

Estadística para ingenieros

Pedro José Ovalles García



Universidad Simón Bolívar
CO3321 - Estadística para ingenieros.

Intensivo 2016

Agenda

- 1 Presentación
- 2 Detalles técnicos
- 3 Temario
- 4 Bibliografía
- 5 Evaluación
- 6 Laboratorios

Detalles del curso

Materia Estadística para ingenieros.

Código CO3321.

Profesor Pedro Ovalles.

e-mail povallesgarcia@usb.ve

Oficina MYS108 (CEsMA).

Detalles técnicos del curso

Horario de clases: 2-3 (8:30 a.m. a 10:30 a.m.) — 4-5 (10:30 a.m. a 12:30 p.m.)

Aula: ENE118 — ENE103

Horas de consulta: programar vía correo.

Aula Virtual – > Estadística para ingenieros / CO3321 / Intensivo
2016 CO3322 - Pedro José Ovalles

Temas a desarrollar

- 1 Estadística Descriptiva.
- 2 Distribuciones muestrales.
- 3 Estimación. Propiedades de los estimadores.
- 4 Médeto de máxima verosimilitud.
- 5 Intervalos de confianza.
- 6 Pruebas de hipótesis.
- 7 Pruebas χ^2 .
- 8 Regresión lineal simple y múltiple.
- 9 Análisis de varianza.

Bibliografía principal

- Guía de la Profa. María Eglee Pérez [Tema 1]
- Wackerly, Mendenhall & Scheaffer. Estadística Matemática con Aplicaciones.
 - Capítulo 7 [Tema 2]
 - Capítulo 8 [Temas 3 y 5]
 - Capítulo 9 [Temas 3 y 4]
 - Capítulo 10 [Tema 6]
 - Capítulo 11 [Tema 8]
 - Capítulo 12 [Tema 9]
 - Capítulo 14 [Tema 7]

Bibliografía complementaria

- Guía del prof. Romulo Mayorca y Giselle Alvarez. (Est. Descrip.)
- Walpole, Myers, Myers & Ye. Probabilidades & Estadística para ingeniería & ciencias.
- De Groot & Schervish. Probabilidades y Estadística.

Evaluación

Evaluación	Porcentaje	Fecha	Semana	Día
Parcial I	40 %	03/08	3	Miércoles
Parcial II	40 %	22/08	6	Lunes
Lab. Estadística descriptiva	4 %	21/07	1	Jueves
Lab. Máxima verosimilitud	2 %	29/07	2	Viernes
Lab. Intervalos de confianza	3 %	03/08	3	Miércoles
Lab. Pruebas de hipótesis	3 %	10/08	4	Miércoles
Lab. Bondad de ajuste	4 %	16/08	5	Martes
Lab. Regresión lineal	4 %	19/08	5	Viernes

Reglas a seguir en los laboratorios:

- 1 Se tendrán distintos grupos de datos, asignados “aleatoriamente” para realizar todos la misma tarea.
- 2 Se debe realizar en forma individual o en parejas. En el caso de las parejas tienen que escoger con cual de los grupos de datos trabajar.
- 3 El documento de entrega debe incluir explícitamente el código utilizado, así como las salidas de los comandos.
- 4 El documento debe estar en formato *.pdf*.
- 5 El documento debe estar debidamente identificado con el nombre de quien(es) haya(n) realizado el trabajo.
- 6 La entrega será en formato digital al correo electrónico *povalles@cesma.usb.ve*, a más tardar 12 horas luego de la publicación.
- 7 No deben enviar correos con adjuntos con extensión *.zip* o de cuentas @hotmail.com; ya que rebotan.
- 8 Los laboratorios NO TIENEN RECUPERACIÓN.

FAQ

Introducción a R y R-Studio

- 1 Paquete estadístico R (3.x):
@ cran.org
- 2 Entorno de desarrollo integrado RStudio (0.99.x):
@ rstudio.com
- 3 Visor de documentos PDF (Okular, Evince, Adobe Acrobat R):
@ kde.org, @ gnome.org y @ adobe.com



Download RStudio

[Home](#) / [Overview](#) / [RStudio](#) / [Download RStudio](#)

RStudio is a set of integrated tools designed to help you be more productive with R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.

If you run R on a Linux server and want to enable users to remotely access RStudio using a web browser [please download RStudio Server](#).

Do you need support or a commercial license?

[Check out our commercial offerings](#)

Download RStudio Desktop v0.98.1091 — [Release Notes](#)

RStudio requires R 2.11.1 (or higher). If you don't already have R, you can download it [here](#).

[Click here to learn more about Shiny!](#)

Installers for ALL Platforms

Installers

[RStudio 0.98.1091 - Windows XP/Vista/7/8](#)

[RStudio 0.98.1091 - Mac OS X 10.6+ \(64-bit\)](#)

[RStudio 0.98.1091 - Debian 6+Ubuntu 10.04+ \(32-bit\)](#)

[RStudio 0.98.1091 - Debian 6+Ubuntu 10.04+ \(64-bit\)](#)

[RStudio 0.98.1091 - Fedora 13+openSUSE 11.4+ \(32-bit\)](#)

[RStudio 0.98.1091 - Fedora 13+openSUSE 11.4+ \(64-bit\)](#)

Size	Date	MD5
45 MB	2014-11-06	910fba345c0555597bda498cad1302b0
38.4 MB	2014-11-06	9c7d2cea702cf478a4a774b79134b3ee
53 MB	2014-11-06	0bc579cbee43a514e3fb456959a0ada
54.9 MB	2014-11-06	1e88e6775993daa8cf7d4d89f76af7e0
53.4 MB	2014-11-06	3ae5923956166f90ecc1cb721b02f90f
55 MB	2014-11-06	6d1ac08ceed731f5750f3de9a911511b

Zip/Tarballs

Ayuda en la web

The screenshot shows a Mozilla Firefox browser window. The address bar displays the URL `rparatodos.wordpress.com/2011/11/22/libros-de-r-gratuitos/`. The page title is "Libros de R gratuitos | R_paratodos - Mozilla Firefox". The browser's status bar at the bottom shows "Scripts Currently Forbidden | <SCRIPT>: 45 | <OBJECT>: 0".

The webpage content is as follows:

R paratodos

Un blog donde podrías encontrar cosas interesantes sobre el uso del lenguaje R

Inicio > Users > Libros de R gratuitos

Libros de R gratuitos

22/11/2011 rparatodos [Go to comments](#) [Deja un comentario](#)

En los siguientes enlaces podéis descargarlos diferentes libros en pdf totalmente gratis:

- Análisis multivariante
- Estadística biomédica
- Series Temporales
- Bioinformática

Y estos que cuelgan de la página del CRAN:

- R para principiantes (en inglés)
- R para principiantes (en español)
- Introducción a R (en español)
- Regresión y ANOVA
- Introduction to the R Project for Statistical Computing for use at ITC
- Objetos S4
- R para biólogos
- Generación automática de informes con Sweave (en español)
- R Commander (en español)
- Curso de R (en español)
- "Chuleta" de comandos básicos
- Otra chuleta

About these ads

Tu voto:

Concurso

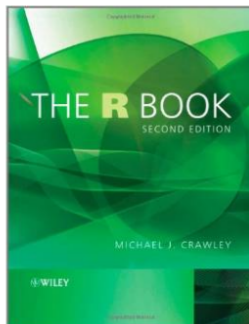
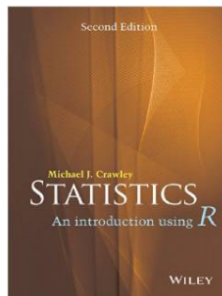
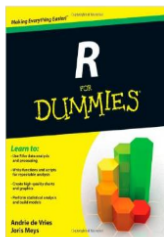
Entradas recientes

