Introducción a R

Monroy Salinas Ramírez Evelyn

2022-1

1. Operaciones básicas

Podemos realizar operaciones básicas como si fuera una calculadora

```
2+2
```

```
## [1] 4
```

```
6-9
```

```
## [1] -3
```

```
9*8
```

```
## [1] 72
```

```
3/2
```

```
## [1] 1.5
```

```
7**6
```

```
## [1] 117649
```

```
7 ^ 6
```

```
## [1] 117649
```

```
(7 - 5) ** 3.1415926535
```

```
## [1] 8.824978
```

Shortcut: comentar o descomentar varias líneas Ctrl + Shift + C

2. Operadores lógicos

TRUE y FALSE son las unidades lógicas básicas, equivalen a 1 y 0, respectivamente

```
TRUE
```

```
## [1] TRUE
```

```
FALSE
```

```
## [1] FALSE
```

La igualdad como expresión lógica se representa con ==

```
1 == TRUE
```

```
## [1] TRUE
```

La negación en R se expresa con un !

```
0 != FALSE
```

```
## [1] FALSE
```

& es el operador lógico conjunción “y”
es el operador lógico disyunción “o”

```
1 & 0
```

```
## [1] FALSE
```

```
0 | 1
```

```
## [1] TRUE
```

```
1 & 1 && 0 & 1 & 1 & 1
```

```
## [1] FALSE
```

```
1 | 1 || 0 | 0 | 0
```

```
## [1] TRUE
```

Como TRUE y FALSE representan 1 y 0, respectivamente, podemos sumarlos
TRUE+TRUE = ?

```
TRUE+TRUE
```

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

```r
TRUE+FALSE+1.5
```

```
## [1] 2.5
```

```r
TRUE + TRUE + FALSE + TRUE
```

```
## [1] 3
```

Esto nos puede ayudar a crear condiciones lógicas

3. Asignación de valores

Shortcut: para poner <- es ALT + -

<- (Alt + -) se usa para asignar valores, el lado izquierdo de <- denota la variable que será asignada y el lado derecho el valor que se le asignará

```r
x <- 7
# = funciona igual que <-
y = 7
# = e == no son lo mismo!
x == y
```

```
## [1] TRUE
```

Convencionalmente se utiliza <-

4. Formas correctas de nombrar una variable

Se inicia con letras (mayúsculas o minúsculas)

- Puedes utilizar el _
- Preferentemente evitar nombres con .

*R distingue mayúsculas de minúsculas

```r
hola.1 = 0
HOLA.1 = 2
M <- 5
m <- FALSE + 5.8
adios_2 = 23
el_nombre_mas_largo_que_se_me_puede_ocurrir = NULL
missing_dato <- c(1,2,3,NA, 5,6)
a <- TRUE
nombre <- "probabilidad"
semestre <- "2022"
```

Evitar! (no causan un error... inmediato, pero puede generar muchos problemas)

Evitar acentos y caracteres como ñ:

sí = 1

ño = 0

aÃ±o = 30

PROHIBIDO (no podemos asignar nombres que empiecen con un número o que contengan caracteres especiales o espacios):

1.hola = 0

1hola = 0

pesitos$ = 5

circunflejo^5 = 0

ay! = 2

arroba@gmail = 5

un mal nombre <- FALSE

Mandar llamar mis objetos con Ctrl + Enter

```
a
```

```
## [1] TRUE
```

```
nombre
```

```
## [1] "probabilidad"
```

```
m
```

```
## [1] 5.8
```

```
z <- 2+2
z
```

```
## [1] 4
```

```
# O bien poniendo el objeto en ()

(y <- 5.6^2)
```

```
## [1] 31.36
```

```
# Podemos mandar llamar varios objetos con ;

a;z
```

```
## [1] TRUE

## [1] 4
```

```r
x <- 4*5; y <- 3/8
x;y
```

```
## [1] 20
```

```
## [1] 0.375
```

5. El objeto báscio de R: El vector

Características:

- Un vector tiene el mismo tipo que los datos que contiene, es decir, un vector sólo puede contener datos de un sólo tipo.
- Podemos obtener la longitud: es el número de elementos que contiene un vector. La longitud es la única dimensión que tiene esta estructura de datos.

