INTRODUCCIÓN AL R

EXPRESIONES ARITMÉTICAS

R puede ser usado como una potente calculadora ya que permite realizar un conjunto amplio de operaciones. R usa los símbolos usuales de adición +, sustracción -, multiplicación *, división / y potencia ^. Los paréntesis pueden ser usados para definir el orden de las operaciones.

```
(1 + 1/100) ^ 100
```

```
## [1] 2.704814
```

```
17/5
```

```
## [1] 3.4
```

```
17 %% 5
```

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

```
17%/%5
```

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

Algunas constantes especiales como exp( ) y pi se encuentran predefinidas en R.

```
exp(1)
```

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

```
pi
```

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

```
sin(pi/6)
```

```
## [1] 0.5
```

```
floor(pi)
```

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

```
ceiling(pi)
```

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

Al igual que en otros lenguajes de programación se pueden almacenar valores en una variable definida por el usuario.

```
x <- 100

(1 + 1/x) ^ x

## [1] 2.704814

y <- (1 + 1/x) ^ x

n <- 1
n <- n + 1
n

## [1] 2
```

Sucesiones

En R existen funciones que sirven para generar sucesiones numéricas.

```
1:10

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

seq(from = 1, to = 9, by = 2)

## [1] 1 3 5 7 9

seq(from = 1, to = 9)

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

seq(1,9,2)

## [1] 1 3 5 7 9

seq(to = 9, from = 1)

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

seq(by = -2, 9, 1)

## [1] 9 7 5 3 1

rep(3, times = 4)

## [1] 3 3 3 3

rep(1:5, times = 4)

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

VECTORES NUMÉRICOS

R utiliza diferentes estructuras de datos. La estructura más simple es el vector, que es una colección ordenada de números.

```
x <- seq(1, 20, by = 2)
x

##  [1]  1  3  5  7  9 11 13 15 17 19

y <- rep(3, times = 4)
y

## [1] 3 3 3 3

z <- c(y, x)
z

##  [1]  3  3  3  3  1  3  5  7  9 11 13 15 17 19
```

Para acceder al elemento i del vector x se usa x[i]. Si los elementos de i son negativos, entonces los valores correspondientes son omitidos.

```
x <- 100:110
x

##  [1] 100 101 102 103 104 105 106 107 108 109 110

x[5]

## [1] 104

i <- c(1, 3, 2)
x[i]

## [1] 100 102 101

j <- c(-1, -2, -3)
x[j]

## [1] 103 104 105 106 107 108 109 110
```

Las operaciones algebraicas sobre vectores actúan sobre cada elemento de forma separada.

```
x <- c(1, 2, 3)
y <- c(4, 5, 6)
x * y

## [1]  4 10 18

x + y

## [1] 5 7 9

y ^ x
```

```
## [1]   4  25 216
```

Cuando se realiza una operación algebraica con dos vectores de diferente longitud, R automáticamente repite el vector más pequeño hasta que tenga la misma longitud del otro vector.

```
c(1, 2, 3, 4) + c(1, 2)
## [1] 2 4 4 6
(1:10) ^ c(1, 2)
##  [1]   1   4   3  16   5  36   7  64   9 100
2 + c(1, 2, 3)
## [1] 3 4 5
2 * c(1, 2, 3)
## [1] 2 4 6
(1:10) ^ 2
##  [1]   1   4   9  16  25  36  49  64  81 100
```

En R existe un conjunto de funciones útiles que usan como argumentos vectores: sum(), prod(), max(), min(), sqrt(), mean(), var(), etc.

```
x <- c(5, 1, 3, 4, 2)
sqrt(x)
## [1] 2.236068 1.000000 1.732051 2.000000 1.414214
sum(x)
## [1] 15
sort(x)
## [1] 1 2 3 4 5
mean(x)
## [1] 3
var(x)
## [1] 2.5
```

VECTORES DE CARACTERES

También es posible definir vectores de caracteres en R.

```
frutas <- c("naranja", "platano", "manzana", "pera")
frutas

## [1] "naranja" "platano" "manzana" "pera"

frutas[3]

## [1] "manzana"

variables <- paste("X", 1:10, sep = "")
variables

##  [1] "X1"  "X2"  "X3"  "X4"  "X5"  "X6"  "X7"  "X8"  "X9"  "X10"

variables <- paste(c("X","Y"), 1:10, sep = "")
variables

##  [1] "X1"  "Y2"  "X3"  "Y4"  "X5"  "Y6"  "X7"  "Y8"  "X9"  "Y10"
```

VECTORES LÓGICOS

Los elementos de un vector lógico solo pueden tomar dos valores: FALSE (falso) y TRUE (verdadero). Estos valores se representan también por F y T.

```
3 < 4

## [1] TRUE

2 + 2 == 5

## [1] FALSE

T == TRUE

## [1] TRUE

x <- 1:20
x %% 4 == 0

##  [1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
## [12]  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE

x <- 1:20
x %% 2 == 0 & x %% 3 == 0

##  [1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
## [12]  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE

x <- 1:20
x %% 2 == 0 | x %% 3 == 0
```

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

```
y <- x[x %% 4 == 0]
y
```

```
## [1]  4  8 12 16 20
```

```
x <- c(1, 8, 3, 4)
x > 2
```

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

```
x[x > 2]
```

```
## [1] 8 3 4
```

```
subset(x, subset = x > 2)
```

```
## [1] 8 3 4
```

```
x <- c(1, 1, 2, 3, 5, 8, 13)
which(x %% 2 == 0)
```

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

MATRICES

Una matriz en R es un arreglo bidimensional que se obtiene usando la función matrix().

```
A <- matrix(1:6, nrow = 2,ncol = 3)
A
```

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

```
A <- matrix(1:6, nrow = 2,ncol = 3, byrow = TRUE)
A
```

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

```
A[1, 3] <- 0
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    0
## [2,]    4    5    6
```

```
A[, 2]
```

```
## [1] 2 5
```

```
A[, 2:3]
```

```
##      [,1] [,2]
## [1,]    2    0
## [2,]    5    6
```

```
B <- diag(c(1, 2, 3))
B
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    2    0
## [3,]    0    0    3
```

Las operaciones usuales actúan elemento por elemento sobre las matrices. Se tienen además algunas funciones sobre matrices como por ejemplo nrow(x), ncol(x), det(x), t(x) y solve(A).

```
A <- matrix(c(3, 5, 2, 3), nrow = 2, ncol = 2)
```

```
B <- matrix(c(1, 1, 0, 1), nrow = 2, ncol = 2)
```

```
A * B
```

```
##      [,1] [,2]
## [1,]    3    0
## [2,]    5    3
```

```
A %*% B
```

```
##      [,1] [,2]
## [1,]    5    2
## [2,]    8    3
```

```
det(A)
```

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

```
solve(A)
```

```
##      [,1] [,2]
## [1,]   -3    2
## [2,]    5   -3
```

```
t(solve(A))
```

```
##      [,1] [,2]
## [1,]   -3    5
## [2,]    2   -3
```

DATOS PERDIDOS

R representa los datos perdidos con el valor NA.

```
a <- NA
is.na(a)

## [1] TRUE

a <- c(11, NA, 13)
is.na(a)

## [1] FALSE  TRUE FALSE

mean(a)

## [1] NA

mean(a, na.rm = TRUE)

## [1] 12
```

Además hay una segunda clase de valores faltantes producidos por las operaciones llamados NaN.

```
0/0

## [1] NaN

Inf - Inf

## [1] NaN
```