Complemento nociones básicas de R

Variables

R no requiere ningún tipo de comando para declarar variables. Sencillamente crea la variable mediante asignación de su valor

```
x <- 3
x
```

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

Una vez declarada la variable podemos utilizar en cálculos

```
x^3
```

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

```
x+5
```

```
## [1] 8
```

Si deseamos cambiar el valor de la variable, solo debemos asignarle un nuevo valor

```
x <- 3+7
x
```

```
## [1] 10
```

```
x^4
```

```
## [1] 10000
```

Vectores

Para representar un nuevo vector de elemento debemos concatenar el vector de la siguiente manera

```
x <- c(1, 4, 9, 2.25, 1/4)
x
```

```
## [1] 1.00 4.00 9.00 2.25 0.25
```

```r
length(x)
```

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

```r
class(x)
```

```
## [1] "numeric"
```

```r
sqrt(x)
```

```
## [1] 1.0 2.0 3.0 1.5 0.5
```

Primeras funciones

```r
class(c)
```

```
## [1] "function"
```

```r
class(length)
```

```
## [1] "function"
```

```r
length
```

```
## function (x)  .Primitive("length")
```

Operaciones sencillas con vectores

```r
x + 1
```

```
## [1]  2.00  5.00 10.00  3.25  1.25
```

```r
y <- 1:10
x + y
```

```
##  [1]  2.00  6.00 12.00  6.25  5.25  7.00 11.00 17.00 11.25 10.25
```

```r
x * y
```

```
##  [1]  1.00  8.00 27.00  9.00  1.25  6.00 28.00 72.00 20.25  2.50
```

```r
x^2
```

```
## [1]  1.0000 16.0000 81.0000  5.0625  0.0625
```

```
x^2 + y^3
```

```
##  [1]    2.0000   24.0000  108.0000   69.0625  125.0625  217.0000
359.0000
##  [8]  593.0000  734.0625 1000.0625
```

```
exp(x)
```

```
## [1]    2.718282   54.598150 8103.083928    9.487736    1.284025
```

```
log(x)
```

```
## [1]  0.0000000  1.3862944  2.1972246  0.8109302 -1.3862944
```

¿Y qué hago cuando necesito ayuda?

Existen varias formas de buscar ayuda en R

- Nombre de la función precedida por signo de interrogación, por ejemplo ?mean
- help(mean)
- example(for)
- help.search("regression") o equivalentemente ??regression
- ??"logistic regression"
- apropos("mean")
- Lo mejor: buscar en inglés lo que se quiere en google y al final la palabra r

Generar vectores con ***seq***

Para ver la ayuda digite *help(seq)*

```
seq(1, 100, by=2)
```

```
##  [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43
45
## [24] 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89
91
```

```
## [47] 93 95 97 99
```

```
seq(1, 100, 10)
```

```
##  [1]  1 11 21 31 41 51 61 71 81 91
```

```
seq(1, 100, length=10)
```

```
##  [1]   1  12  23  34  45  56  67  78  89 100
```

```
x <- seq(1, 100, length=10)
x
```

```
##  [1]   1  12  23  34  45  56  67  78  89 100
```

```
length(x)
```

```
## [1] 10
```

```
y <- seq(2, 100, length=50)
y
```

```
##  [1]   2   4   6   8  10  12  14  16  18  20  22  24  26  28  30  32
34
## [18]  36  38  40  42  44  46  48  50  52  54  56  58  60  62  64  66
68
## [35]  70  72  74  76  78  80  82  84  86  88  90  92  94  96  98 100
```

```
length(y)
```

```
## [1] 50
```

```
z <- c(x, y)
z
```

```
##  [1]   1  12  23  34  45  56  67  78  89 100   2   4   6   8  10  12
14
## [18]  16  18  20  22  24  26  28  30  32  34  36  38  40  42  44  46
48
## [35]  50  52  54  56  58  60  62  64  66  68  70  72  74  76  78  80
82
## [52]  84  86  88  90  92  94  96  98 100
```

```
z + c(1, 2)
```

```
##  [1]   2  14  24  36  46  58  68  80  90 102   3   6   7  10  11  14
15
## [18]  18  19  22  23  26  27  30  31  34  35  38  39  42  43  46  47
50
```

```
## [35]  51  54  55  58  59  62  63  66  67  70  71  74  75  78  79  82
83
## [52]  86  87  90  91  94  95  98  99 102
```

```
z <- c(z, z, z)
z
```

```
##   [1]   1  12  23  34  45  56  67  78  89 100   2   4   6   8  10  12
14
##  [18]  16  18  20  22  24  26  28  30  32  34  36  38  40  42  44  46
48
##  [35]  50  52  54  56  58  60  62  64  66  68  70  72  74  76  78  80
82
##  [52]  84  86  88  90  92  94  96  98 100   1  12  23  34  45  56  67
78
##  [69]  89 100   2   4   6   8  10  12  14  16  18  20  22  24  26  28
30
##  [86]  32  34  36  38  40  42  44  46  48  50  52  54  56  58  60  62
64
## [103]  66  68  70  72  74  76  78  80  82  84  86  88  90  92  94  96
98
## [120] 100   1  12  23  34  45  56  67  78  89 100   2   4   6   8  10
12
## [137]  14  16  18  20  22  24  26  28  30  32  34  36  38  40  42  44
46
## [154]  48  50  52  54  56  58  60  62  64  66  68  70  72  74  76  78
80
## [171]  82  84  86  88  90  92  94  96  98 100
```

```
length(z)
```

```
## [1] 180
```

