UNIVERSITÀ DEGLI STUDI DELLA TUSCIA

INFORMATICA

Linguaggio R
Strutture Dati

Dott. Franco Liberati
franco.liberati@unitus.it

LINGUAGGIO R

Argomenti del corso

LINGUAGGIO R

Strutture dati: generalità

- Una struttura dati definisce il modo in cui sono organizzate le informazioni digitali

- In altre parole una struttura dati è un particolare tipo di dato, caratterizzato dall'organizzazione dei dati, più che dal tipo di dato stesso.
  È quindi composta da un modo sistematico di organizzare i dati ed un insieme di operatori che permettono di manipolare la struttura

Vettore/Vector

LINGUAGGIO R

VETTORE

- Un **vettore** è una struttura dati che corrisponde ad un gruppo di locazioni di memoria contigue, ognuna nota come elemento, che hanno tutte la stessa tipologia

- Per fare riferimento a un elemento è necessario specificare il nome del vettore e la sua posizione, ovvero l'indice. In R l'indice iniziale del vettore è 1

INDICE	VALORE
1	64.64
2	45.44
3	23.12
4	11.24
5	56.23
6	76.55
7	33.78
8	89.12

LINGUAGGIO R

VETTORE: definizione ed inizializzazione

- Un vettore multi-dimensione è una collezione di elementi di uguale tipologia e può essere definito usando:
  - la funzione
    seq(from=init**, to=**end**,by=**step**)**
  - per vettori con elementi numerici, eccetto il tipo COMPLEX il costrutto:
    from**:**to
  - La funzione:
    c(v1,…,vn**)**

ESEMPIO

#Creazione di un vettore di cinque elementi da #1 #a 5

```
Vett1<-seq(from=1,to=5,by=1)
Vett2<- 1:5
Vett3<- c(1,2,3,4,5)
```

LINGUAGGIO R

VETTORE: definizione ed inizializzazione

La funzione

seq(from=init, to=end,by=step)

crea una sequenza numerica che va da un valore iniziale (from) ad uno finale (to) con un incremento dettato dal parametro dell'ultimo argomento (by), anche detto passo incrementale

ESEMPIO

```
#Definizione di un vettore multidimensionale
# numerico con funzione seq

vett_num<-seq(5,10,by=0.5)

print(vett_num)

#Output
[1] 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5 9.0 9.5 10.0
```

LINGUAGGIO R

VETTORE: definizione ed inizializzazione

Il costrutto

from:to

è una semplificazione della funzione seq e consente di definire un vettore dal valore from al valore to avente come passo incrementale uno

Se il valore massimo non è un multiplo raggiungibile con il passo incrementare lo si scarta

ESEMPI

```
vett_num<-5:10
print(vett_num)
#Output
[1] 5 6 7 8 9 10

vett_num<-5.5:10.5
print(vett_num)
#Output
[1] 5.5 6.5 7.5 8.5 9.5 10.5

vett_num<-5.5:12.3
#se lil massimo specificato non appartiene alla sequenza…
print(vett_num) # … lo si scarta e si riporta l'ultimo
#elemento valido
#Output
[1] 5.5 6.5 7.5 8.5 9.5 10.5 11.5
```

LINGUAGGIO R

VETTORE: definizione ed inizializzazione

La funzione

c(v1**,**…**,**vn**)**

crea un vettore di n elementi

Se almeno uno degli elementi è un carattere o una stringa gli altri sono automaticamente convertiti nel tipo CHARACTER

La funzione **c** ha anche lo scopo di concatenare vettori

Scrivere c(1,2,3) infatti equivale a concatenare i vettori atomici (1),(2) e (3)

ESEMPI

#Vettore di numerici

```
vett_int<-c(1,2,56,78,100)
print(vett_int)
[1] 1 2 56 78 100
```

#Vettore di stringhe

```
vett_str<-c("verde", "bianco", "rosso", "blu", "giallo")
print(vett_str)
[1] verde bianco rosso blu giallo
```

#Vettore di numerici e stringhe

```
vett_str<-c("mela", "albicocca", "melone", 3, TRUE, 4L,
3+2i,45.78)
print(vett_str)
[1] "mela", "albicocca", "melone", "3", "TRUE", "4",
"3+2i","45.78" #Tutti gli elementi sono stringhe
```

LINGUAGGIO R

VETTORE: definizione ed inizializzazione

La funzione c ha anche lo scopo di concatenare vettori o vettori con elementi

ESEMPI

```
#Vettore di numerici
vett_int<-c(1,2,56,78,100)
print(vett_int)
[1] 1 2 56 78 100

vett_int<-c(vett_int,45)
print(vett_int)
[1] 1 2 56 78 100 45

_____

str<-c("pippo", "pluto", "paperino")
print(str)
[1] "pippo" "pluto" "paperino"
str2<-c(str, "qui" "quo" "qua")
print(str2)
[1] "pippo" "pluto" "paperino" "qui" "quo" "qua"
```

LINGUAGGIO R

VETTORE: definizione ed inizializzazione

Tra i costruttori di un vettore c'è anche la funzione

rep(x, times, each**)**

che genera un vettore ottenuto replicando l'oggetto x per il numero di volte specificato da times. Con l'argomento each=n si ripete n volte l'oggetto x

ESEMPI

```
x<-1
print(rep(x,7))
[1] 1 1 1 1 1 1 1

x<-1:5
print(rep(x,3))
[1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5

x<-c(1,2,3)
print(rep(x,each=4))
[1] 1 1 1 1 2 2 2 2 3 3 3 3

x<-c(10,20,30)
print(rep(x,times=4,each=2))
[1] 10 10 20 20 30 30 10 10 20 20 30 30 10 10 20 20 30 30
10 10 20 20 30 30
```

LINGUAGGIO R

VETTORE: accesso ad un elemento

Il **prelievo degli elementi** avviene specificando l'indice tra parentesi quadre.

L'indice degli elementi segue l'ordine di scrittura a partire dalla **prima posizione che ha indice uno**

ESEMPIO

```
vett_str<-c("verde", "bianco", "rosso", "blu", "giallo")
print(vett_str[3])
[1] rosso
```

LINGUAGGIO R

VETTORE: accesso a più elementi

Per **prelevare più elementi** si possono specificare più indici utilizzando la funzione c intervallati da una virgola

Formalmente con vett[subvett], dove subvett è un vettore numerico, si estrae da vett il sotto-vettore i cui indici sono esplicitati in subvett

Gli elementi proposti sono separati da uno spazio in quanto il risultato è un vettore a tutti gli effetti

ESEMPIO

```
vett_str<-c("verde", "bianco", "rosso", "blu", "giallo" )
print(vett_str[c(3,5)])
[1] rosso giallo
```

Per escludere degli elementi dal prelievo si può definire un vettore, utilizzando la funzione c, con indici di segno negativo

ESEMPIO

```
vett_str<-c("verde", "bianco", "rosso", "blu", "giallo" )
print(vett_str[c(-1,-3,-4)])
[1] bianco giallo
```

LINGUAGGIO R

VETTORE: accesso a più elementi

Un altro modo per **accedere agli elementi del vettore è la definizione di un vettore di uguale dimensione con elementi booleani**: sono selezionati gli elementi che hanno indice uguale al valore TRUE

ESEMPIO

```
vett_str<-c("verde", "bianco", "rosso", "blu", "giallo" )
print(vett_str[c(TRUE,FALSE,FALSE,FALSE,TRUE)])
[1] verde giallo
```

L'accesso ai dati può essere svolto anche con espressioni logiche matematiche

Ad esempio scrivendo vett[vett>0] si estrae da vett il sotto vettore formato dai soli elementi positivi

ESEMPIO

```
x<-c(-1,2,4,-3,15,-56,7)
vett<-x[x>5]
print(vett)
[1] 15 7
```

LINGUAGGIO R

VETTORE: aggiunta degli elementi

Per **aggiungere un elemento** (values) al vettore (vett) di dimensione n si può usare la funzione

append(vett, values, after**)**

Se after≤0 allora il valore è posizionato in cima

Se after≥n il valore è posizionato in coda

ESEMPIO

```
vett_str<-c(1.2,3.4,5.6)
v1<-append(vett_str,values=6, after=-2)
print(v1)
# 6.0  1.2  3.4  5.6

vett_str<-c(1.2,3.4,5.6)
v1<-append(vett_str,values=6, after=2)
print(v1)
# 1.2  3.4  6.0 5.6

vett_str<-c(1.2,3.4,5.6)
v1<-append(vett_str,values=6, after=7)
print(v1)
# 1.2  3.4  5.6  6.0
```

LINGUAGGIO R

VETTORE: operazioni aritmetiche

Il Linguaggio R estende gli operatori matematici elementari su vettori aventi uguale dimensione e la funzione aritmetica coinvolge gli elementi ad uguale posizione

