Estructuras de control

Cristopher Aguirre

16 de octubre de 2020

1. Ejecución condicional

Cuando se quiere forzar la ejecucion de alguna parte del programa según el resultado de la evaluacion de una condicional lógica se ejecuta una estructura de control. Hay diferentes estructuras que permiten la ejecucion condicional.

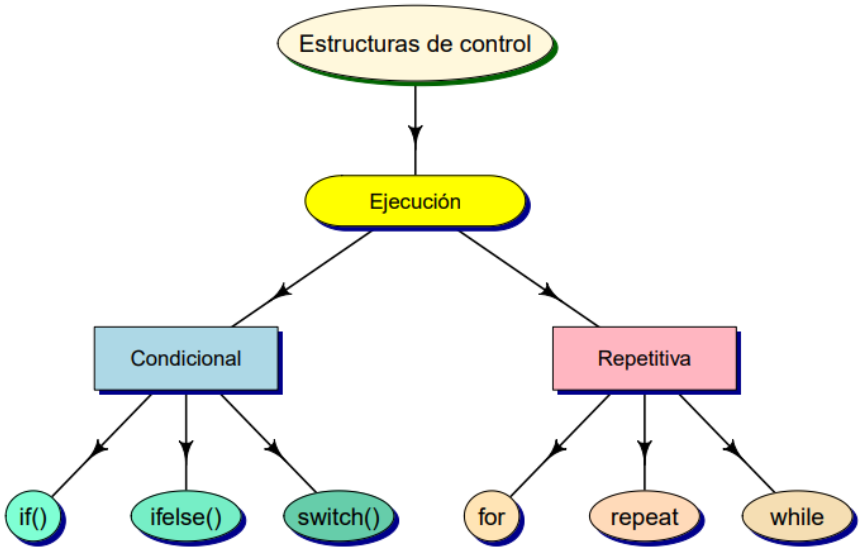

Figure 1: Estructuras de control

1.1. Operadores condicionales if, if else, ifelse

1.1.1. if

Solo evalua una condición lógica y si se cumple se ejecuta un código escrito entre corchetes, si no se cumple salta a la siguiente orden.

```
if (expresion_logica) {
  expression1 ...
}
```

Ejemplo

```
a <-  5
b <-  5
if (a == b){
  print("a es igual a b")
}
```

```
## [1] "a es igual a b"
```

1.1.2 if else

Aquí siempre se ejecuta un bloque de codigo, el bloque uno si se cumple o el bloque dos si no se cumple.

```
if (logical_expression) {
expression_1
...
} else {
expression_2
...
}
```

Ejemplo

```
a <- 7
b <- 5
if (a == b){
  print("a es igual a b")
} else{
  print("a es diferente de b")
}
```

```
## [1] "a es diferente de b"
```

La evaluacion lógica no es vectorizada. Dado un vector de números enteros, escriba un programa que diga si son positivos o negativos.

```
x <- c(2, 3, -5, 6, -2, 8)
if (x > 0) {
  print("positivo")
} else {
  print("negativo")
}
```

```
## Warning in if (x > 0) {: la condición tiene longitud > 1 y sólo el primer
## elemento será usado
```

```
## [1] "positivo"
```

1.1.3. ifelse

Es una versión mas corta de if/else en el que se evalúa si cada uno de los elementos de un vector cumple una condicion; si la cumple se adopta la expresión 'A' si no la cumple se adopta la expresion 'B'.

```
ifelse(logical_expression, A, B)
```

Ejemplo

```
x <- c(2, 3, -5, 6, -2, 8)
ifelse(x > 0,"positivo","negativo")
```

```
## [1] "positivo" "positivo" "negativo" "positivo" "negativo" "positivo"
```

1.2. Ejecucion repetitiva for, repeat, while

1.2.1. for

Cuando se desea una ejecucion un determinado número de veces, o una secuencia de veces se escribe entre parentesis una variable de 'vuelta' y un vector de secuencia. Se ejecutara el código mientras var in seq sea cierto.

```
for(var in seq) expr
```

