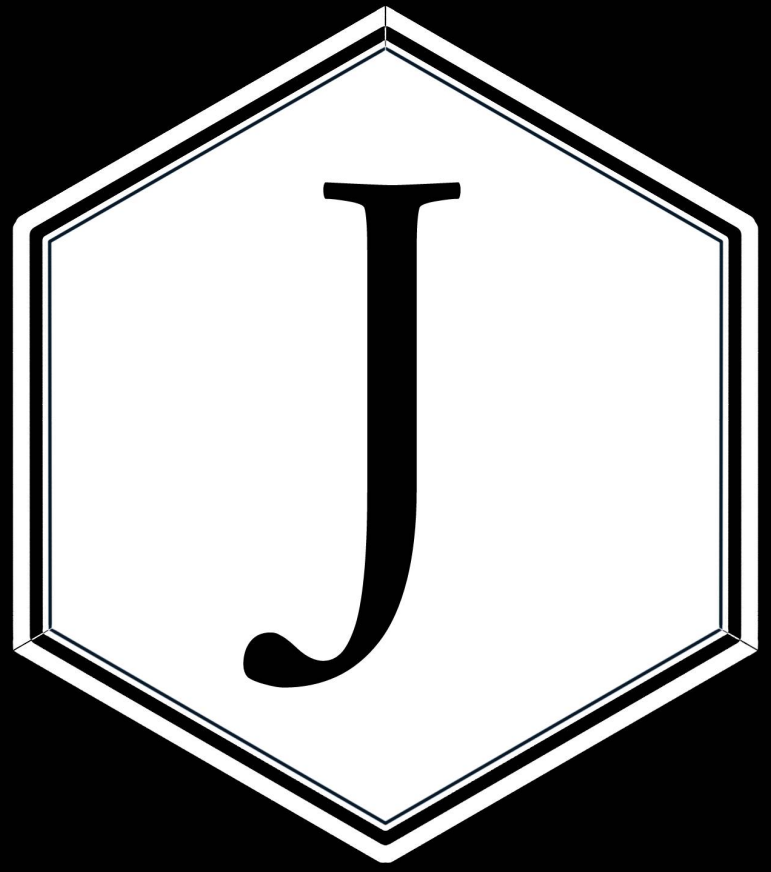
R

Breve introduccion al lenguaje

Jhon

Fecha: 2020-07-27



Funciones basicas

```
# + - / *
# suma, resta, division, multiplicacion

1 + 2
```

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

```
# exp(2), 3^2, sqrt(2), log(2), log10(2)
```

Lo anterior se traduce como:

$$e^2, 3^2, \sqrt{2}, ln(2), log(2)$$

Variables

```r
# Anteriormente se sugirio el uso de `%>%`
# crear x con el valor de 12
x <- 12

# crear y con el valor de 8

y <- 8
```

`c()` funcion

Esta funcion se puede entender como crear

```r
# Creacion de un vector con los elementos 1, 2, 3, 4
elementos <- c(1, 2, 3, 4)

# Lo anterior se puede escribir de varias maneras:
elementos <- 1:4
c(1, 2, 3, 4) -> elementos
```

Extraccion de elementos de un vector `[i]`

[i] indica la posicion del elemento que queremos.

```
# Vector
celular_precios <- c(120, 130, 400, 500)

# Extraer el primer elemento del vector mediante
celular_precios[1]
```

```
## [1] 120
```

```
# Extraer todos menos el primer elemento
celular_precios[-1]
```

```
## [1] 130 400 500
```

```
# Con `c()` extraer el elemento 1 y 4
celular_precios[c(1,4)]
```

```
## [1] 120 500
```

Uniendo todo

Operaciones con variables

```
x <- 12
y <- 8
x - y
```

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

Operaciones con vectores

```
vector_1 <- 6:1
2 * vector_1
```

```
## [1] 12 10  8  6  4  2
```

