Laboratorio di Statistica e Analisi Dati: Lezione 4

Tommaso C. & Marco G.

16 - 18 Novembre 2016

Controllo del flusso di esecuzione di un programma

- In R esistono strutture di controllo specifiche per regolare il flusso di esecuzione di un programma:
  - *Blocchi di istruzioni*
  - *Istruzioni condizionali*
  - *Istruzioni di looping*

Sequenze e blocchi di istruzioni

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

- Un blocco non viene quasi mai usato da solo, ma in combinazione con un'altra istruzione

```
{
x <- 4+2
z <- 4
x+z
x-3
}
## [1] 3
```

Sequenze e blocchi di istruzioni

```
{
x <- 4+2
z <- 4
x+z
x-3
}
## [1] 3
```

Differenze ?

```
x <- 4+2
z <- 4
x+z
## [1] 10
x-3
## [1] 3
```

Si, la valutazione delle singole istruzioni nel caso dei blocchi è di tipo lazy (rimandata al termine del blocco), questo comporta, per esempio, che

l’output dell’operazione `x+z` non sia visibile.

Istruzioni condizionali (IF)

```
x <- 6
if ( x %% 3 == 0 ){
  x <- x + 6
  x - 3
} else {
  x + 3
}
## [1] 9
```

Istruzioni condizionali (IF) (cont.)

Sintassi:

- Se l'istruzione in uno dei blocchi è unica le parentesi graffe possono essere omesse.
- Posso omettere il ramo else

```
{
x <- 120
if (x >= 100)
  x <- x + 4
else
  x <- x - 200
if(x<0)
  x <- x + 2000
x
}
## [1] 124
```

N.B. Questo tipo di sintassi però è valida solo all'interno di un blocco

(potete provare ad incollare il codice senza le parentesi graffe)

Istruzioni di Loop

Esistono diverse forme di istruzioni di loop:

- for
- while
- repeat

Tutte permettono di ripetere un blocco di istruzioni

For

Sintassi:

```
v = c(2,5,7,8)
for (i in v){
  print(i)
}
## [1] 2
## [1] 5
## [1] 7
## [1] 8
```

N.B. La semantica di questo comando è pari al foreach di java, quindi di fatto non avete una variabile `i` come contatore che si incrementa ad ogni ripetizione

For (cont.)

Su cosa posso iterare?

- Si può iterare su una lista di funzioni di R, quello che ottengo è la valutazione di tutte le funzioni
- Si può iterare su una lista di componenti di un dataframe

```
valore = 24
funzioni = list(sqrt, sum1, log, exp)
for (f in funzioni){
  print(f(valore))
}
## [1] 4.898979
## [1] 25
## [1] 3.178054
## [1] 26489122130
```

While

Sintassi:

```
i <- 0
while (i < 2){
  print (i)
  i<-i+1
}
## [1] 0
## [1] 1
```

Repeat

Sintassi:

```
i <- 0
repeat{
  print (i)
  i <- i + 1
  if (i >= 2) break;
}
## [1] 0
## [1] 1
```

N.B. La differenza principale con `while` è che posso eseguire del codice prima di valutare la condizione (`do while`)

Cicli e prestazioni

- Molte delle funzioni di `R` su vettori operareno elemento per elemento
- Utilizzare direttamente operazioni o funzioni vettorizzate è più efficiente che effettuare le medesime operazioni utilizzando cicli `for`

Esempio calcolare il vettore sommma tra due vettori `v` e `w`. Useremo `system.time()` per calcolare il tempo di esecuzione

```
v <- sample (1:400, 100000, replace = TRUE)
w <- sample (400:800, 10000, replace = TRUE )
system.time(for(e in 1:length(w)) v[e] <- v[e] + w[e] )
##    user  system elapsed
##   0.020   0.003   0.023
system.time(v <- v + w)
##    user  system elapsed
##   0.001   0.000   0.000
```

Definizione di funzioni

Ovviamente in `R` è possibile creare delle funzioni personalizzate usando la parola chiave `function`

- Alla funzione è possibile passare dei parametri che vengono passati per valore (se li modificate le modifiche non si propogano all'esterno)
- Il corpo di una funzione è definito da un blocco e può restituire un valore con la parola chiave `return`
- Gli argomenti vengono separati da virgole, è possibile impostare dei valori di default usando gli uguali

```r
# Definizione della funzione successore
succ <- function(x=0){
  return(x+1)
}
```

Chiamare una funzione (opzione 1)

