

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.

```
1 # somma i numeri da 1 a 10  
2 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)
```

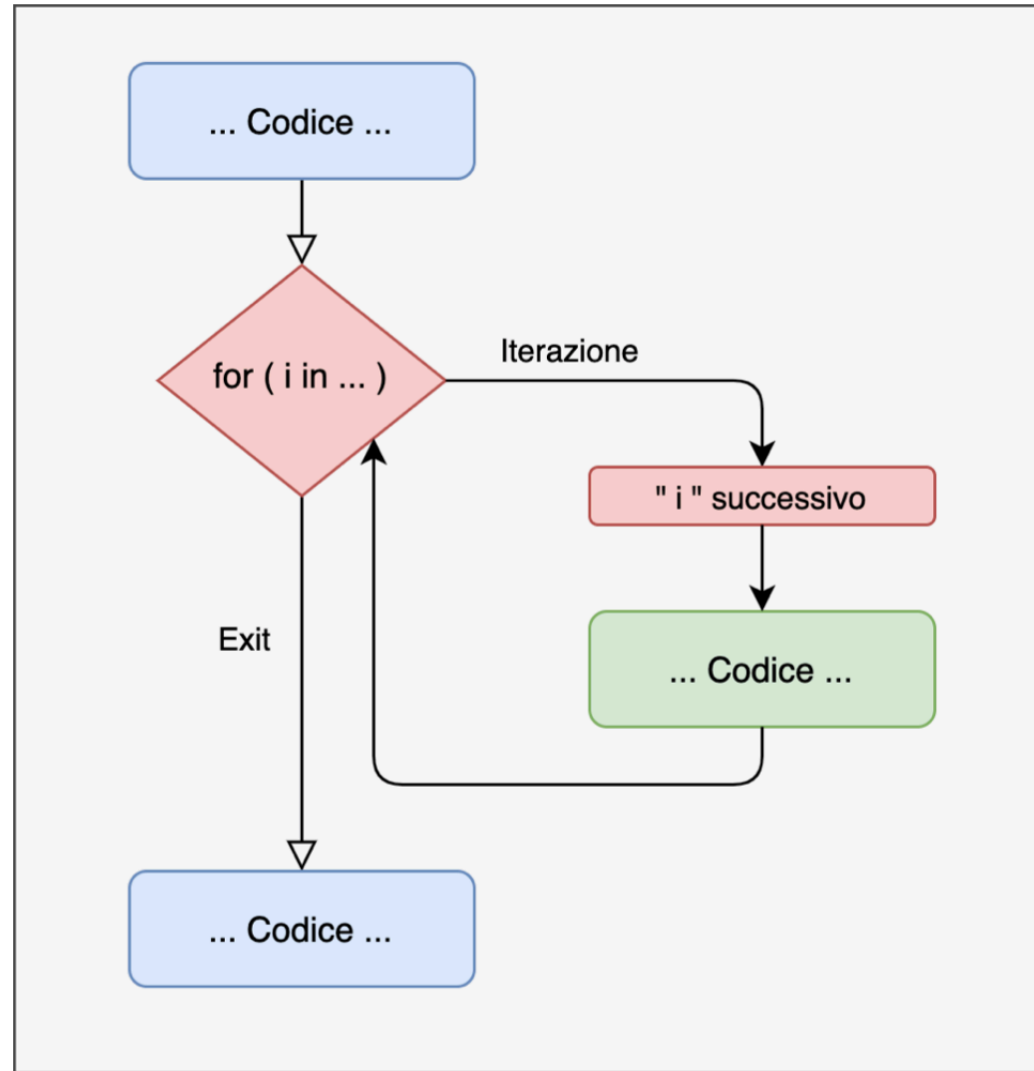
```
[1] "hello world"
```

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:

```
1 n = 10
2 for(i in 1:n){
3   # quali operazioni
4 }
```

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

```
1 for(i in 1:5){
2   print(paste("hello word", i, sep = "_"))
3 }
```

```
[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 il 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:

```
1 for(i in 1:10){  
2   print(i)  
3 }
```

```
[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
```

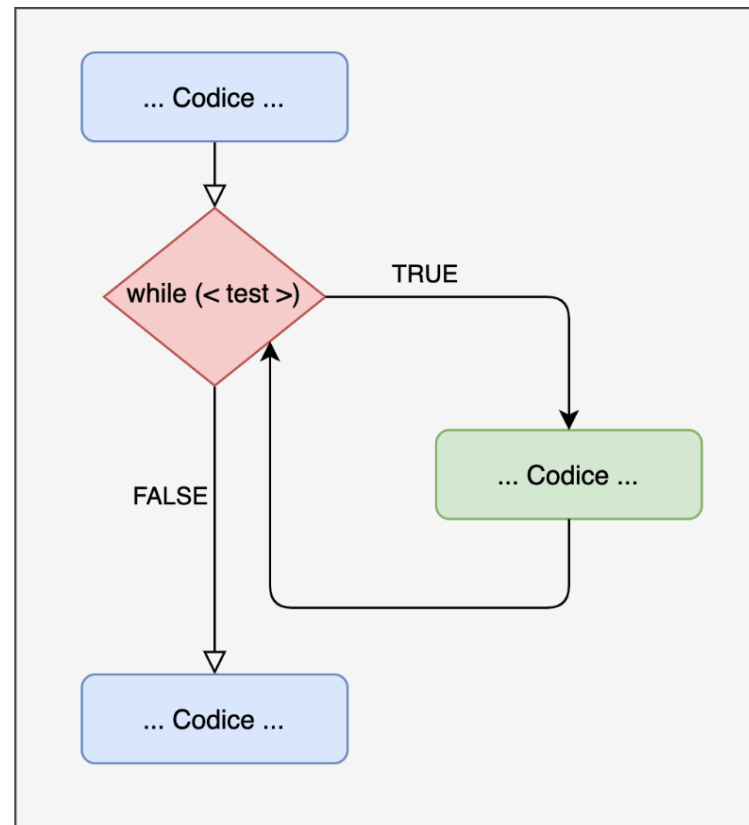
i rappresenta direttamente ogni elemento del vettore vec

```
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.

```
1 x = 5
2
3 while (x < 15) {
4     print(x)
5     x = x + 1
6 }
```

```
[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**

```
1 repeat{  
2   lancia_dado = sample(1:6, 1)  
3   print(lancia_dado)  
4   if(lancia_dado == 6) break  
5 }
```

```
[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 ($\text{somma} = 1 + 2$)
- prendo la somma e la sommo al 3 elemento $\text{somma} = \text{somma} + 3 \dots$

In pratica abbiamo:

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

Somma come iterazione

```
1 somma = 0 # inizializziamo la somma a 0
2 x = 1:10 #vettore x
3
4 for(i in 1:length(x)){ # per tutta la lunghezza di x = 1:10
5     somma = somma + x[i]
6 }
```

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

```
1 my_sum = function(x){  
2   somma = 0 # inizializziamo la somma a 0  
3   for(i in seq_along(x)){ # equivalente a 1:length(x)  
4     somma = somma + x[i]  
5   }  
6   return(somma)  
7 }  
8  
9 x = rnorm(100)  
10 my_sum(x)
```

```
[1] 12.32882
```

```
1 sum(x)
```

```
[1] 12.32882
```

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 **.2**

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

```
'data.frame':   30 obs. of  4 variables:
 $ id      : int  1 2 3 4 5 6 7 8 9 10 ...
 $ age     : int  44 18 32 25 45 33 15 24 30 35 ...
 $ age_cat: chr   "adulto" "adolescente" "adulto" "giovane" ...
 $ age_z   : num  1.1542 -1.1902 0.0721 -0.5591 1.2443 ...
```

```
1 group_mean = .2 # effetto medio atteso
2 var_mean   = .1  # rumore
3 n_subj     = max(mydf$id) #numero soggetti
```