Forma básica de crear un vector, mediante c()

```r
ejemplo <- c(1,2,3,4,5,6,7,8)
vector_1 <- c(1,2,3)
```

Otra forma es con el operador :

```r
vector_2 <- 1:3
vector_1 == vector_2 # operador lógico
```

```
## [1] TRUE TRUE TRUE
```

```r
vector_3 <- 1:7
letras <- c("a","b","c")
b <- c(TRUE,FALSE,FALSE)
c <- c(1.1,1.8,3.56)
```

¿Podemos crear vectores que tengan distintos tipos de variables dentro?

```r
vector_4 <- c(1, "2", 3, "4")

# ¿Qué hizo R?
vector_4
```

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

```r
# Cómo modificamos el tipo de variable que contiene el vector?
as.character(vector_4)
```

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

```r
as.numeric(vector_4)
```

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

```r
as.integer(vector_4)
```

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

```r
nuevo <- as.integer(vector_4)
vector_4 <- as.integer(vector_4)
```

Tipo de datos del vector: class()

```r
class(vector_4)
```

```
## [1] "integer"
```

Otra forma de crear un vector, mediante una secuencia

```r
secuencia <- seq(from = 0, to = 10, by = 0.1)
secuencia2 <- seq(3,7,1)
```

Otra más, mediante una repetición

```r
repeticion_0 <- rep(3, times = 7) # Repite el número 3, el número "times" de veces
```

No confundir con una replicación!

```r
rep_1 <- replicate(2, 3) # Repite 2 veces el número 3
repeticion_1 <- rep(2, 3)

rep_1 == repeticion_1
```

```
## Warning in rep_1 == repeticion_1: longitud de objeto mayor no es múltiplo
## de la longitud de uno menor
```

```
## [1] FALSE FALSE FALSE
```

```r
repeticion_2 <- rep(5, 4)
repeticion_3 <- rep(1, 10)
```

Podemos concatenar vectores y crear uno nuevo

```
repeticiones <- c(repeticion_0, repeticion_1, repeticion_2, repeticion_3)
```

Otros ejemplos de vectores

```
dado <- 1:6
urna <- c('Tauro', "Géminis", 'Virgo', 'Libra', "Cáncer")
nombres <- c("A", "B")
nombres_edades <- c("A", 23, "B", 27)
```

Tamaño o longitud de un vector: length()

```
length(dado)
```

```
## [1] 6
```

```
length(repeticiones)
```

```
## [1] 24
```

¿Cómo funcionan los operadores lógicos y los vectores?

```
nombres == nombres_edades
```

```
## [1]  TRUE FALSE FALSE FALSE
```

```
c(1,0,0,1) & c(1,1,0,1) # Opera entrada a entrada
```

```
## [1]  TRUE FALSE FALSE  TRUE
```

```
c(1,0,0,1) && c(1,1,0,1) # Opera en conjunto
```

```
## [1] TRUE
```

```
c(1,0,0,1) | c(1,1,0,1) # Opera entrada a entrada
```

```
## [1]  TRUE  TRUE FALSE  TRUE
```

```
c(1,0,0,1) || c(1,1,0,1)  # Opera en conjunto
```

```
## [1] TRUE
```

Podemos acceder a las entradas del vector con el operador "[]"

```r
nombres_edades
```

```
## [1] "A"  "23" "B"  "27"
```

```r
length(nombres_edades)
```

```
## [1] 4
```

```r
nombres_edades[3]
```

```
## [1] "B"
```

```r
nombres_edades[1] # En R, el índice empieza en 1
```

```
## [1] "A"
```

```r
nombres_edades[0] # En R, el índice empieza en 1
```

```
## character(0)
```

```r
nombres_edades[5] # NA = Not Available (Missing Values)
```

```
## [1] NA
```

6. Operaciones con vectores

```r
a <- c(2,3,4,5,6,7)
b <- 7:15
c <- c(1,2,1,2,1,2)
```

Operaciones aritméticas

```r
(a+2)
```

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

```r
(a*5)
```

```
## [1] 10 15 20 25 30 35
```

```r
(b%%3) #Módulo
```

```
## [1] 1 2 0 1 2 0 1 2 0
```

```r
(a^2)
```

```
## [1]  4  9 16 25 36 49
```

```r
(a**4)
```

```
## [1]   16   81  256  625 1296 2401
```

