```
############################################################
########### R: caratteristiche del linguaggio #############
############################################################


## le righe che iniziano con il simbolo "#" sono commenti
## (il cancelletto è un carattere speciale che dice
## all'interprete di R di non mandare in esecuzione
## quanto scritto dopo #, fino all'interruzione di riga);
## le righe che non iniziano con il cancelletto possono essere
## copiate ed incollate al prompt dell'interprete per mandarle in esecuzione;
## per qualsiasi problema o chiarimento contattatemi:
## l.maraviglia@provincia.lucca.it



3 + 5

4 / 2


## R (o meglio, il suo interprete) capisce i
## principali simbolici matematici

8 ^ 2


## cosa scrivereste se voleste la radice cubica di 27?

27 ^ (1/3)


## lo sapevate? (io l'ho scoperto pochi mesi fa...)



ciao R, come stai?

## non capisce il linguaggio "naturale"
## (restituisce un messaggio di errore)


## forse abbiamo sbagliato lingua

Hello R, how re you?


## R è un linguaggio artificiale con proprie regole (sintattiche e semantiche)
## se vogliamo utilizzarlo per comunicare con il nostro processore
## siamo noi che dobbiamo attenerci alle sue regole
```

"Ciao R, come stai?"

non dà errore!
se "quotiamo" la frase mettendola fra due virgolette,
l'interprete la accetta

R ammette tre tipi di dati elementari:
- stringhe di caratteri (qualsiasi tipo) circondate da virgolette (character)
- numeri (numeric)
- espressioni logiche (logical)

sto semplificando, ma per i nostri obiettivi odierni va bene così

##
R: Object-oriented
##

entriamo nel vivo della materia
(un primo step nella complessità del linguaggio R)

a

come abbiamo visto, dà errore

a <- 2

secondo voi, cosa succede se schiaccio il tasto di invio (ENTER)?

niente!?

a

secondo voi, cosa succede se adesso schiaccio il tasto di invio?

il trucco è l'avere usato l'operatore di assegnazione `<-`;
l'operatore di assegnazione crea un collegamento fra l'espressione
alla propria sinistra (nome) ed il contenuto alla propria destra (oggetto);
tale collegamento (o referenza) si mantiene stabile per la durata della sessione

di lavoro (e, a certe condizioni, anche oltre)

quando creo in questo modo un "oggetto" e schiaccio l'invio,
se ho fatto le cose per bene (ad esempio, ho rispettato le regole
di accettazione dei nomi) non succede apparentemente nulla;
in realtà, qualcosa è cambiato nell'ambiente di lavoro,
e lo posso vedere digitando

```
ls()
```

l'ambiente di lavoro (workspace) che all'inizio della sessione era vuoto
adesso contiene l'oggetto "a"

la cosa fantastica è che posso assegnare ad un nome (dunque oggettificare)
qualsiasi contenuto, anche complesso, come ad esempio un intero dataset

```
rcfl <- read_tsv(file.choose())
```

posso verificare che l'operazione sia andata a buon fine con

```
ls()
```

posso vedere l'oggetto con
```
View(rcfl)
```

Tutto questo è riassunto nell'espressione "orientato agli oggetti" (object oriented):
R è fondamentalmente un linguaggio "orientato agli oggetti"

##
R as Functional PRogramming Language
##

facciamo un passo indietro

```
3 ^ (1/2)
```

abbiamo visto che possiamo estrarre la radice di un numero
utilizzando l'operatore `^` e l'espediente di esprimere
l'argomento della radice sotto forma di frazione (con num 1);

c'è però un modo più efficiente: usare la funzione `sqrt`

sqrt(3)

oltre ad essere orientato agli oggetti, R è fondamentalmente
un linguaggio funzionale

le due cose sono strettamente collegate fra loro, vediamo perche

in R, scrivere una funzione è estremamente semplice
(forse per questo motivo vi sono migliaia di funzioni!)

adder <- function(x) x + 1

ad esempio, adder è una funzione che aggiunge un'unità al numero che le viene passato come argomento
(x è un segnaposto, o meglio indica una variabile)

la sintassi da utilizzare per scrivere una funzione in R è obbligata:

a) per prima cosa, dobbiamo dichiarare che stiamo scrivendo una funzione
digitando l'espressione `function`;

b) quindi, dobbiamo indicare fra una coppia di parentesi tonde
gli argomenti richiesti dalla funzione;
in questo caso, la funzione "adder" richiede un unico argomento
indicato genericamente con la lettera "x";
x deve essere un valore numerico, ma questo non è dichiarato
(a differenze di quello che dovremmo fare in C)
l'interprete lo capisce dal contesto, ovvero dal corpo (body) della funzione

c) infine dobbiamo fornire il corpo (body) della funzione;
il corpo specifica che cosa fa la funzione;
in questo caso, il corpo `x + 1` dice in modo abbastanza trasparente che
che adder somma un'unità ad x;
in genere, il corpo di una funzione viene inserito fra una coppia di parentesi graffe:

```
adder <- function (x) {
                        x + 1
                      }
```