- FELIZ AÑO (Internacional de la Estadística) 2013!!!
- La analítica como arma profesional
- IV Jornadas de Usuarios de R
- GRADO EN ESTADÍSTICA de la Uva y CURSO DE ADAPTACIÓN AL GRADO para diplomados.
- Curso de iniciación a R...
- Datos geográficos de tipo raster en R
- Estructura de un nuevo paquete en R
- Brew, o cómo mezclar R y texto para generar informes repetitivos
- Design of Experiments in R
- Cloudnumbers: R en la nube

Categorías

- Charles
- Developers

Ayuda en libros



Lenguaje R



Lenguaje R

A screenshot of the RGui (32-bit) application window. The window has a menu bar with 'File', 'Edit', 'View', 'Misc', 'Packages', 'Windows', and 'Help'. Below the menu bar is a toolbar with icons for file operations and execution. The main area is the 'R Console', which displays the R version 3.2.2 startup message, copyright information, and usage instructions. The prompt '>' is visible at the bottom of the console.

```
RGui (32-bit)
File Edit View Misc Packages Windows Help

R Console

R version 3.2.2 (2015-08-14) -- "Fire Safety"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

R: Introducción

R es una implementación abierta del lenguaje S (S-PLUS) desarrollado en AT&T Bell Laboratories (C, UNIX) por Rick Becker, John Chambers y Allan Wilks.

Inicialmente R fue desarrollado por Ross Ihaka y Robert Gentleman (U. Auckland, NZ). Hoy en día el desarrollo esta liderado por el *core team* de cerca de una docena de personas de diversas instituciones a nivel mundial.

R es un conjunto integrado de programas para manipulación de datos, cálculos y gráficos¹ y cuenta con los siguientes aspectos:

- Almacenamiento y manipulación efectiva de datos
- Operadores para cálculo sobre variables (vectores, matrices, ...)
- Amplia, coherente e integrada colección de herramientas para análisis de datos (gráficas, ...)
- Lenguaje de programación bien desarrollado, simple y efectivo
- Muchas de las funciones suministradas con el sistema están escritas en el lenguaje R

¹<http://cran.r-project.org/doc/manuals/R-intro.pdf>

R como una calculadora

Los números son considerados vectores de longitud 1. Las *asignaciones* se hacen mediante `<-` y `=`. Los comandos se ejecutan al presionar ENTER. Las operaciones generalmente son *vectoriales*.

```
> x<- 1 : 4
> y = c(5, 10)
> 2 * x + y
[1] 7 14 11 18
> z=x^2-x
> x/y
[1] 0.2 0.2 0.6 0.4
> sum(z) / length(z) # Promedio
[1] 5
> mean(z)
[1] 5
> sqrt(y - 1)
[1] 2 3
> max(z)
[1] 12
> range(z)
[1] 0 12
```

El vector `y` es **reciclado** hasta llegar a la longitud requerida. Ejemplo:

```
> a = 1 : 2; b = 1 : 4; (b + a)
[1] 2 4 4 6
> c(1, 2, 3, 4) + c(1, 2, 1, 2)
[1] 2 4 4 6
> c(1, 2, 3, 4) + c(1, 2, 1)
[1] 2 4 4 5
Warning message:
In c(1, 2, 3, 4) + c(1, 2, 1) :
  longer object length is not a multiple of shorter object length
```

RStudio



RStudio es un *Entorno de Desarrollo Integrado* para R





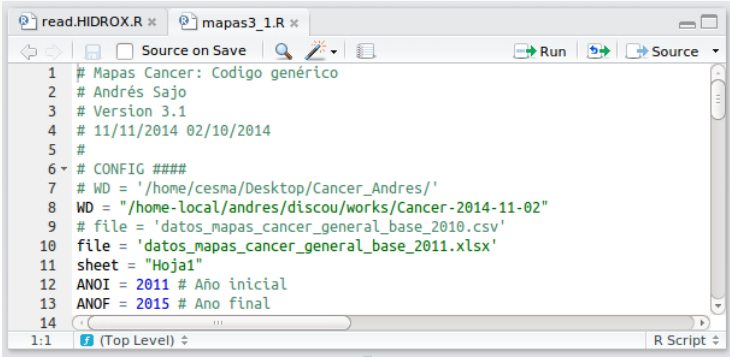
RStudio es un *Entorno de Desarrollo Integrado* para R

The screenshot shows the RStudio IDE interface with four numbered annotations:

- 1** (Green): Points to the source editor showing R code for a script named `mapas3.R`. The code includes comments, a configuration section, and data loading instructions.
- 2** (Blue): Points to the console window showing the output of the R script, including package loading messages and variable assignments.
- 3** (Red): Points to the Environment pane showing the global environment with variables like `mort.main`, `mort.nax`, `mort.min`, `mort.pal`, `mort.ramp`, `mort.top.s`, and `tabla`.
- 4** (Yellow): Points to the Files pane showing the file structure, including `R: Random Samples and Permutations` and `sample (base)`.

RStudio –1– Editor

La ventana 1 constituye el *Editor*, es el lugar donde se redactan y editan los programas y *scripts*. Cuenta con la mayoría de las facilidades de un editor moderno: completación de palabras, definiciones de objetos, ayudas emergentes, búsquedas y reemplazos, etc. Adicionalmente cuenta con un sistema para seccionar códigos fuentes.



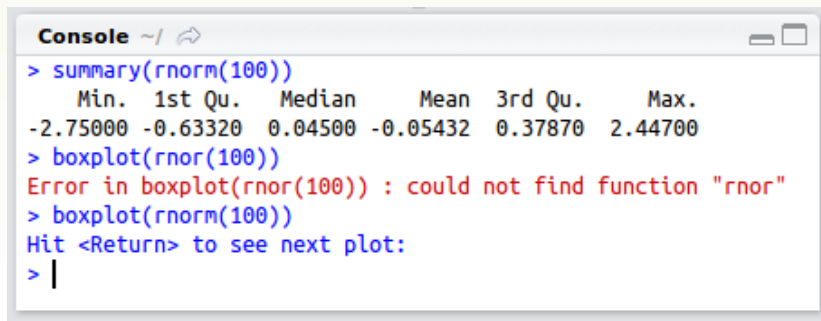
The screenshot shows the RStudio Editor window with two tabs: 'read.HIDROX.R' and 'mapas3_1.R'. The 'mapas3_1.R' tab is active. The script content is as follows:




```
1 # Mapas Cancer: Codigo genérico
2 # Andrés Sajó
3 # Version 3.1
4 # 11/11/2014 02/10/2014
5 #
6 # CONFIG ####
7 # WD = '/home/cesma/Desktop/Cancer_Andres/'
8 WD = "/home-local/andres/discou/works/Cancer-2014-11-02"
9 # file = 'datos_mapas_cancer_general_base_2010.csv'
10 file = 'datos_mapas_cancer_general_base_2011.xlsx'
11 sheet = "Hoja1"
12 ANOI = 2011 # Año inicial
13 ANOF = 2015 # Año final
14
```

The status bar at the bottom indicates '1:1' and '(Top Level)'.

RStudio –2– Consola

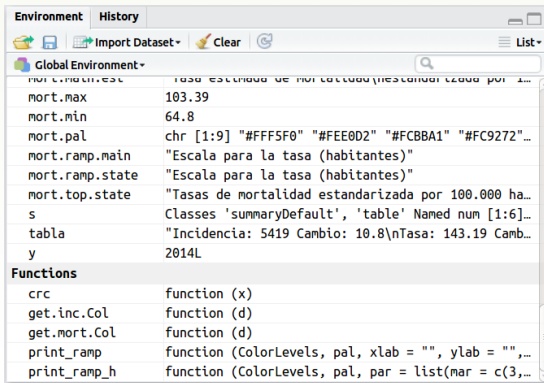
La *Consola*, es el lugar donde se interactúa con el *interpretador* del lenguaje R. Al igual que el editor, la consola permite la completación de palabras, ayudas emergentes y es donde los programas redactados en el editor son ejecutados. La consola identifica el origen y propósito de cada comando o mensaje.