Nota: no es necesario hacer un for en las operaciones con vectores

Entre vectores deben tener la misma longitud

```r
length(a)
```

```
## [1] 6
```

```r
length(b)
```

```
## [1] 9
```

```r
length(c)
```

```
## [1] 6
```

```r
a-b # Error!
```

```
## Warning in a - b: longitud de objeto mayor no es múltiplo de la longitud de
## uno menor
```

```
## [1]  -5  -5  -5  -5  -5  -5 -11 -11 -11
```

```r
a/c
```

```
## [1] 2.0 1.5 4.0 2.5 6.0 3.5
```

Operaciones relacionales

```r
a > 7
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE
```

```r
b >=5
```

```
## [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```r
a >5 | a <=4
```

```
## [1]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
```

Acceder al vector con operaciones: vector[]

```r
a[a > 7]
```

```
## numeric(0)
```

```r
b[b >=5]
```

```
## [1]  7  8  9 10 11 12 13 14 15
```

```r
a[a >5 | a <=4]
```

```
## [1] 2 3 4 6 7
```

7. Matrices

Las matrices son una estructura con forma rectangular, con renglones y columnas.

Se crean matrices con la función: matrix()

Puede aceptar 2 argumentos:

nrow = número de renglones

ncol = número de columnas

```r
matrix(1:12) # Matriz sin especificar el número de renglones ni de columnas
```

```
##       [,1]
##  [1,]    1
##  [2,]    2
##  [3,]    3
##  [4,]    4
##  [5,]    5
##  [6,]    6
##  [7,]    7
##  [8,]    8
##  [9,]    9
## [10,]   10
## [11,]   11
## [12,]   12
```

R arma las matrices por columnas (hacia abajo)

```r
matrix(1:20, nrow=5)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    2    7   12   17
## [3,]    3    8   13   18
## [4,]    4    9   14   19
## [5,]    5   10   15   20
```

```r
matrix(1:20, nrow=5, ncol=4) # Especificar los 2 argumentos está demás
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    2    7   12   17
## [3,]    3    8   13   18
## [4,]    4    9   14   19
## [5,]    5   10   15   20
```

```r
matrix(1:20, ncol=4)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    2    7   12   17
## [3,]    3    8   13   18
## [4,]    4    9   14   19
## [5,]    5   10   15   20
```

```r
matrix(1:30, ncol=10)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    1    4    7   10   13   16   19   22   25    28
## [2,]    2    5    8   11   14   17   20   23   26    29
## [3,]    3    6    9   12   15   18   21   24   27    30
```

Si quiero que la matriz se llene por renglones debo especificar con el argumento : byrow = TRUE

```r
matrix(1:30,3, byrow = TRUE)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    1    2    3    4    5    6    7    8    9    10
## [2,]   11   12   13   14   15   16   17   18   19    20
## [3,]   21   22   23   24   25   26   27   28   29    30
```

También puedo crear una matriz uniendo vectores:

cbind() para unir vectores, usando cada uno como una columna.

rbind() para unir vectores, usando cada uno como un renglón.

Deben tener la misma longitud

```r
vector_1 <- 1:4
vector_2 <- 5:8
vector_3 <- 9:12
vector_4 <- 13:16

matriz <- rbind(vector_1, vector_2, vector_3, vector_4) # por renglones
matriz
```

```
##          [,1] [,2] [,3] [,4]
## vector_1    1    2    3    4
## vector_2    5    6    7    8
## vector_3    9   10   11   12
## vector_4   13   14   15   16
```

```r
matriz2 <- cbind(vector_1, vector_2, vector_3, vector_4) # por columnas
matriz2
```

```
##      vector_1 vector_2 vector_3 vector_4
## [1,]        1        5        9       13
## [2,]        2        6       10       14
## [3,]        3        7       11       15
## [4,]        4        8       12       16
```

Crear una matriz vacía

```r
aux <- matrix(nrow = 5,ncol=6)
aux
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]   NA   NA   NA   NA   NA   NA
## [2,]   NA   NA   NA   NA   NA   NA
## [3,]   NA   NA   NA   NA   NA   NA
## [4,]   NA   NA   NA   NA   NA   NA
## [5,]   NA   NA   NA   NA   NA   NA
```

Puedo obtener el tamaño de una matriz: dim()

Devuelve: el número de renglones, número de columnas

```r
dim(matriz)
```

```
## [1] 4 4
```

```r
dim(aux)
```

```
## [1] 5 6
```

```r
dim(aux)[1] # Primer elemento: renglones
```

```
## [1] 5
```

```r
columnas <- dim(aux)[1] # Segundo elemento: columnas
```

