

Docenti: **Matteo Re** (INFORMATICA)  
**Alessandro di Domizio** (STATISTICA)

UNIVERSITÀ DEGLI  
STUDI DI MILANO



C.d.I. **Biotechnologia**

A.A. 2016-2017 semestre II

# Informatica e Statistica

**1**

**INTRODUZIONE R**

Modulo: **INFORMATICA**

# Caratteristiche di R

- Linguaggio ad alto livello *interpretato*
- Dotato di insiemi di operatori ad alto livello per *calcoli su array e matrici*
- Supporta paradigmi di programmazione *imperativa, object-oriented e funzionale*.
- Fornisce un ambiente per la *elaborazione interattiva* dei dati
- *Ambiente integrato di risorse software* per la gestione ed elaborazione di dati e la visualizzazione di grafici
- Dispone di *interfacce* verso programmi e moduli sw scritti con altri linguaggi
- Ambiente di sviluppo e package open source disponibili liberamente in internet.

# R come linguaggio per la bioinformatica

- Linguaggio ad alto livello orientato alla analisi dei dati
- Permette di strutturare dati complessi ed eterogenei
- Dispone di un ambiente di lavoro e di sviluppo per lavorare interattivamente con i dati
- Dispone di package (librerie) specifiche per la bioinformatica
- R è il linguaggio utilizzato dal progetto internazionale open source *Bioconductor* per la gestione ed elaborazione di dati genomici e proteomici
- E' uno dei linguaggi maggiormente utilizzati dalla comunità internazionale dei bioinformatici

# Breve storia di R

- Deriva da *S*, un linguaggio ed un sistema sviluppati da *John Chambers* e collaboratori negli anni '80 presso i Laboratori Bell.
- *S* è valso l'*ACM Software Systems Award* al suo principale progettista J. Chamber nel 1999.
- *R* è un progetto *Open Source* conforme per la maggior parte ad *S*:
  - Sviluppato inizialmente da *Ross Ihaka and Robert Gentleman* all'Università di Auckland (Nuova Zelanda)
  - Attualmente sviluppato da una comunità internazionale di ricercatori e sviluppatori in ambito sia accademico sia industriale
  - Opera attraverso il web: [www.r-project.org](http://www.r-project.org)
  - Archivi software e documentazione: [cran.r-project.org/](http://cran.r-project.org/)

# Da dove “scaricare” R

**CRAN** - the Comprehensive R Archive Network:  
<http://cran.r-project.org/> (ci sono anche mirror locali)

Sono disponibili distribuzioni binarie per :

- *Windows*
- *Macintosh*
- *Linux*

L’installazione sul proprio PC è (in genere) semplice.

# Documentazione e bibliografia su R

- Materiale didattico scaricabile dalle pagine web del corso: “*Metodi Bioinformatici*”: <http://homes.dsi.unimi.it/~valenti/MB1011.html>
- Un corso introduttivo scaricabile dal web:  
W. Venables and D.M. Smith, *An Introduction to R*: <http://cran.r-project.org/doc/manuals/R-intro.pdf>, 2010
- Libri sulla programmazione “avanzata” in R:
  - 1) Robert Gentleman *R Programming for Bioinformatics*, CRC/Computer Science & Data Analysis Volume 12 , Chapman & Hall, 2008
  - 2) J. Chambers *Software for Data Analysis: Programming with R*, Springer, 2008
- Un libro specifico sull’ utilizzo di R per la bioinformatica e Bioconductor:  
Gentleman, R.; Carey, V.; Huber, W.; Irizarry, R.; Dudoit, S.  
*Bioinformatics and Computational Biology Solutions using R and Bioconductor*, Springer, 2005



C.d.I. Biotecnologia

A.A. 2016-2017 semestre II

# Informatica e Statistica

**2**

**Strutture dati in R**

Modulo: **INFORMATICA**

# Strutture dati

Nei linguaggi di programmazione ad alto livello non si ha accesso diretto alla memoria fisica, ma ad una sua astrazione (*struttura dati*).

Le principali strutture dati fornite da R sono:

- Vettori
- Array e matrici
- Fattori
- Liste
- Data frame



# Vettori

I vettori rappresentano sequenze ordinate di elementi omogenei.  
Un vettore  $v$  è rappresentabile tramite una struttura unidimensionale:

$v$	4	8	2	1	3	1	5	0
-----	---	---	---	---	---	---	---	---

*Esempio:*

- > `c(1, 4, 5)` # crea un vettore di interi
- > `c("A", "B", "C")` # crea un vettore di caratteri
- > `c("gatto", "topo", "12")` # crea un vettore di stringhe

La *funzione* `c(arg1, arg2, arg3, arg4)` concatena i suoi argomenti

# Variabili ed assegnamenti

Un vettore può essere *assegnato* ad una *variabile*.