Operaciones con vectores

```
# 6 elementos
vector_1 <- c(1, 2, 3, 4, 5, 6)
vector_2 <- 6:1
# Suma Vectores

vector_1 + vector_2
```

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

```
# Multiplicacion
vector_1 * vector_2
```

```
## [1]  6 10 12 12 10  6
```

Tipo de variables

typeof() funcion

Character, String

```
x <- "Hola mundo, soy Jhon"
typeof(x)
```

```
## [1] "character"
```

Numeric-Double (Reales o decimales)

```
x <- 12.12
typeof(x)
```

```
## [1] "double"
```

Integer

```
x <- 1:12
typeof(x)
```

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

Logical

```
y <- c(TRUE, TRUE, FALSE, FALSE)
typeof(y)
```

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

```
typeof(as.numeric(y))
```

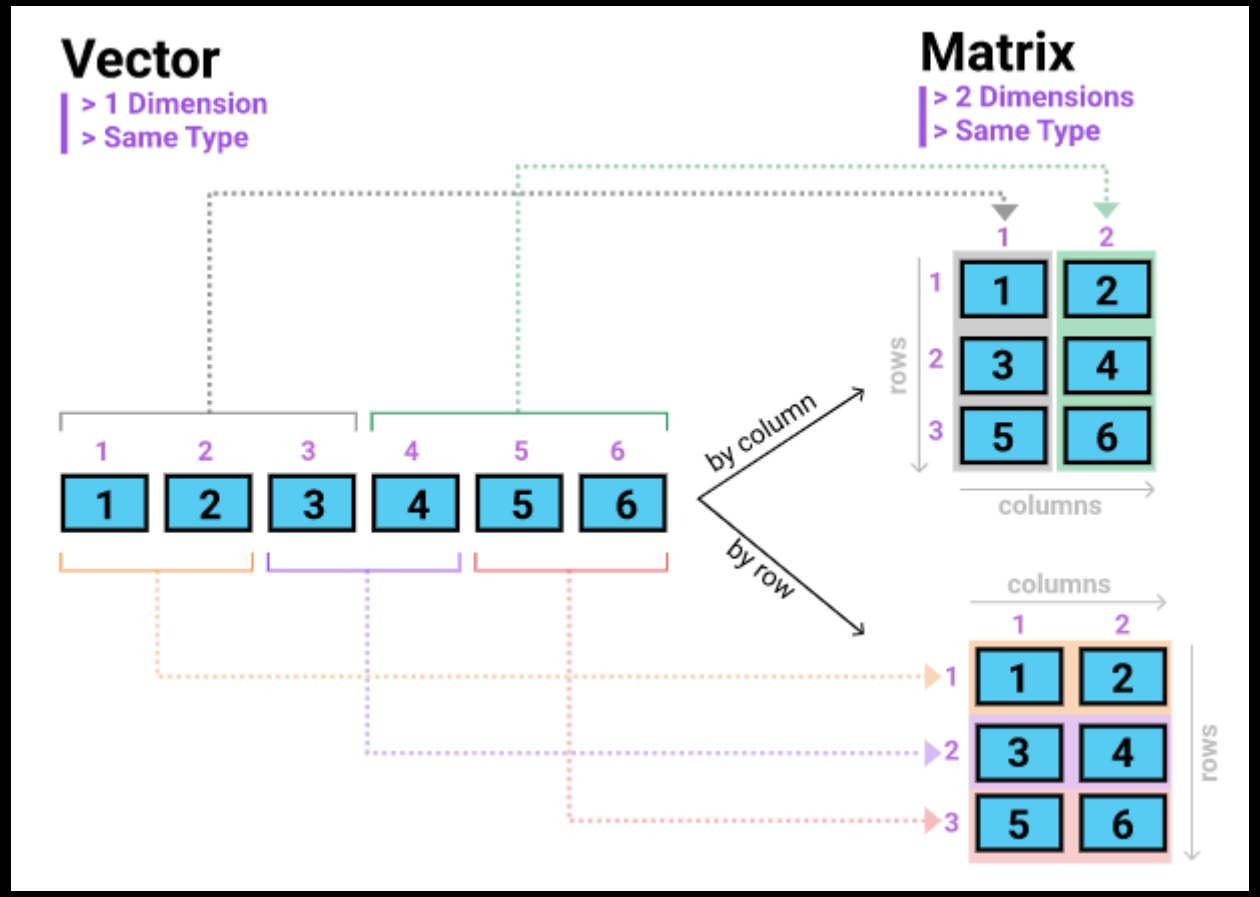
```
## [1] "double"
```

Mezcla de elementos

```
choco <- c(1, 3, 4 , TRUE, FALSE, "ALGO mas")
#typeof(choco)
```

Datos

Matrices


Vector
> 1 Dimension
> Same Type
Matrix
> 2 Dimensions
> Same Type
1 2 3 4 5 6
1 2 3 4 5 6
by column
by row
1 2
1 1 2
2 3 4
3 5 6
rows
columns
columns
1 2
1 1 2
2 3 4
3 5 6
rows

Creacion de matrices

matrix() Function

```
# Vector
v <- 1:12
# Matrix

matrix(v)
```

```
##       [,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
# 2 columnas de arriba abajo
matrix(v, ncol = 2)
```

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

```r
# 2 columnas izquierda derecha
matrix(v, ncol = 2, byrow = T)
```

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

Data frames

```r
# Vectores

a1 <- c(1:12)
a2 <- c(12:1)
a3 <- c(4:15)
a4 <- c(20:9)
# rbind union por filas
# rbind == t(cbind)
valores_r <- rbind(a1, a2, a3, a4)
valores_r
```

```
##    [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
## a1    1    2    3    4    5    6    7    8    9    10    11    12
## a2   12   11   10    9    8    7    6    5    4     3     2     1
## a3    4    5    6    7    8    9   10   11   12    13    14    15
## a4   20   19   18   17   16   15   14   13   12    11    10     9
```

```r