```
Console ~/     
> summary(rnorm(100))  
      Min.   1st Qu.   Median     Mean   3rd Qu.     Max.   
-2.75000 -0.63320  0.04500 -0.05432  0.37870  2.44700   
> boxplot(rnor(100))  
Error in boxplot(rnor(100)) : could not find function "rnor"  
> boxplot(rnorm(100))  
Hit <Return> to see next plot:  
> |
```

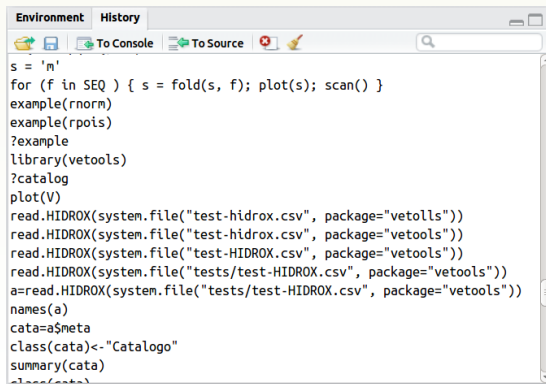
RStudio –3– Ambiente e Historial (Panel A)

El panel 3 está constituido por dos pestañas. Primero está el *Ambiente*, que muestra los *objetos* definidos. Están clasificados en tres categorías: (1) **Data** (cuadro de datos y matrices), (2) **Valores** (escalares, listas y objetos varios) y (3) **Funciones** (lista de funciones definidas por el usuario). Por otro lado, la pestaña de *Historial* presenta un diario de todos los comandos ejecutados.



RStudio –3– Ambiente e Historial (Panel A)

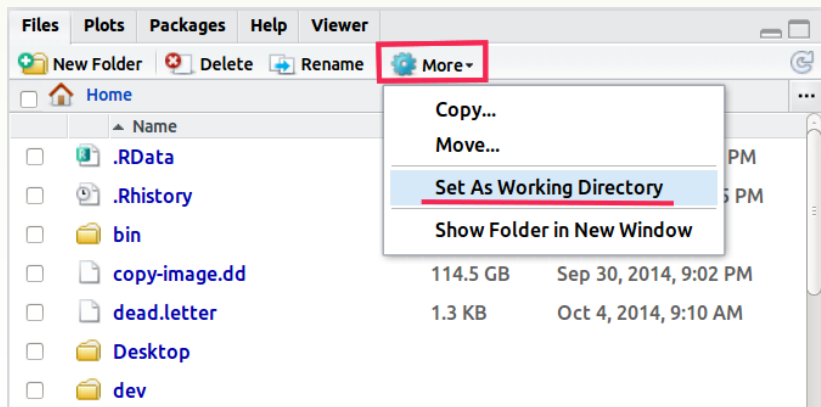
El panel 3 está constituido por dos pestañas. Primero está el *Ambiente*, que muestra los *objetos* definidos. Están clasificados en tres categorías: (1) **Data** (cuadro de datos y matrices), (2) **Valores** (escalares, listas y objetos varios) y (3) **Funciones** (lista de funciones definidas por el usuario). Por otro lado, la pestaña de *Historial* presenta un diario de todos los comandos ejecutados.



```
s = 'm'
for (f in SEQ ) { s = fold(s, f); plot(s); scan() }
example(rnorm)
example(rpois)
?example
library(vetools)
?catalog
plot(V)
read.HIDROX(system.file("test-hidro.csv", package="vetolls"))
read.HIDROX(system.file("test-hidro.csv", package="vetools"))
read.HIDROX(system.file("test-HIDROX.csv", package="vetools"))
read.HIDROX(system.file("tests/test-HIDROX.csv", package="vetools"))
a=read.HIDROX(system.file("tests/test-HIDROX.csv", package="vetools"))
names(a)
cata=a$meta
class(cata)<-"Catalogo"
summary(cata)
class(cata)
```

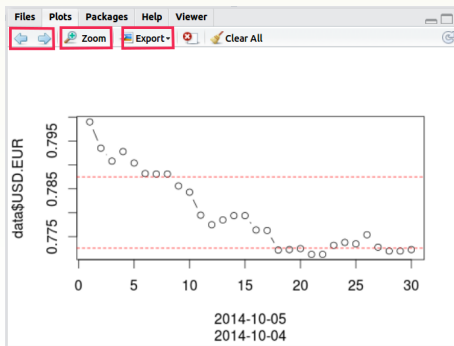
RStudio -4- Panel B

Este panel esta constituido por 4 pestañas: *Archivos* (Files), es un navegador de archivos. *Graficas* (Plots), este panel contiene todas las gráficas que se construyen. *Paquetes* (Packages), á través de esta pestaña es posible administrar los paquetes y/o librerías. *Ayuda* (Help), la pestaña más importante, contiene todas las páginas de ayuda y manuales de referencia.



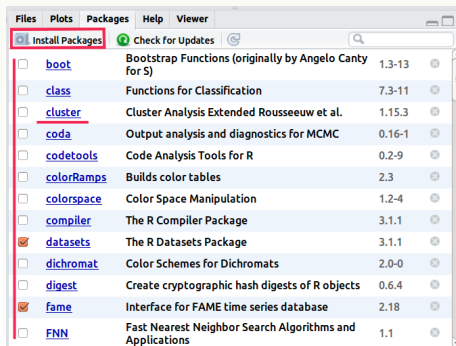
RStudio -4- Panel B

Este panel esta constituido por 4 pestañas: *Archivos* (Files), es un navegador de archivos. *Graficas* (Plots), este panel contiene todas las gráficas que se construyen. *Paquetes* (Packages), á través de esta pestaña es posible administrar los paquetes y/o librerías. *Ayuda* (Help), la pestaña más importante, contiene todas las páginas de ayuda y manuales de referencia.



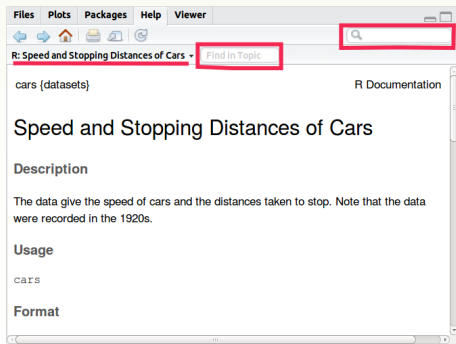
RStudio –4– Panel B

Este panel esta constituido por 4 pestañas: *Archivos* (Files), es un navegador de archivos. *Graficas* (Plots), este panel contiene todas las gráficas que se construyen. *Paquetes* (Packages), á través de esta pestaña es posible administrar los paquetes y/o librerías. *Ayuda* (Help), la pestaña más importante, contiene todas las páginas de ayuda y manuales de referencia.



RStudio -4- Panel B

Este panel esta constituido por 4 pestañas: *Archivos* (Files), es un navegador de archivos. *Graficas* (Plots), este panel contiene todas las gráficas que se construyen. *Paquetes* (Packages), á través de esta pestaña es posible administrar los paquetes y/o librerías. *Ayuda* (Help), la pestaña más importante, contiene todas las páginas de ayuda y manuales de referencia.



Solicitar ayuda sobre un tópico se puede hacer directamente en la pestaña de Ayuda (Help) o a través de la consola:

