3.2_iterativa

Programmazione iterativa

Il concetto di iterazione è alla base di qualsiasi operazione nei linguaggi di programmazione. In R molte delle operazioni sono vettorizzate. Questo rende il linguaggio più efficiente e pulito MA nasconde il concetto di iterazione.

Ad esempio la funzione **`sum()`** permette di sommare un vettore di numeri.

```r
# somma i numeri da 1 a 10
sum(1:10)
```

```
[1] 55
```

Ma cosa si nasconde sotto?

Esempio: se io vi chiedo di usare la funzione **`print()`** per scrivere “hello world” nella console 5 volte, come fate?

```
1 msg = "hello world"
2 print(msg)
```

```
[1] "hello world"
```

```
1 print(msg)
```

```
[1] "hello world"
```

```
1 print(msg)
```

```
[1] "hello world"
```

```
1 print(msg)
```

```
[1] "hello world"
```

```
1 print(msg)
```

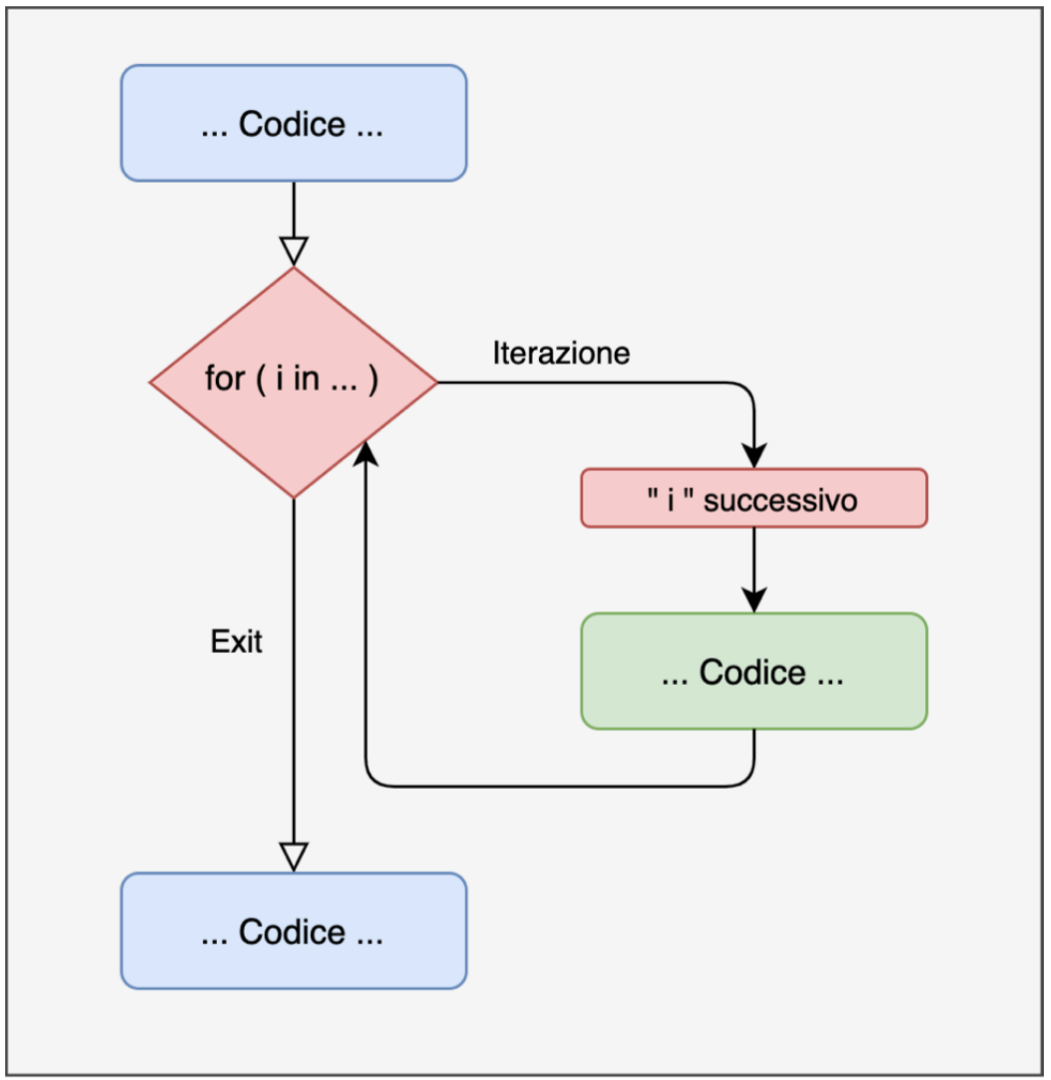
```
[1] "hello world"
```

Quello che ci manca (e che invece fa la funzione `sum()`) è un modo di ripetere una certa operazione, senza effettivamente ripetere il codice manualmente.