# cbind union por columnas
valores_c <- cbind(a1, a2, a3, a4)
valores_c
```

```
##       a1 a2 a3 a4
##  [1,]  1 12  4 20
##  [2,]  2 11  5 19
##  [3,]  3 10  6 18
##  [4,]  4  9  7 17
##  [5,]  5  8  8 16
##  [6,]  6  7  9 15
##  [7,]  7  6 10 14
##  [8,]  8  5 11 13
##  [9,]  9  4 12 12
## [10,] 10  3 13 11
## [11,] 11  2 14 10
## [12,] 12  1 15  9
```

Nombres

```
nombre <- c("Nombre a1", "Otro nombre a2", "Otro mas", "")
colnames(valores_c) <- nombre
valores_c
```

```
##       Nombre a1 Otro nombre a2 Otro mas
##  [1,]         1             12        4 20
##  [2,]         2             11        5 19
##  [3,]         3             10        6 18
##  [4,]         4              9        7 17
##  [5,]         5              8        8 16
##  [6,]         6              7        9 15
##  [7,]         7              6       10 14
##  [8,]         8              5       11 13
##  [9,]         9              4       12 12
## [10,]        10              3       13 11
## [11,]        11              2       14 10
## [12,]        12              1       15  9
```

```
colnames(valores_c)
```

```
## [1] "Nombre a1"      "Otro nombre a2" "Otro mas"       ""
```

Extraer elementos `[ ]`

Recordando: Las coordenadas de una matriz son por fila y columnas en ese orden

```
valores_c[1, ] # Extraer la primera fila
```

```
##      Nombre a1 Otro nombre a2       Otro mas
##              1             12              4             20
```

```
valores_c[, -1] # Extraer todo menos la primera  columna
```

```
##       Otro nombre a2 Otro mas
##  [1,]             12        4 20
##  [2,]             11        5 19
##  [3,]             10        6 18
##  [4,]              9        7 17
##  [5,]              8        8 16
##  [6,]              7        9 15
##  [7,]              6       10 14
##  [8,]              5       11 13
##  [9,]              4       12 12
## [10,]              3       13 11
## [11,]              2       14 10
## [12,]              1       15  9
```

```r
valores_c[1, 4] #valor individual
```

```
##
## 20
```