Ejemplos

```
# Ejemplo 1
for (i in 1:4){
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
```

```
# Ejemplo 2
x <- c(12, 22, 28, 41)
for (i in x) {
  print(i+1)
}
```

```
## [1] 13
## [1] 23
## [1] 29
## [1] 42
```

1.2.2. repeat

Aquí no se utiliza ninguna variable de 'vuelta'; el código se repite indefinidamente hasta que se apague el ordenador, se pulse Ctrl+C o se encuentre una declaracion **break.**

```
repeat expr
```

Ejemplo

```
i <- 1
repeat {
  if (i > 5) break
  else{
    cat('\n', i, "años de edad")
    i <- i + 1
  }
}
```

```
##
##  1 años de edad
##  2 años de edad
##  3 años de edad
##  4 años de edad
##  5 años de edad
```

1.2.3. while

Otras veces se repite el codigo mientras una condicióon es verdadera

```
while(cond) expr
```

Ejemplo

```
"Imprimamos solo los numeros menores a 5"
```

```
## [1] "Imprimamos solo los numeros menores a 5"
```

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

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

2. Familia apply

2.1. apply

Toma el marco de datos o la matriz como entrada y da la salida en vector, lista o matriz. La función apply () se usa principalmente para evitar usos explícitos de construcciones de bucle. Es la más básica de todas las colecciones que se puede utilizar sobre una matriz.

Esta función toma 3 argumentos:

apply (X, MARGIN, FUN)

Aquí:

-x: una matriz o matriz

-MARGIN: toma un valor o rango entre 1 y 2 para definir dónde aplicar la función:

-MARGIN = 1: la manipulación se realiza en filas

-MARGIN = 2: la manipulación se realiza en columnas

-MARGIN = c (1,2): la manipulación se realiza en filas y columnas

-FUN: indica qué función aplicar. Se pueden aplicar funciones integradas como media, mediana, suma, mínimo, máximo e incluso funciones definidas por el usuario.

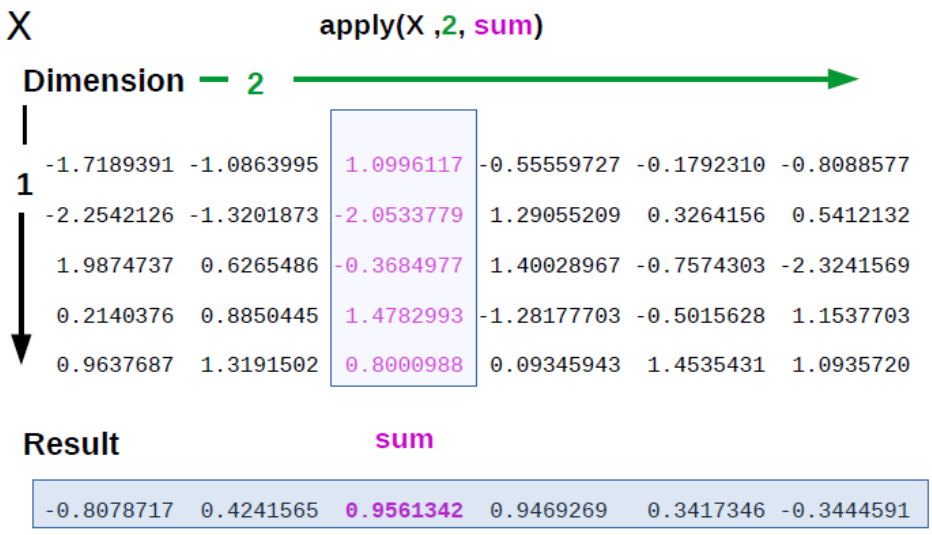

Figure 2: Función apply

```
iris <- data.frame(iris)
apply(X = iris[,-5],MARGIN = 2,FUN = mean)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##     5.843333     3.057333     3.758000     1.199333
```

```
apply(X = iris[,-5],MARGIN = 2,FUN = sum)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##        876.5        458.6        563.7        179.9
```

