Introducción al software estadístico

Módulo VIII

Nicolás Schmidt

nschmidt@cienciassociales.edu.uy

Departamento de Ciencia Política
Facultad de Ciencias Sociales
Universidad de la República

Estructura de la presentación

Estructura de la presentación

Estructuras de control

Hay dos tipos de estructuras de control:

- Condicional
  - → `if()`
  - → `ifelse()`
  - → `switch()`
- Repetitiva
  - → `for()`
  - → `repeat()`
  - → `while()`

Estructura de la presentación

if()

Esta estructura evalúa una sentencia lógica, si es `TRUE` se desarrolla una operación determinada, si es `FALSE` avanza con la siguiente orden.

```
if( expresión_lógica ){ expresión_1...}
```

Ejemplo:

```
if(5 < 8){print("Correcto!")}

## [1] "Correcto!"

if(8 < 5){print("Correcto!")}
```

if else

Esta estructura evalúa una sentencia lógica, si es `TRUE` se desarrolla una operación determinada, si es `FALSE` se desarrolla otra sentencia.

```
if( expresión_lógica ){ expresión_1...
} else {
expresión_2...}
```

Ejemplo:

```
if(5 < 8){print("Correcto!")
    }else{
    print("No es correcto!")}

## [1] "Correcto!"

if(8 < 5){print("Correcto!")
    }else{
    print("No es correcto!")}

## [1] "No es correcto!"
```

switch()

```
resumen <- function(x, operacion){
                      switch(operacion,
                      media = mean(x),
                      mediana = median(x),
                      min = min(x),
                      max = max(x),
                      stop("Operacion desconocida!"))
 }
resumen(sample(1:100, 200, replace = T), "media")

## [1] 47.99

resumen(sample(1:100, 200, replace = T), "Media")

## Error in resumen(sample(1:100, 200, replace = T), "Media"): Operacion
desconocida!

resumen(sample(1:100, 200, replace = T), "mediana")

## [1] 49
```

Estructura de la presentación

for

Esta estructura ejecutara una repetición siempre que contenga secuencia para hacerlo.

```
for(var in seq){exp...}
```

Ejemplo:

```
vec <- numeric()        # este es el objeto sobre el que opera el bucle
for(i in 1:3){          # esta es la secuencia
    vec[i] <- i + 1     # esto es el cuerpo del bucle. La/s operaciones que se hacen.
}
vec

## [1] 2 3 4
```

Entonces: los bucles `for` tienen tres partes 1- output, 2- secuencia, 3- cuerpo.

`for` → output

El bucle `for` *opera* sobre un objeto. Este objeto puede ser de cualquier clase y contener cualquier tipo de dato. Si el objeto no existe, se debe crear ya que va a estar asociado a las operaciones del bucle.

Ejemplo:

```
a <- numeric(); a                          # vector de tipo numérico vació

## numeric(0)

for(i in 1:10){a[i] <- i}; a

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

b <- vector("integer", 10); b              # vector de tipo integer de largo 10

##  [1] 0 0 0 0 0 0 0 0 0 0

for(i in 1:10){b[i] <- i}; b

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

`for` → secuencia

La secuencia indica el valor de 'i' en cada iteración. Es importante la diferencia entre usar `1:length(objeto)` y `seq_along(objeto)`.

Ejemplo:

```
vec <- numeric()

seq_along(vec)

## integer(0)

1:length(vec)

## [1] 1 0
```

Si se tiene un vector vacío la secuencia `1:length(objeto)` va a iterar una vez, mientras que la secuencia `seq_along(objeto)` no va a iterar (lo cual es correcto!).

for → cuerpo

El cuerpo contiene la/s operaciones que se van a realizar. Para que el bucle realice las operaciones es necesario indexar o indicar correctamente donde se quiere que la operación actúe.

Ejemplo:

```
vec <- numeric()
for(i in 1:10){
    vec[i] <- i + 10
}
vec

##  [1] 11 12 13 14 15 16 17 18 19 20
```

1. La primera iteración hace `vec[1] <- 1 + 10`
2. La segunda iteración hace `vec[2] <- 2 + 10`

⋮ ⋮ ⋮

10. La décima iteración hace `vec[10] <- 10 + 10`

for

Ejemplo:

```
fibo <- 1:2
for(i in 3:45){
    fibo[i] <- fibo[i-2] + fibo [i-1]
    }
fibo
```