```
1 ? var
2 ?? mean
3 help(mean)
4 %apropos("sort")
```

Los comandos `? <comando>` y `help(<comando>)` despliegan la ayuda de `<comando>`, mientras que los comandos `?? <comando>` y `apropos('‘<comando>’')` muestran una lista de posibles tópicos relacionados a `<comando>`.

Nombres de variables y funciones

Al definir una variable o función hay que tomar en cuenta:

- ❶ No puede empezar por un dígito y contener caracteres especiales
-, +, *, /, ?, &, #, \$, (, [, {, :, etc...

Nombres aceptados `tabla4`, `HrMin`, `bal.pres`, ...

Nombres no aceptados `4tabla`, `hr+min`, `bal?pres`, ...

- ❷ Debe ser concisa e informativa:

Uso

Nombre

Variable temporal

`tmp`



`VariableDeUsoTemporal`



- ❸ Se sugiere no utilizar nombres que pueden confundirse con comandos

`c` Concatenación

`t` Transpuesta

`T` Valor lógico verdad, `TRUE`

`F` Valor lógico falso, `FALSE`

IMPORTANTE!!!!



En R, se diferencian puntos (.) de comas (,) ($3,17 \neq 3.17$).

La coma (,) se usa para separar valores, en cambio el punto se relaciona con decimales, por ejemplo `a= c(2.2, 13, 1.8, 4, 9)`.

Se diferencian letras mayúsculas de minúsculas. Por ejemplo:

`a= 3+2`

`A= 4*7`

Clases fundamentales de R

Numérico	Comprende los números enteros (-5, 0, 315) y punto flotantes (0.5, 1.333)
Lógico	Comprende los valores Verdad (TRUE ó T), Falso (FALSE ó F) y <i>No disponible</i> NA
Carácter	Son las letras y palabras, están delimitados por comillas ("). Ejemplo "Z" , "Cadena"
Factor	Representación de factores y pueden estar en cualquier clase anterior (número, letra o lógico)
Fórmula	Describen modelos, tiene sintaxis propia con <i>lado derecho</i> y <i>lado izquierdo</i> , separados por tilde (~). Ejemplo: y~x-1
Vector	
Matriz	
Arreglo	
Tabla	data.frame ¹ y tablas de contingencia

¹ Estos objetos pueden a su vez estar conformados por 1 o más clases.

Vectores

Para construir vectores, se usa el comando concatenar: “ c”.

```
edad = c(2, 30, 23, 33, 28, 45, 34, 55, 67, 28, 35, 35, 36, 98)
```

```
sexo = c("F", "F", "M", "M", "F", "M", "M", "F", "F", "F", "F", "M", "F")
```

```
summary(edad) min(edad) max(edad) sort(edad) var(edad) sd(edad) sort(edad)
```

```
length(edad) hist(edad) boxplot(edad)
```

Factores

La construcción de factores se logra con los comandos **factor** y **ordered**:

```
1 > (fac <- factor(letters[1:5]))
2 [1] a b c d e
3 Levels: a b c d e
4 > (ord <- ordered(LETTERS[1:5]))
5 [1] A B C D E
6 Levels: A < B < C < D < E
```

Los factores pueden ser de clase numérica, letra o lógico.

Para conocer la cantidad y los *niveles* de una variable (factor):

```
1 > levels(fac)
2 [1] "a" "b" "c" "d" "e"
3 > nlevels(fac)
4 [1] 5
```

Para factores ordenados, los niveles disponen de *ordenamiento*:

```
1 > ord[2] < ord[1]
2 [1] FALSE
```

Factores (gl)

Un comando útil para la construcción sistemática de factores es el comando `gl` (*generate factor levels*):

```
1 > (gl(3, 2))
2 [1] 1 1 2 2 3 3
3 Levels: 1 2 3
4 > (y = gl(3, 2, 12))
5 [1] 1 1 2 2 3 3 1 1 2 2 3 3
6 Levels: 1 2 3
7 > levels(y) <- c('A', 'B', 'A')
8 > y
9 [1] A A B B A A A A B B A A
10 Levels: A B
```

Los factores pueden ser extendidos:

```
1 > f = factor(c('a', 'b'))
2 [1] a b
3 Levels: a b
4 > levels(f) <- letters[1 : 4]
5 [1] a b
6 Levels: a b c d
```

Vectores y secuencias

Los vectores son la unidad fundamental, existen varios comandos para construirlos

- 1 Secuencias enteras, `" : "`. Ejemplo: `v <- 1 : 5` almacena en `v` la secuencia de 1 a 5
- 2 Concatenación, `c(1, 2, 3, 4, 5)`
- 3 Secuencias generales, `seq(desde, hasta, paso, longitud)`

```
1 seq(1, 5) # mismo que 1 : 5
2 seq(5, 1, -1) # mismo que rev(1 : 5)
3 seq(0, 1, 0.1) # 11
4 seq(0, 1, length.out=10) # 10
```

- 4 Replicas, `rep(x, veces, modo)`, repite `x`, `veces` veces y/o a cada elemento de `x` lo repite según `modo`.

```
1 rep(1 : 3, 2)
2 rep(1 : 3, each = 2)
3 rep(1 : 3, 3, each = 2)
```

- 5 Invertir el orden de vectores `rev(1:5)` construye `[5 4 3 2 1]`

Acceso a los elementos I

Acceso a los elementos de un vector [índice de posición]

El acceso de los elementos se efectúa a través del comando []

```
1 > A <- c(10, 20, 30, 40, 50)
2 > A[1]
3 10
4 > A[3:5]
5 30 40 50
6 > A[c(4, 2, 5, 1, 3)]
7 40 20 50 10 30
8 > A[10]
9 [1] NA
```

Acceso por índice de posición: cuando se indica la posición del elemento dentro del vector a ser accedido.

El comando **length** muestra la longitud del vector, o arreglo:

```
1 > length(A)
2 [1] 5
```

Eliminación de elementos

Con los índices de acceso también se pueden *eliminar* entradas de un vector

Eliminación de elementos

Consiste en prefijar con el signo menos (-) las posiciones que NO se quieren seleccionar:

```
1 > Nombres <- c('Amanda', "Beatriz", 'Carolina', 'Daniela')
3 > Nombres[-3]
4 "Amanda"  "Beatriz" "Daniela"
6 > Nombres
7 "Amanda"  "Beatriz"  "Carolina" "Daniela"
9 > Nombres[-(1:2)]
10 "Carolina" "Daniela"
```

Nota: Esta forma NO modifica el vector **Nombres**.



Práctica

- 1 Construir el vector (11, 12, 13, 14, 15) y llamarlo “esc”
- 2 Construir el vector (1, 3, 5, 7, 9, 11, 13, 15, 17, 19) y almacenarlo en la variable “vec”
- 3 Construir el vector “x” concatenando los vectores “esc” y “vec”
- 4 Sustituir los elementos en las posiciones 2, 3 y 5 por sus respectivos valores negativos
- 5 Eliminar las posiciones 4 y 8
- 6 Mostrar la nueva longitud del vector resultante
- 7 Construir el siguiente vector Nombres=(A, D, X, Z, Y, M, L, B, V, E, R, A, B, T, Z, Z, U)
- 8 Identificar las posiciones donde se encuentra la letra A, B y L
- 9 Extraer conjuntamente las posiciones de las letras A y Z.
Ayuda: `x == 'A' | x == 'Z'`

Operaciones sobre vectores

La mayoría de los operadores aceptan como entrada vectores:

`+`, `-`, `*`, `/`, `^`, `%%`: operaciones conocidas elemento a elemento

`c`: concatenar dos ó mas *objetos*

`rev`: invierte los elementos de un vector

`sum`: suma los elementos de un *objeto*

`cumsum`: suma acumulativa

`mean`, `median`: promedio y mediana

`var`: varianza

`sd`: desviación estándar (equivalente a `sqrt(var(x))`)

`scale`: escalar y centrar

`hist`: histograma

`unique`: muestra los elementos de un vector sin repetición

`sort`: ordenar un vector

`summary`: resumen estadístico de un *objeto*

`str`: resumen estructural de un *objeto*

`aggregate`: resumen categórico de los miembros de *objeto*

`split`: separación categórica de los miembros de *objeto*



Práctica

- 1 Construir el vector 1 a 5 y llamarlo "a"
- 2 Construir el vector 1 a 10 y almacenarlo en la variable "b"
- 3 Efectuar $a + b$, $a - b$ y $a * b$, ¿están bien definidos? ¿por qué?
- 4 Construir el vector "x" concatenando "a" y "b"
- 5 Revertir el orden de "x" y almacenarlos en la variable "y"
- 6 Mostrar los elementos únicos de "x"
- 7 Mostrar la suma acumulativa de "x" y su respectivo histograma
- 8 Estimar la media, varianza y desviación estándar de "y"

El comando summary

El comando **summary** es uno de los comandos mas importantes, para cada clase de variable dispone de un *método* propio:

```
1 > fac = gl(4, 2, 50)
2 > summary(fac)
3 1 2 3 4
4 14 12 12 12
```

```
1 > vec = 1 : 25
2 > summary(vec)
3      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
4         1         7      13     13      19      25
```

```
1 > p <- rnorm(50)
2 > summary(p)
3      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
4 -2.2410 -0.3029  0.4025  0.2625  0.7375  2.4030
```

El comando **summary** tiene muchos otros métodos. Mas adelante conoceremos otros métodos, p. e. el método asociado a los modelos lineales de clase **lm**, entre otros.

Arreglos multidimensionales: Matrices

A parte de los escalares y vectores, R dispone de variables multidimensionales, como son: matrices, arreglos, tablas (**data.frame**) y listas. Las matrices se definen por

```
1 > M <- matrix(c(1, 2, 3, 9, 9, 9, 0, 3, 2),ncol = 3,nrow = 3)
2      [,1] [,2] [,3]
3 [1,]    1    9    0
4 [2,]    2    9    3
5 [3,]    3    9    2
6 > M <- matrix(c(1, 2, 3, 9, 9, 9, 0, 3, 2),ncol = 3,nrow = 3,
7               byrow=TRUE)
8      [,1] [,2] [,3]
9 [1,]    1    2    3
10 [2,]    9    9    9
11 [3,]    0    3    2
12 > rownames(M) <- paste0('Factor', 1:3)
13 > colnames(M) <- LETTERS[1:3]
14      A B C
15 Factor1 1 2 3
16 Factor2 9 9 9
17 Factor3 0 3 2
18 > class(M)
19 [1] "matrix"
```

Arreglos multidimensionales: Arreglos

Arreglos (**array**) son variables con al menos una dimensión.

```
1 > array(letters, dim = c(3,3,2))
2 > a <- array(1:9, dim = c(3,3))
3 > colnames(a) <- paste('fila', 1:3, sep='.')
4 > rownames(a) <- paste0('col', 1:3)
5 > a
6 > dimnames(a)
```

Otros comandos útiles para construir arreglos son **rbind** y **cbind** que unen dos o mas arreglos existentes en forma de fila o columna:

```
1 > (b <- cbind(1:3, 2:4)) # () ext.
2      [,1] [,2]
3 [1,]    1    2
4 [2,]    2    3
5 [3,]    3    4
6 > r <- rbind(1:3, 2:4)
7 > r
8      [,1] [,2] [,3]
9 [1,]    1    2    3
10 [2,]    2    3    4
```

Arreglos multidimensionales: Tablas (data.frame)

Las tablas (**data.frame**) son los objetos usados por defecto para almacenar datos multidimensionales. Muchas de las funciones disponibles asumen que los datos son de esta clase (**lm**, **glm**, ...).

```
1 > D <- data.frame(Nombres=c('Andreina', 'Veronica', '
    Zuleima'), Edades=c(28, 25, 37), IRC=c(4.55, 2.63,
    3.31), stringsAsFactors = FALSE)
2 > View(D) # ó simplemente D
3   Nombres Edades  IRC
4 1 Andreina    28 4.55
5 2 Veronica    25 2.63
6 3  Zuleima    37 3.31
```

Los elementos de cualquier objeto que tenga dimensiones pueden ser accedidos mediante sus *miembros*

```
1 > D$Nombres
2 [1] "Andreina" "Veronica" "Zuleima"
3 > D[, "Edades"] # equivalente D[, 2]
4 [1] 28 25 37
```

Dos tablas se pueden unir a través del comando **merge**.

Arreglos multidimensionales: Tablas de contingencia

Las tablas de contingencia (`table`) son elementos intrínsecos para el análisis de factores.

```
1 > a <- letters[1:4]
2 > table(a, sample(a))

4 a      a b c d
5   a 0 0 0 1
6   b 1 0 0 0
7   c 0 1 0 0
8   d 0 0 1 0

10 > table(rpois(100, 5))

12  1  2  3  4  5  6  7  8  9 10
13  1 10  8 21 26  7 10  6  8  3
```

Para conocer otros comandos más avanzados en la construcción de tablas, consultar `tabulate`, `fTable`, `margin.table`, `prop.table`, `addmargins` y `xtabs`.

Arreglos multidimensionales: Listas (S3)

Las listas son los objetos más genéricos del lenguaje, son de dimensiones arbitrarias donde cada dimensión contiene una colección de cualquier tipo de elementos. La mayoría de los comandos (*métodos S3*) devuelven objetos que en realidad son listas (con nombres de clases particulares).

Las listas se definen con el comando `list()`

```
1 L <- list(Nombre = c('Julia', 'Maria', 'Rosa'), Cargo  
  = factor(letters[1:5]))  
2 names(L)  
3 L
```

Las listas pueden estar anidadas

```
1 > (LL <- list(L, L))  
2 > length(LL) # 2  
3 > class(LL)  
4 [1] "list"
```

Existe una gran cantidad de comandos que operan sobre las listas: la familia `*apply`, la librería `plyr`, ...

Arreglos multidimensionales: Acceso a elementos II

- Para conocer los nombre de los miembros de un objeto: `names(x)`
- Clase de un objeto: `class(x)`
- Nombre de las dimensiones: `dimnames(x)`, `rownames(x)` y `colnames(x)`
- Tamaño de cada dimensión: `dim(x)`, longitud de listas: `length(LL)`

```
1 > data(airquality)
2 > head(airquality)
3   Ozone Solar.R Wind Temp Month Day
4 1     41     190  7.4   67     5   1
5 2     36     118  8.0   72     5   2
6 3     12     149 12.6   74     5   3
7 4     18     313 11.5   62     5   4
8 5      NA      NA 14.3   56     5   5
9 > names(airquality)
10 [1] "Ozone" "Solar.R" "Wind" "Temp" "Month" "Day"
11 > class(airquality)
12 [1] "data.frame"
13 > colnames(airquality)
14 [1] "Ozone" "Solar.R" "Wind" "Temp" "Month" "Day"
```

Arreglos multidimensionales: Acceso a elementos II

- ❶ Al igual que los vectores, los arreglos multidimensionales se pueden acceder mediante índices de posición e índices lógicos:

`x[, "miembro"]`, `x[fi, co]`, `x["miembro",]`, `x[[n]]` y `x[["miembro"]]`.

```
1 > airquality[1:2, ]
2   Ozone Solar.R Wind Temp Month Day
3 1     41     190  7.4   67     5   1
4 2     36     118  8.0   72     5   2
5 > airquality[1:2, "Solar.R"]
6 [1] 190 118
7 > airquality[1:2, c("Solar.R", "Ozone")]
8   Solar.R Ozone
9 1      190    41
10 2      118    36
```

```
1 > airquality[2, c("Solar.R", "Ozone")] <- c(93, 22)
2 > head(airquality, n = 3)
3   Ozone Solar.R Wind Temp Month Day
4 1     41     190  7.4   67     5   1
5 2     22      93  8.0   72     5   2
6 3     12     149 12.6   74     5   3
```

Arreglos multidimensionales: Acceso a elementos II

- 2 Se pueden acceder por los nombres de cada miembro `x$miembro` ó `x$"miembro"`

```
1 > names(airquality)
2 [1] "Ozone" "Solar.R" "Wind" "Temp" "Month" "Day"
3 > head(airquality, n = 3)
4   Ozone Solar.R Wind Temp Month Day
5 1     41     190  7.4   67     5   1
6 2     22      93  8.0   72     5   2
7 3     12     149 12.6   74     5   3
8 > airquality$Solar.R[1:3] <- c(100, 92, 88)
9 > head(airquality, n = 3)
10  Ozone Solar.R Wind Temp Month Day
11 1     41     100  7.4   67     5   1
12 2     36      92  8.0   72     5   2
13 3     12      88 12.6   74     5   3
```