8. Operaciones con matrices

```r
matriz + 5 # Suma de escalares
```

```
##          [,1] [,2] [,3] [,4]
## vector_1    6    7    8    9
## vector_2   10   11   12   13
## vector_3   14   15   16   17
## vector_4   18   19   20   21
```

```r
matriz * 2 # Multiplicación por un escalar
```

```
##          [,1] [,2] [,3] [,4]
## vector_1    2    4    6    8
## vector_2   10   12   14   16
## vector_3   18   20   22   24
## vector_4   26   28   30   32
```

```r
matriz2/3
```

```
##       vector_1 vector_2 vector_3 vector_4
## [1,] 0.3333333 1.666667 3.000000 4.333333
## [2,] 0.6666667 2.000000 3.333333 4.666667
## [3,] 1.0000000 2.333333 3.666667 5.000000
## [4,] 1.3333333 2.666667 4.000000 5.333333
```

```r
t(matriz2) # Transpuesta
```

```
##          [,1] [,2] [,3] [,4]
## vector_1    1    2    3    4
## vector_2    5    6    7    8
## vector_3    9   10   11   12
## vector_4   13   14   15   16
```

Entre matrices deben tener la misma dimensión

```r
A <- matrix(1:9,ncol=3)
B <- matrix(rep(2,9),nrow=3)
C <- matrix(cbind(c(2,3,4),c(1,1,1),c(12,15,3)),ncol=3)

A+B
```

```
##      [,1] [,2] [,3]
## [1,]    3    6    9
## [2,]    4    7   10
## [3,]    5    8   11
```

```r
B %*% C # Multiplicación de matrices
```

```
##      [,1] [,2] [,3]
## [1,]   18    6   60
## [2,]   18    6   60
## [3,]   18    6   60
```

```r
solve(C) # Inversa de una matriz
```

```
##             [,1]       [,2]        [,3]
## [1,] -0.80000000  0.6000000  0.20000000
## [2,]  3.40000000 -2.8000000  0.40000000
## [3,] -0.06666667  0.1333333 -0.06666667
```

```r
det(C) # Determinante de una matriz
```

```
## [1] 15
```

```r
diag(A+B+C) # Diagonal
```

```
## [1]  5  8 14
```

```r
3*A %*% B %*% t(C)
```

```
##      [,1] [,2] [,3]
## [1,] 1080 1368  576
## [2,] 1350 1710  720
## [3,] 1620 2052  864
```

```r
A %*% A # A^2
```

```
##      [,1] [,2] [,3]
## [1,]   30   66  102
## [2,]   36   81  126
## [3,]   42   96  150
```

Seleccionar valores: matriz[renglones, columnas]

```r
A[1,1] # renglón 1, columna 1
```

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

```r
B[2,] # renglón 2, todas las columnas
```

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

```r
C[,3] # todos los renglones, columna 4
```

```
## [1] 12 15  3
```

Asignar un nuevo valor

```r
B
```

```
##      [,1] [,2] [,3]
## [1,]    2    2    2
## [2,]    2    2    2
## [3,]    2    2    2
```

```r
B[2,3] <- 4
B
```

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

```r
ejemplo <- matrix(c(1/2, 1/4,2/3,6/7),ncol=2)
inv <- solve(ejemplo)
ejemplo %*% inv
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

9. Loops y condicionales

Plantilla básica de un if:

if(condicion lógica){ cuerpo del TRUE }

```r
if(1234**2<exp(35)){
  print("Hola")
}
```

```
## [1] "Hola"
```

Plantilla básica de un if-else:

if(condicion lógica){ cuerpo del TRUE }else{ cuerpo de la negación de la condición lógica }

```r
n <- 7
if(n%%2==0){
  print("n es par")
}else{
  print("n es impar")
}
```

```
## [1] "n es impar"
```

Plantilla básica de un if-else if-. . . -else if-else

if(condicion lógica){ cuerpo del TRUE }else if(condicion lógica que no contenga la primer condición lógica){ cuerpo de la segunda condición lógica }else if(condicion lógica que no contenga a las 2 anteriores){ cuerpo de la tercera condición lógica }.... }else{ En caso de que no se cumpla ninguna condición lógica de las anteriores }