LINGUAGGIO R

VETTORE: somma sottrazione prodotto divisione

ESEMPIO SOMMA

```
v1<-c(1,2,3,4,5)
v2<-c(6,7,8,9,10)
somma_v<-v1+v2
print(somma)
[1] 7 9 11 13 15
```

ESEMPIO SOTTRAZIONE

```
v1<-c(1,20,13,11, 5);
v2<-c(4, 7,  8, 9,10);
print(v1-v2)
[1] -3 13 5 2 -5
```

ESEMPIO PRODOTTO (PRODOTTO VETTORIALE)

```
v1<-c(1,5,4,3,16)
v2<-c(5,4,3,5,10)
prod_v<-v1*v2
print(prod_v)
[1] 5 20 12 15 160
```

ESEMPIO DIVISIONE

```
v1<-c(10,5,12,15,16);
v2<-c(5,4,3,5,9)
div_v<-v1/v2
print(div_v)
[1] 2.000000 1.250000 4.000000 3.000000 1.777778
```

LINGUAGGIO R

VETTORE: operazioni con vettori di dimensione diversa

Nel caso si applichino gli operatori aritmetici **a due vettori aventi dimensione diverse** allora il vettore con dimensione più piccola è trasformato in un vettore di pari dimensione al maggiore ripetendo gli elementi partendo dal primo e troncando quelli oltre la dimensione; cioè il vettore minore si completa con un andamento circolare

Quando si effettua questa opzione c'è un messaggio di avvertimento

LINGUAGGIO R

VETTORE: operazioni con vettori di dimensione diversa

ESEMPIO SOMMA

```
v1<-c(10,5,12,15,16,20)
vmin<-c(5,3)
som_v<-v1+vmin  #vmin è impostato a 5 3 5 3 5 3
print(som_v)
[1] 15  8 17 18 21 23
```

ESEMPIO SOTTRAZIONE

```
v2<-c(5,4,3,2,0)
vmin<-c(5,3)
dif_v<-v2-vmin      #vmin è impostato a 5 3 5 3 5
print(dif_v)
[1]  0  1 -2 -1 -5
```

ESEMPIO PRODOTTO (PRODOTTO VETTORIALE)

```
v3<-c(0,1,2,3,4,5,6,7,8,9,10)
vmin<-c(5,3)
pro_v<-v3*vmin
#vmin è impostato a 5 3 5 3 5 3 5 3 5 3 5
print(pro_v)
[1]  0  3 10  9 20 15 30 21 40 27 50
```

ESEMPIO DIVISIONE

```
v4<-c(10,8,15,16)
vmin<-c(5,3)
div_v<-v4/vmin      #vmin è impostato a 5 3 5 3
print(div_v)
[1] 2.000000 2.666667 3.000000 5.333333
```

ESEMPIO WARNING: *In* v2 - vmin : *longer object length is not a multiple of shorter object length*

LINGUAGGIO R

VETTORE: Lunghezza

La funzione **length(**x**)** restituisce il numero di elementi del vettore x

OSS: per conoscere la lunghezza di una stringa x si usa la funzione

nchar(x**)**

ESEMPIO

```
v1<-c(1,2,3,4,5,6)
print(length(v1))
[1] 6


v2<-c("A",5,"CASA",5+3i,"zebra")
print(length(v2))
[1] 5


s1<-"RICHARD BENSON“
print(nchar(s1))
print(length(s1))
```

LINGUAGGIO R

VETTORE: ordinamento

Tra le funzioni impiegabili con i vettori c'è l'ordinamento

sort(vector, **decreasing = FALSE)**

l'argomento decreasing, se impostato a TRUE, consente un ordinamento decrescente

L'ordinamento si applica sia agli interi sia ai caratteri e alle stringhe

ESEMPIO

```
#ORDINAMENTO NUMERI (CRESCENTE)
v1<-c(10,5,12,15,16,20)
result_sort_asc<-sort(v1)
print(result_sort_asc)
[1] 5 10 12 15 16 20

#ORDINAMENTO NUMERI (DECRESCENTE)
result_sort_dec<-sort(v1, decreasing=TRUE)
print(result_sort_dec)
[1] 20 16 15 12 10 5

#ORDINAMENTO STRINGHE (CRESCENTE)
v1<-c("mela", "uva", "ananas","mango", "kiwi")
result_sort<-sort(v1)
print(result_sort)
[1] "ananas" "kiwi" "mango" "mela" "uva"
```

LINGUAGGIO R

VETTORE: ricerca del Massimo e del Minimo

Tra le funzioni impiegabili con i vettori c'è la ricerca del massimo

max(vector**)**

Esiste anche la funzione di ricerca opposta

min(x**)**

Qualora le due funzioni si applicano a un vettore stringa si restituisce l'elemento con letterali aventi codifica ASCII più alta (max) o bassa (min)

ESEMPIO

```
#CALCOLO MASSIMO
v1<-c(10,5,120,15,16,20)
print(max(v1))
[1] 120

#CALCOLO MINIMO
v1<-c(10,5,120,15,16,20)
print(min(v1))
[1] 52 10 5

# Massimo e Minimo di un vettore di elementi stringhe
v1<-c("A",5,"CASA",16,120,"zebra")
print(max(v1))
[1] "zebra" (il carattere 'z' ha valore ASCII 122)
print(min(v1))
[1] "120"       (il carattere '1' ha valore ASCII 49)
```

LINGUAGGIO R

VETTORE: Sommatoria e Produttoria

Tra le funzioni impiegabili con i vettori c'è la sommatoria

$$\sum_{i=1}^{n} v_i = v_1 + \ldots + v_n$$

che ha sintassi:

sum(vector**)**

Esiste anche la funzione di produttoria

$$\prod_{i=1}^{n} v_i = v_1 + \ldots + v_n$$

che ha sintassi:

prod(x**)**

ESEMPIO

```
#CALCOLO SOMMATORIA
v1<-c(1,2,3,4,5,6)
print(sum(v1))
[1] 21


#CALCOLO PRODUTTORIA
v1<-c(1,2,3,4,5,6)
print(prod(v1))
[1] 720
```

LINGUAGGIO R

VETTORE: estrazione di sottovettori

La funzione **head(**vett, n**)** seleziona i primi n elementi.

La procedura **tail(**vett,n**)** ha un comportamento opposto ad head(), e seleziona gli ultimi n elementi

La funzione **subset(**vett,subset**)** estrae un sottovettore da vett in accordo ad una condizione specificata in subset

ESEMPIO

```
vett <- c(1.2, 3.2, 3.3, 2.5, 5, 5.6)
print(head(vett,3))
[1] 1.2, 3.2 3.3
```

```
vett<-c(4.5, 6.7, 8.9, 7.7, 11.2)
print(tail(vett,2))
[1] 7.7 11.2
```

```
#Uso di subset per estrarre gli elementi maggiori di 7
vet <- c(7.8, 6.6, 6.5, 7.4, 7.3, 7, 6.14, 7.1, 6.7,6.8)
print(subset(vet,vet> 7))
[1] 7.8  7.4  7.3  7.1
```

LINGUAGGIO R

VETTORE: applicazione di funzione

Un funzione molto interessante è

sapply(vet, FUN**)**

che prende un vettore e applica ai suoi elementi la subroutine FUN

ESEMPIO

```
MIA_FUNZIONE<-function(x){
return(x^2)
}

vet1<-c(1,2,3,4,5,6,7,8,9,10)
ris<-sapply(vet1,MIA_FUNZIONE)
print(ris)

[1] 1   4   9  16  25  36  49  64  81 100
```

Matrice/Matrix

LINGUAGGIO R

MATRICE: generalità

Una **matrice** è una tabella di valori omogenei

Il numero di righe (R) per il numero di colonne (C) determina la dimensione (o ordine) della tabella e si specifica come M_{RxC}

Per identificare un elemento di una tabella è necessario dover specificare la riga e la colonna

Matrice di ordine 5x4 (o M_{5x4})

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}
a_{51}	a_{52}	a_{53}	a_{54}

1	32	81	12
123	490	72	345
100	58	63	44
9	34	556	33
8	56	77	22

LINGUAGGIO R

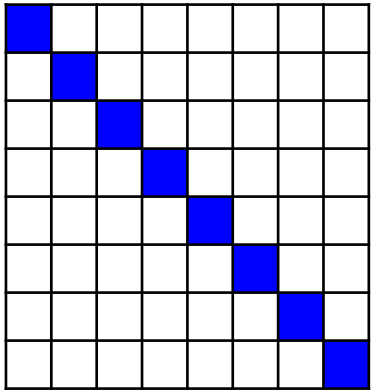
MATRICE: generalità (matrice quadrata)

Se R=C si ha una **matrice quadrata**

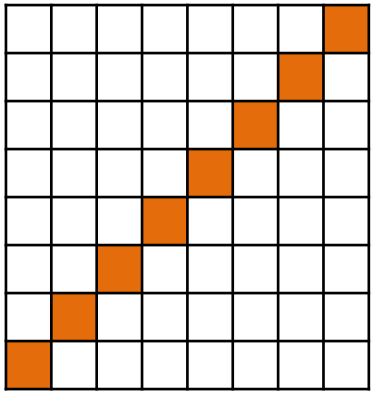
Gli elementi di una matrice quadrata con indici uguali individuano la **diagonale principale**

La **diagonale secondaria** di una matrice quadrata è la diagonale che va dall'angolo in alto a destra a quello in basso a sinistra.

