Università degli Studi di Milano

Laurea Specialistica in Genomica Funzionale e Bioinformatica

Corso di Linguaggi di Programmazione per la Bioinformatica

Strutture di controllo

Giorgio Valentini
e –mail: *valentini@dsi.unimi.it*

DSI – Dipartimento di Scienze dell' Informazione
Università degli Studi di Milano

1

Programmi in R come valutazione sequenziale di istruzioni

```
golub <- function(x,y)  {
  mx <- mean(x);
  my <- mean(y);
  vx <- sd(x);
  vy <- sd(y);
  g <- (mx-my)/(vx+vy);
  g
}
```

- Ogni istruzione è valutata in sequenza.
- Le singole istruzioni possono essere separate da ";" o <new line>

2

Controllo del flusso di esecuzione di un programma

- I programmi sono eseguiti sequenzialmente, istruzione dopo istruzione, ma in alcuni casi il *flusso di esecuzione* può scegliere vie alternative o ripetersi ciclicamente.
- In R esistono strutture di controllo specifiche per regolare il flusso di esecuzione di un programma:
  - *Blocchi di istruzioni*
  - *Istruzioni condizionali*
  - *Istruzioni di looping*

3

Sequenze e blocchi di istruzioni - 1

- Le sequenze di istruzioni possono essere separate da <new line> o ";":

```
> 3+2- # istruz. Terminata da new line
+ 1     # l'interprete avverte che l' istruz. non è
# sintatticamente completa
[1] 4
> 3+2-;  # il ";" forza la valutazione dell'
# espressione
Error: syntax error
```

4

Sequenze e blocchi di istruzioni - 2

- Le istruzioni possono essere raggruppate insieme utilizzando le parentesi graffe. Una sequenza di istruzioni fra parentesi graffe costituisce un blocco.

```
Esempio:
> { x<-0;
+   y<-1;
+   x+y
+   x*y
+   z<-x*y;
+   z-1;
+ }
[1] -1
```

- Si noti che i blocchi vengono valutati solo dopo la chiusura delle parentesi graffe.
- Si può pensare ad un blocco come ad un' unica macro istruzione costitutita da una sequenza di istruzioni

Istruzioni condizionali: l' istruzione if ... else

L' istruzione if ... else permette *flussi alternativi di esecuzione* dipendenti dalla valutazione di una *condizione logica*.

Sintassi:

```
if (condizione)
     blocco1
else
     blocco2
```

Semantica:

se la condizione è vera viene eseguito il `blocco1` altrimenti viene eseguito il `blocco2`.

If..else: esempi

Es.1:

```
if (x>=0)
  print("x è positivo")
else
  print("x è negativo")
```

Es.2:

```
if (x<=0) {
  y <- x^2;
  z <- log2(1+y);
}
else
    z <-log2(x);
```

Es.3:

Il ramo else può anche essere assente:

```
if (x<0)
  x <- -x;
sqrt(x)
```

L' istruzione `sqrt(x)` viene sempre eseguita, mentre `x<--x` viene eseguita solo se `x` è negativo.

Es.4:

Se la condizione assume un valore numerico:

```
if (x)
  print("x è diverso da 0")
else
  print("x è uguale a 0")
```

7

Istruzione if..else innestate

Le istruzioni if..else possono essere innestate:

```
if (condizione1)
  blocco1
else if (condizione2)
  blocco2
...
else if (condizioneN)
  bloccoN
else
  bloccoN+1
```

8

La funzione ifelse

L' istruzione ifelse implementa una versione vettorizzata dell' istruzione if..else

Sintassi: `ifelse (condizione, a, b)`

`Condizione`, `a` e `b` sono vettori.

Semantica: ritorna un vettore i cui elementi sono `a[i]` se la `condizione[i]` è vera, altrimenti `b[i]` .

Es:

```
> v1 <- round(runif(5)*10)
> v1
[1]  5  2  1  0 10
> v2 <- round(runif(5)*10)
> v2
[1] 8 7 7 6 5
> ifelse(v1>v2,v1,v2)
[1]  8  7  7  6 10
```

9

La funzione switch

- La *funzione* switch consente di scegliere fra opzioni multiple.
- La sua semantica è simile a quella dell' omonima struttura di controllo di altri linguaggi di programmazione.

Sintassi:

`switch (istruzione, lista)`

Semantica:

Viene valutata `istruzione` e viene ritornato un `valore`. Se `valore` è un numero compreso fra 1 e lunghezza della lista, allora viene valutato il corrispondente elemento della lista e viene ritornato un risultato. Se `valore` è troppo grande o troppo piccolo viene ritornato NULL.