Generar vectores con ***rep***

Para ver la ayuda digite *help (rep)*

```
rep(1:10, 4)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10  1  2  3  4  5  6  7  8  9 10  1  2
3
## [24]  4  5  6  7  8  9 10  1  2  3  4  5  6  7  8  9 10
```

```
rep(c(1, 2, 3), 10)
```

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

```
rep(c(1, 2, 3), each=10)
```

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

Indexado numérico de vectores

```
x <- seq(1, 100, 2)
x
```

```
##  [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43
45
## [24] 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89
91
## [47] 93 95 97 99
```

```
x[c(1, 2, 3, 4, 5)]
```

```
## [1] 1 3 5 7 9
```

```
x[1:5]
```

```
## [1] 1 3 5 7 9
```

Indexado de vectores con condiciones lógicas

```
condicion <- x>30
condicion
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [23]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [34]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [45]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
class(condicion)
```

```
## [1] “logical”
```

```
x[condicion]
```

```
##  [1] 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73
75
## [24] 77 79 81 83 85 87 89 91 93 95 97 99
```

```
x[x<20]
```

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

```
x[x==9]
```

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

```
x[x!=9]
```

```
##  [1]  1  3  5  7 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45
47
## [24] 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91
93
## [47] 95 97 99
```

Indexado de vectores con *%in%*

```
y <- seq(101, 200, 2)
y %in% c(101, 127, 141)
```

```
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [45] FALSE FALSE FALSE FALSE FALSE FALSE
```

```
y
```

```
##  [1] 101 103 105 107 109 111 113 115 117 119 121 123 125 127 129 131
133
## [18] 135 137 139 141 143 145 147 149 151 153 155 157 159 161 163 165
167
## [35] 169 171 173 175 177 179 181 183 185 187 189 191 193 195 197 199
```

```
y[y %in% c(101, 127, 141)]
```

```
## [1] 101 127 141
```

Indexado de vectores con condiciones múltiples

```
z <- c(x, y)
z
```

```
##   [1]   1   3   5   7   9  11  13  15  17  19  21  23  25  27  29  31
33
##  [18]  35  37  39  41  43  45  47  49  51  53  55  57  59  61  63  65
67
##  [35]  69  71  73  75  77  79  81  83  85  87  89  91  93  95  97  99
101
##  [52] 103 105 107 109 111 113 115 117 119 121 123 125 127 129 131 133
135
##  [69] 137 139 141 143 145 147 149 151 153 155 157 159 161 163 165 167
169
##  [86] 171 173 175 177 179 181 183 185 187 189 191 193 195 197 199
```

```
z>150
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [45] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [56] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [67] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
##  [78]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [89]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [100]  TRUE
```

```
z[z>150]
```

```
##  [1] 151 153 155 157 159 161 163 165 167 169 171 173 175 177 179 181
183
## [18] 185 187 189 191 193 195 197 199
```

```
z[z<30 | z>150]
```

```
##  [1]   1   3   5   7   9  11  13  15  17  19  21  23  25  27  29 151
153
## [18] 155 157 159 161 163 165 167 169 171 173 175 177 179 181 183 185
187
## [35] 189 191 193 195 197 199
```

```
z[z>=30 & z<=150]
```

```
##  [1]  31  33  35  37  39  41  43  45  47  49  51  53  55  57  59  61
63
## [18]  65  67  69  71  73  75  77  79  81  83  85  87  89  91  93  95
97
## [35]  99 101 103 105 107 109 111 113 115 117 119 121 123 125 127 129
131
## [52] 133 135 137 139 141 143 145 147 149
```

```
z[c(1, 10, 40, 80)]
```

```
## [1]   1  19  79 159
```

```
cond  <-  (x>10) & (x<50)
cond
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [12]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [23]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [45] FALSE FALSE FALSE FALSE FALSE FALSE
```

```
cond  <-  (x>=10) & (x<=50)
cond
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [12]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [23]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [45] FALSE FALSE FALSE FALSE FALSE FALSE
```

```
x[cond]
```

```
##  [1] 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
```