Algo mas complejo

```r
valores_c[c(9, 3), c(1, 3,4)]
```

```
##      Nombre a1 Otro mas
## [1,]         9       12 12
## [2,]         3        6 18
```

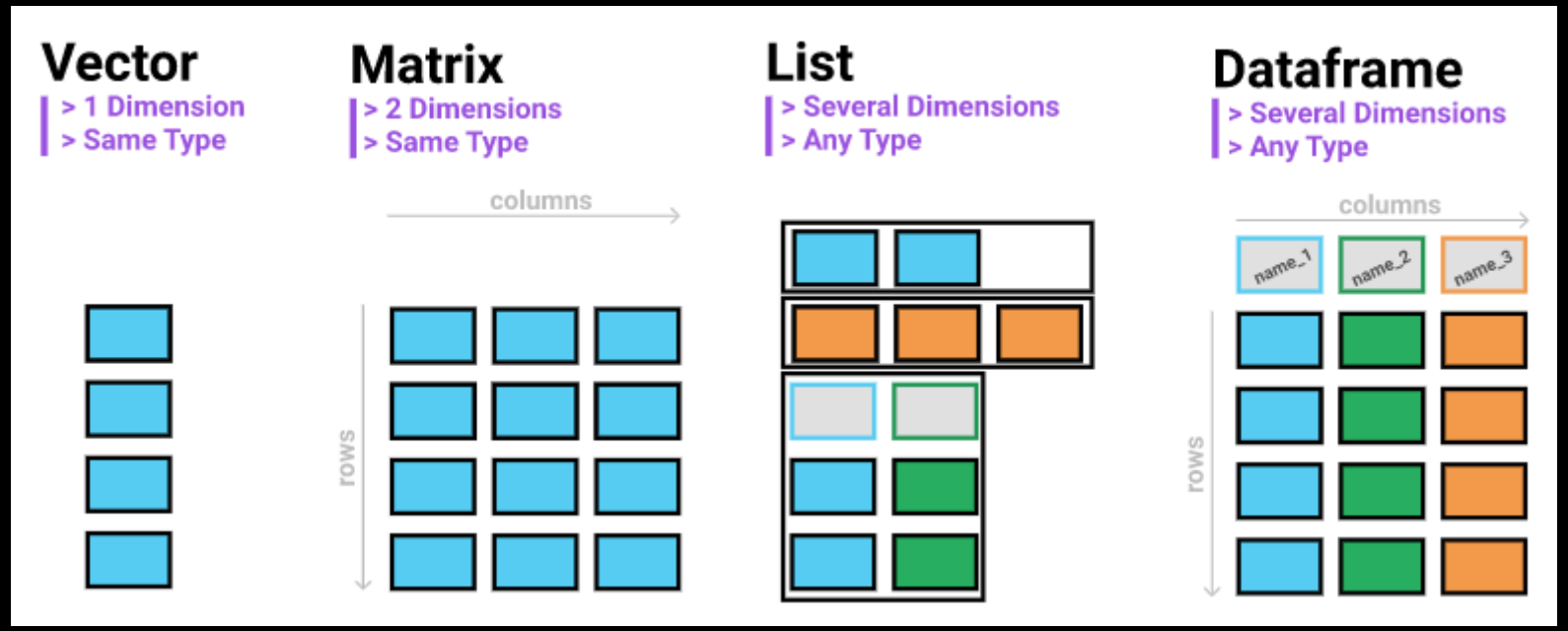
Remplazar valores

```r
valores_c[c(9, 3), c(1, 3,4)] <- 1
valores_c[c(9, 3), c(1, 3,4)]
```

```
##      Nombre a1 Otro mas
## [1,]         1        1 1
## [2,]         1        1 1
```