NB: da notare che nell'espressione abbiamo usato l'operatore di assegnazione `<-`

per assegnare il corpo della funzione al nome "adder";
in R, le funzioni possono essere assegnate a nomi, come qualsiasi altro contenuto;
in altre parole, le funzioni sono (di norma) oggetti alla stregua dei dati (semplici o complessi)

e questo è tutto!

in R le funzioni sono ovunque:
tutto ciò che accade è il risultato dell'invocazione di una funzione

anche gli operatori aritmetici (`+`, `^` ecc.) e perfino le parentesi (`(`, `[` ecc.)
sono in realtà delle funzioni;
ad esempio

3 + 2

è equivalente (in realtà è) a

`+` (3,2)

##
############ R as vectorized Language ####################
##

la terza fondamentale caratteristica di R è la vettorizzazione

la struttura fondamentale di dati in R è il vettore (vector);
un vettore è un insieme ordinato di dati elementari di un o stesso tipo
(ordinato nel senso che l'ordine con cui sono forniti gli elementi
quando è costituito il vettore conta)

per creare un vettore in R si utilizza la funzione `c`

c(1, 2, 3, 4, 5)

o, in un caso come questo, l'operatore `:`

1:5

se proviamo a mettere in un vettore dati di tipo diverso, R applica proprie regole di coercizione
per ripristinare l'omogeneità fra gli elementi

```
c(1:3, "4", 5)


## se vogliamo fare uso di un vettore in un secondo momento, dobbiamo assegnargli un nome
## usando l'operatore di assegnazione `<-`

num_vec <- 1:5


## una funzione vettorizzata è in grado di operare su vettori (oltre che su semplici scalari)

## `adder` è una funzione vettorizzata (ciò è sorprendente data la sua semplicità)

adder(num_vec)


a <- adder(num_vec)

num_vec + a


## la vettorizzazione è una proprietà potentissima, che consente ad R di fornire
## prestazioni sorprendenti (R è per costruzione un linguaggio lento, perchè
## pensato per un uso interattivo; quando però è possibile vettorizzare le operazioni
## R riesce a recuperare una buona parte del gap di efficienza rispetto a linguaggi
## come Python o (persino) C;


#########################################################
##### R as a dynamic, interactive language ##########
#########################################################


## R è stato progettato per l'analisi statistica,
## in modo da consentire un uso dinamico ed interattivo (tramite l'interprete)


## in realtà, anche in una sessione dinamica è utile ed opportuno
## scrivere piccoli blocchi di codice in un editor di testi
## (NB: usate il block notes o simili e non un file tipo word)
## quindi mandarli in esecuzione, copiandoli ed incollandoli al prompt
## (esistono strumenti ottimizzati, ma non è il caso di allargare il quadro)
```

```
## scrivere codice utilizzando un editor di testo è la chiave della riproducibilità di un'analisi;
## in tale contesto è particolarmente utile il simbolo `#`;
## il cancelletto è un carattere che viene trattato dall'editor in un modo speciale:
## tutto ciò che è collocato alla sua destra, fino al termine della riga viene ignorato;
## il pratica, il cancelletto si usa per introdurre commenti nel codice



## pacchetti per l'analisi
library(tidyverse)



## per importare i dati in R (già eseguito)
## NB: il file con i dati è stato precedentemente scaricato,
## dezzippato e salvato in una cartella in locale;

rcfl <- read_tsv(file.choose())

## ho utilizzato la funzione read_tsv perchè i dati
## sono in un file .txt e sono separati da tabulazione
## (più avanti vedremo un esempio di importazione di dati
## da un file .csv)



## per vedere l'oggetto (data frame) tramite il Viewer
View(rcfl)



## per comodità, trasformo i nomi delle variabili da maiuscoli in minuscoli
colnames(rcfl) <- tolower(colnames(rcfl))



## per individuare le variabili che contengono la condizione professionale
grep("cond", colnames(rcfl))



## per estrarre l'informazione relativa alla condizione professionale

rcfl %>%
        select(cond3)



## per la  tabella di frequenza della condizione professionale

rcfl %>%
```

```
        count(cond3)


## per creare una nuova variabile con la condizione professionale esplicita

rcfl <- rcfl %>%
        mutate(cond_prof = ifelse(cond3 == 1, "occupati",
                                  ifelse(cond3 == 2, "disoccupati", "inattivi")))


## per la tabella di frequenza esplicita

 rcfl %>%
        count(cond_prof)


## per aggiungere una colonna con le frequenze percentuali

rcfl %>%
        count(cond_prof) %>%
        mutate(perc = n / sum(n) * 100)



## a me interessa la provincia di Lucca...

grep("prov", colnames(rcfl))



## per la tabella con le frequenze assolute e percentuali per la prov di Lucca

rcfl %>%
        filter(provcm == "046") %>%
        count(cond_prof) %>%
        mutate(perc = n / sum(n) * 100)


## ovviamente a me interessano i dati pesati

grep("coef", colnames(rcfl))



## fortunatamente la funzione "count" contempla come argomento un vettore con i pesi...

rcfl %>%
        filter(provcm == "046") %>%
        count(cond_prof, wt = as.numeric(coefmi)/10)
```