## Esempio 1

```
> X <- c(1,4,5) # il vettore <1 4 5> è assegnato alla  
  variabile X
```

```
> X
```

```
[1] 1 4 5
```

La variabile X rappresenta ora il vettore <1 4 5>: si può pensare come un “contenitore” del vettore stesso

## Es. 2

```
> X <- c(4,7) # il vettore <4 7> è assegnato alla  
  variabile X
```

```
> X
```

```
[1] 4 7
```

Un nuovo assegnamento cancella il contenuto precedente

## Es.3

```
> Y <- 100 # vettore formato da 1 solo elemento
```

```
> Y
```

```
[1] 100
```

# Tipi elementari di vettori

I vettori sono sequenze ordinate i cui elementi possono essere di 3 tipi base:

- **Numerici**: numeri interi o in virgola mobile (floating point)
- **Caratteri**: singoli caratteri o stringhe (sequenze) di caratteri
- **Logici**: TRUE o FALSE

# Operazioni con vettori aritmetici

Le operazioni usuali dell'aritmetica vengono eseguite sui vettori elemento per elemento:

## Addizione e sottrazione

```
> x <- c(1, 2, 3)
```

```
> y <- c(4, 5, 6)
```

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

```
> z
```

```
[1] 5 7 9
```

```
> d <- y - x
```

```
> d
```

```
[1] 3 3 3
```

## Moltiplicazione e divisione

```
> x <- c(1, 2, 3)
```

```
> y <- c(4, 5, 6)
```

```
> p <- x * y
```

```
> p
```

```
[1] 4 10 18
```

```
> q <- y / x
```

```
> q
```

```
[1] 4.0 2.5 2.0
```

# Esempi di funzioni applicate a vettori numerici

```
> x <- c(1,2,3)
> mean(x)
[1] 2
> var(x)
[1] 1
> max(x)
[1] 3
> min(x)
[1] 1
> range(x)
[1] 1 3
> sum(x)
[1] 6
> prod(x)
[1] 6
```

```
> y <- rnorm(10) # generazione ~N(0,1)
> y
[1] -1.5171592  0.5538263 -0.9505327
-0.6218845  0.5113505  0.7547935
[7] -1.5403415  2.3607231  1.3177109
-1.3993465
> sort(y) # ordinamento
[1] -1.5403415 -1.5171592 -1.3993465
-0.9505327 -0.6218845  0.5113505
[7]  0.5538263  0.7547935  1.3177109
2.3607231
> order(y) # indici corrispondenti
# agli elementi ordinati
[1]  7  1 10  3  4  5  2  6  9  8
```

# Generazione di sequenze regolari

R dispone di diversi comandi per generare automaticamente sequenze di numeri:

```
> c(1:10)
```

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

```
> c(5:1)
```

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

```
> seq(1,10)
```

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

```
> seq(from=1, to=4, by=0.5)
```

```
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0
```

La funzione `seq()` può avere 5 argomenti (si veda l'help). Un'altra funzione per generare repliche di vettori è `rep()`:

```
> rep(c(1,2), times=4)
```

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

# Vettori ed operatori logici

Sono vettori i cui elementi possono assumere valore **TRUE** o **FALSE**.

## Operatori logici:

`<`, `<=`, `>`, `>=`, `==` (uguaglianza), `!=` (disuguaglianza)

Es:

```
> x<- 3:8
```

```
> x > 5
```

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

```
> x <= 5
```

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

```
> x == 5
```

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

```
> x != 5
```

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

```
> x != c(5,6) # vale la "regola del riciclo"!
```

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

# Dati mancanti

- In molte situazioni reali i componenti di un vettore possono essere “non noti” o comunque non disponibili.
- In questi casi R riserva il valore speciale **NA** (“Not Available”).
- In generale qualunque operazione che coinvolga valori NA ha come risultato NA.

**Es:**

```
> x <- c(1:4, NA)
```

```
> x + 2
```

```
[1] 3 4 5 6 NA
```

Si noti che NA non è un valore ma un “marcatore” di una quantità non disponibile:

```
> x == NA
```

```
[1] NA NA NA NA NA
```

Per individuare quali elementi siano effettivamente NA in un vettore si deve usare la funzione **is.na()**:

```
> is.na(x)
```

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



# Vettori: selezione e accesso a sottoinsiemi di elementi

Esistono diverse modalità di accesso a singoli elementi o a sottoinsiemi di elementi di un vettore. In generale la selezione e l'accesso avviene tramite l' **operatore []** (parentesi quadre) : sottoinsiemi di elementi di un vettore sono selezionati collegando al nome del vettore un vettore di indici in parentesi quadre. Esistono *4 diverse modalità di selezione/accesso*:

- Vettori di **indici interi positivi**
- Vettore di **indici interi negativi**
- Vettore di **indici logici**
- Vettori di **indici a caratteri**

# Selezione ed accesso tramite vettori di indici interi positivi

Gli elementi di un vettore  $x$  sono selezionati tramite un vettore  $y$  di indici positivi racchiuso fra parentesi quadre :  $x[y]$   
i corrispondenti elementi sono selezionati e concatenati.

```
> x <- 5:10
> x[1] # selezione di un singolo elemento
[1] 5
> x[5]
[1] 9
> length(x) # lunghezza del vettore
[1] 6
> x[7] # accesso ad un elemento "fuori range"
[1] NA
> x[2:4]
[1] 6 7 8
> x[c(1,3,5)]
[1] 5 7 9
```

