VETTORI:
CREAZIONE:

```
          x<- c(1,2,3)              #crea e assegna 3 elementi a x
          x<- c("A","B")            #crea e assegna due caratteri a x
          x<- c("casa","blu")       #crea e assegna due nomi
          x<- c(1:10)               #genera vettore ordinato da 1 a 10
          x<- c(10:1)               #genera vettore ordinato da 10 a 1
          x<- seq(1,10)             #sequenza ordinata
                                      seq( from= 1, to =5, by= 0,5)
          x<- rep( c(1,2), times=4) # ripete 1,2 in successione per 4 volte
          x<- c(vet,NA,NA)          #aggiungere in coda ai valori nel vettore
vet 2 NA
```

le operazioni aritmetiche vengono eseguite membro a membro.
se in un vettore si inseriscono tipi ETEROGENEI di variabili, si tende a tratare tutti come CARATTERI

```
es: x<- c(1,2,3)
    y<- c(4,5,6)

    z<- x+y   o  x-y  o  x*y  o   x/y
   [z] 5 7 9
```

FUNZIONI SU VETTORI NUMERICI:

```
            sd(vet)          #calcolo la deviazione standard
          mean(vet)          #media degli elementi delvettore
          var (vet)          #varianza elementi vettore
          max (vet)          #valore massimo nel vettore
          min (vet)          #valore minimo nel vettore
         range(vet)          #minimo e massimo valore nel vettore
          sum (vet)          #somma tutti gli elementi vettore
         sqrt (vet)          #calcolo la radice
         prod (vet)          #prodotto di tutti gli elementi del vettore
        rnorm (NUM)          #generazione tot numeri casuali da 0 a 1 di una
distribuzione normale
                             rnorm (c(num), mean=num, sd=num)
        runif (NUM)          #genero tot numeri casuali da una distribuzione
uniforme
                              runif(c(num), min=num, max=num)
         sort (vet)          #ordinamento ELEMENTI vettore
        order (vet)          #ordinamento per INDICI DI POSIZIONE
        is.na (vet)          #restituisce un vettore di operatori logici e
individua tutti gli NA
       length (vet)          #lunghezza di un vettore
        names (vet)          #da i nomi a ciascun elemento del vettore vet
        round (vet)          #arrotonda alla prina cifra (se non meglio
specificato) i valori contenuti in vet
        table (vet)          #restituisce per ogni valore contenuto nel vattore
il numero di volte che esso compare,utile per frequenza.
                              es: table(x)/ length(x) *100 = frequenza relativa
percentuale
         abs (vet)           #restituisce il valore assoluto di tutti i numeri
del vettore vet
```

OPERATORI LOGICI:

Mi restituisce un vettore con FLASE e TRUE a seconda del verificarsi o meno della condizione

```
          >                    #maggiore
          <                    #minore
```

```
          >=                    #maggiore uguale
          <=                    #minore uguale
          ==                    #uguale
          !=                    #diverso
          &                     #and logico
          |                     #or logico
```

ACCESSO A ELEMENTI DEL VETTORE:

L'accesso a singoli elementi in un vettore avviene tramite []

```
           x[n]                   #restituisce il valore in posizione n
          x[-n]                   #restituisce tutti i valori tranne quello in
posizione n
           x[i]                   #da i che Ã¨ il vettore logico estrae da x
tutt i valore che in i corrispondono a TRUE
  x[c("num","num")]               #seleziona da x tutti i valori corrispondenti
alla categoria denominata come num e num
```

CONTROLLO DEL FLUSSO DI ESECUZIONE IN R:

Solitamente in R i programmi son eseguiti SEQUENZIALMENTE, a volte, attraverso uso di strutture di controllo Ã¨ possibile fargli scegliere VIE ALTERNATIVE.

In R son presenti strutture di controllo specifiche:
1) Blocchi di istruzioni
2) Istruzioni condizionate
3) istruzioni di looping

RAPPRESENTAZIONE BLOCCHI:

Per rappresentare blocchi di istruzioni bisogna racchiudere le istruzioni tra PARENTESI GRAFFE
Il blocco vene valutato solo quando vengono chiuse le parentesi.
es:

```
{

 x <- runif(10);
 y <- runif(10);
mx <- mean(x);
my <- mean(y);
vx <- sd(x);
vy <- sd(y);
 g <- (mx-my)/(vx+vy);
 g;

}
```

ISTRUZIONI CONDIZIONALI:

```
           if (condizione)           *  #dipendendo dal verificarsi o meno di
una condizione il programma prende due decisioni diverse,con conseguenze con il
risultato diverse.
             {blocco 1}
          else(condizione)              #il ramo else puÃ² essere assente in
questo caso una condizione viene eseguita solo quando si verifica una
condizione.
             {blocco 2}


es:
if (x<=0)
{
```

```
y <- x^2;
z <- log2(1+y);                          #ATTENZIONE AL PUNTO E VIRGOLA
}
else
z <-log2(x);                             #se la condizione Ã¨ solo una posso
omtttere le graffe
```

NB: * bisogna prestare molta attenzione alle condizioni che si scelgono ed Ã¨ importante, per usare queste strutture, imporre condizioni.