a questo punto posso calcolare manualmente il tasso di disoccupazione

l'operatore `%>%` (pipe) e le funzioni della suite "tidyverse"
possono essere utilizzate per molte altre operazioni di wrangling
(ad esempio per fare delle join fra dateframe diversi)
oppure per rappresentare graficamente i risultati
ma questo lo vediamo in un contesto applicativo...

```
##########################################################################
######################## UPI, 16 ottobre 2018 ############################
##########################################################################



#########################################################
############## un esempio applicativo... ###############
#########################################################



### in genere, il territorio delle province è suddiviso in zone
### per finalità amministrative (ad es. zone sanitarie, zone educative ecc.);
### spesso la zonizzazione amministrativa non coincide con le classificazioni statistiche
### (ad es. le zone non coincidono con i sistemi locali del lavoro)

### il problema pratico è quantificare la popolazione in età scolare-infantile (3-5 anni)
### per le 3 zone educative della provincia di Lucca,
### specificando anche l'incidenza di stranieri



# prima di ogni altra cosa, conviene settare opportunamente
# la directory di lavoro (file > cambia directory...)




# per "attivare" i pacchetti che saranno utilizzati
# ------------------------------------------------


library(tidyverse)

## NB: per poter essere attivati, prima i pacchetti devono
## essere installati, ad es.:
## install.packages("tidyverse")



# importazione dei dati direttamente dal web
# ------------------------------------------


# per assegnare l'url dei dati (come stringa di testa)
# ad un oggetto ("url") che posso riutilizzare in seguito
```

```
url <- "http://demo.istat.it/pop2018/dati/comuni.zip"


# per creare un contenitore "vuoto" in cui
# immagazzinare i dati

dest <- basename(url)



# per prelevare i dati dall'url (url) e importarli
# nel file temporaneo di destinazione (dest)

download.file(url, dest)



# si utilizza la funzione "read_csv" assieme alla funzione "unzip" per
# importare i dati contenuti nel file "comuni_2018" in R
# ed assegnarli ad un oggetto denominato "p18"

p18 <- read_csv(unzip(dest), locale = locale(encoding = "UTF-8"))



# basic wrangling
#----------------


# per rendere minuscoli i caratteri dei nomi delle colonne (variabili)

colnames(p18) <- tolower(colnames(p18))


# per rinominare alcune variabili del dataframe

p18 <- p18 %>%
        rename(cod_com = "codice comune",
                com = "denominazione",
                eta = "età",
                totale_m = "totale maschi",
                totale_f = "totale femmine")



# per eliminare dal dataframe l'informazione ridondante
# relativa al totale di abitanti per ciascuna categoria

p18 <- p18 %>%
        filter(eta != 999)
```

```
### i totali (totale celibi, totale coniugati ecc.)
### sono codificati con l'etichetta "999" nella colonna "età";
### in linea di principio, è opportuno eliminare da un dataframe
### le righe ridondanti perchè possono facilmente indurre
### in errore (tipo gli italiani sono 120 milioni!)




##############################################
####### adesso vai con gli stranieri #######
##############################################


### replichiamo le operazioni di download per gli stranieri

url <- "http://demo.istat.it/strasa2018/dati/comuni.zip"


dest <- "comuni_st_2018.zip"

## NB: per evitare di soprascrivere il file con i dati
## non utilizzo la funzione "basename", bensì assegno
## il nome "manualmente"


download.file(url, dest)


p18_st <- read_csv(unzip(dest), skip = 1, locale = locale(encoding = "UTF-8"))



#### wrangling

p18_st <- p18_st %>%
                rename(cod_com = "Codice comune",
                        com = "Comune",
                        eta = "Età",
                        st_m = "Maschi",
                        st_f = "Femmine")

# per eliminare informazione ridondante sui totali

p18_st <- p18_st %>%
        filter(eta != 999)


# per eliminare la colonna "com"

p18_st <- p18_st %>%