# Selezione ed accesso tramite vettori di indici interi negativi

Sono selezionati gli elementi di un vettore  $x$  che devono **essere esclusi** tramite un vettore  $y$  di indici negativi racchiuso fra parentesi quadre :  $x[y]$   
Es:

```
> y <- rep(c("G", "A", "T", "T"), times=3)
> y
[1] "G" "A" "T" "T" "G" "A" "T" "T" "G" "A" "T" "T"
> z <- y[-(1:5)] # selezionati tutti gli elementi di y
# eccetto primi 5
> z
[1] "A" "T" "T" "G" "A" "T" "T"
> z <- z[-length(z)] # cancellazione dell' ultimo
# elemento di z
> z
[1] "A" "T" "T" "G" "A" "T"
```

# Selezione ed accesso tramite vettori indice logici

Il vettore indice deve essere della stessa lunghezza del vettore i cui elementi devono essere selezionati. Sono selezionati gli elementi corrispondenti a TRUE nel vettore degli indici ed omessi quelli corrispondenti a FALSE.

Es:

```
> x <- c(1:5, NA, NA)
> x
[1] 1 2 3 4 5 NA NA
> i <- c(rep(TRUE, times=3), rep(FALSE, times=4))
> i # i è il vettore indice logico
[1] TRUE TRUE TRUE FALSE FALSE FALSE FALSE
> x[i] # selez. elementi tramite vett. indice logico
[1] 1 2 3
> x[!is.na(x)] # selezione elementi che non sono NA
[1] 1 2 3 4 5
> x[!is.na(x) & x > 2]
[1] 3 4 5
```

# Selezione ed accesso tramite vettori di indici a caratteri

E' applicabile quando un vettore possiede un attributo **names** per identificare le sue componenti. In questo caso un sottovettore del vettore **names** può essere utilizzato per selezionare le componenti

Es:

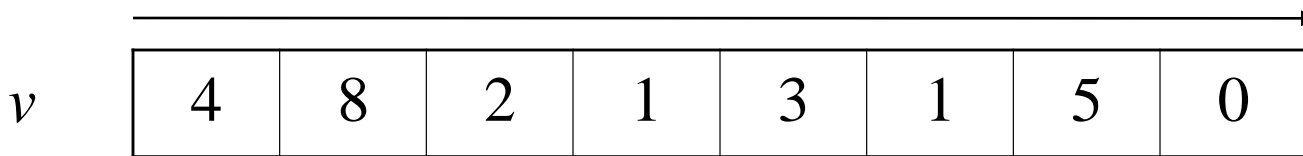
```
> campione <- c(45,210,5.12,73.22,0.82)
> names(campione) <- c("Eta", "Conc.ciclosporina",
"Dose.giornaliera", "Conc.urea", "Conc.bilirubina")
> campione
      Eta Conc.ciclosporina Dose.giornaliera
45.00          210.00          5.12
Conc.urea  Conc.bilirubina
 73.22          0.82
> Eta.Dose <- campione[c("Eta","Dose.giornaliera")]
> Eta.Dose
      Eta Dose.giornaliera
45.00          5.12
```

# Esercizi

- Generare un vettore contenente i primi 100 interi positivi: Calcolare la media, la varianza e la deviazione standard dei suoi elementi (funzioni `mean`, `var`, `sd`)
- Ripetere il precedente esercizio con un vettore di 100 numeri random estratti da una distribuzione normale standard (si veda la funzione `rnorm`)
- (a) Costruire una sequenza `s` costituita da 3 ripetizioni in sequenza della stringa “CGCT”.  
(b) Estrarre dalla sequenza ottenuta una sottosequenza `sub` in cui compaiano tutti gli elementi di `s` eccetto la lettera C.  
(c) Aggiungere in coda alla sequenza ottenuta 3 valori NA.  
(d) Riottenere la sequenza `sub` in (b) tramite la funzione `is.na()`
- Cosa accade se si prova a costruire un vettore eterogeneo di numeri e caratteri ? E di numeri e di valori logici ?
- Generare un vettore `vet` di 100 elementi casuali estratti secondo la distribuzione uniforme in  $[0,1]$  (vedi `runif`). Estrarre da `vet` un vettore `subvet` i cui elementi abbiano valore  $v > 0.2$  e valore  $v < 0.6$ . Estrarre da `subvet` gli elementi di indice pari ed assegnarli al vettore `w`. Trasformare `w` in modo che i suoi elementi siano normalizzati rispetto al loro valore medio ed alla deviazione standard.

# Array e matrici come generalizzazioni multidimensionali di vettori (1)

- I vettori sono sequenze ordinate di elementi omogenei. Un vettore  $v$  è rappresentabile tramite una *struttura unidimensionale*:

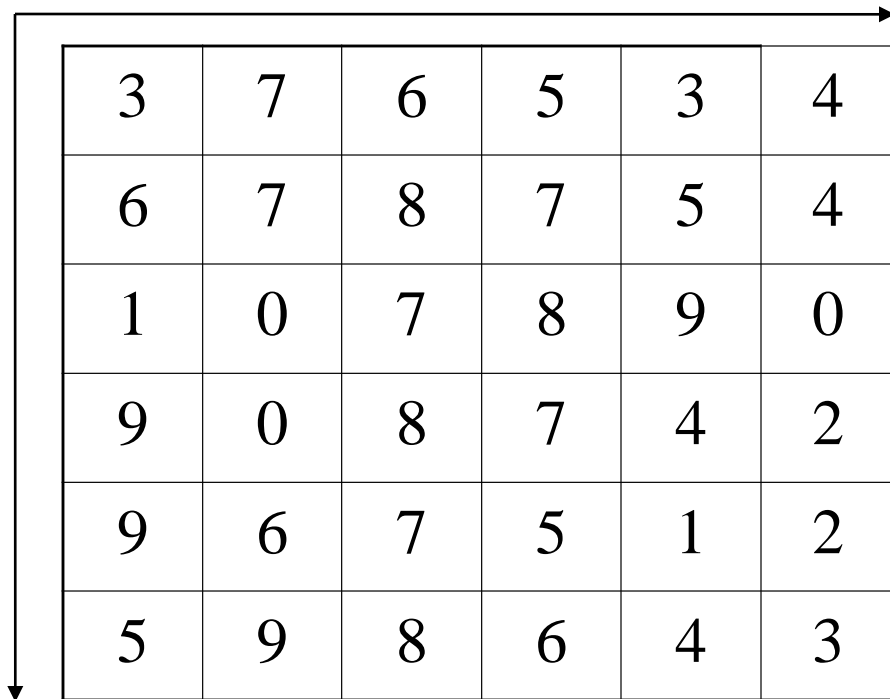


- Essendo strutture unidimensionali è possibile accedere o modificare un elemento di un vettore utilizzando un *unico indice*:

```
> v[2]
8
> v[1] <- 7
> v[1]
7
```

# Array e matrici come generalizzazioni multidimensionali di vettori (2)

- In R è possibile rappresentare *estensioni bidimensionali* di vettori (**matrici**)



3	7	6	5	3	4
6	7	8	7	5	4
1	0	7	8	9	0
9	0	8	7	4	2
9	6	7	5	1	2
5	9	8	6	4	3

- Essendo strutture bidimensionali, è necessaria una *coppia di indici* per accedere o modificare un elemento di una matrice :

```
> m[1,3] # seleziona el. I  
# riga e III colonna  
> 6  
> m[2,4] <- 0
```



# Array e matrici come generalizzazioni multidimensionali di vettori (3)

- In generale in R è possibile rappresentare *estensioni multidimensionali* di vettori (**array**):

Es: array  
tridimensionale



Per accedere ad un elemento sono necessari 3 indici.

Es:

```
> a[1, 3, 4]
```

- In R si possono costruire array di dimensione arbitraria (limitatamente alle disponibilità di memoria)
- Le matrici sono a tutti gli effetti array bidimensionali
- Sugli array sono applicabili le medesime operazioni di accesso e modifica (estese a più dimensioni) utilizzabili per i vettori

# Matrici

- In  $\mathbb{R}$  le matrici sono array bidimensionali:

colonne

righe

3	7	6	5	3	4
6	7	8	7	5	4
1	0	7	8	9	0
9	0	8	7	4	2
9	6	7	5	1	2
5	9	8	6	4	3

- Gli elementi di una matrice sono selezionati tramite una coppia di indici racchiusi fra parentesi quadre:
  - `m[1,2]` (m variabile cui è associata la matrice) identifica l'elemento della riga 1 e colonna 2
  - `m[4,6]` identifica l'elemento della riga 4 e colonna 6

# Costruzione di matrici (1)

Le matrici possono essere costruite tramite la funzione **matrix** a partire da un vettore esistente

```
> x <- 1:24
> m <- matrix (x, nrow=4) # genera una matrice con 4
# righe utilizzando gli elementi del vettore x
> m # si noti l' assegnamento dei valori "per colonne"
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    2    6   10   14   18   22
[3,]    3    7   11   15   19   23
[4,]    4    8   12   16   20   24
> length(m)
[1] 24
> mode(m)
[1] "numeric"
> dim(m)
[1] 4 6
```

L' attributo **dim** mostra che la matrice *m* è un array bidimensionale 4 X 6

# Costruzione di matrici (2)

La funzione **matrix** può avere anche altri argomenti:

```
> x <- 1:12
> m <- matrix (x, ncol=4) # si può specificare il n. delle colonne
> m
      [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> m <- matrix (x, ncol=4, byrow=TRUE) # inserimento el. per righe
> m
      [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
> m <- matrix (x, ncol=5) # anche per le matrici si applica
                        # la "regola del riciclo"
Warning message:
data length [12] is not a sub-multiple or multiple of the number of
columns [5] in matrix
> m
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    4    7   10    1
[2,]    2    5    8   11    2
[3,]    3    6    9   12    3
```

# Le funzioni cbind e rbind

Le matrici possono essere costruite anche tramite le funzioni **cbind** ed **rbind**.