Diagonale principale



Diagonale secondaria



LINGUAGGIO R

MATRICE: definizione

Nel Linguaggio R una **matrice è un oggetto i cui elementi sono disposti in due dimensioni e sono tutti della stessa tipologia**

Una matrice si definisce mediante la funzione:

matrix(data,nrow,ncol,byrow,dimnames)

data: sono gli elementi che costituiscono la matrice;

nrow: specifica il numero di righe;

ncol: indica il numero di colonne,

byrow: se è impostato a TRUE indica che gli elementi in data sono scritti per righe, se è impostato a FALSE gli elementi sono riportati a partire dalla prima colonna

dimname: consente di assegnare dei nomi alle righe e alle colonne

ESEMPIO

```
M <- matrix(1:20, nrow = 4, byrow = TRUE)
print(M)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
[4,]   16   17   18   19   20

N <- matrix(1:20, nrow = 4, byrow = FALSE)
print(N)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
```

LINGUAGGIO R

MATRICE: definizione

Nel Linguaggio R una matrice è un oggetto i cui elementi sono disposti in due dimensioni e sono tutti della stessa tipologia

Una matrice si definisce mediante la funzione:

matrix(data,nrow,ncol,byrow,dimnames**)**

data: sono gli elementi che costituiscono la matrice;

nrow: specifica il numero di righe;

ncol: indica il numero di colonne,

byrow: se è impostato a TRUE indica che gli elementi in data sono scritti per righe, se è impostato a FALSE gli elementi sono riportati a partire dalla prima colonna

dimname: **consente di assegnare dei nomi alle righe e alle colonne**

ESEMPIO

```
rownames = c("Riga 1", "Riga 2", "Riga 3", "Riga 4")
colnames = c("Colonna 1", "Colonna 2", "Colonna 3",
"Colonna 4", "Colonna 5")

P <- matrix(1:20, nrow = 4, byrow = TRUE, dimnames =
list(rownames, colnames))

print(P)
```

	Colonna 1	Colonna 2	Colonna 3	Colonna 4	Colonna 5
Riga 1	1	2	3	4	5
Riga 2	6	7	8	9	10
Riga 3	11	12	13	14	15
Riga 4	16	17	18	19	20

LINGUAGGIO R

MATRICE: prelievo e aggiornamento elemento

Il **prelievo di un elemento** si ottiene con la specifica della matrice e l'uso delle parentesi quadre in cui si specifica l'indice di riga e l'indice di colonna separato da una virgola:

elem<-matrice[indice_riga, indice_colonna]

La **modifica (o aggiornamento)** di una cella con un valore ha sintassi inversa:

matrice[indice_riga, indice_colonna]<-elem

ESEMPIO

```
M <- matrix(1:20, nrow = 4, byrow = TRUE)
print(M)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
[4,]   16   17   18   19   20

print(M[2,3])
[1] 8

M[3,2]<-100
print(M)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11  100   13   14   15
[4,]   16   17   18   19   20
```

LINGUAGGIO R

MATRICE: prelievo elementi

Omettendo il numero di colonna, o il numero di riga, si selezionano gli elementi di una intera riga, o rispettivamente di una intera colonna

ESEMPIO

```
M <- matrix(1:20, nrow = 4, byrow = TRUE)
print(M)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
[4,]   16   17   18   19   20

print([M[2,])
[1]    6     7     8     9       10
print(M[,3])
[1]    3     8    13    18
```

LINGUAGGIO R

MATRICE: operazione SOMMA

Se A e B sono due matrici aventi lo stesso numero di righe e di colonne, si definisce **matrice somma** di A e B e si indica con A+B, la matrice di uguale dimensione i cui elementi sono ciascuno la somma degli elementi corrispondenti delle matrici note

L'addizione fra matrici di uguale ordine gode della proprietà associativa, cioè A+(B+C)=(A+B)+C; della proprietà commutativa: A+B=B+A; e ammette come elemento neutro la matrice nulla (ovvero una matrice con tutti elementi 0) dello stesso tipo per ogni matrice A, ovvero A+0 = 0+A=A.

ESEMPIO

```
A <- matrix(1:20, ncol = 4, byrow = TRUE)
B <- matrix(11:30, ncol= 4, byrow = TRUE)
S<-A+B
print(A)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[4,]   13   14   15   16
[5,]   17   18   19   20
print(B)
     [,1] [,2] [,3] [,4]
[1,]   11   12   13   14
[2,]   15   16   17   18
[3,]   19   20   21   22
[4,]   23   24   25   26
[5,]   27   28   29   30
print(S)
     [,1] [,2] [,3] [,4]
[1,]   12   14   16   18
[2,]   20   22   24   26
[3,]   28   30   32   34
[4,]   36   38   40   42
[5,]   44   46   48   50
```

LINGUAGGIO R

MATRICE: operazione DIFFERENZA

Se A e B sono due matrici aventi lo stesso ordine, si definisce **matrice differenza** di A e B e si indica con A - B, la matrice di uguale dimensione i cui elementi sono ciascuno la differenza degli elementi corrispondenti delle matrici note

ESEMPIO

```
A <- matrix(1:20, nrow = 5, byrow = TRUE)
B <- matrix(11:30, nrow = 5, byrow = TRUE)
D<-A-B
print(A)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[4,]   13   14   15   16
[5,]   17   18   19   20
print(B)
     [,1] [,2] [,3] [,4]
[1,]   11   12   13   14
[2,]   15   16   17   18
[3,]   19   20   21   22
[4,]   23   24   25   26
[5,]   27   28   29   30
print(D)
     [,1] [,2] [,3] [,4]
[1,]  -10  -10  -10  -10
[2,]  -10  -10  -10  -10
[3,]  -10  -10  -10  -10
[4,]  -10  -10  -10  -10
[5,]  -10  -10  -10  -10
```

LINGUAGGIO R

MATRICE: matrice opposta

La **matrice opposta** di A, che si indica con –A, è la matrice i cui elementi sono gli elementi opposti dei corrispondenti in A

La matrice differenza si può ricavare come somma della prima matrice con l'opposta della seconda: A–B=A+(-B)

ESEMPIO

```
A <- matrix(1:16, nrow = 4, byrow = TRUE)
O<-A*-1

print(A)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[4,]   13   14   15   16

print(O)
     [,1]   [,2]   [,3]  [,4]
[1,]   -1    -2    -3    -4
[2,]   -5    -6    -7    -8
[3,]   -9   -10   -11   -12
[4,]  -13   -14   -15   -16
```

LINGUAGGIO R

MATRICE: operazione PRODOTTO

Se A e B sono due matrici aventi la stessa dimensione, si definisce **matrice prodotto** di A e B e si indica con A·B, la matrice di uguale dimensione i cui elementi sono ciascuno il prodotto degli elementi corrispondenti delle matrici note

ESEMPIO

```
A <- matrix(1:20, nrow = 5, byrow = TRUE)
B <- matrix(11:30, nrow = 5, byrow = TRUE)
P<-A*B
print(A)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[4,]   13   14   15   16
[5,]   17   18   19   20
print(B)
     [,1] [,2] [,3] [,4]
[1,]   11   12   13   14
[2,]   15   16   17   18
[3,]   19   20   21   22
[4,]   23   24   25   26
[5,]   27   28   29   30
print(P)
     [,1] [,2] [,3] [,4]
[1,]   11   24   39   56
[2,]   75   96  119  144
[3,]  171  200  231  264
[4,]  299  336  375  416
```

LINGUAGGIO R

MATRICE: operazione DIVISIONE

Se A e B sono due matrici aventi la stessa dimensione, si definisce la funzione di **divisione** di A con B e si indica con A/B, la matrice di uguale dimensione i cui elementi della matrice A sono dividendi degli elementi corrispondenti alla matrice B che sono i divisori

Non vale la proprietà commutativa né quella commutativa né quella neutra

ESEMPIO

```
A <- matrix(1:20, nrow = 5, byrow = TRUE)
B <- matrix(11:30, nrow = 5, byrow = TRUE)
P<-A/B
print(A)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[4,]   13   14   15   16
[5,]   17   18   19   20
print(B)
     [,1] [,2] [,3] [,4]
[1,]   11   12   13   14
[2,]   15   16   17   18
[3,]   19   20   21   22
[4,]   23   24   25   26
[5,]   27   28   29   30
print(P)
          [,1]      [,2]      [,3]      [,4]
[1,] 0.09090909 0.1666667 0.2307692 0.2857143
[2,] 0.33333333 0.3750000 0.4117647 0.4444444
[3,] 0.47368421 0.5000000 0.5238095 0.5454545
[4,] 0.56521739 0.5833333 0.6000000 0.6153846
```