2.2. lapply

La función *lapply* es útil para realizar operaciones en objetos de lista y devuelve un objeto de lista de la misma longitud que el conjunto original. lappy () devuelve una lista de longitud similar a la del objeto de lista de entrada, cada elemento del cual es el resultado de aplicar FUN al elemento correspondiente de la lista. lapply () toma una lista, un vector o un marco de datos como entrada y da una salida en la lista.

lapply (X, FUN)

Argumentos:

-X: un vector o un objeto

-FUN: Función aplicada a cada elemento de x

l en lapply () significa lista. La diferencia entre lapply () y apply () se encuentra entre el retorno de salida. La salida de lapply () es una lista. lapply () se puede usar para otros objetos como marcos de datos y listas.

`La función lapply () no necesita MARGEN.`

Ejemplos

```
lapply(iris[,-5],FUN = mean)
```

```
## $Sepal.Length
## [1] 5.843333
##
## $Sepal.Width
## [1] 3.057333
##
## $Petal.Length
## [1] 3.758
##
## $Petal.Width
## [1] 1.199333
```

```
sapply(iris, function(x) summary(x))
```

```
## $Sepal.Length
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.300   5.100   5.800   5.843   6.400   7.900
##
## $Sepal.Width
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.000   2.800   3.000   3.057   3.300   4.400
##
## $Petal.Length
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.600   4.350   3.758   5.100   6.900
##
## $Petal.Width
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.100   0.300   1.300   1.199   1.800   2.500
```

```
##
## $Species
##     setosa versicolor  virginica
##         50         50         50
```

```
#sobre un vector
lapply(1:10, function(x) x^2)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 4
##
## [[3]]
## [1] 9
##
## [[4]]
## [1] 16
##
## [[5]]
## [1] 25
##
## [[6]]
## [1] 36
##
## [[7]]
## [1] 49
##
## [[8]]
## [1] 64
##
## [[9]]
## [1] 81
##
## [[10]]
## [1] 100
```

2.3. sapply

La función sapply () toma una lista, vector o marco de datos como entrada y da salida en vector o matriz. Es útil para operaciones en objetos de lista y devuelve un objeto de lista de la misma longitud que el conjunto original. La función sapply () hace el mismo trabajo que la función lapply () pero devuelve un vector.

sapply (X, DIVERSIÓN)

Argumentos:

-X: un vector o un objeto

-FUN: Función aplicada a cada elemento de x

```
min.sl <- sapply(iris[,-5],min)
min.sl #devuelve un vector
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##          4.3          2.0          1.0          0.1
```

```
min.sl <-lapply(iris[,-5],min)
min.sl #devuleve una lista
```

```
## $Sepal.Length
## [1] 4.3
##
## $Sepal.Width
## [1] 2
##
## $Petal.Length
## [1] 1
##
## $Petal.Width
## [1] 0.1
```

```
avg <- function (x) {
  (min(x) + max(x)) / 2
}
resultado <- sapply (iris[,-5], avg)
resultado
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##         6.10         3.20         3.95         1.30
```

2.4. tapply

tapply () calcula una medida (media, mediana, mínima, máxima, etc.) o una función para cada variable de factor en un vector. Es una función muy útil que le permite crear un subconjunto de un vector y luego aplicar algunas funciones a cada uno de los subconjuntos.

tapply (X, INDEX, FUN = NULL)

Argumentos:

-X: un objeto, generalmente un vector

-INDEX: una lista que contiene el factor

-FUN: Función aplicada a cada elemento de x

```
tapply (iris$Sepal.Width, iris$Species, median)
```

```
##     setosa versicolor  virginica
##        3.4        2.8        3.0
```