**cbind** forma matrici legando insieme vettori o matrici “orizzontalmente”:

```
> x <- 1:3
> y <- 4:6
> m <- cbind(x, y)
> m
```

	x	y
[1, ]	1	4
[2, ]	2	5
[3, ]	3	6

**rbind** forma matrici legando insieme vettori o matrici “verticalmente”:

```
> x <- 1:3
> y <- 4:6
> m <- rbind(x, y)
> m
```

	[, 1]	[, 2]	[, 3]
x	1	2	3
y	4	5	6

Se i vettori costituenti non sono della stessa lunghezza si applica la regola del riciclo

# Operazioni con le matrici (1)

## *Somma e prodotto con costanti*

```
> x <- 1:12
> A <- matrix(x,nrow=3)
> A + 2
      [,1] [,2] [,3] [,4]
[1,]    3    6    9   12
[2,]    4    7   10   13
[3,]    5    8   11   14
> B <- A * 2
> B
      [,1] [,2] [,3] [,4]
[1,]    2    8   14   20
[2,]    4   10   16   22
[3,]    6   12   18   24
```

## *Somma e prodotto “elemento per elemento”*

```
> A+B
      [,1] [,2] [,3] [,4]
[1,]    3   12   21   30
[2,]    6   15   24   33
[3,]    9   18   27   36
> A*B
      [,1] [,2] [,3] [,4]
[1,]    2   32   98  200
[2,]    8   50  128  242
[3,]   18   72  162  288
```

# Operazioni con le matrici (2)

Prodotto di matrici “righe X  
colonne”: Operatore `%*%`

Si ricordi che A e B sono  
matrici 3X4:

```
> A%*%B # il numero delle  
# colonne di A deve essere  
# uguale al numero di righe di  
# B!
```

```
Error in A %*% B : non-  
conformable arguments
```

```
> A%*%t(B) # t(B) indica la  
# trasposta di B
```

```
      [,1] [,2] [,3]  
[1,]  332  376  420  
[2,]  376  428  480  
[3,]  420  480  540
```

Prodotto di matrici per vettori:

```
> z <- 1:4
```

```
> A%*%z
```

```
      [,1]  
[1,]   70  
[2,]   80  
[3,]   90
```

```
> z%*%A
```

```
Error in z %*% A : non-  
conformable arguments
```

```
> z%*%t(A)
```

```
      [,1] [,2] [,3]  
[1,]   70   80   90
```

# Altri operatori e funzioni per le matrici

In R esiste un repertorio amplissimo di operatori e funzioni specifiche per le matrici (si veda la documentazione):

- Ad es: la funzione *diag*, che ha un significato diverso a seconda che il suo argomento sia un intero, un vettore ed una matrice (si provi).
- La funzione *solve* consente di invertire una matrice e può essere usata per risolvere sistemi lineari. Ad es, se  $A$  è una matrice quadrata non singolare e  $b$  un vettore compatibile con  $A$ , allora `x<-solve(A,b)` risolve il sistema lineare  $Ax=b$ . (si provi ...)
- Esistono inoltre funzioni per il calcolo degli autovalori ed autovettori di matrici simmetriche, per il calcolo della svd di una matrice e molte altre.



# Accesso agli elementi di array e matrici

Le regole di accesso per array e matrici seguono quelle già viste per i vettori, considerando però l' esistenza di più indici e quindi la possibilità di utilizzare un vettore per ogni dimensione:

- Vettori di **indici interi positivi**
- Vettore di **indici interi negativi**
- Vettore di **indici logici**
- Vettori di **indici a caratteri**

Si utilizza quindi un vettore di indici *per ogni dimensione* dell' array

# Esempi di accesso agli elementi di una matrice

```
> m <- matrix (1:12,nrow=2)
> m
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    3    5    7    9   11
[2,]    2    4    6    8   10   12
> m[1,2]
[1] 3
> m [1,3:4] # el. I riga,col. 3 e 4
[1] 5 7
> m [1:2,4:6] # I e II riga, col.
              # dalla IV alla VI
      [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
> m[1,] #tutti gli el. della I riga
[1] 1 3 5 7 9 11
> m[,3] #tutti gli el. III col.
[1] 5 6
> m[,c(1,2,6)]
      [,1] [,2] [,3]
[1,]    1    3   11
[2,]    2    4   12
```

```
> m[,-4] # esclusione el. IV
# colonna
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    9   11
[2,]    2    4    6   10   12
> m[m>4] # el.>4 (si ottiene un
# vettore)
[1] 5 6 7 8 9 10 11 12

> dimnames(m)<list(c("r1","r2"),
paste("c",1:6,sep="")) # viene
# dato un nome alle componenti
> m
      c1 c2 c3 c4 c5 c6
r1    1  3  5  7  9 11
r2    2  4  6  8 10 12
> m["r1","c2"] # vettori indice
              # a caratteri
[1] 3
> m["r1",]
c1 c2 c3 c4 c5 c6
 1  3  5  7  9 11
```

