Introduzione ad R

Per caricare uno script in R: `source("R/paths.r")` sia da console sia da uno script per includere altro codice

Istruzioni in R

Non richiedono un ; alla fine, ma se volessimo utilizzare più espressioni sulla stessa eriga dobbiamo utilizzare il ;

Non abbiamo bisogno di definire una variabile, il tipo viene definito in base al contenuto. Per assegnare le variabile si usa il simbolo `<-` ad esempio `x<-10` o anche `6->y` e volendo anche `z=5` ma è sconsigliato.

Per i nomi delle variabili R è case-sensitive, i nomi delle variabili possono contenere lettere, numeri, . e _ ma non possono iniziare per un numero o il punto.

Se il nome di una variabile inizia con il . viene considerata come una variabile nascosta e non verrà considerata nell'elenco delle variabili.

Per elencare le variabili esistenti si usa la funzione `ls()` .

Per ripulire l'ambiente si usa la funzione `rm(list=ls())`

Tipi di dati principali in R

Tipo numerico, si suddivide in:

- intero (integer): 1L, 2L, 3L
- double (double): 1.5, -2.3, 0.0
- complessi (complex): 1+2i, 3-4i

Stringhe (character): "ciao", 'mondo'

Valori logici (logical): TRUE, FALSE

Ci sono anche tipi avanzati che permettono di rappresentare strutture più complesse:

- Vettori (vector): Sequenze di elementi tutti dello stesso tipo
- Matrici (matrix): Tabele bidimensionali di elementi dello stesso tipo
- Array (array): Tabelle multidimensionali tutti dello stesso tipo
- Liste (list): Collezioni di elementi di tipo diverso
- Data frame (data.frame): tabelle bidimensionali utilizzate per rappresentare dati sperimentali
- Fattori (factor): Variabili categoriali (con un numero limitato di valori come il sesso, un piccolo insieme di colori)

Per conoscere il tipo di una variabile si usa la funzione class() ad esempio

```
a<-10
class(a) #numeric
```

Le variabili in R possono assumere dei valori speciali:

- NULL: Rappresenta un valore indefinito
- NA: Valore mancante, Not Available
- NaN: Valore numerico non definito, Not a Number, tipicamente è il risultato di una operazione matematica non valida
- Inf: Infinito positivo
- -Inf: Infinito negativo

Per verificare se una variabile è NULL si usa la funzione `is.null(nomevariabile)`

Per verificare se una variabile è NA si usa la funzione `is.na(nomevariabile)` ma questa restituisce TRUE anche in casi di valori NaN e NULL

Operatori in R

Operatori aritmetici: +, -, *, /, ^ (potenza), %% (modulo), %/% (divisione intera)

Operatori di confronto: ==, !=, <, >, <=, >=

Operatore di appartenenza: %in% verifica se un elemento è presente in un vettore

Gli operatori di confronto restituiscono un valotr logico True o False come risultato e si possono utilizzare per tutti i tipi di base in R (numeric, character, logical)

Operatori logici: & (And), | (Or), ! (Not). Gli operatori logici a differenza di altri linguaggi non hanno il doppio simbolo

Definire un operatore: `%not_in% <- Negate(%in%)`

Costrutti di controllo flusso, if, while, for

Costrutto if

```
if(a>0){
    print("a è positivo")
} else{
    print("a è negativo")
}
```

Costrutto while

```
i<-1
while(i<=5){
    print(i)
    i<-i+1
}
```

Per saltare un'interazione si usa la funzione next

Costrutto for

```
for (variabile in sequenza){
    istruzioni da eseguire
}
```

Con min:max si crea una sequenza di numeri interi

```
for(i in 1:5){
    print(i)
}
```

Posso generare sequenze di numeri con la funzione seq(from, to, by, length.out) dove:

- from: Valore iniziale
- to: Valore finale
- by: Incremento (default 1)
- length.out: Lunghezza della sequenza (se specificato ignora by o to)

```
seq(1,10,by=2) #1, 3, 5, 7, 9
seq(1,10,length.out=5) # 1, 3.25, 5.5, 7.75, 10
seq(1, by=2, length.out=5) # 1, 3, 5, 7, 9
seq(10, 1, by=-2) # 10, 8, 6, 4, 2
```

Definzione di funzioni

La sintassi per definire una funzione in R è:

```
nome_funzione <- function(arg1, arg2, ...){
    istruzioni
    istruzione finale # valore di ritorno
}
```

