Sintaxis de la sentencia if else en R

La sintaxis del if else en R es como sigue:

```
Syntax
if (Condición) { # La condición debe devolver TRUE o FALSE
    # Ejecuta un código
} else {
    # Ejecuta otro código
}
```

También puedes escribir sentencias if **en una sola línea sin llaves**, pero generalmente no se recomienda, ya que podrías cometer errores de sintaxis en el código.

```
if (Condición) print("Código") else print("Más código")
```

Si necesitas que se cumplan varias condiciones puedes **crear if anidados** (if else if).

```
if (Condición) { # La condición debe devolver TRUE o FALSE
    # Código
} else {
    # Código
    if(Condición 2) { # La condición debe devolver TRUE o FALSE
        # Código
    } else {
        # Más código
    }
}
```

Sintaxis del bucle FOR en R

La **sintaxis del bucle for en R** es muy simple:

```
Sintaxis
for (i in lista) {
  # Código
}
```

También puedes escribir un bucle for en una sola línea de código sin corchetes. Sin embargo, no es recomendable escribir así los bucles for.

Como primer ejemplo, podrías pensar en imprimir i + 1, siendo i = 1, ..., 5 en cada iteración del bucle. En este caso, el bucle for comenzará con i = 1 y finalizará con i = 5, por lo tanto la salida será la que se muestra a continuación:

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

# Equivalente a :
# for (i in 1:5) print (i + 1)
Output
2
3
4
5
```

```
6
```

También puedes escribir **sentencias for dentro de otras**. A esto se le llama **bucles anidados**. La sintaxis se representa en el siguiente bloque de código:

```
Sintaxis
for (i in lista) {
        # Código
    for(j in lista) {
        # Código
    }
}
```

A veces es necesario **detener el ciclo** en algún índice si se cumple alguna condición o evitar evaluar algún código para algún índice o condición. Para ese propósito puedes usar las funciones **break** y **next**.

En el siguiente ejemplo, el bucle se interrumpirá en la sexta iteración (que ya no será evaluada) a pesar de que el ciclo completo tiene 15 iteraciones, y además se saltará la tercera iteración.

```
for (iter in 1:15) {

  if (iter == 3) {
        next
    }

    if (iter == 6) {
        break
    }

  print(iter)
}

Output
1
2
4
5
```

Preasignar espacio para ejecutar bucles

Los bucles son **especialmente lentos** en R. Si ejecutas o planeas ejecutar tareas computacionalmente costosas, debes preasignar memoria. Esta técnica consiste en reservar espacio para los objetos que estás creando o rellenando dentro de un bucle. Veamos un ejemplo.

Primero, puedes crear una variable llamada almacenar sin indicar el tamaño de la variable final una vez que se haya llenado dentro del bucle. La **función Sys.time** almacenará la hora exacta en el que se ejecuta la función en sí, así que asegúrate de llamar al siguiente código de una vez, no línea por línea.

```
inicio <- Sys.time()

almacenar <- numeric() # Sin preasignación

for (i in 1:1000000){
 almacenar[i] <- i ** 2
}

fin <- Sys.time()
fin - inicio  # El código tardó en ejecutarse ¿??
```

Segundo, copia el código anterior y preasigna la **variable almacenar** con la longitud final que tendrá el vector.

```
inicio <- Sys.time()

almacenar <- numeric(1000000) # Con preasignación

for (i in 1:1000000){
 almacenar[i] <- i ** 2
}

fin <- Sys.time()
fin - inicio   # El código tardó en ejecutarse XX segundos
```

Bucle for vectorizado

La función **foreach** es una alternativa del bucle for del paquete foreach. Ten en cuenta que también necesitarás usar el operador **%do%**. Esta función puede hacer que tus bucles sean más rápidos, pero la velocidad final podría depender del bucle que realices.

En el siguiente ejemplo creamos una función llamada for_each donde ejecutamos la raíz cuadrada del valor correspondiente de cada iteración. Como foreach devuelve una lista de forma predeterminada, puedes usar el argumento **.combine = 'c'** para que la salida se concatene.

```
install.packages("foreach")
library(foreach)
res <- foreach(i = 1:8, .combine = 'c') %do% {
        sqrt(i)
    }
Output
1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
```

Otra opción es devolver el resultado envuelto por la función unlist.

```
res <- foreach(i = 1:8) %do% {
        sqrt(i)
    }

unlist(res)
Output
1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
```

Bucle for con proceso en paralelo

Cuando se trata de **tareas computacionalmente muy intensivas**, como estudios de simulación, **puede que necesites hacer que tus bucles sean paralelos**. Para eso necesitarás hacer uso de los paquetes parllel y doParallel. Sin embargo, el segundo paquete se carga cuando carga el primero, por lo que no necesitas cargar ambos.

En el siguiente ejemplo, configuramos nuestra ejecución paralela **con todos los núcleos disponibles,** pero puedes usar tantos como quieras. Luego registramos la paralelización y al acabar, debes recordar detener el clúster.

```
library(parallel)

    cl <- parallel::makeCluster(detectCores())
    doParallel::registerDoParallel(cl)

    res <- foreach(i = 1:9, .combine = 'c') %dopar% {
        sqrt(i)
    }

  parallel::stopCluster(cl)
```

Output

```
1.0      1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427 3.000000
```

Bucle while en R

La sintaxis del bucle while en R es como sigue:

Sintaxis

```
while (condición_lógica) {
    # Código
}
```

Supongamos que queremos saber el primer número entero positivo cuyo cuadrado excede 4000. Para ello podríamos escribir:

```
# Inicializamos las variables
n <- 0
cuadrado <- 0

# Bucle while
while(cuadrado <= 4000) {
    n <- n + 1
    cuadrado <- n ^ 2
}

# Resultados
n          # 64
cuadrado   # 4096
```