10

La funzione switch: esempi

```
> x <- 3
> switch(x, 2+2, mean(1:100), rnorm(3))
[1] -0.3393166  0.1595591 -0.2016252
> x <- 2
> switch(x, 2+2, mean(1:100), rnorm(3))
[1] 50.5
> x <- 5
> switch(x, 2+2, mean(1:100), rnorm(3))
NULL
```

Se in `switch (espressione, lista con nomi)`
la valutazione di `espressione` ritorna un vettore di caratteri che corrisponde al nome associato ad un elemento della lista, tale elemento viene valutato.

Esempio:

```
> y <- "frutto"
> switch(y, frutto="pera", ortaggio="cavolo",
   legume="fagiolo")
[1] "pera"
```

11

La funzione switch per eseguire funzioni a "scelta multipla"

```
centro <- function (x, tipo="media") {
 switch(tipo,
        media = mean(x),
        mediana = median(x),
        media_c = mean(x, trim=0.2))
}
```

```
> x<-c(1:5,seq(6,10,by=0.5)); x
 [1]  1.0  2.0  3.0  4.0  5.0  6.0  6.5  7.0  7.5  8.0  8.5  9.0
9.5 10.0
> centro(x)
[1] 6.214286
> centro(x,"mediana")
[1] 6.75
```

12

Istruzioni di loop

- Permettono di ripetere ciclicamente blocchi di istruzioni per un numero prefissato di volte o fino a che una determinata condizione logica viene soddisfatta
- Sono istruzioni la cui struttura sintattica è del tipo:
  loop `{ blocco di istruzioni}`
- Esistono diverse forme di istruzioni di loop:
  1. `for`
  2. `while`
  3. `repeat`

13

Istruzione for

Sintassi:

for (*nome* in *v*)
  blocco di istruzioni

v può essere un vettore o una lista

Semantica:

Gli elementi di *v* sono assegnati ad uno ad uno alla variabile *nome* ed il *blocco di istruzioni* viene valutato ciclicamente fino a che non sono stati esauriti tutti gli elementi di *v*.

14

Istruzione for: esempi

```
> v = round(runif(50)*5)
> for (i in 1:5) cat(v[i],"  ")
4   4   5   2   3
> for( i in (1: 10)* 5)  cat(v[i],"  ")
3   5   2   5   2   1   1   3   4   0
> for( j in c( 3,1,4,1,5,9,2,7))  cat(v[j],"  ")
5   4   2   4   3   3   4   1
```

L' istruzione *for* può ciclare su qualsiasi tipo di sequenza:

- Es: accedere in sequenza alle componenti di un data frame

```
> for( var in names(data)) {... ; comp<- data$var; ... }
```

- Es: accedere in sequenza a funzioni diverse:

```
> x <- c(pi, pi/2, pi/4) # pi corrisponde a π
> for( f in c(sin, cos, tan)) print(f(x))
[1] 1.224606e-16 1.000000e+00 7.071068e-01
[1] -1.000000e+00  6.123032e-17  7.071068e-01
[1] -1.224606e-16  1.633178e+16  1.000000e+00
```

15

Istruzione while

Sintassi:

while (*condizione*)
  blocco di istruzioni

condizione è un' espressione logica

Semantica:

condizione viene valutata: se il suo valore è TRUE allora viene eseguito il *blocco di istruzioni*.
Il blocco di istruzioni continua ad essere eseguito ciclicamente se *condizione* rimane TRUE.
Quando *condizione* diventa FALSE allora si esce dal ciclo.

16

Istruzione while - esempi

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

```
> f(1)
[1] 0
> f(2)
[1] 1
```

```
> f(10)
[1] 4
> f(1000)
[1] 10
```

```
> i<-1; while (a[i] < 0) i <- i+1;
```

Ciclo infinito:

```
while (TRUE) {... }
```

Ricerca della prima occorrenza di "UAG" nel vettore di caratteri d:

```
> i<-1; while (d[i] != "UAG" & i <=length(d)) i <- i+1;
```

17

Istruzione repeat

Sintassi:

repeat
  blocco di istruzioni

Semantica:

blocco di istruzioni viene eseguito ciclicamente all' infinito a meno che non venga incontrata una istruzione break che forzi l' uscita dal loop

18

Istruzione repeat - esempi

```
f1 <-function(y) {
  i <- 0;
  repeat {
    if (y<=1)
      break;
    y <- y/2;
    i <- i + 1;
  }
  i
}
```

