchetti R tra cui dplyr e ggplot2 (su cui ci concentreremo).*

# INTRODUZIONE AD R

**ggplot2**

# ggplot2 - Filosofia

The simple graph has brought more information to the data analyst's than any other device.

(John Tukey)

Potete anche trovare una spiegazione più dettagliata sul funzionamento di questo pacchetto consultando il libro *ggplot2: Elegant Graphics for Data Analysis*.

*ggplot2 vs base plot: <https://twitter.com/i/status/991400769235468288>*

# ggplot2 - Introduzione

Iniziamo caricando il pacchetto

```
library(ggplot2)
```

Nei prossimi esempi utilizzeremo il dataset *mpg*. Carichiamolo in memoria e leggiamo la pagina di help associata.

# ggplot2 – Il primo grafico

Creiamo ora il nostro primo grafico:

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy))
```

Che relazione esiste quindi tra hwy (kilometri percorsi con un litro di carburante) e displ (cilindrata del veicolo)?

# ggplot2

Quindi come funziona *ggplot2*? La prima funzione che abbiamo utilizzato, *ggplot*, crea un grafico vuoto. A questo grafico vuoto vanno ad aggiungersi mano a mano diversi layer (definiti dall'utente) che vanno poi a modificare il grafico vuoto. Ad esempio, nel grafico precedente la funzione *geom\_point* aggiunge un layer al grafico vuoto rappresentando lo scatterplot di *hwy* vs *displ*.

Per definire come un layer viene creato (e quindi come le variabili presenti nel dataset vengono mappate sul grafico) devo utilizzare la funzione *aes*, dentro cui specifico quali valori mappare sull'asse x e sull'asse y.



# ggplot2

In generale, quindi, un grafico in *ggplot2* viene creato utilizzando il seguente comando:

```
ggplot(data = DATA) +
```

```
  GEOM_FUNCTION(mapping = aes(MAPPINGS))
```

dove *GEOM\_FUNCTION* è una funzione che crea un layer e *MAPPINGS* sono i parametri che passiamo alla funzione. Man mano che l'analisi diventa più complicate andremo ad aggiungere sempre più layers.

# Esercizio

1. Pulire il proprio Environment riavviando R.
2. Caricare il pacchetto `ggplot2` e il dataset *faithful* disponibile dalla libreria *datasets*. Leggere l'help associato al dataset.
3. Stampare il dataset a schermo (basta scrivere *faithful* sulla console di Rstudio e far eseguire il comando). NB: Viene stampato tutto il dataset, i nomi delle variabili sono all'inizio.
4. Creare uno scatterplot in cui si studia la relazione tra *eruptions* e *waiting*. Cosa ne riuscite a dedurre?

# ggplot2 – Aesthetic Mappings

Data una particolare GEOM\_FUNCTION (come *geom\_point*), esistono diversi tipi di aesthetics che è possibile modificare per personalizzare molti aspetti di ogni layer. L'elenco complete è disponibile leggendo l'help associate ad ogni funzione.

Ad esempio, guardiamo l'help della funzione *geom\_point*. Vediamo che i possibili aesthetics non sono solamente *x* ed *y* (cioè le coordinate del punto) ma anche *colour*, *size* e *shape*.

# ggplot2 – Aesthetic Mappings

Creiamo ora il nostro secondo grafico:

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy, color = class))
```

Come cambia la relazione tra *hwy* e *displ* al variare della tipologia di veicolo?

## ggplot2 – Aesthetic Mappings

Possiamo anche associare la variabile `class` a diverse caratteristiche di un punto, come la sua grandezza:

```
ggplot(data = mpg) +
```

```
  geom_point(mapping = aes(x = displ, y = hwy, size = class))
```

o la sua forma:

```
ggplot(data = mpg) +
```

```
  geom_point(mapping = aes(x = displ, y = hwy, shape = class))
```

# ggplot2 – Aesthetic Mappings

E' possibile anche modificare le caratteristiche di tutti i punti in uno scatterplot (o di una qualsiasi GEOM\_FUNCTION) senza doverli necessariamente mappare ad una variabile:

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy), col = "blue")
```

# Esercizio

1. Pulire il proprio Environment riavviando R.
2. Caricare il pacchetto `ggplot2` e il dataset *iris* disponibile dalla libreria *datasets*. Leggere l'help associato al dataset.
3. Creare uno scatterplot in cui si studia la relazione tra *Sepal.Length* e *Sepal.Width* al variare di *Species*. Si modifichi il colore dei punti, la forma e la grandezza. Cosa riuscite a dedurne?
4. Cosa restituisce il seguente codice?