Ci sono vari costrutti che ci permettono di ripetere operazioni, i più utilizzati sono:

- Ciclo `for`
- Ciclo `while`
- Ciclo `repeat`
- `*apply` family

Il ciclo `for`



Il ciclo **`for`** è una struttura che permette di ripetere un numero finito e pre-determinato di volte una certa porzione di codice.

La scrittura di un ciclo for è:

```
n = 10
for(i in 1:n){
# quali operazioni
}
```

Se voglio stampare una cosa 5 volte, posso tranquillamente usare un ciclo for:

```
for(i in 1:5){
print(paste("hello word", i, sep = "_"))
}
```

```
[1] "hello word_1"
[1] "hello word_2"
[1] "hello word_3"
[1] "hello word_4"
[1] "hello word_5"
```

Scomponiamo il ciclo `for`

- `for(){ }`: è l'implementazione in R (in modo simile all'if statement)
- `i`: questo viene chiamato iteratore o indice. E' un indice generico che può assumere qualsiasi valore e nome. Per convenzione viene chiamato i, j etc. Questo tiene conto del numero di iterazioni che il nostro ciclo deve fare
- `in <valori>`: questo indica i valori che assumerà l'iteratore all'interno del ciclo
- `{ # operazioni }`: sono le operazioni che i ciclo deve eseguire

La potenza del ciclo for sta nel fatto che l'**iteratore** i assume i valori del vettore specificato dopo `in`, uno alla volta:

```
for(i in 1:10){
  print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

`For` con iteratore vs senza

`i` rappresenta l'indice degli elementi del vettore vec

```
1  vec=1:5
2
3  for(i in 1:length(vec)){
4
5    print(vec[i])
6
7    }
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

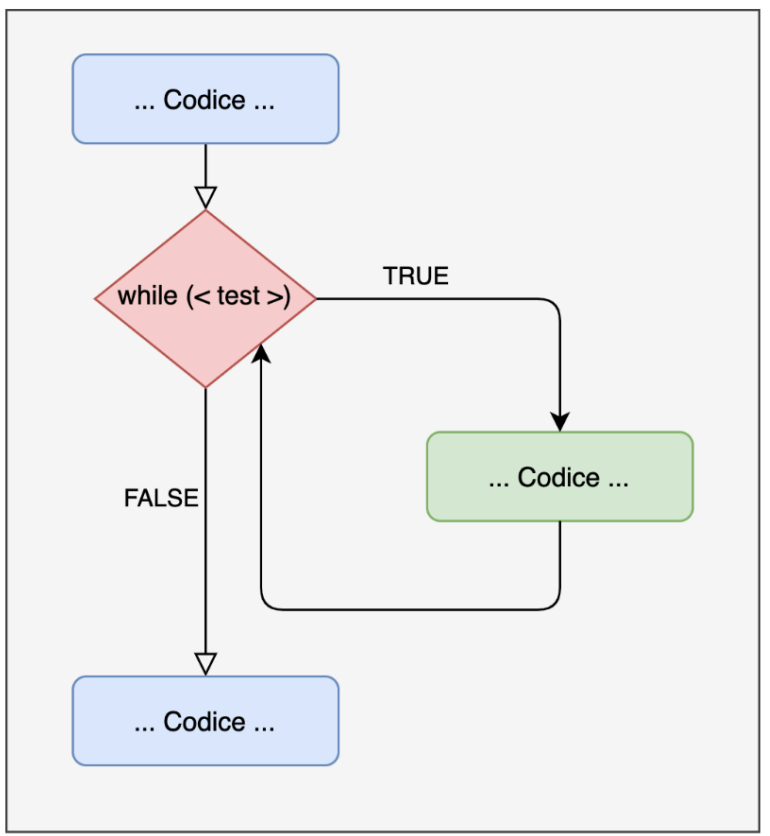
`i` rappresenta direttamente ogni elemento del vettore vec

```
1  vec=1:5
2
3  for(i in vec){
4
5    print(i)
6
7  }
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Ciclo `while`

Il ciclo **`while`** utilizza una condizione logica e non un iteratore e un range di valori come nel for. Il ciclo continuerà fino a che la condizione è **`TRUE`**:



Provate a scrivere questo ciclo **`while`** e vedere cosa succede e capire perchè accade

```
x = 10

while (x < 15) {

  print(x)

}
```

Il ciclo **`while`** è un ciclo non pre-determinato e quindi necessita sempre di un modo per essere interrotto, facendo diventare la condizione falsa.