Listas

Para explicar que es una lista se debe conocer todo lo anterior. Ya que cada uno de los anteriores elemento puede ser parte de una lista, inclusive una lista es puede se parte de una lista



Ejemplo

Base de datos

```
knitr::kable(head(iris), format = 'html')
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

La funcion lm()

```
# Creacion del modelo
modelo <- Sepal.Length ~ Sepal.Width + Petal.Length
# Minimos cuadrados ordinarios
resultado <- lm(modelo, data = iris)
typeof(resultado)
```

```
## [1] "list"
```

```
summary(resultado)
```

```
##
## Call:
## lm(formula = modelo, data = iris)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.96159 -0.23489  0.00077  0.21453  0.78557
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.24914    0.24797    9.07 7.04e-16 ***
## Sepal.Width   0.59552    0.06933    8.59 1.16e-14 ***
## Petal.Length  0.47192    0.01712   27.57  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3333 on 147 degrees of freedom
## Multiple R-squared:  0.8402,    Adjusted R-squared:  0.838
## F-statistic: 386.4 on 2 and 147 DF,  p-value: < 2.2e-16
```

Vector

```
resultado$call
```

```
## lm(formula = modelo, data = iris)
```

Matrices

```
resultado$coefficients
```

```
##  (Intercept)  Sepal.Width Petal.Length
##    2.2491402    0.5955247    0.4719200
```

Data frames

```
head(resultado$model)
```

```
##   Sepal.Length Sepal.Width Petal.Length
## 1          5.1         3.5          1.4
## 2          4.9         3.0          1.4
## 3          4.7         3.2          1.3
## 4          4.6         3.1          1.5
## 5          5.0         3.6          1.4
## 6          5.4         3.9          1.7
```

Listas dentro de listas

```
summary(resultado$xlevels)
```

```
## Length  Class   Mode
##      0   list   list
```

Creacion de listas list()

```
lista <- list("UNCP", "economia", c(1, 1, 23, 4, 5), TRUE, valores_c)
lista
```

```
## [[1]]
## [1] "UNCP"
##
## [[2]]
## [1] "economia"
##
## [[3]]
## [1]  1  1 23  4  5
##
## [[4]]
## [1] TRUE
##
## [[5]]
##       Nombre a1 Otro nombre a2 Otro mas
##  [1,]          1             12        4 20
##  [2,]          2             11        5 19
##  [3,]          1             10        1  1
##  [4,]          4              9        7 17
##  [5,]          5              8        8 16
##  [6,]          6              7        9 15
##  [7,]          7              6       10 14
##  [8,]          8              5       11 13
##  [9,]          1              4        1  1
## [10,]         10              3       13 11
## [11,]         11              2       14 10
## [12,]         12              1       15  9
```

Desordenado! muchos `[ [ ] ]`

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

```
## [1] "UNCP"
```

```r
mi_lista <- list(universidad  = "UNCP", facultad = "economia", vector = c(1, 1, 23, 4, 5), logi
mi_lista
```

```
## $universidad
## [1] "UNCP"
##
## $facultad
## [1] "economia"
##
## $vector
## [1]  1  1 23  4  5
##
## $logical
## [1] TRUE
##
## $matrix
##       Nombre a1 Otro nombre a2 Otro mas
##  [1,]         1             12        4 20
##  [2,]         2             11        5 19
##  [3,]         1             10        1  1
##  [4,]         4              9        7 17
##  [5,]         5              8        8 16
##  [6,]         6              7        9 15
##  [7,]         7              6       10 14
##  [8,]         8              5       11 13
##  [9,]         1              4        1  1
## [10,]        10              3       13 11
## [11,]        11              2       14 10
## [12,]        12              1       15  9
##
## $lista
## [1] TRUE
```

Ahora se puede extraer con `$`