LINGUAGGIO R

MATRICE: operazione PRODOTTO MATRICIALE

Il **prodotto matriciale**, C=AxB, è definito tra una matrice A è di ordine mxn e una matrice B di dimensione nxp e genera una matrice C di ordine mxp, il cui elemento c_{hk} è ottenuto dal prodotto della riga numero h della prima matrice per la colonna numero k della seconda matrice

Nel Linguaggio R il prodotto matricale si effettua con l'operatore **%*%**

Il prodotto matriciale gode della proprietà associativa, cioè (AxB)xC=Ax(BxC); della proprietà distributiva (a sinistra e a destra) della moltiplicazione rispetto all'addizione cioè Ax(B+C)=AxB+AxC e anche (A+B)xC=AxC+BxC ma non della proprietà commutativa (perché se si inverte l'ordine delle matrici non è detto valga ancora la condizione di definizione)

ESEMPIO

```
A <- matrix(c(2,1,2,0,1,1), nrow = 2, byrow = TRUE)
# 2 1 2
# 0 1 1
B <- matrix(c(0,1,2,1,1,0), nrow = 3, byrow = TRUE)
# 0  1
# 1  1
# 2  0
P<-A%*%B
```

```
a b c          k  n     a*k+b*l+c*m   a*n+b*o+c*p
d e f          l  o     d*k+e*l+f*m   d*n+e*o+f*p
               m  p
#2·0+1·2+2·1                  2·1+1·1+2·0
#0·0+1·2+1·1                  0·1+1·1+1·0
print(P)
     [,1] [,2]
[1,]    4    3
[2,]    3    1
```

LINGUAGGIO R

MATRICE: Prodotto matriciale per sistemi equazioni

- Qualora si abbia un sistema di 2 equazioni e 2 incognite del tipo:

$$\begin{cases} ax & +by & = e \\ cx & +dy & = f \end{cases}$$

- Il sistema può essere rappresentato come il prodotto matriciale di:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \%*\% \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} e \\ f \end{bmatrix}$$

Cioè la matrice dei coefficienti per il vettore delle variabili posto uguale al vettore dei risultati

- Questa forma si presta ad una soluzione semplice senza ricorrere alla sostituzione delle variabili, calcolando il **determinate della matrice** dei coefficienti
- Il **determinante di una matrice** è un numero che descrive alcune proprietà algebriche e geometriche della matrice ed è usato nella risoluzione dei sistemi di equazioni

LINGUAGGIO R

MATRICE: Prodotto matriciale per sistemi equazioni

- Se il determinante della matrice dei coefficienti è diverso da zero allora il sistema ha soluzione:

$$x = \frac{det\begin{bmatrix} e & b \\ f & d \end{bmatrix}}{det\begin{bmatrix} a & b \\ c & d \end{bmatrix}} \qquad y = \frac{det\begin{bmatrix} a & e \\ c & f \end{bmatrix}}{det\begin{bmatrix} a & b \\ c & d \end{bmatrix}} \qquad con\begin{bmatrix} a & b \\ c & d \end{bmatrix} diverso\ da\ 0$$

- Il determinante di una matrice 2x2 si calcola moltiplicando gli elementi della diagonale principale meno il prodotto degli elementi della diagonale secondaria

$$det\begin{bmatrix} a & b \\ c & d \end{bmatrix} = (a * d) - (b * c)$$

LINGUAGGIO R

MATRICE: Prodotto matriciale per sistemi equazioni (esempio)

- Si supponga si abbia un sistema di 2 equazioni e 2 incognite del tipo:

$$\begin{cases} 4x & +2y & = 8 \\ 5x & +3y & = 9 \end{cases}$$

- Il sistema può essere rappresentato come il prodotto matriciale di:

$$\begin{bmatrix} 4 & 2 \\ 5 & 3 \end{bmatrix} \%*\% \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 8 \\ 9 \end{bmatrix}$$

- Il determinate è diverso da zero perché 4*3-2*5=2… e quindi

$$x = \frac{det\begin{bmatrix} 8 & 2 \\ 9 & 3 \end{bmatrix}}{det\begin{bmatrix} 4 & 2 \\ 5 & 3 \end{bmatrix}} = \frac{(8*3)-(2*9)}{(4*3)-(2*5)} = \mathbf{3}$$

$$y = \frac{det\begin{bmatrix} 4 & 8 \\ 5 & 9 \end{bmatrix}}{det\begin{bmatrix} 4 & 2 \\ 5 & 3 \end{bmatrix}} = \frac{(4*9)-(8*5)}{(4*3)-(2*5)} = \mathbf{-2}$$

LINGUAGGIO R

MATRICE: operazione TRASPOSTA

Nel Linguaggio R la **trasposta di una matrice** (cioè la matrice ottenuta scambiandone le righe con le colonne) si ottiene applicando la funzione

t(matrix**)**

ESEMPIO

```
A <- matrix(c(2,1,2,0,1,1), nrow = 2, byrow = TRUE)
# 2 1 2
# 0 1 1
Atrasposta<-t(A)
print(Atrasposta)
     [,1] [,2]
[1,]    2    0
[2,]    1    1
[3,]    2    1
```

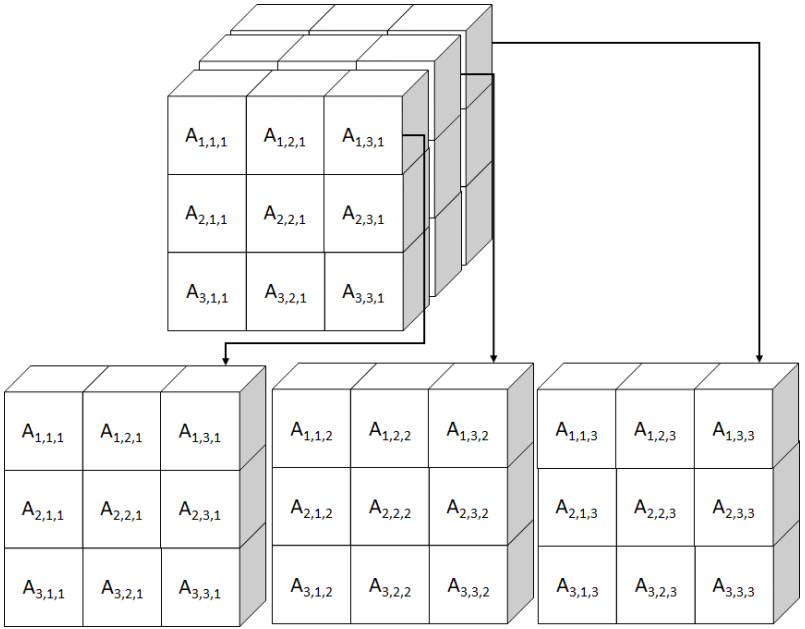
Vettore Multidimensionale/array

LINGUAGGIO R

ARRAY: generalità

Un vettore multi dimensionale definisce una **matrice multidimensionale (array)**

Nel caso di un array tridimensionale gli indici della notazione M_{RxCxL} indicano rispettivamente il numero di riga (R), il numero di colonna (C) e il livello (L), cioè il numero della matrice tra la collezione di matrici



LINGUAGGIO R

ARRAY: definizione

Nel Linguaggio R un vettore multidimensionale è un oggetto che descrive dati in più di due dimensioni

Un vettore multidimensionale a tre dimensioni è anche detto cubo; ad esempio l'array (2,3,4) può essere rappresentato con quattro matrici di ordine 2 righe e 3 colonne.

Nel caso in cui si superino le tre dimensioni si parla di iper-cubo

Un vettore multidimensionale si definisce mediante la funzione:

array(data,dim,dimnames)

dove data è il vettore numerico; dim è un vettore che ne specifica la dimensione; dimnames riporta le etichette delle dimensioni.