# Esercizi

- Costruire una matrice  $10 \times 10$  composta da numeri casuali in almeno 2 modi diversi utilizzando le funzioni `matrix` ed `array`.
- Costruire 2 matrici di caratteri a piacere  $x$  e  $y$ , la prima di dimensione  $3 \times 4$ , la seconda di dimensione  $5 \times 3$ . Modificare con un unico assegnamento la matrice  $x$  in modo da sostituire le sue 2 prime colonne con le ultime 2 righe di  $y$ .
- Costruire due matrici  $M$  ed  $N$  che abbiano entrambe 5 colonne. Costruire, se possibile, tramite `rbind` una matrice di 5 colonne che abbia come righe le righe di entrambe le matrici. Utilizzando  $M$  ed  $N$ , è possibile costruire una matrice tramite `cbind`?
- Date due matrici quadrate  $A$  e  $B$  di dimensione  $3 \times 3$  costituite da numeri casuali, calcolarne il prodotto elemento per elemento e “righe per colonne”. Scelto un vettore  $b$  di 3 elementi, risolvere il sistema lineare  $Ax=b$  che ne deriva.
- Quale struttura dati si potrebbe utilizzare per modellare un insieme di dati numerici relativi a diverse tipologie di analisi per un insieme di diversi pazienti? Se tali dati dovessero essere rilevati ad intervalli di tempo differenti, quale struttura dati si potrebbero utilizzare?

# Fattori

- Strutture dati per rappresentare valori che *possono assumere solo valori discreti*
- In R i diversi valori che possono assumere i dati discreti sono definiti come *livelli*
- Es: dati qualitativi, o dati ordinali.
- Spesso si usano per “etichettare” dati

# Costruzione di fattori

I fattori sono costruiti tramite la funzione **factor**:

```
> trt <- factor( rep( c(" Control", "Treated"), c( 3,
  4)))
> trt
[1] Control Control Control Treated Treated
   Treated Treated
Levels: Control Treated
> levels(trt) # visualizza i livelli del fattore
[1] " Control" "Treated"
> str( trt) # visualizza in modo succinto la struttura
             # di un oggetto R
Factor w/ 2 levels "Control"," Treated": 1 1 1 2 2 2 2
> summary( trt) # la funzione summary fornisce una
# tabella della frequenze dei due livelli Control e #
Treated
Control Treated
3         4
```

# Esercizi

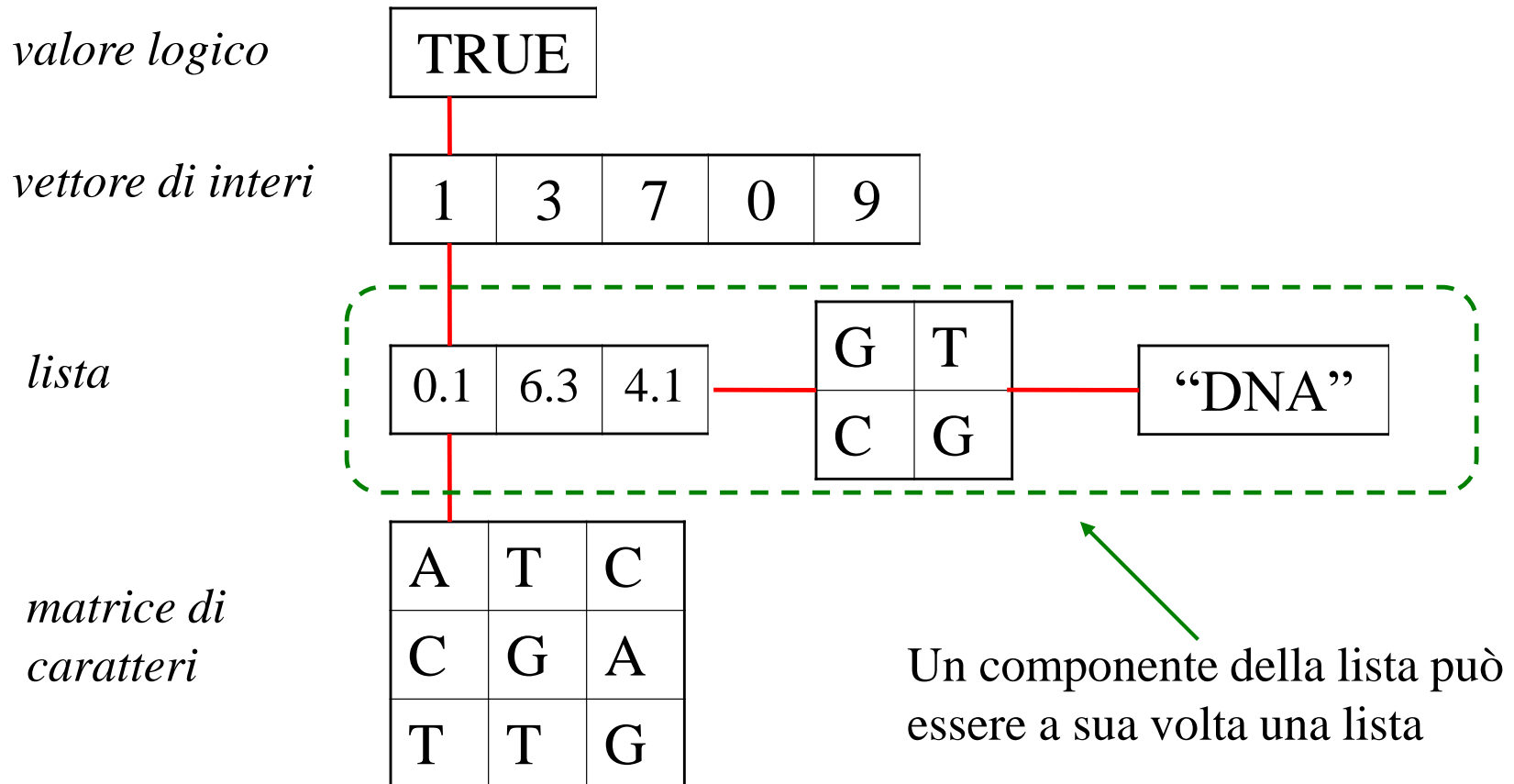
- Costruisci un fattore di 25 elementi (stringhe), caratterizzato da 3 diversi livelli
- Costruisci un fattore ordinato di 15 elementi di livelli basso < medio < alto.  
Genera un corrispondente vettore di misurazioni di lunghezza 15 e calcola il valor medio e la deviazione standard (funzione sd) di ogni livello tramite la funzione tapply.
- Quali strutture dati si potrebbero scegliere per modellare un data set caratterizzato da un insieme di pazienti sani e malati sottoposti a 5 diversi tipi di analisi cliniche?