```
n <- 98
if(n%%7==0){
  print("n es multiplo de 7")
} else if(n%%5==0){
  print("n es multiplo de 5")
} else if(n%%2==0){
  print("n es multiplo de 2")
} else{
  print("n no es multiplo de 7, 5 o 2")
}
```

```
## [1] "n es multiplo de 7"
```

Plantilla básica de un for

for(contador in vector de contadores){ Qué hacer para cada uno de los contadores (en orden) }

```
auxiliar <- 1:20
for(i in 1:length(auxiliar)){
  cat("La tercera potencia de ", i, "es ",i^3,"\n") # "\n" se usa para dar enter
}
```

```
## La tercera potencia de  1 es  1
## La tercera potencia de  2 es  8
## La tercera potencia de  3 es  27
## La tercera potencia de  4 es  64
## La tercera potencia de  5 es  125
## La tercera potencia de  6 es  216
## La tercera potencia de  7 es  343
## La tercera potencia de  8 es  512
## La tercera potencia de  9 es  729
## La tercera potencia de  10 es  1000
## La tercera potencia de  11 es  1331
## La tercera potencia de  12 es  1728
## La tercera potencia de  13 es  2197
## La tercera potencia de  14 es  2744
## La tercera potencia de  15 es  3375
## La tercera potencia de  16 es  4096
## La tercera potencia de  17 es  4913
## La tercera potencia de  18 es  5832
## La tercera potencia de  19 es  6859
## La tercera potencia de  20 es  8000
```

Alternativa de lo anterior en R con operaciones con vectores

```
paste("La tercera potencia de ", auxiliar, "es", auxiliar^3)
```

```
##  [1] "La tercera potencia de  1 es 1"
##  [2] "La tercera potencia de  2 es 8"
##  [3] "La tercera potencia de  3 es 27"
##  [4] "La tercera potencia de  4 es 64"
##  [5] "La tercera potencia de  5 es 125"
##  [6] "La tercera potencia de  6 es 216"
##  [7] "La tercera potencia de  7 es 343"
##  [8] "La tercera potencia de  8 es 512"
##  [9] "La tercera potencia de  9 es 729"
## [10] "La tercera potencia de  10 es 1000"
## [11] "La tercera potencia de  11 es 1331"
## [12] "La tercera potencia de  12 es 1728"
## [13] "La tercera potencia de  13 es 2197"
## [14] "La tercera potencia de  14 es 2744"
## [15] "La tercera potencia de  15 es 3375"
## [16] "La tercera potencia de  16 es 4096"
## [17] "La tercera potencia de  17 es 4913"
## [18] "La tercera potencia de  18 es 5832"
## [19] "La tercera potencia de  19 es 6859"
## [20] "La tercera potencia de  20 es 8000"
```

Plantilla básica de un while

contador_inicial while(condición lógica acerca del contador){ Qué hacer para el paso contador en turno contador <- contador + 1 # contador = contador + 1 }

0j0, si la condición lógica no llega a su fin el proceso quedará colgado
Y tendremos que detener el proceso manualmente

```
# contador_inicial = 1
# while (contador_inicial > 0) {
#   print("hola")
#   contador_inicial <-  contador_inicial + 1
# }


contador_inicial = 1
while (contador_inicial > 0 & contador_inicial<15) {
  print("hola")
  contador_inicial <-  contador_inicial + 1
}
```

```
## [1] "hola"
## [1] "hola"
## [1] "hola"
## [1] "hola"
## [1] "hola"
## [1] "hola"
## [1] "hola"
```

```
## [1] "hola"
## [1] "hola"
## [1] "hola"
## [1] "hola"
## [1] "hola"
## [1] "hola"
## [1] "hola"
```

10. Funciones propias de R

Recordemos el vector

```
a
```

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

```
sum(a) # Suma todos los datos
```

```
## [1] 27
```

```
cumsum(a) # Suma acumulada
```

```
## [1]  2  5  9 14 20 27
```

```
cos(a)
```

```
## [1] -0.4161468 -0.9899925 -0.6536436  0.2836622  0.9601703  0.7539023
```

```
cosh(a)
```

```
## [1]   3.762196  10.067662  27.308233  74.209949 201.715636 548.317035
```

```
exp(1)
```

```
## [1] 2.718282
```

```
pi
```

```
## [1] 3.141593
```

```
exp(a*pi)
```

```
## [1] 5.354917e+02 1.239165e+04 2.867513e+05 6.635624e+06 1.535529e+08
## [6] 3.553321e+09
```