```
##  [1]          1          2          3          5          8         13
##  [7]         21         34         55         89        144        233
## [13]        377        610        987       1597       2584       4181
## [19]       6765      10946      17711      28657      46368      75025
## [25]     121393     196418     317811     514229     832040    1346269
## [31]    2178309    3524578    5702887    9227465   14930352   24157817
## [37]   39088169   63245986  102334155  165580141  267914296  433494437
## [43]  701408733 1134903170 1836311903
```

for

Ejemplo: usando next y break

```
for(i in 1:5){
    if(i == 3){
        next}
        print(i)
}

## [1] 1
## [1] 2
## [1] 4
## [1] 5

for(i in 1:5){
    if(i == 3){
        break}
        print(i)
}

## [1] 1
## [1] 2
```

repeat

Esta estructura permite repetir indefinidamente una operación. Para frenarlo se debe utilizar `break`.

Ejemplo:

```
x <- 1                          # objeto que se va a usar para la repetición
repeat {                        # inicia la repetición
        print(x)                # imprime el valor de x
        x <- x+1                # se va modificando el valor de x...
        if (x == 10){           # condición de parada
            break               # si la condición es TRUE se hace un corte
            }
    }
```

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

while

Esta estructura permite repetir una operación siempre que una expresión sea verdadera

Ejemplo:

```
x <- 0
while (x != 10){
        print(x)
        x <- x + 1
}
```

```
## [1] 0
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
```

Estructura de la presentación

Estructura de la presentación

Estructura

Una función en R es un objeto. Las funciones están compuestas de tres partes:

`formals()` → Argumentos: la lista de argumentos que contiene la función.

`body()` → Cuerpo: todo el código que que está dentro de la función.

`environment()` → Entorno: determina la forma como la función encuentra los valores asociados con los nombres.

Ejemplo: función `tabulate`

```
tabulate

## function (bin, nbins = max(1L, bin, na.rm = TRUE))
## {
##     if (!is.numeric(bin) && !is.factor(bin))
##         stop("'bin' must be numeric or a factor")
##     if (typeof(bin) != "integer")
##         bin <- as.integer(bin)
##     if (nbins > .Machine$integer.max)
##         stop("attempt to make a table with >= 2^31 elements")
##     nbins <- as.integer(nbins)
##     if (is.na(nbins))
##         stop(gettextf("invalid value of %s", "'nbins'"), domain = NA)
##     .Internal(tabulate(bin, nbins))
## }
## <bytecode: 0x40e4a10>
## <environment: namespace:base>
```

Ejemplo: función tabulate

```
formals(tabulate)

## $bin
##
##
## $nbins
## max(1L, bin, na.rm = TRUE)

body(tabulate)

## {
##     if (!is.numeric(bin) && !is.factor(bin))
##         stop("'bin' must be numeric or a factor")
##     if (typeof(bin) != "integer")
##         bin <- as.integer(bin)
##     if (nbins > .Machine$integer.max)
##         stop("attempt to make a table with >= 2^31 elements")
##     nbins <- as.integer(nbins)
##     if (is.na(nbins))
##         stop(gettextf("invalid value of %s", "'nbins'"), domain = NA)
##     .Internal(tabulate(bin, nbins))
## }

environment(tabulate)

## <environment: namespace:base>
```

Estructura de la presentación

Creación

Las funciones en R se crean con la función `function()`. Comúnmente están asociadas a un nombre. Cuando se crea una función se deben establecen los argumentos (`formals()`) que tiene y el código que compone la función (`body()`). El ambiente de una función depende de donde fue creada la función y es una operación implícita.

La creación de una función es:

```
nombre_de_la_función <- function(argumentos){ cuerpo }
```

Nombres

Como cualquier objeto R, las funciones siguen las reglas de nombramiento habituales. No se puede iniciar con nombre, guión bajo ni usar palabras reservadas.

Si es realmente importante iniciar el nombre que va a asignar a una función o a un valor con alguno de estos criterios debe usar comillas simples.

→ Palabras reservadas:

```
?Reserved      # leer ayuda
?make.names    # nombres válidos
```

→ Usar palabras o signos reservados:

```
`1` <- 10 ; `1`

## [1] 10

`NA` <- "HOLA"; `NA`

## [1] "HOLA"
```

Funciones anónimas

Las funciones anónimas son aquellas que no llevan nombre. Comúnmente estas funciones son argumento de otra función.

Ejemplo:

```
args(apply)

## function (X, MARGIN, FUN, ...)
## NULL

apply(cars, 2, function(x){mean(x)/sd(x)})

##    speed     dist
## 2.912450 1.667871
```

La función que se aplica a las columnas de la base `cars` no tiene nombre ya que se ejecuta para esa operación y no es necesario nombrarla.

Creación de funciones

Ejemplo:

```
vec <- c(1, 4, 6, 7, NA, 9, 8, 12, NA, 12, 3, 5, 12, 10, 12)
mean(vec)

## [1] NA

promedio1 <- function(x){mean(x, na.rm = TRUE)}
promedio1(vec)

## [1] 7.769231

promedio2 <- function(z){
                  z <- na.omit(z)
                  sum(z) / length(z)
    }
promedio2(vec)

## [1] 7.769231
```

Creación de funciones: moda

Ejemplo: creando una función que calcule la moda

```
vec

##  [1]  1  4  6  7 NA  9  8 12 NA 12  3  5 12 10 12

moda <- function(x) {
            z <- unique(x)
            z[which.max(tabulate(match(x, z)))]
}
moda(vec)

## [1] 12

moda(c(1,1,2,2)) # dos modas

## [1] 1
```

→ Esta función tiene el problema que no logra identificar si el vector de datos tiene dos o más modas.

Creación de funciones: moda

```
moda2 <- function(x){
            tab <- table(x)
            as.numeric(names(tab)[tab == max(tab)])
}
moda2(vec)

## [1] 12

moda2(c(1,1,2,2))

## [1] 1 2
```

Creación de funciones: moda

```
moda2(c("a", "a", "b"))

## Warning in moda2(c("a", "a", "b")): NAs introduced by coercion

## [1] NA
```

→ No funciona con un vector de caracteres, lo cual es lógico. Pero si nos interesas saber cual es el valor más frecuente de un vector de caracteres?.

```
moda2 <- function(x) {
            tab <- table(x)
            if(is.numeric(x)){
                return(as.numeric(names(tab)[tab == max(tab)]))
            }
            if(is.character(x)){
                return(names(tab)[tab == max(tab)])
            }
}
moda2(c("a", "a", "b"))

## [1] "a"
```

Creación de funciones: moda

¿Si queremos que además del valor más frecuente la función nos devuelva la frecuencia?

```
moda2 <- function(x, freq=FALSE) {
                    tab <- table(x)
                    nam <- names(tab)[tab == max(tab)]
                    fre <- max(tab)

                    if(is.numeric(x)){
                        if(freq == TRUE){
                             return(list(Moda = as.numeric(nam), Frecuencia = fre))
                        }else{
                                return(as.numeric(nam))
                            }
                    }

                    if(is.character(x)){
                        if(freq == TRUE){
                            return(list(Moda = nam, Frecuencia = fre))
                        }else{
                                return(nam)
                            }
                    }
}
```

Creación de funciones: moda

¿Si queremos que además del valor más frecuente la función nos devuelva la frecuencia?

```
vec
```

```
##  [1]  1  4  6  7 NA  9  8 12 NA 12  3  5 12 10 12
```

```
moda2(vec, freq = TRUE)
```

```
## $Moda
## [1] 12
##
## $Frecuencia
## [1] 4
```

```
moda2(c("a", "b", "a"), freq = TRUE)
```

```
## $Moda
## [1] "a"
##
## $Frecuencia
## [1] 2
```

Estructura de la presentación

Búsqueda de nombres

Cuando se crea una función, se crea un ambiente que va a contener los nombres asociados a valores (objetos) que usa esa función. Si una función hace referencia a un nombre que no existe dentro de la función lo va a ir a buscar fuera de ella. Pero lo contrario no es cierto, si se quiere buscar un nombre que está dentro de una función por fuera de ella no es posible acceder a el. Estas reglas están dentro de lo que en R se llama 'alcance léxico'.

Ejemplos:

```
x <- 1
f1 <- function(){
        x <- 2
        x
}
f1()

## [1] 2
```

El objeto `x <- 1` está fuera de la función y dentro de la función hay un `x <- 2` definido.

```
f2 <- function(){
        y <- 2
        c(x, y)
}
f2()

## [1] 1 2
```

En este caso dentro de la función no hay un nombre `x` definido por lo que la ejecución lo va a buscar por fuera de la función.

Estructura de la presentación

Ambientes y búsqueda de nombres

```
ls() # lista de objetos en el entorno global

## character(0)

f <- function() {
        objeto_adentro <- 1
        f <- 2
        list(clase_f = class(f), objetos = ls())
}
f()

## $clase_f
## [1] "numeric"
##
## $objetos
## [1] "f"              "objeto_adentro"

list(clase_f = class(f), objetos = ls())

## $clase_f
## [1] "function"
##
## $objetos
## [1] "f"
```