La funzione `f1` è semanticamente equivalente alla funzione `f` precedentemente vista negli esempi per l' istruzione *while*

Ciclo infinito:

```
repeat {... }
```

Si possono prevedere anche più punti di uscita da un repeat:

```
repeat {
   if (a[i]> 0.1) break;
   i<-i+1;
   ...
   if (i>length(a)) break;
   ...
}
```

punti di uscita dal loop

Istruzioni per forzare il ciclo di escuzione dei loop

- L'istruzione break forza l' uscita dai loop:

```
findcodon <-function(s,codon) {
 i<-1;
 while (s[i]!=codon) {
   if (s[i] == "UAA" | s[i] == "UAG" | s[i] == "UGA")  {
     print("Stop codon found");
     break;
   }
   i <- i+1;
 }
 cat("Codon ", s[i], " found in position ",i,"\n");
}
```

- L' istruzione next forza il flusso di esecuzione direttamente al ciclo successivo:

```
for (i in 1:length(a)) {
 if (a[i] == 0)
    next;
 b[i] <- b[i]/a[i];
}
```

Iterazioni e operazioni/funzioni vettorizzate -1

- Molte operazioni e funzioni in R sono *vettorizzate* ed operano elemento per elemento su interi oggetti.
- Utilizzare direttamente operazioni o funzioni vettorizzate *è più efficiente* che effettuare le medesime operazioni utilizzando cicli for.

Esempio: prodotto scalare di due vettori:

```
> a <-rnorm(5); b<-runif(5)
```

A. Calcolo con cicli for:

```
> for(i in 1:5) d[i]<- a[i]* b[i];
> s <-0; for(i in 1:5) s<-s+d[i];
```

B. Calcolo con funzioni vettorizzate:

```
s <- sum(a*b)
```

21

Iterazioni e operazioni/funzioni vettorizzate -2

- Esistono almeno due buone ragioni per rimpiazzare (dove sia possibile) i cicli for con funzioni/operazioni vettorizzate:
  1. *La velocità*: il loop `for` è molto più lento perchè deve essere valutato ogni volta dall' interprete
  2. *La chiarezza*: è molto più semplice e sintetica l' espressione `sum(a*b)` piuttosto di una serie di cicli `for`.
- Le funzioni vettorizzate includono:
  1. Gli operatori & , | , ! , + , -, * , / , ^ , %%
  2. Funzioni matematiche. Ad es: sin, cos, log, pnorm, choose
  3. Generatori di numeri casuali: rnorm, runif, rpois, ...
  4. L'istruzione ifelse per la valutazione vettorizzata di condizioni logiche.

22

I comandi "ciclici" della famiglia apply

- I *comandi della famiglia apply* iterano una funzione specificata su insiemi di oggetti.
- La loro sintassi generale è del tipo:
  `comando_apply (insieme_di_oggetti, f)`
  La funzione `f` viene applicata ciclicamente a ciascun oggetto contenuto nell' `insieme_di_oggetti.`
- Sono semanticamente equivalenti ad un ciclo for del tipo:

```
for (i in insieme_di_oggetti)
    f(insieme_di_oggetti[i])
```

- In generale la loro esecuzione è più efficiente del corrispondente ciclo for.
- Ne esistono diverse varianti (si veda l' help in linea):
  lapply ed sapply si applicano a liste; apply si applica ad array; tapply si usa con fattori.

23

Esercizi

1. Scrivere una funzione *find_value* che ritorni la prima occorrenza (cioè i corrispondenti indici numerici) di un valore arbitrario *val* in una matrice generica *m*.
2. Scrivere una funzione *find_all_values* che ritorni tutte le occorrenze (cioè i corrispondenti indici numerici) di un valore arbitrario *val* in una matrice generica *m*.
3. Scrivere una funzione *translate* che riceva in ingresso un vettore numerico *num* e ritorni in uscita un vettore a caratteri *out* tale che *out[i]="P"* se *num[i]>0*, altrimenti *out[i]="N"*.
4. Scrivere una funzione *op* che esegua una sola fra le operazioni di somma, sottrazione, divisione o moltiplicazione dei suoi argomenti, a seconda di un opportuno parametro scelto dall' utente.
5. Tradurre la funzione *switch* (istruzione, lista) nella corrispondente serie di istruzioni if..else. Implementare una funzione *myswitch*(istruzione,lista) semanticamente equivalente alla funzione R switch.
6. Scrivere una funzione *calc_mean_var* che calcoli la media e la varianza di una lista di vettori ricevuti in ingresso.

24