# Lista come insieme ordinato di oggetti eterogenei (1)

- Le liste rappresentano un *insieme ordinato di oggetti* (**componenti**)
- Le componenti possono non essere dello stesso tipo o modo. Quindi le liste rappresentano *insiemi di oggetti eterogenei*.
- I componenti possono essere ad es: un vettore numerico, un valore logico, una matrice, un array di caratteri, una funzione o anche un' altra lista.
- La lista è quindi una *struttura dati ricorsiva*, poichè una sua componente può essere a sua volta una lista (e la lista componente può avere come componente un' altra lista).

# Lista come insieme ordinato di oggetti eterogenei (2)

Una lista composta da oggetti eterogenei: un valore logico, un vettore di interi, un'altra lista ed una matrice di caratteri





# Costruzione di una lista

Per costruire le liste si usa la funzione **list**:

I componenti delle liste sono sempre **numerati**:

```
> li <- list(TRUE, c(1,3,7,0,9))
> li
[[1]]
[1] TRUE
[[2]]
[1] 1 3 7 0 9
```

E' però possibile assegnare alle componenti un **nome**:

```
> li <- list(val=TRUE,
vector=c(1,3,7,0,9))
> li
$val
[1] TRUE
$vector
[1] 1 3 7 0 9
```

# Accesso alle componenti di una lista

Esistono 3 modalità di accesso alle componenti di una lista:

- Accesso tramite *indice numerico*
- Accesso tramite il *nome delle componenti*
- Accesso tramite *indice “a caratteri”*

# Accesso tramite indice numerico

I componenti delle liste sono numerati ed è possibile accedere ad essi tramite indice numerico racchiuso fra doppie parentesi quadre.

Es:

```
> li <- list(val=TRUE, vector=c(1,3,7,0,9),  
m=matrix(1:12,nrow=2))
```

```
> li[[1]]
```

```
[1] TRUE
```

```
> li[[2]]
```

```
[1] 1 3 7 0 9
```

```
> li[[3]]
```

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

# Gli operatori “[[]]” e “[ ]”

- L’ accesso alle componenti tramite gli operatori “[[]]” e “[ ]” produce risultati sostanzialmente differenti.
- “[[]]” è l’ operatore che seleziona l’ oggetto contenuto nella lista ( e l’ eventuale nome associato all’ oggetto non è incluso)
- “[ ]” è l’ operatore che seleziona una sottolista (si riferisce quindi ad un elemento di modo “lista”, e l’ eventuale nome associato all’ oggetto viene incluso )

# Accesso tramite il nome delle componenti

Se le componenti hanno un nome è possibile accedere ad esse direttamente tramite il nome stesso

Es:

```
> li <- list(val=TRUE, vector=c(1,3,7,0,9),  
m=matrix(1:12,nrow=2))
```

```
> li$val
```

```
[1] TRUE
```

```
> li$vector
```

```
[1] 1 3 7 0 9
```

Quindi `li$val` è equivalente a `li[[1]]` e `li$vector` a `li[[2]]`

Tramite la notazione `lista$nome` è possibile accedere anche ai singoli elementi delle componenti:

```
> li$vector[4]
```

```
[1] 0
```

```
> li$m[1,1]
```

```
[1] 1
```

# Accesso tramite indice “a caratteri”

Se le componenti hanno un nome è possibile accedere ad esse tramite indice “a caratteri”

Es:

```
> li <- list(val=TRUE, vector=c(1,3,7,0,9),  
m=matrix(1:12,nrow=2))  
> li[["m"]]  
      [,1] [,2] [,3] [,4] [,5] [,6]  
[1,]    1    3    5    7    9   11  
[2,]    2    4    6    8   10   12
```

Questa modalità può essere utile quando il nome della componente è memorizzato in un'altra variabile:

```
> v<-"vector"; li[[v]]  
[1] 1 3 7 0 9
```

# Allungare e concatenare liste

Come qualsiasi altro oggetto accessibile tramite indici, le liste possono essere estese, aggiungendo specifiche componenti:

```
> li<-list(0.6798,  
paste(c(rep("A",4),rep("T",4)),  
collapse=""))  
  
> li[3] <- list(TRUE)  
  
> li  
[[1]]  
[1] 0.6798  
[[2]]  
[1] "AAAATTTT"  
[[3]]  
[1] TRUE
```