Con las condiciones se pueden hacer operaciones

```
sum(cond)
```

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

```
!cond
```

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

```
## [34]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [45]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
sum(!cond)
```

```
## [1] 30
```

```
length(x[cond])
```

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

```
length(x[!cond])
```

```
## [1] 30
```

```
as.numeric(cond)
```

```
##  [1] 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0
## [36] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Funciones predefinidas

```
summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     1.0    25.5    50.0    50.0    74.5    99.0
```

```
mean(x)
```

```
## [1] 50
```

```
sd(x)
```

```
## [1] 29.15476
```

```
median(x)
```

```
## [1] 50
```

```
max(x)
```

```
## [1] 99
```

```
min(x)
```

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

```
range(x)
```

```
## [1]  1 99
```

```
quantile(x)
```

```
##   0%  25%  50%  75% 100%
##  1.0 25.5 50.0 74.5 99.0
```

Matrices

Construir una matriz

Para la ayuda de la función escriba *help (matrix)*

```
z <- 1:12
M  <-  matrix(z, nrow=3)
M
```

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

```
z
```

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

```
class(M)
```

```
## [1] "matrix"
```

```
dim(M)
```

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

```
summary(M)
```

```
##        V1            V2            V3            V4
##  Min.   :1.0   Min.   :4.0   Min.   :7.0   Min.   :10.0
##  1st Qu.:1.5   1st Qu.:4.5   1st Qu.:7.5   1st Qu.:10.5
##  Median :2.0   Median :5.0   Median :8.0   Median :11.0
##  Mean   :2.0   Mean   :5.0   Mean   :8.0   Mean   :11.0
```

```
##  3rd Qu.:2.5   3rd Qu.:5.5   3rd Qu.:8.5   3rd Qu.:11.5
##  Max.   :3.0   Max.   :6.0   Max.   :9.0   Max.   :12.0
```

Matrices a partir de vectores: rbind y cbind

```
x <- 1:10
y <- 1:10
z <- 1:10
z <- y <- x <- 1:10

M <- cbind(x, y, z)
M
```

```
##        x  y  z
##  [1,]  1  1  1
##  [2,]  2  2  2
##  [3,]  3  3  3
##  [4,]  4  4  4
##  [5,]  5  5  5
##  [6,]  6  6  6
##  [7,]  7  7  7
##  [8,]  8  8  8
##  [9,]  9  9  9
## [10,] 10 10 10
```

```
M <- rbind(x, y, z)
M
```

```
##   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## x    1    2    3    4    5    6    7    8    9    10
## y    1    2    3    4    5    6    7    8    9    10
## z    1    2    3    4    5    6    7    8    9    10
```

```
rbind(M, M)
```

```
##   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## x    1    2    3    4    5    6    7    8    9    10
## y    1    2    3    4    5    6    7    8    9    10
## z    1    2    3    4    5    6    7    8    9    10
## x    1    2    3    4    5    6    7    8    9    10
## y    1    2    3    4    5    6    7    8    9    10
## z    1    2    3    4    5    6    7    8    9    10
```

```
cbind(M, M)
```

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

Transponer una matriz

```
M
```

```
##   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## x    1    2    3    4    5    6    7    8    9    10
## y    1    2    3    4    5    6    7    8    9    10
## z    1    2    3    4    5    6    7    8    9    10
```

```
t(M)
```

```
##        x  y  z
##  [1,]  1  1  1
##  [2,]  2  2  2
##  [3,]  3  3  3
##  [4,]  4  4  4
##  [5,]  5  5  5
##  [6,]  6  6  6
##  [7,]  7  7  7
##  [8,]  8  8  8
##  [9,]  9  9  9
## [10,] 10 10 10
```

```
class(t)
```

```
## [1] “function”
```

```
dim(t(M))
```

```
## [1] 10  3
```

Multiplicación entre matrices

Para multiplicar entre matrices utilice lo siguiente **%*%:**

```
M*M
```

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

```
M%*%t(M)
```

```
##     x   y   z
## x 385 385 385
## y 385 385 385
## z 385 385 385
```

Operaciones con matrices: funciones predefinidas