```
mi_lista$universidad
```

```
## [1] "UNCP"
```

Funciones en R

Creacion de funciones

```
nombre_funcion <- function(variable1, variable2){
    # lo que se quiere hacer con las variables
}
```

```
mi_suma <- function(x, y, z){
  x + y + z
}
mi_suma(1, 3, 4)
```

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

Usando funciones dentro de las funciones

```r
# mean() calcula el promedio de los elementos
x <- 1:12

mean(x)
```

```
## [1] 6.5
```

```r
mi_promedio <- function(a) {
  a1 <- sum(a)
  a2 <- length(a)
  promedio <- a1/a2
  paste0("El promedio de los datos es: ", promedio)
}

mi_promedio(x)
```

```
## [1] "El promedio de los datos es: 6.5"
```

If, else

Verdadero o falso?

```
x <- 12
x < 3
```

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

```
x > 5
```

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

```
x == 12 # en R se usa `==` para preguntar si es igual
```

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

```
x != 12 # Desigual
```

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

```
x <= 13 # menor o igual
```

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

```
x >= 1 # mayour o igual
```

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

La funcion `If` evalua si el enunciado o la proposicion solo si es verdadera y hace lo que especificas dentro del `{}`, por otro lado `else` evalua en algun otro caso.

```
Individuo <- "yo"
if(Individuo == "yo"){
  print("Aquel individuo era yo")
} else {
  print("No recuerdo")
}
```

```
## [1] "Aquel individuo era yo"
```

```
Individuo <- "yo"
if(Individuo != "yo"){
  print("Aquel individuo era yo")
} else {
  print("Ese individuo no era yo")
}
```

```
## [1] "Ese individuo no era yo"
```

For loops

Evitar hacer codigos reiterativos

Se quiere este resultado final:

$$Demanda = 60P - P^2$$

Con precios = [10, 20, 30, 40, 50]

Manera manual o metodo excel

```
demanda <- c(
  -1 * 10^2 + 60 * 10,
  -1 * 20^2 + 60 * 20,
  -1 * 30^2 + 60 * 30,
  -1 * 40^2 + 60 * 40,
  -1 * 50^2 + 60 * 50)
demanda
```

```
## [1] 500 800 900 800 500
```

For - loops

```
precio <- c(10, 20, 30, 40, 50)
demanda <- c()

for (price in precio) {
  calc <- -1 * price^2 + 60 * price
  demanda <- c(demanda, calc)
}
demanda
```

```
## [1] 500 800 900 800 500
```

Packages

Para explorar la estructura de las funciones de los paquetes instalados, solo se necesita cargar el paquete y escribir la funcion sin `()`.

Explorar la funcion de minimos cuadrados ordinarios

```
stats::lm
```

```
## function (formula, data, subset, weights, na.action, method = "qr",
##     model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
##     contrasts = NULL, offset, ...)
## {
##     ret.x <- x
##     ret.y <- y
##     cl <- match.call()
##     mf <- match.call(expand.dots = FALSE)
##     m <- match(c("formula", "data", "subset", "weights", "na.action",
##         "offset"), names(mf), 0L)
##     mf <- mf[c(1L, m)]
##     mf$drop.unused.levels <- TRUE
##     mf[[1L]] <- quote(stats::model.frame)
##     mf <- eval(mf, parent.frame())
##     if (method == "model.frame")
##         return(mf)
##     else if (method != "qr")
##         warning(gettextf("method = '%s' is not supported. Using 'qr'",
```

Gracias!

Diapositivas creadas mediante **xaringan**.

Para descargar los actuales y futuros materiales

Ejecutar el siguiente comando via `git bash`

```
git clone https://github.com/TJhon/R-curso.git
```

Las diapositivas estan en el siguiente link https://github.com/TJhon/R-curso