```
x = 5

while (x < 15) {
  print(x)
  x = x + 1
}
```

```
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
[1] 11
[1] 12
[1] 13
[1] 14
```

Ciclo `repeat`

La logica del ciclo **`repeat`** è molto simile a quella del ciclo **`while`**, ma con 3 importanti differenze:

1. esegue sempre almeno un'interazione
2. enfatizza la ripetizione fino a che una condizione non viene raggiunta
3. utilizza il comando **`break`** per terminare

Ciclo repeat

```
repeat{
  lancia_dado = sample(1:6, 1)
  print(lancia_dado)
  if(lancia_dado == 6) break
}
```

```
[1] 2
[1] 6
```

`repeat` vs. `while`

`repeat` valuta la condizione una volta **finita** l'iterazione, mentre **`while`** all'**inizio**. Se la condizione non è **`TRUE`** all'inizio, il while non parte mentre repeat si:

```
1  i=1
2  repeat {
3    print(i)
4    i=i+1
5    if(i > 3)break
6    }
```

```
[1] 1
[1] 2
[1] 3
```

```
1  i=1
2  while (i < 4) {
3    print(i)
4    i=i+1
5    }
```

```
[1] 1
[1] 2
[1] 3
```

Esempio funzione `sum()`

Immaginiamo di non avere la funzione `sum()` e di volerla ricreare, come facciamo? Idee?

Somma come iterazione

Scomponiamo concettualmente la somma, sommiamo i numeri da 1 a 10:

- prendo il primo e lo sommo al secondo (somma = 1 + 2)
- prendo la somma e la sommo al 3 elemento somma = somma + 3 …

In pratica abbiamo:

- il nostro vettore da sommare
- un oggetto somma che accumula progressivamente le somme precedenti

Somma come iterazione

```
somma = 0 # inizializziamo la somma a 0
x = 1:10 #vettore x

for(i in seq_along(x)){ # per tutta la lunghezza di x = 1:10
  somma = somma + x[i]
}
```

Mettiamo tutto dentro una funzione (i.e., creo una funzione che replichi quello che fa la funzione somma)

```r
my_sum = function(x){
  somma = 0 # inizializziamo la somma a 0
  for(i in seq_along(x)){
    somma <- somma + x[i]
  }
  return(somma)
}

x = rnorm(100)
my_sum(x)
```

```
[1] -6.248785
```

```r
sum(x)
```

```
[1] -6.248785
```

Iterazione, applicazioni

Il ciclo **`for`** è anche utile per simulare dei dati per esempio, supponiamo che in media (tra i nostri 30 partecipanti) ci aspettiamo un'effetto dell'età **b_mu** uguale a **.25**, e con poca variabilità tra i soggetti (**.1**)

```
mydf = read.csv(file = "data/mydf_2.csv") #carico il dataset
str(mydf)
```

```
'data.frame':   30 obs. of  4 variables:
 $ id     : int  1 2 3 4 5 6 7 8 9 10 ...
 $ age    : int  40 15 14 35 32 30 41 38 21 19 ...
 $ age_cat: chr  "adulto" "adolescente" "adolescente" "adulto" ...
 $ age_z  : num  0.7246 -1.7015 -1.7985 0.2394 -0.0518 ...
```