ESEMPIO

```
etichette<-list(c("Riga 1", "Riga 2"), c("Colonna 1","Colonna 2"),
c("Matrice 1", "Matrice 2", "Matrice 3"))

MM<-array(1:12,dim = c(2,2,3),dimnames = etichette)

print(MM)
, , Matrice 1
      Colonna 1 Colonna 2
Riga 1         1         3
Riga 2         2         4

, , Matrice 2
      Colonna 1 Colonna 2
Riga 1         5         7
Riga 2         6         8

, , Matrice 3
      Colonna 1 Colonna 2
Riga 1         9        11
Riga 2        10        12
```

LINGUAGGIO R

ARRAY: prelievo e aggiornamento elemento

Il prelievo di un elemento si ottiene con l'identificativo della matrice e l'uso delle parentesi quadre in cui si specifica l'indice di riga, di colonna e di livello (e altre dimensioni) separato da una virgola

elem<-M[indice_riga, indice_colonna, indice_matrice, indici_supplementari,…]

L'aggiornamento di un valore ha sintassi inversa

M[indice_riga, indice_colonna, indice_matrice, indici_supplementari,…]<-elem

LINGUAGGIO R

ARRAY: prelievo e aggiornamento elemento

ESEMPIO

```
etichette<-list(c("Riga 1", "Riga 2"), c("Colonna 1","Colonna 2"),
c("Matrice 1", "Matrice 2", "Matrice 3"))

MM<-array(1:12,dim = c(2,2,3),dimnames = etichette)

print(MM)
, , Matrice 1
      Colonna 1 Colonna 2
Riga 1         1         3
Riga 2         2         4

, , Matrice 2
      Colonna 1 Colonna 2
Riga 1         5         7
Riga 2         6         8

, , Matrice 3
      Colonna 1 Colonna 2
Riga 1         9        11
Riga 2        10        12
```

```
val<-MM[1,2,3]
print (val)
[1] 11

MM[2,2,2]<-10
print(MM)

, , Matrice 1
      Colonna 1 Colonna 2
Riga 1         1         3
Riga 2         2         4

, , Matrice 2
      Colonna 1 Colonna 2
Riga 1         5         7
Riga 2         6        10

, , Matrice 3
      Colonna 1 Colonna 2
Riga 1         9        11
Riga 2        10        12
```

LINGUAGGIO R

ARRAY: prelievo e aggiornamento elemento

Nel caso di un array cubico ommettendo uno (o due) dei tre parametri tra le parentesi graffe c'è il prelevamento o l'aggiornamento della matrice (o vettore) ottenuta dagli elementi con gli indici non specificati

La modalità operativa è analoga anche nel caso di iper-cubi

ESEMPIO

```
etichette<-list(c("Riga 1", "Riga 2"), c("Colonna 1","Colonna 2"),
c("Matrice 1", "Matrice 2", "Matrice 3"))
MM<-array(1:12,dim = c(2,2,3),dimnames = etichette)
, , Matrice 1
       Colonna 1 Colonna 2
Riga 1         1         3
Riga 2         2         4

, , Matrice 2
       Colonna 1 Colonna 2
Riga 1         5         7
Riga 2         6         8

, , Matrice 3
       Colonna 1 Colonna 2
Riga 1         9        11
Riga 2        10        12

val<-MM[, ,2]
print(val)
       Colonna 1 Colonna 2
Riga 1         5         7
Riga 2         6         8

val<-MM[,1,2]
print(val)
Riga 1 Riga 2
     5      6
```

LINGUAGGIO R

ARRAY: funzioni

È possibile applicare una funzione a tutti gli elementi di un vettore multi dimensionale attraverso:

apply(x, margin, FUN**)**

x è un array

margin specifica gli elementi da trattare (1, riga, 2 colonna, 3 matrice, 4…n altri livelli)

FUN è la procedura da applicare agli elementi.

LINGUAGGIO R

ARRAY: funzioni

ESEMPIO

```
etichette<-list(c("Riga 1", "Riga 2"), c("Colonna 1","Colonna 2"), c("A 1", "A 2", "A 3"))

A<-array(1:12,dim = c(2,2,3),dimnames = etichette)
print(A)

, , A1
      Colonna 1 Colonna 2
Riga 1         1         3
Riga 2         2         4
, , A 2
      Colonna 1 Colonna 2
Riga 1         5         7
Riga 2         6         8
, , A 3
      Colonna 1 Colonna 2
Riga 1         9        11
Riga 2        10        12
```

```
#Sommatoria degli elementi di ciascuna riga delle tre #matrici
val<-apply(A,1,sum)
Riga 1 Riga 2
    36     42

# Sommatoria degli elementi di ciascuna colonna delle tre #matrici
val<-apply(A,2,sum)
print(val)
Colonna 1 Colonna 2
       33        45

 #Sommatoria degli elementi ci ciascuna delle tre matrici
val<-apply(A,3,sum)
print(val)
A 1 A 2 A 3
 10  26  42
```

Lista/List

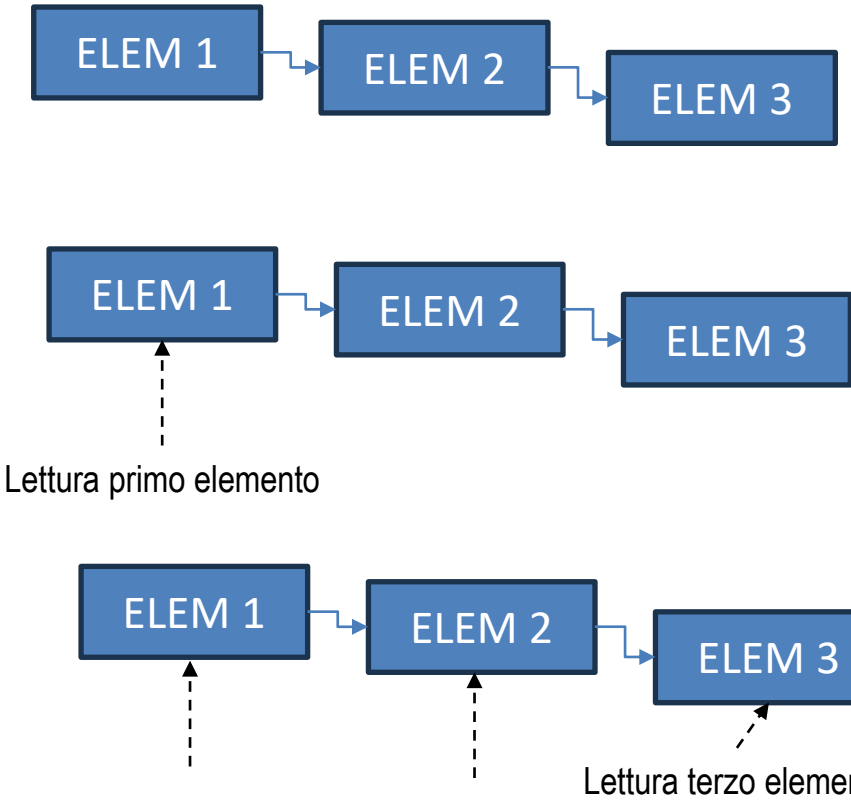
LINGUAGGIO R

LISTA: generalità

Una **Lista** è una struttura dati che denota una collezione omogenea di elementi

L'accesso a un nodo della struttura avviene direttamente a partire dal primo elemento della sequenza

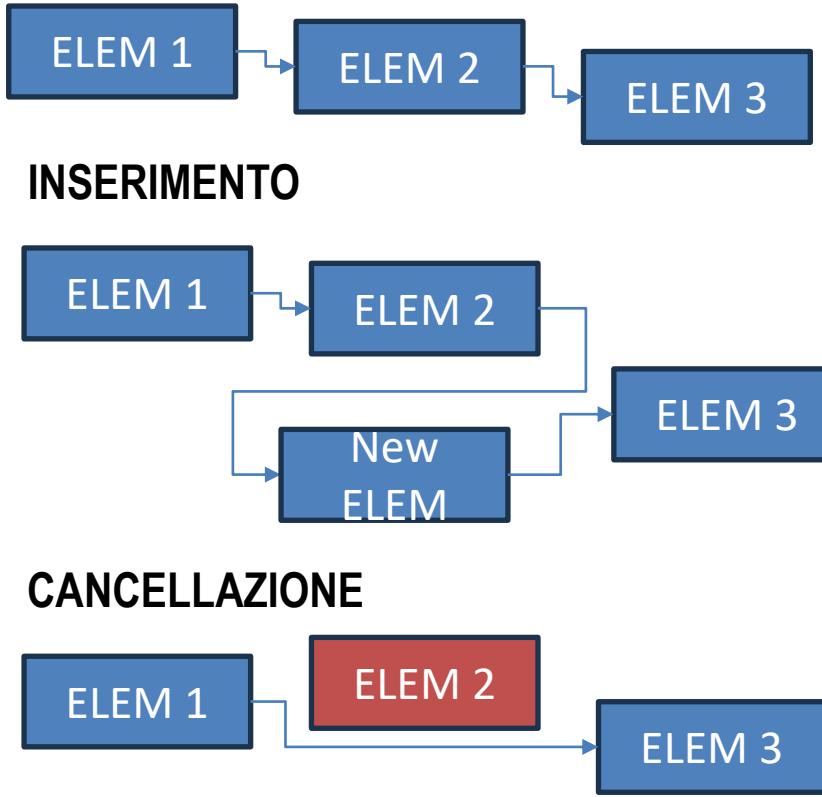
Per prelevare o ricercare un qualunque elemento, occorre scandire sequenzialmente tutti i nodi che precedono il primo



