



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;
- insieme di operatori per calcolo su array e matrici;
- 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 preconditione 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 preconditione 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

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

“The R Inferno”: https://www.burns-stat.com/pages/Tutor/R_inferno.pdf

Altre risorse

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

- 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/> (tra qualche 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? Usandolo!



via boredpanda, bbc, reddit

Your taste develops faster than your ability.

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

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

Help...



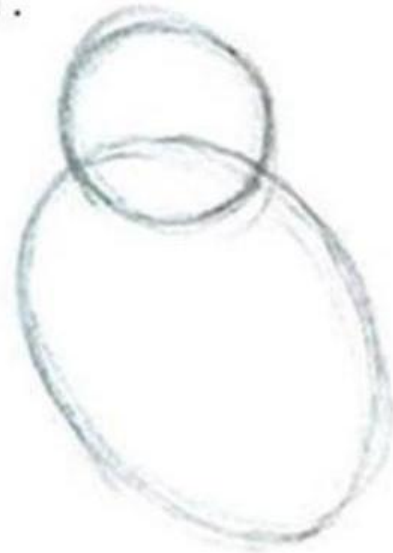
Andrew Hall @andrewhall_NU · 14 Std.
Reading R help documentation.



 Tweet übersetzen

How to draw an owl

1.



2.