```
group_mean = .25 #effetto medio atteso
var_mean = .1   #variabilità nell'effetto atteso
n_subj = max(mydf$id) #numero soggetti
```

Ora possiamo simulare, attraverso un ciclo `for` , la variabile y assumendo che dipenda dall'età (**age_z**)

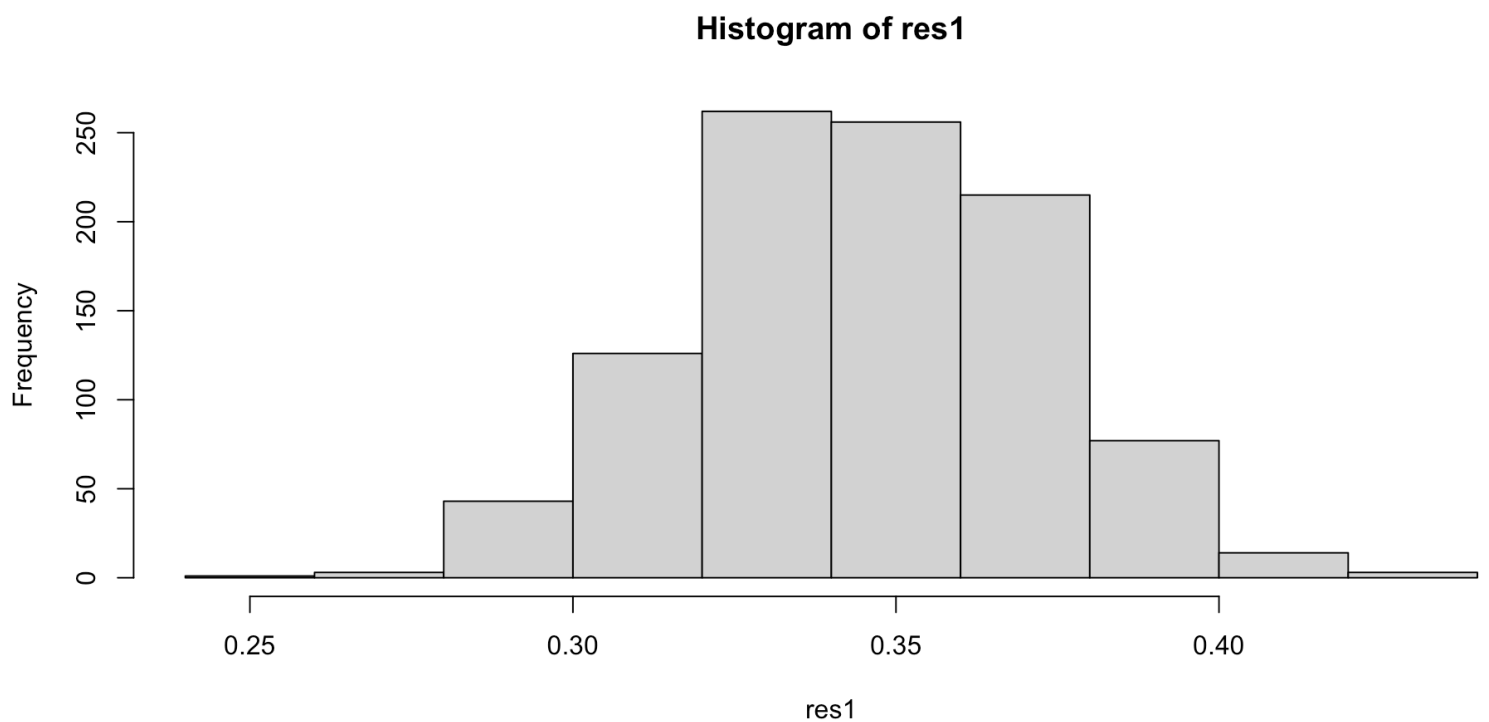
```
set.seed(111) # x replicabilità della simulazione

niter = 1000 # numero di iterazioni
res1 = rep(NA, niter) # vettore di risultati
age_z = mydf$age_z
residui = rnorm(n_subj, 0, .5)

for(i in 1:niter) {

  # effetto
  b_mu = rnorm(n_subj, mean = group_mean, sd = var_mean)
  # Simulo un vettore y per ogni iterazione,
  # che avrà tanti elementi quanti soggetti
  y = 0 + b_mu * age_z + residui
  # Modello lineare
  mod = lm(y ~ 1 + age_z)
  res1[i] = mod$coefficients[[2]] # Estraggo l'effetto dell'età
}
```

Risultato:

```
1 # effetto medio stimato
2 mean(res1)
```

```
[1] 0.3454555
```

```
1 # istogramma dei risultati
2 hist(res1)
```



Aumentiamo la variabilità individuale:

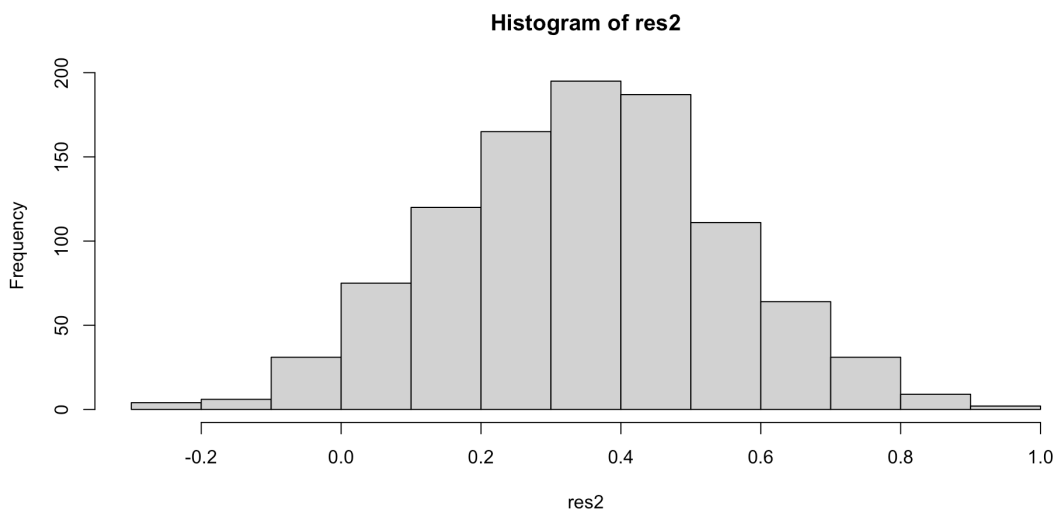
```
1  var_mean = .8  #variabilità nell'effetto atteso
```

Risultato:

```
1  # effetto medio stimato
2  mean(res2)
```

```
[1] 0.3482321
```

```
1  # istogramma dei risultati
2  hist(res2)
```



Lo stesso concetto si può applicare al numero di partecipanti, o alla grandezza dell'effetto atteso. Proprio grazie al for loop è possibile effetuare una ***power simulation***.

Per saperne di più date un'occhiata a questa pagina! :)

Ciclo `for` nested

Una volta compresa la struttura iterativa, si può espandere facilmente inserendo un ciclo dentro un altro:

```
for(i in 1:3){ # livello 1
  for(j in 1:3){ # livello 2
      print(paste(i, j))
  }
}
```

```
[1] "1 1"
[1] "1 2"
[1] "1 3"
[1] "2 1"
[1] "2 2"
[1] "2 3"
[1] "3 1"
[1] "3 2"
[1] "3 3"
```

Supponiamo di avere due gruppi sperimentali (ad esempio, Gruppo di controllo e Gruppo di trattamento), ciascuno con 10 partecipanti. Si vuole misurare una variabile (ad esempio, il tempo di reazione) per ogni partecipante e registrare i risultati in un dataframe:

```
# Definire parametri della ricerca
gruppi=c("Controllo", "Trattamento")  # Nomi dei gruppi
num_soggetti=10                       # Numero di soggetti per gruppo

# Creare una matrice per memorizzare i dati
res=data.frame(
  Gruppo = factor(),
  Soggetto = integer(),
  RT = numeric()
)
```

```r
# Ciclo nested per simulare raccolta dati
set.seed(42)  # Per riproducibilità dei dati simulati

for (gruppo in gruppi) {

  for (soggetto in 1:num_soggetti) {

    # Simulare un tempo di reazione
    RT=ifelse(gruppo == "Controllo",
                                rnorm(1, mean = 500, sd = 50),
              # Media diversa per il gruppo Trattamento
                                rnorm(1, mean = 450, sd = 50))

    # Aggiungere i dati al dataframe, aggiungendo una riga
    res=rbind(res,
        data.frame(Gruppo = gruppo, Soggetto = soggetto, RT = RT))
  }
}
```

```r
str(res)
```

```
'data.frame':   20 obs. of  3 variables:
 $ Gruppo  : chr  "Controllo" "Controllo" "Controllo" "Controllo" ...
 $ Soggetto: int  1 2 3 4 5 6 7 8 9 10 ...
 $ RT      : num  569 472 518 532 520 ...
```

```r
head(res)
```

```
     Gruppo Soggetto       RT
1 Controllo        1 568.5479
2 Controllo        2 471.7651
3 Controllo        3 518.1564
4 Controllo        4 531.6431
5 Controllo        5 520.2134
6 Controllo        6 494.6938
```

```r
mean(res$RT[res$Gruppo == "Controllo"])
```

```
[1] 527.3648
```

```r
mean(res$RT[res$Gruppo == "Trattamento"])
```

```
[1] 441.8272
```

Ora facciamo un po’ di pratica!

Aprite e tenete aperto questo link:

https://etherpad.wikimedia.org/p/arca-corsoR