La concatenazione di liste è possibile tramite la funzione **c**:

```
> li1 <- list(TRUE,2)  
> li2 <- list("pippo")  
> li3<-list( c(1,2,3),  
c("T","A","C"))  
> li123 <- c(li1,li2,li3)  
> li123  
[[1]]  
[1] TRUE  
[[2]]  
[1] 2  
[[3]]  
[1] "pippo"  
[[4]]  
[1] 1 2 3  
[[5]]  
[1] "T" "A" "C"
```

# Esercizi

- Costruire una lista *li* composta da una matrice numerica 4X4, da un vettore di caratteri con 32 elementi, dalla stringa “topo”, e da un’ ulteriore lista composta da un vettore di 10 elementi numerici e dal valore logico FALSE.
- Si estragga dalla lista  

```
li<-list(m=matrix(rnorm(64),nrow=8),s=c(rep("T",3),rep("G",5)))
```

(a) la II colonna della matrice (b) le “G” del vettore *s*.  
Si aggiunga quindi alla lista un vettore composto da 10 numeri casuali.
- Accedi in 3 modi diversi alla II componente della lista *li* dell’ es.2
- Spiega la differenza fra le due diverse modalità di accesso al primo elemento della lista *li*, *li[1]* e *li[[1]]*. Tramite la funzione *diag* si estragga la diagonale della matrice memorizzata nella lista *li*.
- Una serie di campioni è sottoposto ad una serie di analisi, i cui risultati sono espressi in modalità diverse a seconda del tipo stesso delle analisi: come valori interi, valori reali, stringhe di caratteri, un insieme di 3 valori (debole, medio, intenso), come vettore di numeri reali, e come sequenze di lunghezza variabile di numeri reali. Proponi una struttura dati R per modellare i risultati delle analisi relative ai diverse campioni, motivando la scelta effettuata.







# Componenti dei data frame

- Formalmente i **data frame** sono *liste* di *classe* **data.frame**
- I *componenti* (colonne) del data frame possono essere costituiti da:
  - Vettori (numerici, a caratteri, logici)
  - Fattori
  - Matrici numeriche
  - Liste
  - Altri data frame

# Costruzione dei data frame

I data frame sono costituiti tramite la funzione

**data.frame:**

```
> x<-1:4
> y<-5:8
> z<-paste("A",1:4,sep="")
> da.fr<-data.frame(x,y,z)
> da.fr
  x y  z
1 1 5 A1
2 2 6 A2
3 3 7 A3
4 4 8 A4
```

```
> m1 <-matrix(1:12,nrow=2)
> v <- c("A","C")
> daf3<-data.frame(m1,v)
> daf3
  x1 x2 x3 x4 x5 x6 v
1  1  3  5  7  9 11 A
2  2  4  6  8 10 12 C
> v1<- c("A","C","G")
> daf4<-data.frame(m1,v1)
Error in data.frame(m1, v1) :
arguments imply differing
number of rows: 2, 3
```

# Accesso alle componenti ed agli elementi dei data frame

Esistono due modalità generali di accesso alle componenti ed agli elementi dei data frame:

- I data frame sono liste, e quindi è possibile accedere ad essi secondo le *modalità di accesso tipiche delle liste* stesse.
- Come classe data frame, sono definiti *operatori di accesso tramite vettori di indici*, simili a quelli utilizzati per le matrici e gli array.



# Accesso alle componenti tramite vettori di indici

Sono definiti operatori di accesso specifici per la classe *data.frame*: si tratta di vettori di indici con una semantica simile a quella delle matrici ordinarie:

Es:

```
> x<-1:4; y<-5:8
> z<-paste("A",1:4,sep="")
> da.fr<-data.frame(x,y,z)
> da.fr
  x y  z
1 1 5 A1
2 2 6 A2
3 3 7 A3
4 4 8 A4
```

```
> da.fr[1,2]
[1] 5
> da.fr[2,2:3]
  y  z
2 6 A2
> da.fr[3,]
  x y  z
3 3 7 A3
> da.fr[2:4,1:2]
  x y
2 2 6
3 3 7
4 4 8
```





# Esercizi

- Costruire un data frame `da.fr` che abbia come componenti un vettore numerico casuale `v` di lunghezza 20, una matrice casuale `m` con 4 colonne ed una lista `i` i cui componenti siano 3 matrici a piacere.
- Costruire una lista che abbia come componenti 3 vettori a caratteri. Trasformare la lista in un data frame tramite la funzione `as.data.frame`. Quali sono le restrizioni che si devono applicare alle liste perchè siano dei data frame?
- Selezionare dal data set `iris` le osservazioni relative alle specie “virginica” con `Petal.Length > 5.890`.
- Tramite la funzione `summary` ricavare informazioni statistiche di base sulla specie “versicolor” del data set `iris`.
- Si dispone di un insieme di dati sperimentali (ad es: dati clinici e dati bio-molecolari) da utilizzare a fini diagnostici, relativi ad un insieme di pazienti.  
Si discuta se ed in quali condizioni i dati siano rappresentabili tramite data frame.