L'uso di questo controlo Ã¨ comodo perchÃ¨ possono essere anche ANNIDATE, avere cioÃ¨ un if dentro un if o dentro un else

switch(istruzione, lista di...) #consente di scegliere fra OPZIONI MULTIPLE quella che rispetta la condizione.
praticamente il valore assegnato all'istruzione viene confrontato con la lunghezza della lista: se la lista Ã¨ abbastanza lunga da contenere il valore di "istruzione" allora il programma restituisce come risposta l'oggetto contenuto alla posizione che corrisponde al valore di istruzione

es:

```
x <- 3
switch(x, 2+2, mean(1:100), rnorm(3))
[1] -0.3393166  0.1595591 -0.2016252

y <- "frutto"                                                    #se
l'espressione corrisponde a un vettore di caratteri allora lo switch restituisce il valore associato a quel carattere nella lista.
switch(y, frutto="pera", ortaggio="cavolo", legume="fagiolo")
[1] "pera"
```

ISTRUZIONI LOOP:

Sono tipi di strutture, caratterizzati da BLOCCHI DI ISTRUZIONI che ciclano fino a quando una condizione Ã¨ soddisfatta.

1) for
2) while
3) repeat

for(nome in v) #v contiene gli elementi di un vettore che vengono valutati uno ad uno fino all'esaurimento.
{blocco istruzioni}

es:

```
v<- round(runif(50)*5)
for(i in 1:5) cat(v[i]," ")
```

while(condizione)
{blocco istruzioni} #se la valutazione della condizione restituisce un valore TRUE allora si esegue il blocco istruzioni altrimenti si esce dal ciclo.

NB: Ã¨ bene prestare attenzione alla condisione perchÃ¨ si potrebbero generare ccli infiniti.

es: while(TRUE)

es1:
```
f <-function(y)
{ i <- 0;
  while (y > 1)
   {
     y <- y/2;
     i <- i + 1;
   }
}
```

```
                        repeat
                             {blocco istruzioni}     #il blocco di istruzioni
```
viene eseguito all'infinito fino a quando non si incontra l'istruzione break che frza l'uscita.

I BREAK possono essere piÃ¹ si uno se nel caso vogliamo associare l'uscita dal blocco a piÃ¹ condizioni

es:
```
f<- function(y)
{
  i<- 0                 #inizializzo i
  repeat
   {
     if(y<= 1)
      break;
      y<- y/2;
      i<- i+1;
   }
  i                    #restituisce il valore di i che indica quante volte ha
```
dovuto ciclare prima del raggiungimento della condizione.
```
}
```

FAMIGLIA DI COMANDI "APPLY":

Iterano funzioni specifiche su insiemi di oggetti

```
                          (a-s-l)_apply(insieme_oggetti)         #a,s,l sono
```
delle lettere che a second adella struttura o necessitÃ si antepongono a aplly.

funziona in qunto il data frame ha una dimensione

MATRICI

sono estensioni bidimensionali o miltidimensionali, per accedere agli elementi di queste strutture son necessari tanti indici quante son le DIMENSIONI della struttura.

nelle matrici il PRIMO elemento identifica la RIGA e il secondo la COLONNA
es: m[r,c]

Ã¨ bene notare che anche per le matrici esiste la regola del riciclo, se abbiamo numeri insufficienti il programma gira ricominciando da capo