                        select(-(com))
```

```
### join(t)

# per unire (left_join) p18 e p18_st in base alle keys "cod_com" e "età"

p18 <- p18 %>%
        left_join(p18_st, by = c("cod_com", "eta"))



# per tenere solo le colonne necessarie

p18 <- p18 %>%
        select(cod_com, com, eta, totale_m, totale_f, st_m, st_f)



# per calcolare le coorti relative agli italiani

p18 <- p18 %>%
        mutate(it_m = totale_m - st_m,
               it_f = totale_f - st_f)



# per estrarre soltanto i comuni della provincia di Lucca

lu18 <- p18 %>%
        filter(cod_com %in% 46001:46037)



# vettori da usare in seguito

Piana <- c(46001, 46007, 46017, 46021, 46022, 46026, 46034)
Versilia <- c(46005, 46018, 46033, 46013, 46024, 46028, 46030)
Piana_Versilia <- c(Piana, Versilia)
Valle_Serchio <- setdiff(46001:46037, Piana_Versilia)



# per aggiungere una variabile che identifica la zona
# di appartenenza dei comuni

lu18 <- lu18 %>%
        mutate(zona = ifelse(cod_com %in% Piana, "Piana",
```

```
                        ifelse(cod_com %in% Versilia, "Versilia", "Valle_del_Serchio")))

lu18 %>%
        filter(eta %in% 3:5) %>%
        group_by(zona) %>%
        summarise(it_m = sum(it_m),
                it_f = sum(it_f),
                st_m = sum(st_m),
                st_f = sum(st_f)) %>%
        mutate(tot_3_5 = it_m + it_f + st_m + st_f)




# per salvare i data frame nella working directory
# come file .csv

write_csv(p18, "p18.csv")




################################
#### let's do some plotting ####
################################



### discretize age in classes

lu18$eta_class <- cut(lu18$eta, breaks=c(0,5,10,15,20,25,30,35,40,45,50,
                                        55,60,65,70,75,80,85,90,95,101), include.lowert=T, right=F)




# histogram

a <- lu18 %>%
        group_by(eta_class) %>%
        summarise(italiani = sum(it_m) + sum(it_f),
                stranieri = sum(st_m) + sum(st_f)) %>%
                                                ggplot(aes(x = eta_class, y = italiani)) +
                                                geom_bar(stat = "identity") +
                                                coord_flip()
```

```
b <- lu18 %>%
        group_by(eta_class) %>%
        summarise(italiani = sum(it_m) + sum(it_f),
                  stranieri = sum(st_m) + sum(st_f)) %>%
                                                    ggplot(aes(x = eta_class, y = stranieri)) +
                                                    geom_bar(stat = "identity") +
                                                    coord_flip()

library(gridExtra)

grid.arrange(a, b, ncol=2)



### effetto "piramide"


aa <- a + scale_x_discrete(position = "top")

grid.arrange(aa, b, ncol=2)


aaa <- a + scale_y_reverse() + scale_x_discrete(position = "top")


grid.arrange(aaa, b, ncol=2)




######### confronto stranieri e straniere a Viareggio


viareggio_st_m <- lu18 %>%
                        filter(cod_com == 46033) %>%
                        group_by(eta_class) %>%
                        summarise(stranieri_m = sum(st_m)) %>%
                        ggplot(aes(x = eta_class, y = stranieri_m)) +
                        geom_bar(stat = "identity") +
                        coord_flip() +
                        ggtitle("Viareggio_stranieri_M") +
                        scale_y_reverse() +
                        scale_x_discrete(position = "top")


viareggio_st_f <- lu18 %>%
                        filter(cod_com == 46033) %>%
                        group_by(eta_class) %>%
                        summarise(stranieri_f = sum(st_f)) %>%
                        ggplot(aes(x = eta_class, y = stranieri_f)) +
```

```
                        geom_bar(stat = "identity") +
                        coord_flip() +
                        ggtitle("Viareggio_stranieri_F")

grid.arrange(viareggio_st_m, viareggio_st_f, ncol=2)



#---------------------------------------------------------
# import nationalities
#---------------------------------------------------------



# url dove sono i dati

url <- "http://demo.istat.it/str2017/dati/Lucca.zip"


# nome del file di destinazione in wd

dest <- "lucca_nazioni_2017.zip"