LINGUAGGIO R

LISTA: generalità

Diversamente dal vettore la lista è una struttura dati dinamica perché può cambiare di dimensione grazie alle operazioni di inserimento e di eliminazione



LINGUAGGIO R

LISTA: definizione

Nel Linguaggio R **una lista è un oggetto che contiene elementi di diversi tipi elementari nonché altri oggetti** come vettori, array, matrici e funzioni)

Una lista si definisce mediante la funzione:

list(data)

dove data sono gli elementi che costituiscono la lista

ESEMPIO

```
lista<-list("Blu","Rosso", "Verde", c(21,32,11), TRUE, 51.23, 3+5i)
print(lista)
[[1]] #Primo elemento della lista (una stringa)
[1] "Blu"
[[2]] #Secondo elemento della lista (una stringa)
[1] "Rosso"
[[3]] #Terzo elemento della lista (una stringa)
[1] "Verde"
[[4]] #Quarto elemento della lista (un vettore)
[1] 21 32 11
[[5]] #Secondo elemento della lista (un booleano)
[1] TRUE
[[6]] #Secondo elemento della lista (un numerico)
[1] 51.23
[[7]] #Settimo elemento della lista (un numero complesso)
[1] 3+5i
```

LINGUAGGIO R

LISTA: identificatori elementi

Gli elementi di una lista possono essere etichettati per migliorare la comprensione e facilitare l'accesso

Per **identificare gli elementi** della lista si può usare la funzione **names(**list**)** che va inizializzato con un vettore di stringhe

ESEMPIO

```
lista<-list(c("Margherita","Boscaiola", "Focaccia"),
matrix(1:9,ncol=3),rep(FALSE,9))
names(lista)<-c("Tipo di pizze", "Numerazione tavoli", "Prenotazione
tavoli")

print(lista)
$`Tipo di pizze`
[1] "Margherita" "Boscaiola"  "Focaccia"

$`Numerazione tavoli`
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

$`Prenotazione tavoli`
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
```

LINGUAGGIO R

LISTA: lettura e aggiornamento elemento

L'**accesso agli elementi** di una lista avviene specificando la posizione utilizzando le doppie parentesi quadre

elemento<-lista[[n]]

Post-ponendo le parentesi quadre singole, all'interno delle parentesi quadre, si può estrarre il singolo elemento dell'oggetto selezionato come elemento dalla lista

valore_elemento<-lista[[n]][m]

(prende lo m-esimo valore dell' n-esimo elemento della lista)

ESEMPIO

```
lista<-list(c("Margherita","Boscaiola", "Focaccia"),
matrix(1:9,ncol=3),rep(FALSE,9))

elem1<-lista[[1]]
print(elem1)
[1] "Margherita" "Boscaiola"  "Focaccia"

val<-lista[[1]][2]
print(val)
[1] "Boscaiola"

elem2<-lista[[3]]
print (elem2)
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
```

LINGUAGGIO R

LISTA: lettura e aggiornamento elemento

Qualora gli elementi abbiano delle etichette è **possibile prelevarne il valore specificando l'identificatore** interponendo tra il nome della lista e l'etichetta il simbolo $

elemento<-lista**$identificatore**

Post-ponendo le parentesi quadre singole, all'interno delle parentesi quadre, si può estrarre il singolo elemento dell'oggetto selezionato come elemento dalla lista

elemento<-lista**$identificatore[**m**]**

(prende lo m-esimo valore dell' elemento della lista relativo all'identificatore definito con la funzione names() *)*

ESEMPIO

```
lista<-list(c("Margherita","Boscaiola", "Focaccia"),
matrix(1:9,ncol=3),rep(FALSE,9))
names(lista)<-c("Tipo di pizze", "Numerazione tavoli", "Prenotazione
tavoli")

elem<-lista$"Tipo di pizze"
print(elem)
"Margherita" "Boscaiola"  "Focaccia"


valore<-lista$"Numerazione tavoli"[5]
print(valore)
5
```

LINGUAGGIO R

LISTA: aggiunta elemento

Nel Linguaggio R è possibile aggiungere, eleminare o aggiornare un elemento

A differenza della definizione formale di lista, il Linguaggio R consente l'**aggiunta** di un elemento solamente alla fine della lista; mentre l'aggiornamento (precisando il nuovo valore) e la cancellazione (impostando il valore NULL) può avvenire in ogni posizione

L'elemento aggiunto deve essere di tipo elementare

ESEMPIO

```
lista<-list(c("Margherita","Boscaiola", "Focaccia"),
matrix(1:9,ncol=3),rep(FALSE,9))

lista[[4]]<-"Suppli"  #Inserimento elemento nella lista

print(lista) #Stampa ultimo elemento immesso
[[1]]
[1] "Margherita" "Boscaiola"  "Focaccia"
[[2]]
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
[[3]]
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
[[4]]
[1] "Suppli"
```

LINGUAGGIO R

LISTA: aggiornamento elemento

Nel Linguaggio R è possibile aggiungere, eleminare o aggiornare un elemento

A differenza della definizione formale di lista, il Linguaggio R consente l'aggiunta di un elemento solamente alla fine della lista; mentre l'**aggiornamento** (precisando il nuovo valore) e la cancellazione (impostando il valore NULL) può avvenire in ogni posizione

L'elemento aggiunto deve essere di tipo elementare

ESEMPIO

```
lista<-list(c("Margherita","Boscaiola", "Focaccia"),
matrix(1:9,ncol=3),rep(FALSE,9))

lista[[2]]<- c("TavoloRosa ", " TavoloBianco", " TavoloVerde",
"TavoloArancione" )
#aggiornamento elemento nella lista

print(lista) #Stampa ultimo elemento immesso
[[1]]
[1] "Margherita" "Boscaiola"  "Focaccia"

[[2]]
[1] "TavoloRosa "     " TavoloBianco"   " TavoloVerde"
"TavoloArancione"

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

LINGUAGGIO R

LISTA: cancellazione elemento

Nel Linguaggio R è possibile aggiungere, eleminare o aggiornare un elemento

A differenza della definizione formale di lista, il Linguaggio R consente l'aggiunta di un elemento solamente alla fine della lista; mentre l'aggiornamento (precisando il nuovo valore) e la **cancellazione** (**impostando il valore NULL**) può avvenire in ogni posizione

L'elemento aggiunto deve essere di tipo elementare

ESEMPIO

```
lista<-list(c("Margherita","Boscaiola", "Focaccia"),
matrix(1:9,ncol=3),rep(FALSE,9))
lista[[4]]<-"Crocchetta" #Inserimento di un elemento
lista[[2]]<-NULL #Cancellazione di un elemento

print(lista)
[[1]]
[1] "Margherita" "Boscaiola"  "Focaccia"

[[2]]
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE

[[3]]
[1] "Crocchetta"
```

LINGUAGGIO R

LISTA: operazioni - unione di liste

Due liste possono essere unite mediante la funzione **c**

ESEMPIO

```
list1<-list(1,2.5,”Pluto”,4)
list2<-list("Pippo", c(TRUE,FALSE, TRUE),"Paperino")
unione_liste<-c(list1,list2)
print(unione_liste)
[[1]]
[1] 1
[[2]]
[1] 2.5
[[3]]
[1] "Pluto"
[[4]]
[1] 4
[[5]]
[1] "Pippo"
[[6]]
[1]  TRUE FALSE  TRUE
[[7]]
[1] "Paperino"
```

LINGUAGGIO R

LISTA: operazioni – da lista a vettore

Attraverso la funzione **unlist(**list**)** è possibile **convertire una lista in un vettore**

In particolare ogni sotto elemento (di ciascun è elemento della lista) diventa un elemento del vettore risultante

Se la lista è composta di soli valori numerici il risultato è un vettore numerico

ESEMPIO

```
lista<-list(c("Margherita","Boscaiola", "Focaccia"),
matrix(1:9,ncol=3),rep(FALSE,9))
vett<-unlist(lista)
print(vett)
[1] "Margherita" "Boscaiola"  "Focaccia"   "1"          "2"          "3"
"4"          "5"          "6"
[10] "7"          "8"          "9"          "FALSE"      "FALSE"      "FALSE"
"FALSE"      "FALSE"      "FALSE"
[19] "FALSE"      "FALSE"      "FALSE«
```

```
lista<-list(c(1,2,3,4,5), matrix(1:9,ncol=3),rep(5,9))
vett<-unlist(lista)
print(vett)

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

LINGUAGGIO R

LISTA: operazioni – applicazione di una funzione

Sulla lista è possibile applicare una funzione a ciascun elemento (come visto per i vettori) mediante la procedura

lapply(x, FUN = sd**)**

dove x è una lista e FUN è la subroutine che deve operare sugli elementi

ESEMPIO

```
lista<-list(c(1,2,3,4,5),c(5,6,7,8),c(10,20,30))
ris<-lapply(lista, mean)
print(ris) #stampa il valore medio di ciascun elemento della lista
[[1]]
[1] 3

[[2]]
[1] 6.5

[[3]]
[1] 20
```