```
sum(M)
```

```
## [1] 165
```

```
rowSums(M)
```

```
##  x  y  z
## 55 55 55
```

```
colSums(M)
```

```
##  [1]  3  6  9 12 15 18 21 24 27 30
```

```
rowMeans(M)
```

```
##   x   y   z
## 5.5 5.5 5.5
```

```
colMeans(M)
```

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

La función **apply**

Para mayor información utilice lo siguiente *help (apply)*

```
apply(M, 1, sum)
```

```
##  x  y  z
## 55 55 55
```

```
apply(M, 2, sum)
```

```
##  [1]  3  6  9 12 15 18 21 24 27 30
```

```
apply(M, 1, mean)
```

```
##   x   y   z
## 5.5 5.5 5.5
```

```
apply(M, 2, mean)
```

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

```
apply(M, 1, sd, na.rm=TRUE)
```

```
##       x       y       z
## 3.02765 3.02765 3.02765
```

```
apply(M, 2, sd)
```

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

Indexado de matrices

```
M
```

```
##   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## x    1    2    3    4    5    6    7    8    9    10
## y    1    2    3    4    5    6    7    8    9    10
## z    1    2    3    4    5    6    7    8    9    10
```

```
M[1, ]
```

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

```
M[, 1]
```

```
## x y z
## 1 1 1
```

```
M[1:2, ]
```

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

```
M[1:2, 2:3]
```

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

```
M[1, c(1, 4)]
```

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

```
M[-1,]
```

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

```
M[-c(1, 2),]
```

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

Valores Perdidos

Un valor perdido se denota como **NA** *('Not Available' / Missing Values)*

```
class(NA)
```

```
## [1] "logical"
```

```
x <- rnorm(100)
idx <- sample(length(x), 10)
idx
```

```
##  [1] 23  7 34 41 66 56 79 42 99 67
```

```
x[idx]
```

```
##  [1]  0.48140246 -1.25438824  0.43637005  0.10170100 -0.76756173
##  [6]  0.51871286 -0.04006789 -0.34833641 -1.02026599 -0.34850833
```

```
x2 <- x
x2[idx] <- NA
x2
```

```
##   [1]  2.460491400 -0.989672187 -0.733562158  0.446869438 -0.197429659
##   [6] -2.089056325           NA  2.003798474  2.879552568  0.942935009
##  [11] -0.146436980  0.293767454  2.376869437 -0.709277219 -0.663822788
##  [16] -0.552640828  1.114521816  0.275690979 -0.476080386 -0.423466980
##  [21] -0.055346313 -1.220338566           NA  0.203108405 -0.252088015
##  [26]  0.414756468 -0.674746617  0.968628374  0.391101042  0.740616795
##  [31]  0.148390378 -0.724031294  1.584410433           NA -1.970245136
##  [36]  0.312395375 -0.550137672  0.078965341  0.226975622 -0.714285714
##  [41]           NA           NA -0.468640864  0.132528834  0.864668985
##  [46] -0.109778264  0.462969932  0.198948808 -0.914050237 -0.427736403
##  [51] -0.925615008  0.445396509 -0.067783435  0.479883253 -0.272116990
##  [56]           NA -1.727086193  1.339306531  1.148439872 -0.007695203
##  [61] -1.089754411  0.430632420  0.098041925 -0.607280524  1.594838531
##  [66]           NA           NA  0.432061797  0.040583975  0.091827187
##  [71] -1.087568794  0.236805285 -1.297783394 -0.127382414  0.578551249
##  [76] -1.845351783  0.211207141  0.061888953           NA -1.173977016
##  [81]  1.707006915 -0.610353010  0.544673138  1.267711209 -0.519728338
##  [86]  2.123144597  0.835974335 -1.798179950  0.512522698 -0.905948078
##  [91] -2.534751466  0.305806101  1.890898979 -0.123456187 -0.630145362
##  [96]  1.424617821 -0.224555669 -0.971919271           NA -0.558528015
```

NA en las funciones

Cuando en los objetos hay valores **NA** las funciones no trabajan de forma adecuada.

```
summary(x)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
```

```
## -2.53475 -0.63856  0.01644  0.00914  0.46720  2.87955
```

```
mean(x)
```

```
## [1] 0.009140085
```

```
sum(x)
```

```
## [1] 0.9140085
```

```
summary(x2)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.     NA's
## -2.53475 -0.62520  0.05124  0.03506  0.47565  2.87955       10
```

```
mean(x2)
```

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

```
sum(x2)
```

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

Para ello se sugiere revisar las opciones de las funciones, por ejemplo

```
mean(x2, na.rm=TRUE)
```

```
## [1] 0.03505501
```

```
sum(x2, na.rm=TRUE)
```

```
## [1] 3.154951
```

```
sd(x2, na.rm=TRUE)
```

```
## [1] 1.043517
```

Funciones

Definición de funciones

Para definir una función usamos la función `function`.

Forma general