# download dei dati e creazione del file in wd

download.file(url, dest)



# importazione in R e creaone di data frame

lu17_naz <- read_csv2(unzip(dest), skip=41, locale=locale(encoding="UTF-8"))

### l'argomento skip = 41 ci consente di saltare le prime righe,
### quelle con i dati di bilancio demografico in senso stretto
### (a noi qui interessano le nazionalità)


# per rinominare le colonne

lu17_naz <- lu17_naz %>%
                        rename(cod_com = `Codice Comune`,
                                cod_citt = "Codice cittadinanza",
                                citt = "Cittadinanza",
                                st_m = "Cittadini stranieri - Maschi",
                                st_f = "Cittadini stranieri - Femmine",
                                st_t = "Cittadini stranieri - Totale")
```

```
# per la classifica della nazionalità presenti in
# provincia di Lucca

print(lu17_naz %>%
                group_by(citt) %>%
                summarise(tot = sum(st_t, na.rm=T)) %>%
                arrange(desc(tot)),
                        n = Inf)



# per il dettaglio dei rumeni (ogni riga è il totale di un comune)

lu17_naz %>%
                filter(citt =="Romania") %>%
                arrange(desc(st_t)) %>%
                mutate(perc = st_t / sum(st_t) * 100)



# per il dettaglio dei cingalesi

lu17_naz %>%
                filter(citt =="Sri Lanka") %>%
                arrange(desc(st_t)) %>%
                mutate(perc = st_t / sum(st_t) * 100)



# per un confronto fra rumeni e cingalesi

sri_lanka <- lu17_naz %>%
                filter(citt == "Sri Lanka") %>%
                mutate(perc = st_t /sum(st_t) * 100) %>%
                ggplot(aes(x = reorder(cod_com, perc), y = perc)) +
                geom_bar(stat = "identity") +
                coord_flip()


romania <- lu17_naz %>%
                filter(citt == "Romania") %>%
                mutate(perc = st_t /sum(st_t) * 100) %>%
                ggplot(aes(x = reorder(cod_com, perc), y = perc)) +
                geom_bar(stat = "identity") +
                coord_flip()

grid.arrange(romania, sri_lanka, ncol = 2)
```

```
# preserving proportions

sri_lanka <- lu17_naz %>%
                filter(citt == "Sri Lanka") %>%
                mutate(perc = st_t /sum(st_t) * 100) %>%
                ggplot(aes(x = reorder(cod_com, desc(perc)), y = perc)) +
                geom_bar(stat = "identity") +
                ylim(0,75) +
                coord_flip()


romania <- lu17_naz %>%
                filter(citt == "Romania") %>%
                mutate(perc = st_t /sum(st_t) * 100) %>%
                ggplot(aes(x = reorder(cod_com, desc(perc)), y = perc)) +
                geom_bar(stat = "identity") +
                ylim(0,75) +
                coord_flip()

grid.arrange(romania, sri_lanka, ncol = 2)




############################################################
##### let's do some mapping ################################
############################################################


# stringa di testo con indirizzo

url <-
"http://www.istat.it/storage/cartografia/confini_amministrativi/generalizzati/Limiti01012018_g.zip"



# per creare un contenitore "vuoto" in cui
# immagazzinare i dati

dest <- "poligoni.zip"


download.file(url, dest)



library(sf) # per importare e manipolare i file vettoriali (.shp)

shp18 <- st_read(file.choose())
```

```
shplu <- shp18 %>%
                filter(COD_PROV == 46)


plot(shplu[0]) # plot only polygons



sex_ratio <- lu17_naz %>%
                        group_by(cod_com) %>%
                        summarise(st_m = sum(st_m, na.rm = T),
                                st_f = sum(st_f, na.rm = T),
                                sex_ratio = st_m/st_f)


sex_ratio$cod_com <- as.double(sex_ratio$cod_com)



shpsex <- left_join(shplu, sex_ratio, by = c(PRO_COM = "cod_com"))



plot(shpsex["sex_ratio"])


library(tmap)


map_sex = tm_shape(shpsex) +
  tm_fill(col = "sex_ratio", style = "fixed", breaks = c(0, 0.5, 0.8, 0.9, 1.0, Inf)) + tm_borders() +
        tm_layout(legend.position = c("right", "top"), frame = F)



tmap_mode("view")
```