CREAZIONE MATRICI:

```
              x<- 1:24                          #posso costruire matrici
partendo da vettori e imponendo un numero di righe o colonne a mia scelta
              m<- matrix(x, nrow=4)              l'importante Ã¨ che siano
divisibili per il numero di elementidel vettore
              m<- matrix(x, ncol=4)
              m<- matrix(x,ncol=4,byrow=TRUE)    #riempie una matrice con 4
colonne seguendo la successione per righe



FUNZIONI:
              array( vet, c(r,c,))           #creazione matrice con la funzione
array, al primo posto inserisco i valori, al secondo le dimensioni (prima righe
e poi colonne)
              matrix (nrow=num, ncol=num)    #creazione matrici
              mode(m)                        #da quali elementi Ã¨ formato
               dim(m)                        #dimensioni della matrice (n righe
e colonne)
            length(m)                        #numero di elementi della matrice
       *     cbind(x,y)                      #costruisce una matrice legamÃ¬ndo
ORIZZONTALMENRE vettori (i vettori diventano colonne)
       *     rbind(x,y)                      #costruzione matrice legando
VERTICALMENTE vettori (i vettori diventano righe)
                t(m)                         #matrice trasposta (le colonne
diventan righe e le righe colonn)
              diag(m)
             solve(m)                        #risolve sistema lineare
          dinames(m)                         #per dare un nome alle componenti
della matrice
                                             <- list(c("r1","r2"),
paste("c",1:n, sep="")) sto assegnando a ciascuna riga il nome r1 e r2 e a ogni
colonna il nome c da 1 a n per il num di colonne


*NB: se i vettori non son della stessa lunghezza si applica la regola del
riciclo

OPERAZIONI:
                m+n          #somma membro a membro delle variabili nella
stessa posizione
                m+2          #somma ogni elemento alla costante
                m*n          #prodotto elemento per elemento (con stessi
indici)
                m*2          #moltiplica ogni elemento per la costante
               m%*%n         #prodotto riga per colonna il numero delle
COLONNE di m deve essere uguale al numero di RIGHE di n

ACCESSO ELEMENTI MATRICE:

In questo caso Ã¨ necessario indicare un indice per ogni dimensione della
struttura di dati

                m[1,2]       #estrae l'elemento alla prima RIGA e seconda
COLONNA
                m[1,2:4]     #estrae elementi prima RIGA e colonne dalla 2
alla 4
                m[,1]        #tutti gli elementi della prima COLONNA
                m[1,]        #tutti gli elementi della prima RIGA
                m[,c(1,2,3)] #tutti gli elementi delle colonne 1,2,3
                m[c81,2,3),] #tutti gli elementi delle righe 1,2,3
                m[,-1]       #tutti gli elementi esclusi quelli della prima
colonna
                m[m>3]       #si ottiene un vettore in cui tutti gli elementi
```

rispettano la condizione

LISTE:

Rappresentano un insieme ORDINATO di OGGETTI ETEROGENEI.

es: vet numerico, vet logico, matrice, lista ....

Ã¨ una struttura dati ricorsiva in quanto al suo interno puÃ² contenere un'altra lista.

CREZIONE

```
x<- list( TRUE, c(1,2,3))          #creazione di una lista formata da vettore logico e numerico
x<- list( val=TRUE, vec=c(1,2,3))  #assegno al nominativo val il valore di TRUR e a quello vec il vettore numerico
```

FUNZIONI

```
x[4]<- list(TRUE)                  #agggiungo alla lista x in posizione 4 una nuova struttura del tipo lista che Ã¨ un vettore logico
x123<- c(li1,li2,li3)              #possibile concatenare liste, formare una lista a partire da liste piÃ¹ piccole
```

ACCESSO

```
li[[val]]            #accesso atomico alla componente della lista
* li$val             #accesso atomico alla componente della lista
li[val]              #l'elemento non Ã¨ trattato come vettore ma una sottolista della lista dalla quale proviene
li[["m"]]            #accesso a un elemento della lista tramite indice a caratteri
li$v[li$v=="A"]      #estrarre da una lista tutte le componenti di un vettore della lista uguali a "A"
```

NB: * il carattere $ consete di accedere poi usando la notazione[] a un particolare elemento della struttura che si manipola estratta dalla lista

es: li$vector[4] #dalla lista "li" estraggo atomicamente il vettore "vector", e da questo estraggo l'elemento con indice uguale a 4

DATA FRAME:

Sono strutture dati che possono contenere variabili eterogenee, ma in questo caso le variabili sono inserite per colonna.
Ogni data frame present
a puÃ² presentare su ciascuna colonna dati appartenenti a tipi di strutture e di formato duversi

CREAZIONE:

```
*  x<- data.frame( c(1:9),c(2:10),paste("A",1:9, sep=""))
```

#creazione di un data frame composto da tre colonne, due che corrispondono ai vettori e una che Ã¨ un vettore di caratteri.