Fattore/Factor

LINGUAGGIO R

FATTORE: generalità

Nel Linguaggio R il fattore è un oggetto che cataloga dati di tipo CHARACTER o NUMERIC e li classifica in livelli

Il suo impiego è molto utile quando si gestiscono collezione di dati nel quale, ad esempio, alcuni campi hanno un limitato numero di valori univoci (Uomo/Donna o Vero/Falso)

Questi casi sono spesso presenti nei modelli statistici

Fattore	Valore
Uomo	Valerio
Donna	Martina
Donna	Flavia
Donna	Benedetta
Donna	Patrizia
Uomo	Guido

LINGUAGGIO R

FATTORE: creazione

Una collezione di dati è resa un fattore attraverso la funzione **factor(vett)** che ha come argomento un vettore

ESEMPIO

```
#Creazione di un fattore di interi
vett1<-c(1,2,3,4,5,7,3,4,2,1,4,5,7)
fac1<-factor(vett1)
print(fac1)
[1] 1 2 3 4 5 7 3 4 2 1 4 5 7
Levels: 1 2 3 4 5 7


#Creazione di un fattore di reali
vett2<-c(1.6,2.4,3.3,4.1,5.5,7.6,3.6,3.8,2.9,1.5,4.3,3.3,7.6)
fac2<-factor(vett2)
print(fac2)
[1] 1.6 2.4 3.3 4.1 5.5 7.6 3.6 3.8 2.9 1.5 4.3 3.3 7.6
Levels: 1.5 1.6 2.4 2.9 3.3 3.6 3.8 4.1 4.3 5.5 7.6
```

LINGUAGGIO R

FATTORE: creazione

Una collezione di dati è resa un fattore attraverso la funzione **factor(vett)** che ha come argomento un vettore

ESEMPIO

```
#Creazione di un fattore di stringhe
vett3<-c("Pippo","Pluto","Paperino","Minnie","Qui","Qua",
"Paperino","Minnie","Qui", "Quo","Qua")
fac3<-factor(vett3)
print(fac3)
[1] Pippo    Pluto    Paperino Minnie   Qui      Qua      Paperino
Minnie   Qui      Quo      Qua
Levels: Minnie Paperino Pippo Pluto Qua Qui Quo
```

LINGUAGGIO R

FATTORE: ordinamento

Il **fattore è impiegato anche per ordinare vettori di stringhe non in ordine alfabetico**

ESEMPIO

```
mesi <- c("Dic", "Apr", "Gen", "Mar", "Feb","Giu", "Ott", "Lug", "Nov", "Mag", "Set", "Ago")
order_m<-sort(mesi)
print(order_m)
[1]"Ago", "Apr", "Dic", "Feb", "Gen", "Giu", "Lug", "Mag", "Mar", "Nov", "Ott", "Set"

#USO FATTORI
mesi <- c("Dic", "Apr", "Gen", "Mar", "Feb","Giu", "Ott", "Lug", "Nov", "Mag", "Set", "Ago")
livelli_mensili<-c("Gen","Feb","Mar", "Apr", "Mag", "Giu", "Lug", "Ago","Set", "Ott",  "Nov", "Dic")
fact_mesi<- factor(mesi, levels = livelli_mensili)
print(fact_mesi)
[1] Dic Apr Gen Mar Feb Giu Ott Lug Nov Mag Set Ago
Levels: Gen Feb Mar Apr Mag Giu Lug Ago Set Ott Nov Dic

print(sort(fact_mesi))
[1] Gen Feb Mar Apr Mag Giu Lug Ago Set Ott Nov Dic
Levels: Gen Feb Mar Apr Mag Giu Lug Ago Set Ott Nov Dic
```

LINGUAGGIO R

FATTORE: elementi non definti

L'uso dei livelli consente anche di segnalare **alcune condizioni anomale qualora uno degli elementi non sia presente tra quelli definiti nei livelli** (si ha il valore NA, cioè *Not Available*)

ESEMPIO

```
alcuni_mesi <- c("Dic", "Apr", "Ces", "Mar")
livelli_mensili<-c("Gen","Feb","Mar", "Apr", "Mag", "Giu", "Lug", "Ago","Set", "Ott",  "Nov", "Dic")
fact_mesi<- factor(alcuni_mesi, levels = livelli_mensili)
print(fact_mesi)
[1] Dic  Apr  <NA> Mar
Levels: Gen Feb Mar Apr Mag Giu Lug Ago Set Ott Nov Dic
```

Segmento Dati/ Data Frame

LINGUAGGIO R

SEGMENTO DATI: generalità

Nel Linguaggio R un **segmento di dati** (*data frame*) è una tabella nella quale ogni **colonna contiene i valori di una variabile** (attributo) e **ogni riga rappresenta un record**

Un segmento di dati non può avere una colonna vuota o priva di nome (ma può avere elementi mancanti o not available, NA)

Le righe devono essere uniche; i dati immagazzinati sono di tipo NUMERIC, FACTOR o CHARACTER

Le colonne hanno lo stesso numero di elementi

Un segmento di dati è assimilabile ad un foglio elettronico con intestazioni

Att1	Att2	Att3	Att4	Att5	Att6
23	12	74	18	6	NA
90	NA	25	2	NA	202
-56	45	NA	-76	-90	NA
NA	NA	-98	NA	89	67
NA	-67	NA	NA	34	89
NA	-88	NA	101	NA	77
36	0	89	1	7098	NA

LINGUAGGIO R

SEGMENTO DATI: definizione

La creazione di un segmento di dati avviene con la funzione **data.frame()** nel quale sono definiti gli oggetti. Ogni oggetto può essere inizializzato con un numero di valori predefinito

Le variabili di tipo stringa sono convertite nel tipo FACTOR (perché di solito sono le intestazioni che specificano la natura dei dati); per lasciare il tipo CHARACTER si setta **stringsAsFactors=FALSE**.

ESEMPIO

```
classe <- data.frame(
  id = c (1:5),
  name = c("Massimo","Roberto","Arianna","Giorgia","Simona"),
  data_nascita = as.Date(c("2000-04-13", "2000-01-01", "2000-09-
11", "2000-08-15","2000-12-25")),
  media = c(9.5,8.7,9.2,4.5,6.7),
  stringsAsFactors = FALSE
)
print(classe)
  id    name data_nascita media
1  1 Massimo  2000-04-13   9.5
2  2 Roberto   2000-01-01   8.7
3  3 Arianna   2000-09-11   9.2
4  4 Giorgia   2000-08-15   4.5
5  5  Simona   2000-12-25   6.7
```

LINGUAGGIO R

SEGMENTO DATI: prelievo e assegnamento elementi

L'**estrazione di un elemento** può avvenire specificando riga e colonna, separate da una virgola, tra parentesi quadre singole

valore<-df[*num_record*,*num_attrib*]

L'**assegnamento** ha sintassi opposta:

df[*num_record*,*num_attrib*]<-valore

È possibile anche **utilizzare dei vettori per prelevare più elementi**

ESEMPIO

```
classe <- data.frame(
  id = c (1:5),
  name = c("Massimo","Roberto","Arianna","Giorgia","Simona"),
  data_nascita = as.Date(c("2000-04-13", "2000-01-01", "2000-09-11",
"2000-08-15","2000-12-25")),
  media = c(9.5,8.7,9.2,4.5,6.7),
  stringsAsFactors = FALSE
)
val<-classe[3,2]
print(val)
"Arianna"

df1<-classe[c(1,5),c(2,4)]
print(df1)
     name media
1 Massimo   9.5
5  Simona   6.7
```

LINGUAGGIO R

SEGMENTO DATI: prelievo e assegnamento elementi (con fattori)

Quando **i nomi degli oggetti sono fattori, si può accedere ai dati interponendo, tra il nome del segmento data e quello dell'oggetto, il simbolo $**

valore<-df$fatt

L'**assegnamento** ha sintassi opposta

df$fatt<-valore

ESEMPIO

```
classe <- data.frame(
  id = c (1:5),
  name = c("Massimo","Roberto","Arianna","Giorgia","Simona"),
  data_nascita = as.Date(c("2000-04-13", "2000-01-01", "2000-09-
11", "2000-08-15","2000-12-25")),
  media = c(9.5,8.7,9.2,4.5,6.7),
  stringsAsFactors = TRUE
)
df<-data.frame(classe$name,classe$media)
print(df)
 classe.name classe.media
1     Massimo         9.5
2     Roberto         8.7
3     Arianna         9.2
4     Giorgia         4.5
5      Simona         6.7
```

LINGUAGGIO R

SEGMENTO DATI: prelievo e assegnamento elementi (con fattori)

