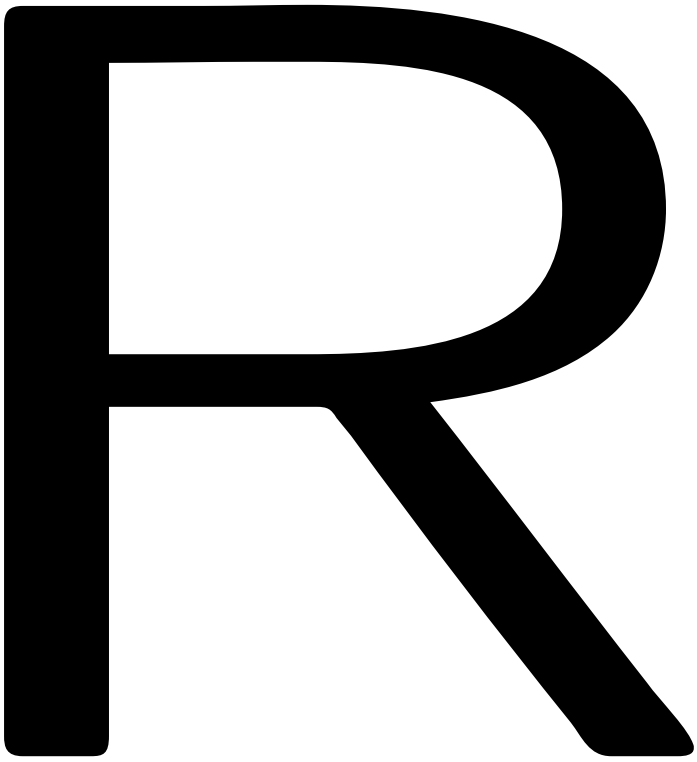

# Appunti di Programmazione

*Sabato De Maio*

# Indice

# Elenco delle tabelle

# Elenco delle figure

# Capitolo 1

# Le Basi

## 1.1 The name of the Game

Cercare un punto da cui partire per introdurre un argomento come **R**, che come vedremo è vastissimo, non è cosa facile. Un primo punto da cui partire è capire innanzitutto come ragione R e come sono le "parole" che usa, al fine di poter comunicare con questo software[1]

La prima cosa da da capire sono i tipi di oggetti, con cui possiamo è possibile lavorare in **R**. Il primo tipo è costituito da "atomic vector", chiamati così in quanto sono l'unità fondamentale alla base di **R**. Proviamo ad esempio a creare e a stampare, semplicemente scrivendo il suo nome, un elemento che chiameremo first, costituito dal solo numero 1.

```
1 > first <- c(1)
2 > first
3 [1] 1
```

Per quanto semplice e banale possa sembrare l'esempio precedente esso introduce sicuramente alcune fondamentali caratteristiche che analizzeremo e potrebbe far sorgere anche qualche interessante domanda. Andiamo per ordine.

Per prima cosa notiamo che il prompt di **R** (o la shell di **R**) restituisce sempre un segno > quando diamo dei comandi, mentre restituisce il numero del primo elemento della riga quando "stampiamo" a video un oggetto. In questo caso abbiamo un elemento di nome first che abbiamo stampato, semplicemente scrivendo il suo nome, ed il numero uno tra parentesi quadre ci indica appunto che quella riga inizia con il primo elemento di first. Da notare che **R** non produce nessun output con la prima stringa di codice.

Tornando leggermente indietro, analizziamo il codice che ci ha permesso di creare l'elemento first.

Il primo importante operatore che abbiamo usato è stato <- composto da un segno minore ed un segno meno. Importante è non inserire alcuno spazio tra questi due simboli. Questo operatore permette di definire oggetti, attribuire loro determinati valori in senso lato (quindi anche funzioni, matrici, ecc.).

[1] **R** è si un software ma è anche un linguaggio derivato a sua volta da altri linguaggi di più basso livello, S ed S-Plus. Per ora, al fine di non introdurre troppi concetti si passi questa definizione superficiale.

Un modo semplice per comprendere concettualmente questo operatore è quello di interpretarlo come una freccia (cosa che da un punto di vista grafico ricorda molto) che punta verso sinistra. In pratica è come se dicessimo ad R "vedi questa stringa che ho digitato, first, assegnale il valore che è alla destra della freccia".

Il secondo, ed importantissimo, elemento introdotto è c( ). Questa è una funzione che permette di combinare elementi e di generare, dalla combinazione di questi, un vettore.

Ecco ora la domanda che, si spera, a qualcuno sia balenata nella mente: "Un vettore con un solo elemento, perché non uno scalare?". La risposta è semplice: R non ha molta simpatia per gli scalari, non li conosce e non è in grado di riconoscerli; semplicemente per lui non esistono.

Da questa importante caratteristica ne deriva che anche un solo elemento, in questo caso numerico, è considerato come un vettore di lunghezza uno ma pur sempre un vettore. Questa caratteristica ha, come si vedrà, importanti effetti.

Ricapitolando:

- con l'operatore <- abbiamo creato un oggetto chiamato first, alla sinistra dell'operatore
- al quale abbiamo attribuito valore 1
- mediante la funzione c( )

Trattandosi un vettore costituito da un solo elemento avremmo potuto usare anche il seguente codice, che si è preferito trascurare per presentare al lettore la funzione c( ), che come detto tornerà utile praticamente sempre nell'uso di R.

```
1 > first1 <- 1
2 > first1
3 [1] 1
```

Per verificare che sia il primo codice che il secondo producono un vettore[2] usiamo il seguente codice mediante il quale, sostanzialmente, chiediamo ad R "l'argomento della funzione is.atomic è un atomic vector?".

```
1 > is.atomic(first)
2 [1] TRUE
3 > is.atomic(first1)
4 [1] TRUE
```

Proviamo ora a creare e stampare a video un elemento, di nome second, composto dai numeri 1, 3, 5, 6, 9, con il codice seguente.

```
1 > second <- c(1,3,5,6,9)
2 > second
3 [1] 1 3 5 6 9
```

[2]Numerico, ma per ora è meglio tralasciare questo aspetto.

In questo caso avendo un numero di elementi maggiore di uno l'uso della funzione `c( )` si rende necessario. Senza di essi infatti avremmo un errore.

# Aggiornamenti delle edizioni