Il valore di rotorno di una funzione è l'ultima istruzione eseguita

Esempio: Funzione che raddoppia un numero

```
raddoppia <- function(x){
    x*2
}

raddoppia(5)
```

Le funzioni possono essere ricorsive

```
fattoriale <- function(n){
    if(n==0)
    {
        1
    } else {
        n * fattoriale(n-1) # Valore di ritorno
    }
}

fattoriale(5) # 120
```

Il valore di ritorno può essere esplicitato con la funzione return()

```
somma <- function(a, b){
    return(a+b)
}

somma(3,4)
```

return() è una funzione quindi vuole le parentesi, non è una parola chiave come in C

Gli argomenti di una funzione possono avere dei valori di default che vengono specificati nella definizione di della funzione utilizzando il simbolo =

```
porenza <- function(base, esponente=2){
    base ^ esponente
}

potenza(3) #9
potenza(3,3) #27
potenza(esponente=3, base=3) #27
potenza(4, esponente=3) #64
```

Per visualizzare l'help di una funzione si usa la funzione help() o il ?

Strutture dati in R

Vettori

Si crea utilizzando la funzione c

```
v <- c(1,2,3,4,5)
```

Per conoscere la lunghezza di un vettore si utilizza la funzione length() `length(v)`

> ATTENZIONE: R PARTE DA 1 E NON DA 0

Per ottenere un elemento di un array si fa con `v[1]` posso usare gli indici negativi per escludere elementi es: `v[-1] #2, 3, 4, 5`

In caso di array booleani si selezionano solo quegli elementi TRUE. Posso anche impostare una condizione da soddisfare es: `v[v>3] #4, 5`

Posso aggiungere elementi al vettore utilizzando la funzione c, posso utilizzarla anche per concatenare vettori

```
v <- c(v, 6) # Aggiungo 6 alla fine del vettore
v <- c(0, v) # Aggiungo 0 all'inizio del vettore
```

Le operazioni aritmetiche tra due vettori si eseguono sempre elemento per elemento, se i vettori hanno lunghezze diverse R ricicla gli elementi del vettore più piccolo per il riempimento. Se il vettore più lungo non è un multiplo del vettore più corto si genera un warning.

Posso usare le funzioni predefinite per i vettori:

- sum(a)
- mean(a)
- sd(a) deviazione standard
- min(a)
- max(a)
- sort(a)
- rev(a) inverte l'ordine degli elementi
- unique(a) restituisce solo gli lemeenti unici

Matrici

Le matrici possono essere considerate come elementi bidimensionali di elementi dello stesso tipo. Si crea copn la funzione `matrix(data, nrow, ncol, byrow, dimanmes)`

- nrow: numero di righe
- ncol: numero di colonne
- byrow: se TRUE riempie la matrice per righe (default FALSE)
- dimnames: lista di nomi per righe e colonne

Tutte le funzioni messe a disposizione per i vettori funzionano anche per le matrici altri metodi sono:

- dim(m) restituisce un vettore di 2 elementi, il numero di righe ed il numero di colonne
- nrow(m) restituisce il numero di righe
- ncol(m) restituisce il numero di colonne

Per accedere agli elementi di una matrice buisogna utilizzare due indici uno per le righe edu no per le colonne es `m[1,2]`, posso usare gli indici negativi come per i vettori.

Per quanto riguarda l'indicizzazione se oemtto di specificare una delle due dimensioni R la calcolerà automaticamente andando a prendere tutti gli elementi necesari dalla matrice. In particolare se ometto l'indice di riga R considera tutte le righe, se ometto l'indoice di colonna R considera tutte le colonne

```
m[,2] # restituisce la seconda colonna
m[1,] #restituisce la prima colonna
```

Tutte le funzioni viste per i vettori si possono anche utilizzare per le matrici (sum(), prod(), ...). Le operazioni aritmetiche si effettuano elemento per elemento a meno che non si stia usando l'operatore %*% che è la **moltiplicazione matriciale** riga per colonna.

Si possono usare tutti gli operatori tranne il prodotto matriciale.