```
1 > tmp = airquality$Temp
2 > length(tmp)
3 [1] 153
4 length(airquality$Temp)
5 [1] 153
```

Arreglos multidimensionales: Acceso a elementos II

- 3 Para la extracción de un subconjunto de una matriz o tabla se usa **subset**

```
1 > data(airquality)
2 > head(airquality)
3   Ozone  Solar.R Wind Temp Month Day
4 1     41     190  7.4   67     5   1
5 2     36     118  8.0   72     5   2
6 3     12     149 12.6   74     5   3
7 4     18     313 11.5   62     5   4
8 5      NA      NA 14.3   56     5   5
9 > S=subset(airquality,Temp>80,select=Ozone:Day)
10 > head(S)
11   Ozone  Solar.R Wind Temp Month Day
12 29     45     252 14.9   81     5  29
13 35      NA     186  9.2   84     6   4
14 36      NA     220  8.6   85     6   5
15 38     29     127  9.7   82     6   7
16 subset(airquality,Temp>80,select=c(Ozone,Temp))
```

```
1 subset(airquality,Day == 1,select = -Temp)
2 subset(airquality,select = Ozone:Wind)
```

Alcance de variables I: attach y search

Hay oportunidades donde el acceso individual de miembros de un objeto es preferible. Accesar `x$miembro` a través de `miembro`. Esto se logra con `attach` del objeto de interés a la *ruta de búsqueda*.

```
1 x = list(Letras=letters, Numeros=seq_along(letters))
2 x$Letras
3 attach(x)
4 Letras
```

```
1 > data(women); names(women)
2 [1] "height" "weight"
3 > search()
4 [1] ".GlobalEnv" "package:datasets" "package:base"
5 > mean(women$height)
6 [1] 65
7 > attach(women)
8 > search()
9 [1] ".GlobalEnv" "women" "package:datasets" "package:
   base"
10 > mean(height)
11 [1] 65
```

Alcance de variables I: detach

Una vez finalizado el uso de los **miembros**, se quita el vínculo a la ruta de búsqueda, con el comando **detach**

```
1 > x = list(Letras=letters, Numeros=seq_along(letters))
2 > x$Letras
3 [1] "a" "b" "c" ... "x" "y" "z"
4 > attach(x)
5 > search()
6 [1] ".GlobalEnv" "x" "package:datasets" "package:base"
7 > Letras
8 [1] "a" "b" "c" ... "x" "y" "z"
```

```
1 > detach(x)
2 > Letras
3 Error: object 'Letras' not found
4 > search()
5 [1] ".GlobalEnv" "package:datasets" "package:base"
6 > x$Letras
7 [1] "a" "b" "c" ... "x" "y" "z"
```

Alcance de variables I: Eclipse

Hay que tener cuidado al efectuar un `attach` a `x$miembro` cuando ya existe una variable con nombre `miembro`. En este caso `x$miembro` a través del `attach` es *eclipsado* por la variable ya existente en el entorno `miembro`.

```
1 > Nombre = 'Manuel'
2 > lista = data.frame(Nombre='Pedro')
3 > Nombre
4 [1] "Manuel"
5 > attach(lista)
6 The following object is masked _by_ '.GlobalEnv': Nombre
7 > Nombre
8 [1] "Manuel"
```

Nota: a través del comando `attach` no es posible modificar las variables usando simplemente el nombre del `miembro`. Este comando solo es para leer sus contenidos usando `miembro` y no `x$miembro`.

Los comandos `str`, `head`, `tail`

El comando `str` muestra de forma compacta la estructura de un objeto y es uno de los primeros comandos llamados al cargar o vaciar una base de datos:

```
1 > data(airquality)
2 > str(airquality)
3 'data.frame':   153 obs. of   6 variables:
4 $ Ozone   : int   41 36 12 18 NA 28 23 19 8 NA ...
5 $ Solar.R: int   190 118 149 313 NA NA 299 99 ...
6 $ Wind    : num   7.4 8 12.6 11.5 14.3 14.9 8.6 ...
7 $ Temp    : int   67 72 74 62 56 66 65 59 61 69 ...
8 $ Month   : int    5 5 5 5 5 5 5 5 5 5 ...
9 $ Day     : int    1 2 3 4 5 6 7 8 9 10 ...
```

Los comandos `head` y `tail` muestran las primeras y últimas n líneas de un objeto (tabla, matriz, función, ...)

```
1 > head(airquality)
2   Ozone Solar.R Wind Temp Month Day
3 1     41     190  7.4   67     5   1
4 ...
5 5     12     149 12.6   74     5   3
6 6     NA      NA 14.3   56     5   5
```




Práctica

El objetivo de esta práctica es entender el efecto que produce el comando `attach`

- 1 Cargar los datos cars: `data(cars)`
- 2 Ejecute `plot(speed, dist)`
- 3 Guardar el vector de 1 a 25 en paso 0.5 en la variable `speed`
- 4 Construya el vector `dist` como el cuadrado de `speed`
- 5 Ejecute el comando `plot(speed, dist)`
- 6 ¿Cómo se puede volver a acceder a las variables `speed` y `dist` asociados a `cars`?

Operadores sobre arreglos multidimensionales

Existen varios comandos que operan sobre las dimensiones de los arreglos multidimensionales

Aritméticos Suma por filas (**rowSums**) o columnas (**colSums**)

Estadísticos Promedio por filas (**rowMeans**) o columnas (**colMeans**)

Arbitrario Otras operaciones se pueden construir para que se ejecuten de forma repetida a lo largo de las dimensiones definidas, usando el comando **apply**

```
1 apply(X, MARGIN=1, FUN=median)
2 apply(D, 2, sd)
```

Para las listas también hay comandos que iteran sobre cada miembro

```
1 L <- list(Edad = c(26, 33, 57, 28), Peso = c(72, 88,
      56, 64))
2 LM <- lapply(L, mean)
3 names(LM) <- paste0('Prom.', names(L))
4 LM
```

Métodos y datos faltantes para arreglos multidimensionales

No siempre los datos a ser analizados están completos, aquellas observaciones faltantes se denotan por la palabra clave **NA**.

- Generalmente al contener un **NA**, el resultado de la operación suele ser **NA**:

```
1 > data <- c(NA, 1:5}  
2 > sum(data)  
3 [1] NA  
4 > mean(data)  
5 [1] NA
```

- El manejo de **NA** se puede especificar a cada comando:

```
1 > sum(data, na.rm = TRUE)  
2 [1] 15  
3 > mean(data, na.rm = TRUE)  
4 [1] 3
```

- Un comando para omitir **todas** aquellas entradas de una tabla (matriz, etc.) que contengan **NA** es **na.omit**:

```
1 > na.omit(data)  
2 [1] 1 2 3 4 5
```



Práctica

Score de jueces norteamericanos (John Hartigan, 1977).

Dado los datos `USJudgeRatings`

- 1 Entienda la estructura de los datos, `?USJudgeRatings` muestra los detalles de cada columna
- 2 Estime el juez con mayor y/o menor score `rowSums`, `min`, `max`
- 3 Estime el promedio grupal (`summary`) y los promedios de cada prueba
- 4 Ordene los jueces según su score `sort`, `matrix`, `rownames`

El comando aggregate

El comando **aggregate** llama cierta función (**mean**, **summary**) para cada nivel (factor, valor, etc.) de alguna variable.

Ejemplo:

Se quiere el promedio de la variable y (**d\$y**) según los niveles de x (**d\$x**)