```r
mean(a)
```

```
## [1] 4.5
```

Hay funciones con más de un parámetro (no todos se tienen que ocupar)

```r
dado <- 1:6
sample(dado, size = 3, replace = TRUE)
```

```
## [1] 6 3 6
```

```r
sample(dado, 3, TRUE)
```

```
## [1] 5 4 4
```

```r
moneda <- c("águila","sol")
sample(moneda, size = 2, replace = FALSE)
```

```
## [1] "águila" "sol"
```

```r
sample(moneda, size = 6, replace = TRUE)
```

```
## [1] "sol"    "sol"    "águila" "águila" "águila" "águila"
```

LAs funciones se pueden componer

```r
round(mean(dado))
```

```
## [1] 4
```

```r
round(mean(dado), digits = 3)
```

```
## [1] 3.5
```

Otras funciones básicas muy importantes

Fijar semillas: set.seed()

```r
set.seed(27)
```

11. Pedir ayuda

?sample

help("mean")

Usamos doble para buscar coincidencias: ??samp

12. Obtener la ruta de la carpeta

¿Dónde estoy?

Utilizo: getwd() para saber la ruta de la carpeta

Utilizo: setwd() para fijar la ruta

setwd("/home/evelyn/PASE")

13. Creando funciones

Plantilla básica de una función

nombre_de_mi_funcion <- function(parametro1, parametro2, parametro3 = "algo_fijo"){ # Leve descripción de los parámetros o la dinamica de mi función cuerpo de mi función return(valor a regresar) # Situacional }

Nuestra primera función, la media de un vector

```
media_de_un_vector = function(vector){ # vector numérico al cual se le extraera la media
  media <- sum(vector)/length(vector)
  media
}

#usamos la función
v <- c(2,3,4,5,5,6,6,7,6,7)
media_de_un_vector(v)
```

```
## [1] 5.1
```

```
#Comparamos con la de R
mean(v)
```

```
## [1] 5.1
```

¿Sirve return()?

Sí, pero hay que saber cuando lo utilizamos

```
media__y_varianza_de_un_vector = function(vector){ # vector numérico al cual se le extraera la media
  media <- sum(vector)/length(vector)
  media
  varianza <- sum((vector - media)**2)/(length(vector)-1)
  varianza
}

#usamos la función
media__y_varianza_de_un_vector(v)
```

```
## [1] 2.766667
```

```r
#En R la varianza es var()
var(v)
```

```
## [1] 2.766667
```

Notemos que solo nos devolvió la varianza y no la varianza y la media

Para que regrese lo que queremos

```r
media__y_varianza_de_un_vector = function(vector){ # vector numérico al cual se le extraera la media
  media <- sum(vector)/length(vector)
  varianza <- sum((vector - media)**2)/(length(vector)-1)
  return(c(Media = media, Varianza = varianza))
}

# Usamos la función
media__y_varianza_de_un_vector(v)
```

```
##    Media Varianza
## 5.100000 2.766667
```

La función puede devolver una lista

```r
media__y_varianza_de_un_vector = function(vector){ # vector numérico al cual se le extraera la media
  media <- sum(vector)/length(vector)
  varianza <- sum((vector - media)**2)/(length(vector)-1)
  return(list(Media = media, Varianza = varianza))
}

# Usamos la función
media__y_varianza_de_un_vector(v)
```

```
## $Media
## [1] 5.1
##
## $Varianza
## [1] 2.766667
```

Ejemplo de funciones 2

Haremos una función para girar un dado

```r
girar_dado <- function(){
  sample(1:6, 1)
}

# Usar la función
girar_dado()
```

```
## [1] 5
```

¿Cuál es más adecuada a nuestros propósitos?

```
girar_dado_n_veces <- function(n_veces = 1){
  sample(1:6, n_veces) #No hay reemplazo
}

girar_dado_n_veces <- function(n_veces = 1){
  sample(1:6, n_veces, replace = TRUE) # Hay reemplazo
}

# Usamos la función
girar_dado_n_veces(10)
```

```
##  [1] 2 6 1 3 1 1 1 1 1 5
```

Ejemplo 3

Creamos una función especial del valor absoluto

```
absoluto <- function(x){
  if(x>0){
    return(x)
  }else if(x==0){
    print("Es cero")
  } else{
    return(-x)
  }
}

absoluto(7)
```

```
## [1] 7
```