Si può accedere al singolo dato interponendo, tra il nome del segmento data e quello dell'oggetto, il simbolo $ e specificando il numero di riga

valore <-df$fatt[*num_elem*]

L'**assegnamento** ha sintassi opposta

df$fatt[*nun_elem*]<-valore

NB: se il fatt è una matrice si può usare la notazione

df$fatt[*nun_rig,num_col*]<-valore

ESEMPIO

```
classe <- data.frame(
  id = c (1:5),
  name = c("Massimo","Roberto","Arianna","Giorgia","Simona"),
  data_nascita = as.Date(c("2000-04-13", "2000-01-01", "2000-09-11", "2000-08-
15","2000-12-25")),
  media = c(9.5,8.7,9.2,4.5,6.7),
  stringsAsFactors = FALSE
)
media=classe$media[2]
print(media)
[1] 8.7
classe$name[2]="Isabella"
print(classe)
 id     name data_nascita media
1  1  Massimo   2000-04-13   9.5
2  2 Isabella   2000-01-01   8.7
3  3  Arianna   2000-09-11   9.2
4  4  Giorgia   2000-08-15   4.5
5  5   Simona   2000-12-25   6.7
```

LINGUAGGIO R

SEGMENTO DATI: aggiunta attributo

Un **segmento di dati può essere incrementato di una o più colonne** (attributi) specificando il nome del dataframe, interponendo il simbolo $, e riportando l'etichetta (cioè il fattore) della nuova colonna

df$nuova_colonna=definizione_colonna

ESEMPIO

```
classe <- data.frame(
  id = c (1:5),
  name = c("Massimo","Roberto","Arianna","Giorgia","Simona"),
  data_nascita = as.Date(c("2000-04-13", "2000-01-01", "2000-09-11", "2000-08-15","2000-12-25")),
  media = c(9.5,8.7,9.2,4.5,6.7),
  stringsAsFactors = FALSE
)
classe$genere=c("M","M","F","F","F")
print(classe)
```

id	name	data_nascita	media	genere
1	Massimo	2000-04-13	9.5	M
2	Roberto	2000-01-01	8.7	M
3	Arianna	2000-09-11	9.2	F
4	Giorgia	2000-08-15	4.5	F
5	Simona	2000-12-25	6.7	F

LINGUAGGIO R

SEGMENTO DATI: aggiunta record 1

Un segmento di dati può essere **incrementato di una o più righe (record) usando la funzione rbind(df1,df2)** che ha come argomenti due strutture uguali

ESEMPIO

```
classe <- data.frame(
  id = c (1:5),
  name = c("Massimo","Roberto","Arianna","Giorgia","Simona"),
  data_nascita = as.Date(c("2000-04-13", "2000-01-01", "2000-09-11", "2000-
08-15","2000-12-25")),
  media = c(9.5,8.7,9.2,4.5,6.7),
  stringsAsFactors = FALSE
)

nuova_classe <- data.frame(
  id = c (6:7),
  name = c(“Isabella”,”Adriano”),
  data_nascita = as.Date(c("2000-05-05", "2000-08-08")),
  media = c(8.8,6.2),
  stringsAsFactors = FALSE
)
```

LINGUAGGIO R

SEGMENTO DATI: aggiunta record 2

Un segmento di dati può essere **incrementato di una o più righe (record) usando la funzione rbind(df1,df2)** che ha come argomenti due strutture uguali

```
classe<-rbind(classe,nuova_classe)
print(classe)
  id     name data_nascita media
1  1  Massimo   2000-04-13   9.5
2  2  Roberto   2000-01-01   8.7
3  3  Arianna   2000-09-11   9.2
4  4  Giorgia   2000-08-15   4.5
5  5   Simona   2000-12-25   6.7
6  6 Isabella   2000-05-05   8.8
7  7  Adriano   2000-08-08   6.2
```

LINGUAGGIO R

SEGMENTO DATI: informazioni

Utilizzando la funzione **summary(**df**)**, che ha come argomento df cioè un segmento di dati, si ottengono informazioni statistiche

ESEMPIO

```
print(summary(classe)
      id          name                data_nascita           media
 Min.   :1.0   Length:7            Min.   :2000-01-01    Min.   :4.500
 1st Qu.:2.5   Class :character    1st Qu.:2000-04-24    1st Qu.:6.450
 Median :4.0   Mode  :character    Median :2000-08-08    Median :8.700
 Mean   :4.0                       Mean   :2000-07-03    Mean   :7.657
 3rd Qu.:5.5                       3rd Qu.:2000-08-28    3rd Qu.:9.000
 Max.   :7.0                       Max.   :2000-12-25    Max.   :9.500
```

LINGUAGGIO R

SEGMENTO DATI: fusione segmenti 1

Due segmenti di dati con un attributo in comune possono essere uniti con la funzione **merge(**df1,df2**)**

Si conserva quello del primo dataframe

ESEMPIO

```
# Crea il primo dataframe.
nome <- c("Benigni", "Bergonzoni", "Grillo")
età <- c(70, 64, 74)
df1 <- data.frame(nome, età)
```

	nome	età
1	Benigni	70
2	Bergonzoni	64
3	Grillo	74

```
# Crea il secondo dataframe.
nome <- c("Benigni", "Bergonzoni", "Grillo")
altezza <- c(1.68, 1.86, 1.74)
df2 <- data.frame(nome, altezza)
```

	nome	altezza
1	Benigni	1.68
2	Bergonzoni	1.86
3	Grillo	1.74

LINGUAGGIO R

SEGMENTO DATI: fusione segmenti 1

Due segmenti di dati con ugual attributo di colonna con elementi in comune (non interessa l'attributo della colonna, si conserva quello del primo dataframe) possono essere uniti con la funzione **merge(**df1,df2**)**

```
#Unione dei due segmenti di dati
df3<-merge(df1,df2)
print(df3)

nome età altezza
1    Benigni  70    1.68
2 Bergonzoni  64    1.86
3     Grillo  74    1.74

df4<-merge(df2,df1)
print(df4)
nome altezza età
1    Benigni    1.68  70
2 Bergonzoni    1.86  64
3     Grillo    1.74  74
```

LINGUAGGIO R

SEGMENTO DATI: fusione segmenti 2

Due segmenti di dati con un attributo in comune ma con elementi diversi possono essere uniti con la funzione **merge(**df1,df2**)**

```
# Crea il primo dataframe.
nome <- c("Benigni", "Bergonzoni", "Grillo")
età <- c(70, 64, 74)
df1 <- data.frame(nome, età)


# Crea il secondo dataframe.
nome <- c("Benigni", "Zalone", "Grillo")
altezza <- c(1.68, 1.86, 1.74)
df2 <- data.frame(nome, altezza)
```

```
df3<-merge(df1,df2)
print(df3)
  nome età altezza
1 Benigni  70    1.68
2  Grillo  74    1.74

df4<-merge(df2,df1)
print(df3)
  nome età altezza
1 Benigni  70    1.68
2  Grillo  74    1.74
```

LINGUAGGIO R

SEGMENTO DATI: fusione segmenti 2

Se l'**attributo di colonna è uguale** e non ci sono elementi in comune il **merge(df1,df2) da un risultato vuoto**

```
# Crea il primo dataframe.
nome <- c("Ale&Franz", "Bisio", "Forest")
età <- c(70, 64, 74)
df1 <- data.frame(nome, età)


# Crea il secondo dataframe.
nome <- c("Benigni", "Zalone", "Grillo")
altezza <- c(1.68, 1.86, 1.74)
df2 <- data.frame(nome, altezza)
```

```
df3<-merge(df1,df2)
print(df3)
[1] nome    età     altezza
<0 righe> (o 0-length row.names)


df4<-merge(df2,df1)
print(df3)
[1] nome    età     altezza
<0 righe> (o 0-length row.names)
```

LINGUAGGIO R

SEGMENTO DATI: fusione segmenti 3

Se non ci sono attributi di colonna uguali il **merge(**df1,df2**)** effettua il prodotto cartesiano degli elementi

```
# Crea il primo dataframe.
comico <- c("Benigni", "Bergonzoni", "Grillo")
età <- c(70, 64, 74)
df1 <- data.frame(nome, età)


# Crea il secondo dataframe.
attore <- c("Deniro", "Pacino", "Pitt")
altezza <- c(1.68, 1.86, 1.74)
df2 <- data.frame(ident, altezza)
```

```
df3<-merge(df1,df2)
print(df3)
 nome età      ident altezza
1    Benigni  70    Deniro    1.68
2 Bergonzoni  64    Deniro    1.68
3     Grillo  74    Deniro   1.68
4    Benigni  70 Pacino    1.86
5 Bergonzoni  64 Pacino    1.86
6     Grillo  74 Pacino    1.86
7    Benigni  70     Pitt    1.74
8 Bergonzoni  64     Pitt    1.74
9     Grillo  74     Pitt    1.74
```


Fine