```
NombreDeFuncion <-function(arg 1, arg 2, ...) expresión
```

Ejemplo

```
myFun <- function(x, y) x + y
myFun(3, 4)
```

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

```
class(myFun)
```

```
## [1] "function"
```

También se puede definir una función a partir de otras funciones

```
foo  <-  function(x, ...){
  mx <- mean(x, ...)
  medx <- median(x, ...)
  sdx <- sd(x, ...)
  c(mx, medx, sdx)
}

foo(1:10) # Función que calcula la media, mediana y la desviación estándar
```

```
## [1] 5.50000 5.50000 3.02765
```

Lo anterior también funciona con matrices

```
M <- matrix(c(1:30),nrow = 3,byrow = TRUE)
M
```

```
##      [,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
```

```r
apply(M, 1, foo) # Aplicando la función foo en forma fila
```

```
##         [,1]     [,2]     [,3]
## [1,] 5.50000 15.50000 25.50000
## [2,] 5.50000 15.50000 25.50000
## [3,] 3.02765  3.02765  3.02765
```

```r
apply(M, 2, foo) # Aplicando la función foo en forma columna
```

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

Funciones con valores predeterminados

```r
function.suma<-function(A=10,B=5) A+B
function.suma()
```

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

```r
function.suma(10,3)
```

```
## [1] 13
```

```r
x
```

```
##   [1]  2.460491400 -0.989672187 -0.733562158  0.446869438 -0.197429659
##   [6] -2.089056325 -1.254388240  2.003798474  2.879552568  0.942935009
##  [11] -0.146436980  0.293767454  2.376869437 -0.709277219 -0.663822788
##  [16] -0.552640828  1.114521816  0.275690979 -0.476080386 -0.423466980
##  [21] -0.055346313 -1.220338566  0.481402461  0.203108405 -0.252088015
##  [26]  0.414756468 -0.674746617  0.968628374  0.391101042  0.740616795
##  [31]  0.148390378 -0.724031294  1.584410433  0.436370045 -1.970245136
##  [36]  0.312395375 -0.550137672  0.078965341  0.226975622 -0.714285714
##  [41]  0.101701005 -0.348336413 -0.468640864  0.132528834  0.864668985
##  [46] -0.109778264  0.462969932  0.198948808 -0.914050237 -0.427736403
##  [51] -0.925615008  0.445396509 -0.067783435  0.479883253 -0.272116990
##  [56]  0.518712855 -1.727086193  1.339306531  1.148439872 -0.007695203
##  [61] -1.089754411  0.430632420  0.098041925 -0.607280524  1.594838531
##  [66] -0.767561732 -0.348508327  0.432061797  0.040583975  0.091827187
##  [71] -1.087568794  0.236805285 -1.297783394 -0.127382414  0.578551249
##  [76] -1.845351783  0.211207141  0.061888953 -0.040067888 -1.173977016
```

```
##  [81]  1.707006915 -0.610353010  0.544673138  1.267711209 -0.519728338
##  [86]  2.123144597  0.835974335 -1.798179950  0.512522698 -0.905948078
##  [91] -2.534751466  0.305806101  1.890898979 -0.123456187 -0.630145362
##  [96]  1.424617821 -0.224555669 -0.971919271 -1.020265985 -0.558528015
```

```
function.suma(x)
```

```
##   [1] 7.460491 4.010328 4.266438 5.446869 4.802570 2.910944 3.745612
##   [8] 7.003798 7.879553 5.942935 4.853563 5.293767 7.376869 4.290723
##  [15] 4.336177 4.447359 6.114522 5.275691 4.523920 4.576533 4.944654
##  [22] 3.779661 5.481402 5.203108 4.747912 5.414756 4.325253 5.968628
##  [29] 5.391101 5.740617 5.148390 4.275969 6.584410 5.436370 3.029755
##  [36] 5.312395 4.449862 5.078965 5.226976 4.285714 5.101701 4.651664
##  [43] 4.531359 5.132529 5.864669 4.890222 5.462970 5.198949 4.085950
##  [50] 4.572264 4.074385 5.445397 4.932217 5.479883 4.727883 5.518713
##  [57] 3.272914 6.339307 6.148440 4.992305 3.910246 5.430632 5.098042
##  [64] 4.392719 6.594839 4.232438 4.651492 5.432062 5.040584 5.091827
##  [71] 3.912431 5.236805 3.702217 4.872618 5.578551 3.154648 5.211207
##  [78] 5.061889 4.959932 3.826023 6.707007 4.389647 5.544673 6.267711
##  [85] 4.480272 7.123145 5.835974 3.201820 5.512523 4.094052 2.465249
##  [92] 5.305806 6.890899 4.876544 4.369855 6.424618 4.775444 4.028081
##  [99] 3.979734 4.441472
```