```
ggplot(data = mpg) +
```

```
geom_point(mapping = aes(x = displ, y = hwy, col = "blue"))
```

# ggplot2 – GEOM\_FUNCTIONS

Esistono molte altre GEOM\_FUNCTIONS oltre a *geom\_point* che servono per creare diversi tipi di grafici (es: istogrammi, grafici a barre, grafici con linee di errore...). Potete leggerne un elenco al seguente link:

<https://ggplot2.tidyverse.org/reference/index.html>



## ggplot2 – GEOM\_FUNCTIONS

Tutte le possibili GEOM\_FUNCTIONS necessitano di uno o più aesthetics, ma non tutte possono funzionare con gli stessi aesthetics (ad esempio non ha senso parlare di shape in un grafico a barre). Ad esempio:

```
ggplot(data = mpg) +  
  geom_smooth(mapping = aes(x = displ, y = hwy))
```

crea una sorta di linea di tendenza utilizzando un diverso tipo di geometria rispetto a *geom\_point*. Leggiamo prima l'help della funzione.

## ggplot2 – GEOM\_FUNCTIONS

Proviamo a modificare linetype:

```
ggplot(data = mpg) +
```

```
  geom_smooth(mapping = aes(x = displ, y = hwy, linetype = drv))
```

ottenendo quindi una linea di tendenza per i diversi valori di *drv*.

*Modifichiamo anche il colore dei punti!*

```
ggplot(data = mpg) +
```

```
  geom_smooth(mapping = aes(x = displ, y = hwy, linetype = drv,  
                             colour = drv))
```

## ggplot2 – GEOM\_FUNCTIONS

Una delle cose più belle di ggplot2 è la possibilità di rappresentare molto facilmente due o più geometrie sullo stesso grafico:

```
ggplot(data = mpg) +  
  geom_smooth(mapping = aes(x = displ, y = hwy)) +  
  geom_point(mapping = aes(x = displ, y = hwy))
```

## ggplot2 – GEOM\_FUNCTIONS

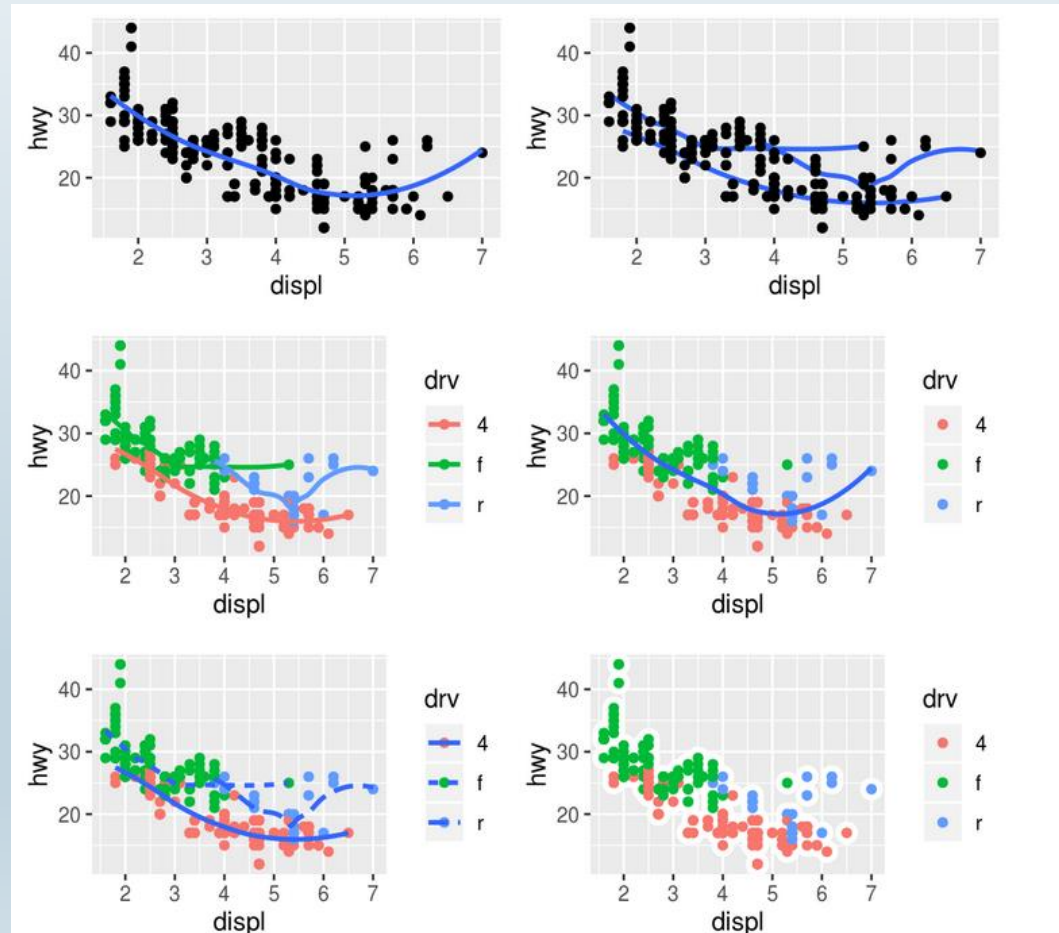
In questo caso, però, vediamo che il codice risulta inutilmente ripetuto.

Possiamo ovviare a questo problema specificando gli aesthetics comuni dentro la funzione *ggplot* e gli aesthetics unici dentro la GEOM\_FUNCTION

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +  
  geom_point(mapping = aes(col = class)) +  
  geom_smooth()
```

# Esercizio

1. Pulire il proprio Environment riavviando R.
2. Caricare il pacchetto ggplot2. Si provi a replicare i seguenti grafici:



# Ringraziamenti

Si ringrazia la Dott.ssa Chiesa per aver fornito la maggior parte del materiale presentato in queste slide. Altre fonti non citate in precedenza:

- <https://www.facebook.com/Rmemes0/>

**Contatti:** a.gilardi5@campus.unimib.it