Liste

Una lista è una collezione di elementi di tipo diverso. Si crea con la funzione list

```
l <- list(nome="Mario", eta=30, altezza = 1.75, figli = c("Anna","Luca"))
```

Per conoscere la lunghezza di una lista si utilizza la funzione length(). Per accedere agli elmenti di una lista si usa l'indicizzazione con le parentesi quadre [] o con il simbolo $ o con le doppie parentesi [[]]

Differenze:

- [] restituisce una sottolista
- [[]] restituisce l'elemento vero e proprio
- $ è un modo più comodo per accedere agli elementi con nome

> ATTENZIONE: Le funzioni viste per i vettori non si possono utilizzra direttamente sulle liste

Lezione 2: Mercoledì 23 ottobre 2025

Una lista è una collezione eterogenea di elementi, per creare una lista in R dobbiamo utilizzare la funzione `list()` che prende tra parentesi l'elenco degli elementi che noi vogliamo inserire nella lista.

```
l<-list(nome="Mario",eta=30,altezza=1.75,figli=c("Anna","Luca"))
```

Se voglio conoscere la lunghezza di una lista utilizzo `length(l)`

La differenza tra [] e [[]] è che mentre [] restituisce una sottolista (sempre una lista), [[]] restituisce l'elemento vero e proprio

Esempio:

```
l[1] # Sottolista con il primo elemento
l [[1]] # Stringa "Mario"
l$nome # Della lista l prendi l'elemento che si chiama nome
l[c(1,3)] # restituisce il primo ed il terzo elemento della lista
```

Posso modificare gli elementi di una lista usando l'indicizzazione con [[]] o con $ o tramite assegnazione <-

```
l[[2]] <- 31
l$altezza <- 1.80
```

Tutte le altre operazioni viste per le matrici non si possono applicare alle liste in quanto eterogenee e non hanno una struttura fissa

Funzioni lapply() ed sapply()

- lapply(): Applica una funzione ad ogni elemento della lista e restituisce una nuova lista
- sapply(): Applica una funzione ad ogni elemento della lista e restituisce un vettore o una matrice

```
l<-list(a=1:5,b=6:10,c=11:15)
print(lapply(l,sum)) # Somma degli elementi di ogni elemento della lista
print(saply(l,mean)) # Media degli elementi di ogni elemento della lista
print(sapply(l, function(x) (x^2))) # Quadrato degli elementi di ogni elemento della lista
```

Se l'output della funzione per ogni elemento della list in input è un singolo valore, restituisce un vettore altrimenti se sono vettori dello stesso tipo e della stessa lunghezza restituisce una matrice.

Funzione sample()

Restituisce elementi random, ad esempio `sample(c(1,3,5,7), 1)` restituisce un elemento a caso tra 1,3,5 e 7

Data frame (data.frame)

Rappresenta una delle strutture più importanti in R è una tabella bidimensionale per rappresentare dati sperimentali, le righe rappresentano le osservazioni le colonne le variabili.

Si utilizza la funzione `data.frame()`

Esempio:

```
df <- data.frame(
  nome = c("Mario", "Luigi", "Peach", "Toad", "Yoshi", "Bowser"),
  eta = c(30, 28, 25, 22, 20, 35),
  altezza = c(1.75, 1.80, 1.60, 1.65, 1.58, 1.62)
)
```

Per ottenere specifici elementi

```
df[1,2] # Elemento alla riga 1, colonna 2
df[,2] # Tutte le righe seconda colonna
df[2,] # Tutte le colonne seconda riga
```

Per accedere a tutti i valori della colonna nome possiamo usare `df$nome`, ad esempio `df$eta <- df$eta + 1` incrementa a tutti l'età di 1

Funzioni rbind() e cbind()

Servono ad aggiungere rispettivamente nuove righe e nuove colonne

```
nuova_riga <- data.frame(nome="Daisy", eta="30", altezza="1.20")
df <- rbind(df,nuova_riga)
```

Posso rimuovere colonne con `df$eta <- NULL`.

Tutto ciò che abbiamo visto con l'operatore $ per accedere o modificare data frame funziona anche con [[]]

```
df[["eta"]] <- df[["eta"]]+1
```

Possiamo usare `data.matrix()` per convertire un dataframe in una matrice numeric, funziona solo per valori numerici

Grafici in R

R contiene una serie di metodi per disegnare grafici come istogrammi, grafici a torte, grafici a barre, boxplot... esistono anche pacchetti come ggplot2 che permette di creare grafici avanzati.

Utilizzeremo il dataset di esempio già integrato in R chiamato `mtcars`.

La funzione `plot()` prende due parametri:

- x: Vettore contenente l'asse x del grafico
- y: vettore contenente l'asse y del grafico

A seconda del valore passato ad x possiamo non specificare y.