1. Draw some circles

2. Draw the rest of the fucking owl

Help

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")
```

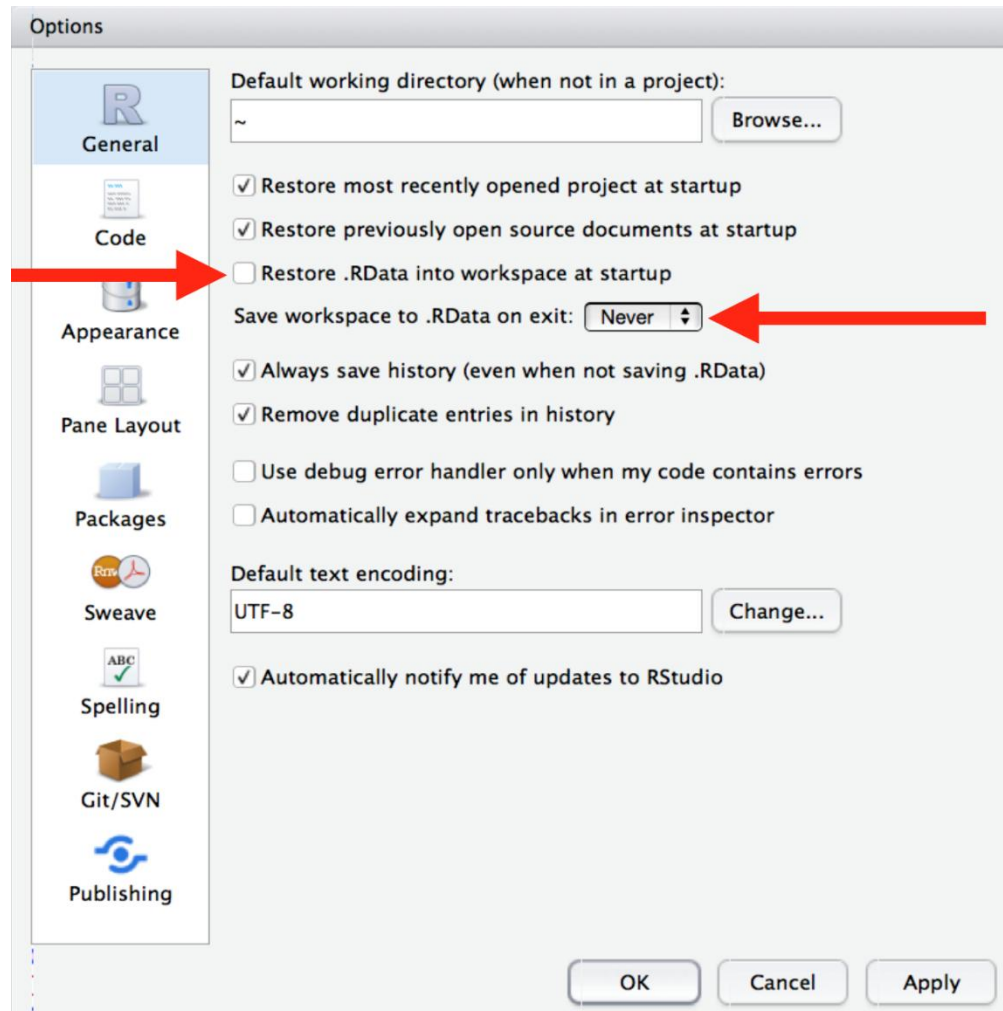
I* will come into your office and
SET YOUR COMPUTER ON FIRE 🔥.

* 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

In pacchetti ancora in fase di sviluppo o in beta-testing vengono solitamente salvati su github.

- *install_github("nome package")* # installare un pacchetto da github. Richiede il pacchetto *devtools/remotes*.
- *library(nome package)* #caricare pacchetto. Restituisce un errore se non è presente in *.libpath()*

Packages - Esercizio

1. Proviamo ad installare dal CRAN il pacchetto *dplyr*.
2. Proviamo ad installare da github il pacchetto *trump* disponibile al link:
<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
- 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 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 ...

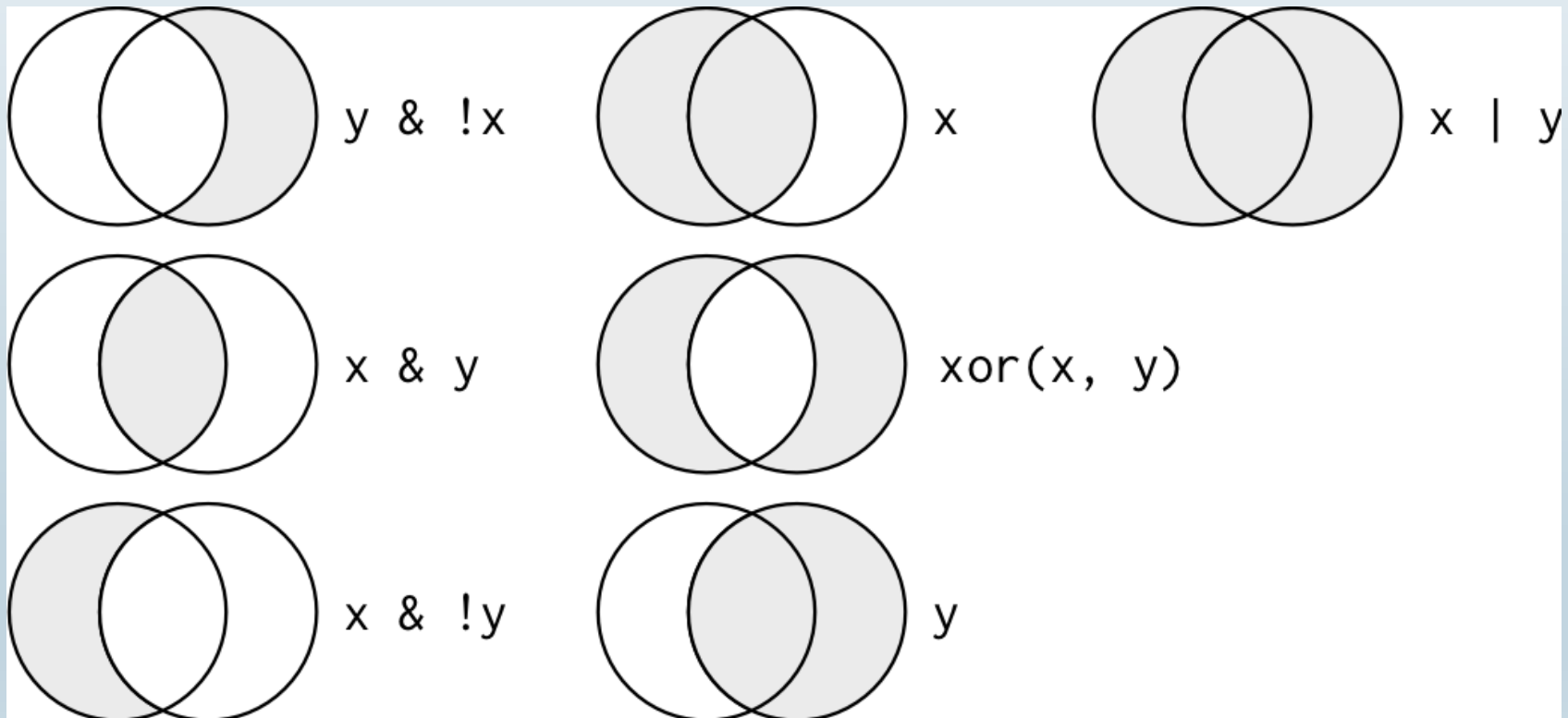
- Operazioni semplici: $+$, $-$, $*$, $/$, $^$, $\%/%$ (divisione intera tra due numeri), $\%\%$ (resto della divisione intera)
- Operatori relazionali (ritornano TRUE o FALSE): $>$, $<$, $>=$, $<=$, $==$, $!=$
- Operatori logici: $!$, $\&$ (AND), $|$ (OR)
- Il logaritmo di un numero viene calcolato 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*.

Prima di cominciare ...

Operatori logici

l'insieme x è il cerchio a sinistra

l'insieme y è il cerchio a destra



Input

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

➤ `3 * 2; exp(3); log(10)`

➤ `x <- 1; x`

➤ `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 180.
- Calcolare il BMI per quella persona.

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; $1/\text{Inf}$ è uguale a 0. Esiste anche $-\text{Inf}$, es: $\log(0)$.