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

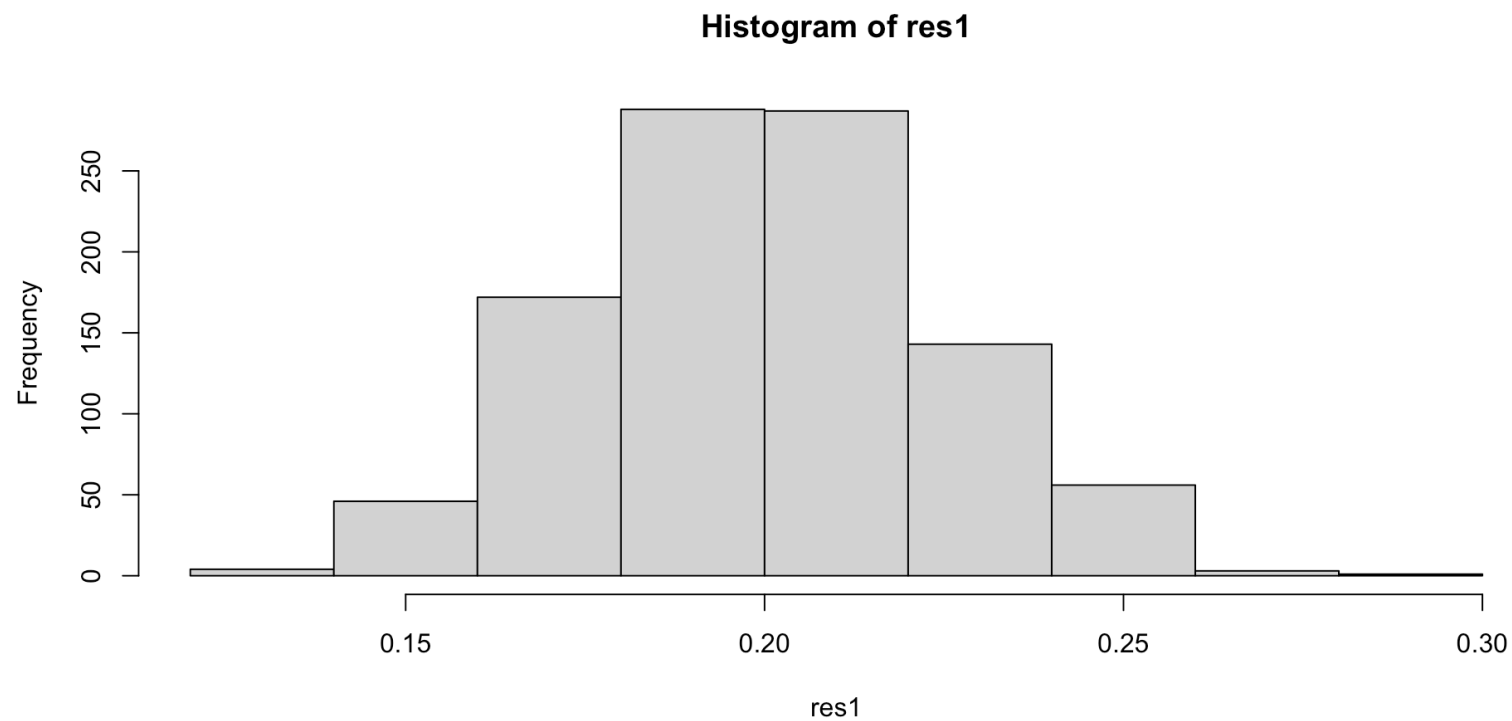
```
1  set.seed(111) # x replicabilità della simulazione
2
3  niter = 1000 # numero di iterazioni
4  res1 = rep(NA, niter) # vettore di risultati
5  age_z = mydf$age_z
6
7  for(i in 1:niter) {
8
9    # effetto
10   b_mu = rnorm(n_subj, mean = group_mean, sd = var_mean)
11   # Simulo un vettore y per ogni iterazione,
12   # che avrà tanti elementi quanti soggetti
13   y = 0 + b_mu * age_z
14   # Modello lineare
15   mod = lm(y ~ 1 + age_z)
16   res1[i] = mod$coefficients[[2]] # Estraggo l'effetto dell'età
17 }
```

Risultato:

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

```
[1] 0.19965
```

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



Aumentiamo il rumore:

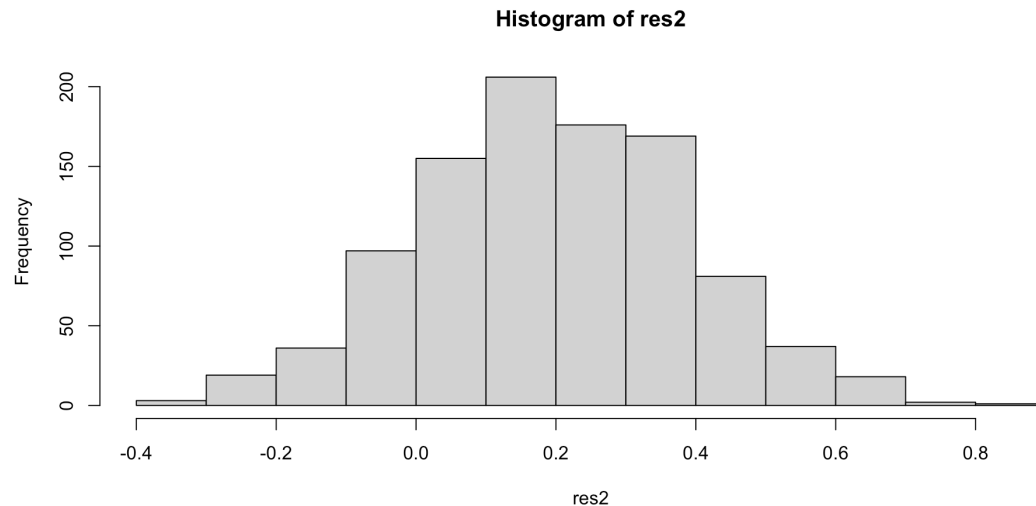
```
1 var_mean = .8
```

Risultato:

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

```
[1] 0.1960141
```

```
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 effettuare 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:

```
1  for(i in 1:3){ # livello 1
2    for(j in 1:3){ # livello 2
3      print(paste(i, j))
4    }
5  }
```

```
[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:

```
1 # Definire parametri della ricerca
2 gruppi=c("Controllo", "Trattamento") # Nomi dei gruppi
3 num_soggetti=10                      # Numero di soggetti per gruppo
4
5 # Creare una matrice per memorizzare i dati
6 res=data.frame(
7     Gruppo = factor(),
8     Soggetto = integer(),
9     RT = numeric()
10 )
```

```
1 # Ciclo nested per simulare raccolta dati
2 set.seed(42) # Per riproducibilità dei dati simulati
3
4 for (gruppo in gruppi) {
5
6   for (soggetto in 1:num_soggetti) {
7
8     # Simulare un tempo di reazione
9     RT=ifelse(gruppo == "Controllo",
10              rlnorm(1, mean = .500, sd = .050),
11              # Media diversa per il gruppo Trattamento
12              rlnorm(1, mean = .450, sd = .050))
13
14     # Aggiungere i dati al dataframe, aggiungendo una riga
15     res=rbind(res,
16              data.frame(Gruppo = gruppo, Soggetto = soggetto, RT = RT))
17   }
18 }
```

```
1 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   1.77 1.6  1.68 1.7  1.68 ...
```

```
1 head(res)
```

	Gruppo	Soggetto	RT
1	Controllo	1	1.765701
2	Controllo	2	1.602821
3	Controllo	3	1.678930
4	Controllo	4	1.701726
5	Controllo	5	1.682387
6	Controllo	6	1.639996

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

```
[1] 1.695801
```

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

```
[1] 1.56018
```

Equivalente (e forse più intuitivo)

```
1 num_soggetti = 10
2 gruppo = character()
3 res2 =data.frame(Gruppo = character(num_soggetti),
4                 Partecipante = integer(num_soggetti), RT = numeric(num_so
5
6 for (i in 1:num_soggetti) {
7
8     gruppo[i] = ifelse(i < 5, "Controllo", "Trattamento")
9
10    RT = ifelse(gruppo[i] == "Controllo",
11               rlnorm(1, .500, .050), rlnorm(1, .450, .050))
12
13    res2$Gruppo[i] = gruppo[i]
14
15    res2$Partecipante[i] = i
16
17    res2$RT[i] = RT
18 }
```

```
1 str(res2)
```

```
'data.frame':  10 obs. of  3 variables:
 $ Gruppo      : chr  "Controllo" "Controllo" "Controllo" "Controllo" ...
 $ Partecipante: int   1  2  3  4  5  6  7  8  9 10
 $ RT          : num  1.62 1.51 1.63 1.75 1.72 ...
```

```
1 head(res2)
```

	Gruppo	Partecipante	RT
1	Controllo	1	1.623636
2	Controllo	2	1.508227
3	Controllo	3	1.634610
4	Controllo	4	1.751958
5	Trattamento	5	1.724194
6	Trattamento	6	1.534917

```
1 mean(res2$RT[res2$Gruppo == "Controllo"])
```

```
[1] 1.629607
```

```
1 mean(res2$RT[res2$Gruppo == "Trattamento"])
```

```
[1] 1.56118
```

Ora facciamo un po' di pratica!

Aprirete e tenete aperto questo link:

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