```
1 > d <- data.frame(x = rep(1:3, each=3), y = rep(1:3, each=3)
  + rnorm(9))
2 > d
3   x      y
4 1 1 1.8027201
5 2 1 0.4231649
6 3 1 1.7083813
7 4 2 0.4208163
8 5 2 3.1780815
9 6 2 1.3696267
10 7 3 2.5009258
11 8 3 1.8170497
12 9 3 1.4221910
```

El comando aggregate

El comando **aggregate** llama cierta función (**mean**, **summary**) para cada nivel (factor, valor, etc.) de alguna variable.

Ejemplo:

Se quiere el promedio de la variable **y** (**d\$y**) según los niveles de **x** (**d\$x**)

```
1 > d <- data.frame(x = rep(1:3, each=3), y = rep(1:3, each=3)
  + rnorm(9))
2 > d
3   x      y
4 1 1 1.8027201
5 2 1 0.4231649
6 3 1 1.7083813
7 4 2 0.4208163
8 5 2 3.1780815
9 6 2 1.3696267
10 7 3 2.5009258
11 8 3 1.8170497
12 9 3 1.4221910
```

```
1 > aggregate(y ~ x, data = d, mean)
2   x      y
3 1 1 1.311422
4 2 2 1.656175
5 3 3 1.913389
```

El comando aggregate

Se quiere el promedio de la variable y ($d\$y$) según los niveles de x ($d\$x$)

```
1 > aggregate(y ~ x, data = d, mean)
2      x      y
3 1 1 1.311422
4 2 2 1.656175
5 3 3 1.913389
```

Diferencia entre `mean` y `aggregate/mean`

Ejemplo del comando aggregate

Ejemplo:

Se quiere un resumen de la variable y ($d\$y$) según los niveles de x ($d\$x$).

```
1 > aggregate(d$y ~ d$x, FUN=summary)
2   d$x d$y.Min. d$y.1st Qu. d$y.Mean d$y.3rd Qu. d$y.Max.
3 1 1    0.169    0.847          1.091    1.552      1.579
4 2 2    2.489    2.507          2.884    3.082      3.638
5 3 3    1.644    2.192          2.605    3.086      3.433
```

```
1 > summary(d)
2           x           y
3 Min.      :1   Min.      :0.169
4 1st Qu.:1   1st Qu.:1.579
5 Median :2   Median :2.489
6 Mean    :2   Mean     :2.193
7 3rd Qu.:3   3rd Qu.:2.739
8 Max.    :3   Max.     :3.638
```

Se muestra la diferencia entre `summary` y `aggregate/summary`.

Funciones

Las funciones, al igual que en matemáticas, definen bajo un *nombre* una serie de cálculos y/o tareas. Las funciones reciben la información a procesar a través de los *parámetros de entrada* y la *salida* de la función puede variar, puede ser: una variable, una gráfica, etc. . .

y	=	colores	(genero)
Salida		Nombre		Entrada	

Funciones

Las funciones son muy útiles para efectuar tareas repetitivas que dependen de algunas pocas variables, por ejemplo:

- ➊ Dado un vector de datos, reportar la media y desviación estándar en una tabla (`summary`)
- ➋ Dado un vector x calcular $y = x \pm 0,5 * \sqrt{x}$
- ➌ Construir un vector de colores según el género:
`col = colores(género)`

Funciones

Las funciones son muy útiles para efectuar tareas repetitivas que dependen de algunas pocas variables, por ejemplo:

- ❶ Dado un vector de datos, reportar la media y desviación estándar en una tabla (`summary`)
 - ❷ Dado un vector x calcular $y = x \pm 0,5 * \sqrt{x}$
 - ❸ Construir un vector de colores según el género:
`col = colores(género)`
- Las funciones pueden tener ninguna o al menos una entrada y cero o mas salidas.
 - `z = matrix()`: una salida sin entradas
 - `plot(x)`: un parámetro de entrada, vector `x` y una salida
 - `plot(x, y, col = 'red')`: tres entradas, vectores `x`, `y` y una cadena de caracteres `'red'`

Funciones

Las funciones son muy útiles para efectuar tareas repetitivas que dependen de algunas pocas variables, por ejemplo:

- ❶ Dado un vector de datos, reportar la media y desviación estándar en una tabla (`summary`)
 - ❷ Dado un vector x calcular $y = x \pm 0,5 * \sqrt{x}$
 - ❸ Construir un vector de colores según el género:
`col = colores(género)`
- Las funciones pueden tener ninguna o al menos una entrada y cero o mas salidas.
 - `z = matrix()`: una salida sin entradas
 - `plot(x)`: un parámetro de entrada, vector `x` y una salida
 - `plot(x, y, col = 'red')`: tres entradas, vectores `x`, `y` y una cadena de caracteres `'red'`
 - Cada parámetro de entrada esta separado por una coma (,)

Funciones

Las funciones son muy útiles para efectuar tareas repetitivas que dependen de algunas pocas variables, por ejemplo:

- ❶ Dado un vector de datos, reportar la media y desviación estándar en una tabla (`summary`)
 - ❷ Dado un vector x calcular $y = x \pm 0,5 * \sqrt{x}$
 - ❸ Construir un vector de colores según el género:
`col = colores(género)`
- Las funciones pueden tener ninguna o al menos una entrada y cero o mas salidas.
 - `z = matrix()`: una salida sin entradas
 - `plot(x)`: un parámetro de entrada, vector `x` y una salida
 - `plot(x, y, col = 'red')`: tres entradas, vectores `x`, `y` y una cadena de caracteres `'red'`
 - Cada parámetro de entrada esta separado por una coma (,)
 - Los parámetros de entrada pueden o no tener una *denominación*:
`x = rnorm(20, mean=0, sd=2)`: en este caso el primer parámetro es *anónimo* y los otros dos son *denominados*.

Definiendo funciones nuevas

Nuevas funciones se definen por

```
1 nombre = function(arg1,arg2,arg3=val3) { comandos }
```

Por ejemplo

```
1 > potencia <- function(x, n = 2) { return(x^n) }
2 > potencia(2)
3 [1] 4
4 > potencia(2, 3)
5 [1] 8
7 > potencia()
8 Error in potencia() : argument "x" is missing, with no
  default
10 > args(potencia)
11 function (x, n = 2)
```

Notas

El comando **return** indica la salida de la función.

El comando **args** informa sobre los argumentos de entrada de una función.



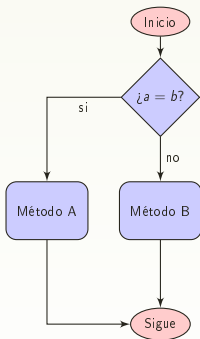
Práctica de funciones

- 1 Construir una función que dados dos parámetros x_1 y x_2 , calcule $x_1 - x_2$
- 2 Modifique la función tal que el parámetro de entrada x_2 sea opcional, con valor por defecto de cero
- 3 Modifique la función del apartado anterior para que el argumento *opcional* x_2 sea por defecto 2 veces el parámetro x_1

Programación estructurada

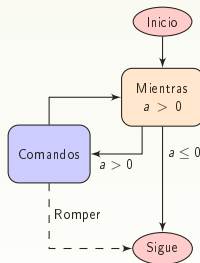
La programación estructurada tiene dos categorías

Ejecución selectiva



if, switch

Ejecución repetitiva



for, while, break, next,
repeat, replicate

Condicional “si... entonces” (if)

Este comando permite ejecutar selectivamente una serie de comandos.
Tiene tres formas generales

Forma simple “si”

```
1 if (condición) {  
2     comando1  
3     ...  
4 }
```

Condicional “si... entonces” (if)

Este comando permite ejecutar selectivamente una serie de comandos.
Tiene tres formas generales

Forma simple “si”

```
1 if (condición) {  
2     comando1  
3     ...  
4 }
```

Forma “si...de otro modo”

```
1 if (condicion1) {  
2     comando1  
3     ...  
4 } else {  
5     comando2  
6     ...  
7 }
```

Condicional “si... entonces” (if)

Este comando permite ejecutar selectivamente una serie de comandos.

Tiene tres formas generales

Forma simple “si”

```
1 if (condición) {  
2     comando1  
3     ...  
4 }
```

Forma “si...de otro modo”

```
1 if (condicion1) {  
2     comando1  
3     ...  
4 } else {  
5     comando2  
6     ...  
7 }
```

Forma completa

```
1 if (condicion1) {  
2     comando1  
3     ...  
4 } else if  
5     (condicion2) {  
6     comando2  
7     ...  
8 } else if  
9     (condicion3) {  
10    comando3  
11    ...  
12 } else {  
13    comando4  
14    ...  
15 }
```

Condicional “si... entonces” (if)

Para ilustrar el uso de los condicionales, calculamos las raíces de una ecuación de segundo grado

$$P_2(x) = ax^2 + bx + c$$

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
1 x1 <- ( -b + sqrt(b^2-4*a*c) ) / ( 2*a )  
2 x2 <- ( -b - sqrt(b^2-4*a*c) ) / ( 2*a )
```

¿Qué sucede si **a** es cero?