- **NaN**, rappresenta un valore indefinito (**N**ot **A** **N**umber) (rappresentato come double); $0/0$
- **NA**, valore mancante

$3 * \text{NA}$ # restituisce NA

$\text{NA} == \text{NA}$ # restituisce NA

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

Warning – Finite Precision Arithmetic

Un errore molto comune è il seguente:

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

Warning – Finite Precision Arithmetic

Un errore molto comune è il seguente:

- `a <- exp(10^500)`

Cosa restituirà la precedente espressione?

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=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

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

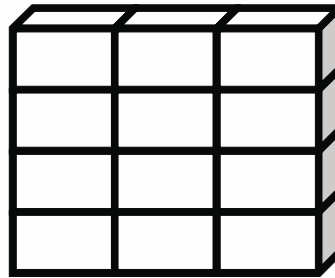
Strutture dati

Strutture dati

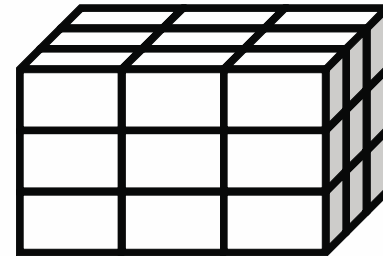
(a) Vector



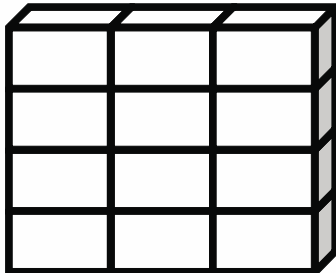
(b) Matrix



(c) Array

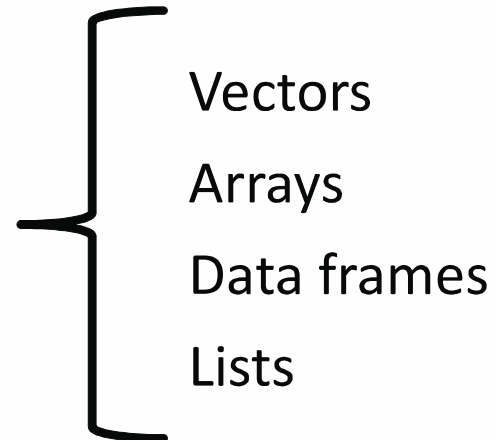


(d) Data frame



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=10* posso utilizzare la funzione *numeric()* a cui devo 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

La maggior parte delle 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

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
```

Questo fenomeno viene comunemente chiamato ***recycling***. E' importante conoscerlo perchè può generare *warnings* ed errori.

Strutture dati: VETTORI

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.

L'ordine di conversione è:

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

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

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/>