Grafico di dispersione o a punti o scatter plot

```
mtcars$qsec

plot(mtcars$qsec)

plot(mtcars$qsec, mtcars$mpg) # Come si comporta qsec in base al valore di mpg
```

Grafico a linee

```
x<-seq(0,10,0.5)
plot(x,sin(x),type='l') # omettendo type='l' viene visualizzato con i puntini
```

Box Plot

```
mtcars$cyl <- factor(mtcars$cyl)
plot(mtcars$cyl, mtcars$mpg)
```

Funzioni lines() e points()

La funzione plot() crea ogni volta un grafico se voless iaggiungere ad un grafico linee o punti posso utilizzare le funzioni `lines()` e/o `points()`

```
x<-seq(0,10,0.5)
plot(x,sin(x))
lines(x,cos(x),col="blue")
```

Istogrammi

```
hist(sample(0:100,1000,replace=T),breaks=50,probability=T) # probability = T equivalente a relative=True di python
```

Grafici a torta

```
table(mtcars$cyl)

pie(table(mtcars$cyl),col=c("red","green","blue"),labels=c("cyl=4","cyl=6","cyl=8"))
```

Grafici a barre

```
barplot(table(mtcars$cyl))
```

con il parametro `horiz=T` posso stampare le barre orizzontalmente

Conteggio dei dati nel dataset

Tramite `table(mtcars$cyl, mtcars$gear)` si ottiene una matrice che indica il numero di macchine per ogni valore di cilindrata è possibile stamparlo con:

```
barplot(table(mtcars$cyl,mtcars$gear),legend=c("cyl4","cyl6","cyl8"),col=c("red","green","blue"),names.arg=c("gear=3","gear=4","gear
```

Se volessimo ottenere le barre uno accanto all'altra possiamo aggiungere il parametro `beside=T`

Boxplot tramite boxplot()

Boxplot tramite boxplot()

La funzione boxplot() in R serve per rappresentare graficamente la distribuzione di una variabile continua in relazione a una o più variabili categoriche. I principali parametri sono:

- formula: indica la relazione tra la variabile dipendente e quella (o quelle) di raggruppamento.
- data: specifica il dataset da cui prelevare le variabili.

- names: consente di assegnare etichette personalizzate ai box.

```
boxplot(mpg ~ cyl, data = mtcars, names = c("cyl=4", "cyl=6", "cyl=8"))
```

In questo caso vengono visualizzati anche gli outliers.

Allineamento di sequenze in R

Tutti gli strumenti per poter eseguire l'allineamento di sequenze in R si trovano con il pacchetto `pwalign` di Bioconductor.

Per installare bioconductor:

```
if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install(version = "3.21")
```

e per installare pwalign: `BiocManager::install("pwalign")` per importarlo `library(pwalign)`.

Per eseguire l'allinemento si usa la funzione pairWiseAlignent che prende i parametri:

- pattern: Un vettore di sequenza da allineare
- subject: La sequenza di riferimento
- type: tipo di allineamento (global, local o overlap)
- gapOpening: costo di apertura di un gap
- gapExtension: costo di estensione di un gap
- scoreOnly: se true restituisce solo il punteggio dell'allineamento

```
myFirstAlignment <- pairwiseAlignment(
    pattern=c("succeed","precede"),
    subject="supersede"
)
```

per ottenere i dettagli dell'allineamento `summary(myFirstAlignment)`, il risultato è un oggetto di classe `PairwiseAlignmentsSingleSubject`. Per ottenere il numero di allineamenti si utilizza il metodo `length()`.

Per interrogare l'oggetto possiamo utilizzare i metodi:

- score: restituisce il punteggio dell'allinemento
- nmatch: restituisce il numero di match
- nmismatch: restituisce il numero di mismatch
- nindel: restituisce il numero di allineamenti

Per ottenere le sequenze allineate si usa il metodo `aligned`, per convertire l'allineamento in stringa si usa il metodo `as.character` o in matrice con `as.matrix`.

Posso fare l'allineamento locale con `type=local`

```
myFirstAlignment <- pairwiseAlignment(
    pattern=c("succeed","precede"),
    subject="supersede",
    type="local"
)
```

Matrici di sostituzione

Il pacchetto `pwalign` supporta le matrici PAM e BLOSUM, sono le matrici che permettono l'allineamento di amminoacidi come ad esempio la matrice PAM120

```
myFirstAlignment <- pairwiseAlignment(
    pattern=c("succeed","precede"),
    subject="supersede",
    substitutionMatrix=PAM120
)
```