NB: *Il data fraem essendo visualizzabile coem una matrice Ã¨ soggetto alle stesse problemanÃ¬tiche di essa, dunque necessita che ci siano , alla sua creazione , lo stesso numero di elementi in tutte le strutture inserite (STESSO NUMERO DI RIGHE)

es: daf4<-data.frame(c(1:9),c(2:12)Error in data.frame(m1, v1) : arguments imply DIFFERING NUMBER OF ROWS

ACCESSO:

Possiamo accedere alle componenti di una matrice taremite accessi simili a quelli per le matrici o le liste.

```
da.fr[[1]]              #accesso atomico alla struttura presente in prima colonna
da.fr[["v]]             #accesso atomico PER NOME agli elementi del data frame contenuti nella colonna diÃ¬enominata "v"
da.fr["v"]              #accesso non atomico alla struttura contenita nella colonna denominata "v", in questo caso ottengo una struttura di tipo DATA.FRAME
da.fr[1]                #accesso non atomico agli elementi nella prima colonna
da.fr[[1,2]]            #accesso atomico alla componente del data frame che corrisponde alla prima riga e seconda colonna
da.fr[2:4,1:2]          #accesso non atmico
da.fr[[3,]]             #accessoa tomico alle componenti in terza riga in modo atomico
 * da.fr[da.fr$v condizione]          #accesso atomico con condizione
** subset(da.fr,v condizione)         #acceeso al data frame in modo atomico
subset(da.fr,v %in% condizione)      #selezionare gli elementi da un insieme.
                                      es: subset(da.fr,v %in% c("A","T")) seleziona tutte le righe in cui ci sono tutte le lettre "A" e "T".
as.data.frame(lista)                 #trasforma una lista in un data.frame ATTENZIONEil numero di righe delle componenti della lista deve coincidere altrimenti o non viene trasformata oppure c'e risorsivitÃ .
summary(.....)                       #ricava le informazioni statistiche riguardanti un dato dato fornito
                                       * se viene usato con subset cos' restituisce il numero di tipi diversi di quella variabile contenuta in quella colonna
                                      ** se viene usato cosÃ¬ invece restituisce tutte le info riguardanti tutti i valori delle righe che sono associate al verificarsi di quella condizione.
```

NB: ATTENZIONE al numero di componeneti, se una componenete a piÃ¹ righe si rischia di avere RICICLAGGIO

FATTORI

Sono strutture dati per variabili qualitative o categoriche che solitamente son usate come etichette.

CREAZIONE

```
x<- factor(c("str"))      #crea un fattore con un elemento di nome str
```

FUNZIONI

```
levels(fact)              #visualizza il numero di "livelli" del fattore, ovvero il numero di eticette
str(fact)                 #visualizza la struttura dell'elemento fator
summary(fact)             #visualizza una tabella composta dalla frequenza dei livelli del fattore
```

vedi bene

PROGRAMMI E FUNZIONI:

R per eseguire i programmi o fa eseguire l'istruzione direttamente dal programma, oppure legge le istruzioni tramite la funzione SOURCE ed esegue.

CREAZIONE:

```
                  function( argomento )                    #l'argomento Ã¨ una sorta di lista di argomenti separati da virgola (arg1,arg2...), mentre il corpo sono le espressioni valide in R
                          { corpo funzione }               l'argomento puÃ² contenere valori di default, che vengonoinseriti a meno che non vengono scelti dal programmatore
```

es: funzione statistica Golub:

```
golub<- function(x,y)                      #gli aromenti nel programma possono non esser messi in ordine se assegnati per nome es: (y=b,x=a)
{
  mx<- mean(x);                            #mx Ã¨ una VARIABILE LIBER in quanto non Ã¨                                          ne variabile formale ne lovcale e la sua funzione rimane correlata all'esistenza del corpo della funzione
  my<- mean(y);
  vx<- sd(x);
  vy<- sd(y);
   g<- (mx-my)/(vx-vy);
   return(g);                                #quello che la funzione restutuisce Ã¨ il valore di g, se non ci fosse qualunque variabile inserita nella funzione non modificherebbe EFFETTIVAMENTE nel programma il suo valore
}


source("golub.R")                           #attingo al file memorizzato contenete il codice della funzioneprecedentemente salvato
a<- runif(5)                                #creo un vettore con 5 elementi
b<- runif(5)
golub(a,b)                                  #calcolo golub con questi 2 vettori
```

NB: x,y son ARGOMENTI FORMALI mentre a,b son ARGOMENTI ATTUALI, i primi si usano nella definizione della funzione, i secondi son quelli che portano i dati veri e propri.

```
x<<- z-1                                    #all'interno di un prgramma la variabile di SUPERASSEGNAMENTO fa variare il parametro a livello superiore
```

la PROGRAMMAZIONE TOP DOWN Ã¨ un tipo di programmazione modulare e risulta assai facile con tali strutture in quanto le funzioni possono richiamarsi tra loro.