```
y
```

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

```
function.suma(,y)
```

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

```
function.suma(x[1:length(y)],y)
```

```
##  [1]  3.460491  1.010328  2.266438  4.446869  4.802570  3.910944
5.745612
##  [8] 10.003798 11.879553 10.942935
```

Listas y data.frame

Una lista es un objeto con diferentes entradas. Para crear una lista usamos la función list

```
dias.semana = c("Lunes" ,"Martes", "Miercoles", "Jueves","Viernes", "Sabado",
"Domingo")
dias.semana
```

```
## [1] “Lunes”     “Martes”    “Miercoles” “Jueves”    “Viernes”
“Sabado”
## [7] “Domingo”
```

```
lista<-list(A=dias.semana,B=1:7,C=matrix(1:8,2))
lista
```

```
## $A
## [1] “Lunes”     “Martes”    “Miercoles” “Jueves”    “Viernes”
“Sabado”
## [7] “Domingo”
##
## $B
## [1] 1 2 3 4 5 6 7
##
## $C
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
```

```
class(lista)
```

```
## [1] “list”
```

Para acceder a las diferentes entradas se utiliza lo siguiente:

Por su nombre

```
lista
```

```
## $A
## [1] “Lunes”     “Martes”    “Miercoles” “Jueves”    “Viernes”
“Sabado”
## [7] “Domingo”
##
## $B
## [1] 1 2 3 4 5 6 7
##
## $C
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
```

```
lista$A
```

```
## [1] “Lunes”     “Martes”    “Miercoles” “Jueves”    “Viernes”
“Sabado”
## [7] “Domingo”
```

```
lista$B
```

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

```
lista$C
```

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

O por su índice

```
lista[1]
```

```
## $A
## [1] “Lunes”     “Martes”    “Miercoles” “Jueves”    “Viernes”
“Sabado”
## [7] “Domingo”
```

```
lista[[1]]
```

```
## [1] “Lunes”     “Martes”    “Miercoles” “Jueves”    “Viernes”
“Sabado”
## [7] “Domingo”
```

```
class(lista[1])
```

```
## [1] “list”
```

```
class(lista[[1]])
```

```
## [1] “character”
```

```
lista[2]
```

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

```
lista[[2]]
```

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

```
class(lista[2])

## [1] “list”

class(lista[[2]])

## [1] “integer”
```

Cada elemento es diferente

```
class(lista)

## [1] “list”

class(lista$A)

## [1] “character”

class(lista$B)

## [1] “integer”

class(lista$C)

## [1] “matrix”
```

Para matrices `apply`, para listas `lapply` y `sapply`

```
lapply(lista, length)

## $A
## [1] 7
##
## $B
## [1] 7
##
## $C
## [1] 8

sapply(lista, length)

## A B C
## 7 7 8
```

```r
lista <- list(x = 1:10,
              y = seq(0, 10, 2),
              z = rnorm(30))
lista
```

```
## $x
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $y
## [1]  0  2  4  6  8 10
##
## $z
##  [1] -0.41658611 -0.83013640  1.10167786  0.68325583  0.49594178
##  [6] -0.07426454 -0.23379710 -1.33677471  0.06467726 -0.74572178
## [11] -1.62684762  2.28484463  0.02984230  0.30768145  0.84309814
## [16]  1.15550167 -0.07825055 -0.39122516 -0.21332397  0.35319109
## [21] -0.53319250  0.31669720 -0.40276853  0.16259041 -0.85742955
## [26]  1.39756270 -0.98746295  1.29900422 -0.58351906  0.45349357
```

```r
lapply(lista, sum)
```

```
## $x
## [1] 55
##
## $y
## [1] 30
##
## $z
## [1] 1.63776
```

```r
lapply(lista, median)
```

```
## $x
## [1] 5.5
##
## $y
## [1] 5
##
## $z
## [1] -0.02221112
```

```
lapply(lista, foo)
```

```
## $x
## [1] 5.50000 5.50000 3.02765
##
## $y
## [1] 5.000000 5.000000 3.741657
##
## $z
## [1]  0.05459199 -0.02221112  0.86931560
```

Para crear un data.frame

```
df <- data.frame(x = 1:10,
                 y = rnorm(10),
                 z = 0)
length(df)
```

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

```
dim(df)
```

```
## [1] 10  3
```

Para acceder a los elementos se realiza lo siguiente:

Por su nombre