Condicional “si... entonces” (if)

Para ilustrar el uso de los condicionales, calculamos las raíces de una ecuación de segundo grado

$$P_2(x) = ax^2 + bx + c$$

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
1 x1 <- ( -b + sqrt(b^2-4*a*c) ) / ( 2*a )  
2 x2 <- ( -b - sqrt(b^2-4*a*c) ) / ( 2*a )
```

¿Qué sucede si **a** es cero?

```
1 if ( a == 0 ) {  
2     x1 <- -c / b  
3     x2 <- x1  
4 } else {  
5     x1 = ( -b + sqrt(b^2-4*a*c) ) / ( 2*a )  
6     x2 = ( -b - sqrt(b^2-4*a*c) ) / ( 2*a )  
7 }
```

¿Qué sucede si **a** y **b** son cero?

Condicional “si... entonces” (if)

```
1 if ( a == 0 ) {  
2     if ( b == 0 ) {  
3         stop('P_no_tiene_raices')  
4     } else {  
5         x1 <- -c / b  
6         x2 <- x1  
7     }  
8 } else {  
9     x1 = ( -b + sqrt(b^2-4*a*c) ) / ( 2*a )  
10    x2 = ( -b - sqrt(b^2-4*a*c) ) / ( 2*a )  
11 }
```

¿Qué sucede si el discriminante $d = b^2 - 4ac$ es positivo, cero ó negativo?

Condicional “si... entonces” (if)

```
1 if ( a == 0 ) {  
2     if ( b == 0 ) {  
3         stop('P_no_tiene_raices')  
4     } else {  
5         x1 <- -c/b  
6         x2 <- x1  
7     }  
8 } else {  
9     d <- b^2 - 4*a*c  
10    if ( d == 0 ) {  
11        x1 <- -b / ( 2*a )  
12        x2 <- x1  
13    } else if ( d > 0 ) {  
14        x1 <- ( -b + sqrt(d) ) / ( 2*a )  
15        x2 <- ( -b - sqrt(d) ) / ( 2*a )  
16    } else {  
17        x1 = ( -b + i * sqrt(-d) ) / ( 2*a )  
18        x2 = ( -b - i * sqrt(-d) ) / ( 2*a )  
19    }  
20 }
```

Condicional “si-entonces” (ifelse)

El comando `ifelse` es útil para ejecuciones selectivas de dos ramas

```
1 ifelse(prueba, si, no)
```

Ejemplos

```
1 > ifelse(-1 : 1 < 1, TRUE, FALSE)
2 [1] TRUE TRUE FALSE
```

```
1 x <- rand(5)
2 limite = 0.75
3 idx = ifelse(x <= limite, T, F)
4 # idx = x <= limite
```

```
1 x <- c(6 : -4)
2 sqrt(x) # warning
3 sqrt(ifelse(x >= 0, x, NA)) # ok
```


Comando stopifnot

El comando `stopifnot` es útil para detener la ejecución del programa si no se cumplen las condiciones básicas supuestas (necesarias)

```
1 stopifnot(...)
```

Ejemplo

```
1 > x = c(1, NA, 3)
2 > stopifnot( ! is.na(x) )
3 Error: !is.na(x) are not all TRUE
```

Nota: el comando `stopifnot(A, B)` es conceptualmente equivalente a

```
1 {
2   if(any(is.na(A)) || !all(A)) stop(...)
3   if(any(is.na(B)) || !all(B)) stop(...)
4 }
```

También existen los comandos `stop` y `warning` para parar y/o notificar sobre algún error o problema.

Condicional switch

Este condicional permite ejecutar secciones de códigos según algún parámetro

```
1 switch(expr, ...)  
2 switch(expr, a=1, b=2)  
3 switch(expr, a=1, b=2, NA)  
4 switch(expr, a=1, b=2, c=, d=4)  
5 switch(expr, a=1, b=2, c=, d=4, NA)
```

Notas

- ❶ Si algún parámetro de entrada no tiene valor, se utiliza el valor del siguiente parámetro que si esté especificado

```
1 > switch("c", a=1, b=2, c=, d=4)  
2 [1] 4
```

- ❷ El valor por defecto es **NULL**

```
1 > switch("c", a=1, b=2, c=, d=)  
2 [1] NULL
```

Ciclos “para” (for)

Este comando repite la secuencia de comandos entre las palabras claves `for { y }` reemplazando el valor de la variable (`i`) en cada iteración del ciclo por el siguiente elemento de la secuencia `secuencia`, su sintaxis es

```
1 for ( i in secuencia ) {  
2     comando1  
3     ...  
4 }
```

El siguiente ejemplo imprime el valor de la variable `i` en cada *iteración*

```
1 for ( i in 1 : 5 ) {  
2     print(i)  
3 }
```

```
1 > Nombres = c('María','Leo','Bea','Luis')  
2 > for ( i in Nombres ) {  
3 +     print(toupper(i))           # espera hasta  
4 + }                               # encontrar }  
5 [1] "MARÍA" "LEO" "BEA" "LUIS"
```

Ciclo “mientras” (while)

Este comando repite la secuencia de comandos entre las palabras claves **while { y }** siempre que se cumpla la condición del ciclo, su sintaxis es

```
1 while ( condición ) {  
2     comando1  
3     ...  
4 }
```

El siguiente ejemplo imprime los números del 10 al 1 menos el 7

```
1 k = 10  
2 while ( k > 0 ) {  
3     if ( k == 7 ) { k <- k - 1; next }  
4     print(k)  
5     k <- k - 1  
6 }
```

Ciclos “para” (for) y “mientras” (while)

Algunos comentarios

- Los ciclos se pueden *interrumpir* en cualquier momento con el comando **break** (romper).
- Parte del ciclo puede ser *saltado* con el comando **next**
- De ser posible, evitar el uso de ciclos: una fortaleza de este lenguaje es justamente *vectorizar* las operaciones repetitivas.

Ejemplo de salto de iteración y ruptura prematura para ambos tipos de ciclo

```
1 for (i in secuencia) {  
2     comando1  
3     ...  
4     if (nueva condición) {  
5         next  
6     }  
7 }
```

```
1 while (condicion)  
2     comando1  
3     ...  
4     if (nueva condición) {  
5         break  
6     }  
7 }
```

Comando replicate

Este comando es un *envoltorio* para el comando **sapply** para evaluar de forma repetitiva alguna expresión que generalmente involucra el uso de números aleatorios.

```
1 replicate(n, expr, simplify = "array")
```

Estimar π utilizando el método de Monte Carlo

```
1 > p <- replicate(10, {
2     x = runif(1000)
3     y = runif(1000)
4     4 * sum( y^2 <= (1 - x^2) ) / 1000
5 })
6 > print(p)
7 [1] 3.16 3.168 3.09 3.172 3.04 3.1 3.08 3.11 3.02 3.11
8 > mean(p)
9 [1] 3.1524
```