- Scrivere la funzione direttamente sull'interprete come nella slide precedente e chiamarla usando il nome che le è stato assegnato

```
expY <- function(x=0,y){
  return(y^x)
}
expY(,3)
## [1] 1
expY(3,2)
## [1] 8
expY(y = 2, x = 3)
## [1] 8
```

N.B. La seconda e la terza chiamata sono assolutamente identiche, in `expY(3,2)` la posizione del valore indica a quale parametro della funzione associarlo, in `expY(y = 2, x = 3)` il valore del parametro viene definito per nome

Chiamare una funzione (opzione 2)

- Scrivere su un file `.R` la vostra funzione e usare il comando `source("nomeFile.R")` per importare il file

```r
# esempioScript.R
expY <- function(x=0,y){
  return(y^x)
}
```

Chiamare una funzione (opzione 2)

- Scrivere su un file `.R` la vostra funzione e usare il comando `source("nomeFile.R")` per importare il file

```
rm(list = ls())
source("esempioScript.R")
expY(3,2)
## [1] 8
```

Comandi sapply() e lapply()

In una delle slide precendenti abbiamo visto che è possibile ciclare su una lista di funzioni, ovvero applicare una dopo l'altra una serie di funzioni ad un singolo valore. Qualora invece sia necessario applicare una singola funzione a tutti i valori di un vettore/matrice/lista/datframe è possibile usare le funzioni `lapply` e `sapply`; mentre `lapply` restituisce una sempre una lista, `sapply` restituisce una matrice se applicato ad un dataframe e un vettore altrimenti

- In generale la loro esecuzione è più efficiente del corrispondente ciclo for

```
y <- sample(1:10, 2, replace=TRUE)
f <- function(el){
  if(el%%2==0)
  return(TRUE)
  else
  return(FALSE)
}
lapply(y, f)
```

```
## [[1]]
## [1] TRUE
##
## [[2]]
## [1] TRUE
sapply(y, f)
## [1] TRUE TRUE
```

N.B. lapply

Perché non applichiamo direttamente la funzione f sul vettore?

- Il motivo sta nella condizione dell'if, infatti non è possibile inserire come condizione un vettore di booleani

```
f(y)
## Warning in if (el%%2 == 0) return(TRUE) else return(FALSE): the condition
## has length > 1 and only the first element will be used
## [1] TRUE
```

Esercizio 1

Usando le istruzioni viste oggi, scrivere le seguenti funzioni `R` e salvarle nel file `mia_libreria.R`

1. Calcolare la frequenza assoluta dei dati contenuti in un vettore (non usare `table`)
2. Calcolare la frequenza relativa dei dati contenuti in un vettore (non usate `table`)
3. Calcolare la moda dei dati contenuti in un vettore (non usate `table`)

N.B. Come ottenere una copia di un vettore senza duplicati

```
vector = c ("pippo", "pluto", "pluto", "paperino", "paperino", "pluto", "paperino", "paperino")
unique (vector)
## [1] "pippo"    "pluto"    "paperino"
```

Esercizio 1 (cont.)

4. Calcolare la media campionaria dei dati contenuti in un vettore usando le frequenze relative
5. Calcolare la covarianza campionaria di due vettori
6. Calcolare (senza usare `quantile()`) il percentile k-esimo dei dati un vettore (k deve essere un parametro della funzione)
7. Calcolare l'indice di eterogeneità di Gini e Entropia **normalizzati** dei dati contenuti in un vettore

Esercizio 2

1. Scaricare il dataset reddito.csv
2. Importare il dataset creando un dataframe
3. Usare la funzione scritta in precendenza per calcolare Entropia e indice di Gini dei fattori StatoCivile e Nazione
4. Calcolare la moda dei due fattori

Esempio: passaggio per valore

```
foo <- function(x,y){
  x <- x + y
  print(paste("x dentro la funzione foo", x, sep = " "))
}
x <- 3
y <- 3
foo(x, y)
## [1] "x dentro la funzione foo 6"
x
## [1] 3
```

Esempio: superassegnamento

In R è possibile bypassare il passaggio per valore utilizzando il doppio simbolo di minore: la variabile x viene **modificata solo nell'ambiente da cui è stata effettuata la chiamata**, non dentro la funzione; se non viene trovata, viene creata e inizializzata

```
foo <- function(x,y){
  x <<- x + y
  print(paste("x dentro la funzione foo", x, sep = " "))
}
x <- 3
y <- 3
foo(x, y)
## [1] "x dentro la funzione foo 3"
x
## [1] 6
```

Questionario