```
df$x
```

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

```
df$y
```

```
##  [1] -0.4148927  0.3277869 -0.9386344 -0.2912153  2.2444974 -1.1127245
##  [7] -0.2061698  0.6014926  0.9038058 -1.9882452
```

```
df$z
```

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

Por su índice

```
df
```

```
##     x          y z
## 1   1 -0.4148927 0
## 2   2  0.3277869 0
## 3   3 -0.9386344 0
## 4   4 -0.2912153 0
## 5   5  2.2444974 0
## 6   6 -1.1127245 0
## 7   7 -0.2061698 0
## 8   8  0.6014926 0
## 9   9  0.9038058 0
## 10 10 -1.9882452 0
```

```
df[1,]
```

```
##   x          y z
## 1 1 -0.4148927 0
```

```
df[,1]
```

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

```
df[,2]
```

```
##  [1] -0.4148927  0.3277869 -0.9386344 -0.2912153  2.2444974 -1.1127245
##  [7] -0.2061698  0.6014926  0.9038058 -1.9882452
```

Construyendo una base

```
year <- 2011
month <- 1:12
class <- c('A', 'B', 'C')
vals <- rnorm(12)
dats <- data.frame(year, month, class, vals)
dats
```

```
##    year month class       vals
## 1  2011     1     A  0.8815282
## 2  2011     2     B -0.9953855
## 3  2011     3     C -1.6119932
## 4  2011     4     A  1.3041921
```

```
## 5  2011     5     B  0.6888027
## 6  2011     6     C -1.0100371
## 7  2011     7     A  3.0549135
## 8  2011     8     B -0.3445440
## 9  2011     9     C -1.0215725
## 10 2011    10     A -0.6462395
## 11 2011    11     B -0.5435808
## 12 2011    12     C -0.2551541
```

La función expand.grid

La función expand.grid es muy util para completar bases de datos

```
df <- expand.grid(edad=c(36,25), peso=c(75,60), sexo=c("Hombre","Mujer"))
head(df)
```

```
##   edad peso   sexo
## 1   36   75 Hombre
## 2   25   75 Hombre
## 3   36   60 Hombre
## 4   25   60 Hombre
## 5   36   75  Mujer
## 6   25   75  Mujer
```

```
tail(df)
```

```
##   edad peso   sexo
## 3   36   60 Hombre
## 4   25   60 Hombre
## 5   36   75  Mujer
## 6   25   75  Mujer
## 7   36   60  Mujer
## 8   25   60  Mujer
```

```
summary(df)
```

```
##       edad           peso          sexo
##  Min.   :25.0   Min.   :60.0   Hombre:4
##  1st Qu.:25.0   1st Qu.:60.0   Mujer :4
##  Median :30.5   Median :67.5
##  Mean   :30.5   Mean   :67.5
##  3rd Qu.:36.0   3rd Qu.:75.0
```

```
##  Max.   :36.0   Max.   :75.0
```

```
dim(df)
```

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

```
names(df)
```

```
## [1] "edad" "peso" "sexo"
```

Seleccionando un subconjunto del data.frame

```
datos1<-data.frame(Peso=c(90,120,56), Altura=c(1.90,1.87,1.70))
datos2<-data.frame(datos1,Sexo=c("Hombre","Hombre","Mujer"))
datos2$Nombres<-c("Pepe","Paco","Pepita")
datos2
```

```
##   Peso Altura   Sexo Nombres
## 1   90   1.90 Hombre    Pepe
## 2  120   1.87 Hombre    Paco
## 3   56   1.70  Mujer  Pepita
```

```
subset(datos2,select=c(Sexo,Nombres))
```

```
##     Sexo Nombres
## 1 Hombre    Pepe
## 2 Hombre    Paco
## 3  Mujer  Pepita
```

```
subset(datos2,subset=c(Sexo=="Mujer"))
```

```
##   Peso Altura  Sexo Nombres
## 3   56    1.7 Mujer  Pepita
```

Modificando un data.frame

```
transform(datos2,logPeso=log(Peso))
```

```
##   Peso Altura   Sexo Nombres  logPeso
## 1   90   1.90 Hombre    Pepe 4.499810
## 2  120   1.87 Hombre    Paco 4.787492
## 3   56   1.70  Mujer  Pepita 4.025352
```

```
transform(datos2,IMC=Peso/(Altura)^2)
```

```
##   Peso Altura   Sexo Nombres      IMC
## 1   90   1.90 Hombre    Pepe 24.93075
## 2  120   1.87 Hombre    Paco 34.31611
## 3   56   1.70  Mujer  Pepita 19.37716
```

Elementos para programar

Bucles

Operadores de control de flujo

```
if(cond) expr
if(cond) cons.expr  else  alt.expr

for(var in seq) expr
while(cond) expr
repeat expr
break
next
```

De un vector de valores aleatorios normales, elevar al cuadrado cada uno de ellos.

```
for(n in c(2,5,10,20,50)) {
  x <- rnorm(n)
  cat(n,":", sum(x^2),"\n")
}
```

```
## 2 : 0.6625061
## 5 : 3.575397
## 10 : 9.409962
## 20 : 25.53235
## 50 : 50.6243
```

Del listado de números del-5 a 5, elevar a la tres cada término.

```
for(i in -5:5)
  {
  cat(i,"\t", i^3,"\n")
}
```

```
## -5    -125
## -4    -64
## -3    -27
## -2    -8
## -1    -1
## 0     0
## 1     1
## 2     8
## 3     27
## 4     64
## 5     125
```

De un vector de valores aleatorios normales asignarles 1 aquellos valores mayores que 0.

```
x <- rnorm(10)
x2 <- as.numeric(x>0)
for (i in 1:length(x2)){
  if (x[i]<0) x2[i] <- 0 else x2[i] <- 1
}
cbind(x, x2)
```

```
##                x x2
##  [1,] -1.4392661  0
##  [2,]  0.5533265  1
##  [3,]  0.7663564  1
##  [4,] -0.1669569  0
##  [5,]  0.5184635  1
##  [6,]  1.0587377  1
##  [7,] -0.9082484  0
##  [8,] -1.0661614  0
##  [9,] -0.3008825  0
## [10,] -1.1290116  0
```

Otra forma

```
x <- rnorm(10)
ifelse(x>0, 1, 0)
```

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

Ejemplos

Empezando con elevar al cuadrado cada valor hasta que:

```
i<-4
while(i<=10){
  print(c(i,i^2))
  i=i+1
}
```

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

Otra forma

```
i<-4
repeat{print(c(i,i^2))
  i=i+1
  if(i==10)break
}
```

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

Condiciones

Operadores lógicos

```
! x
x & y
x && y
x | y
x || y
xor(x, y)
```

Operadores de sintaxis

```
:: :::   #access variables in a namespace
$ @  #component / slot extraction
[ [[     #indexing
^    #exponentiation (right to left)
- +  #unary minus and plus
:    #sequence operator
%any%    #special operators (including %% and %/%)
* /  #multiply, divide
+ -  #(binary) add, subtract
< > <= >= == !=  #ordering and comparison
!    #negation
& &&     #and
| ||     #or
~    #as in formulae
-> ->>   #rightwards assignment
<- <<-   #assignment (right to left)
=    #assignment (right to left)
?    #help (unary and binary)
```

Ejemplos

Creamos una función que calcule el logaritmo de un número:

```
logaritmo<-function(x){
 if(is.numeric(x)&& min(x)!=0)
log(x)
else{stop("x no es numérico o es cero")}
}

logaritmo(3)

## [1] 1.098612

logaritmo(10)

## [1] 2.302585

logaritmo(exp(1))

## [1] 1
```

Creamos una función que calcule el inverso de un número

```
Inverso<-function(x) ifelse(x==0,NA,1/x)

Inverso(-2:3)

## [1] -0.5000000 -1.0000000         NA  1.0000000  0.5000000  0.3333333

Inverso(-10:10)

##  [1] -0.1000000 -0.1111111 -0.1250000 -0.1428571 -0.1666667 -0.2000000
##  [7] -0.2500000 -0.3333333 -0.5000000 -1.0000000         NA  1.0000000
## [13]  0.5000000  0.3333333  0.2500000  0.2000000  0.1666667  0.1428571
## [19]  0.1250000  0.1111111  0.1000000
```

Factorial de un número

```
factorial<-function(n){
  f<-1
  if(n>1)
    for(i in 1:n)
      f<-f*i
    return(f)
}

factorial(3)

## [1] 6

factorial(25)

## [1] 1.551121e+25

factorial(0)

## [1] 1
```

Progresión aritmética

```
#Formula explícita
arit.1<-function(n=1,a1=1,d=1) a1+d*(n-1)

#Formula recursiva
arit.2<-function(n=1,a1=1,d=1){
  a<- numeric(n)
  a[1]<- a1
  if(n>1){
    for(i in 2:n) a[i]=a[i-1]+d
  }
  return(a[n])
}

#Formula vectorial
arit.3<-function(n=1,a1=1,d=1){
  A1<-rep(a1,n)
  D<-rep(d,n)
  N<-0:(n-1)
  A<-A1+N*D
  return(A[n])
}
```