



## Introducción a R

Flavio E. Spetale  
spetale@cifasis-conicet.gov.ar



# ¿Qué es R?

**R** es un lenguaje de programación, creado por Ross Ihaka y Robert Gentleman, cuya característica principal es que forma un entorno de análisis estadístico para la manipulación de datos, su cálculo y creación de gráficos.

La página principal del proyecto “**R-project**” <https://cran.r-project.org/>, en ella podremos conseguir gratuitamente el programa, manuales, paquetes y demás elementos que forman la gran familia que es R.

**R** es un proyecto vivo, ya que los usuarios pueden contribuir al proyecto implementando funciones, librerías, ... Ningún otro programa estadístico en la actualidad reúne la cantidad de recursos y manejabilidad que posee **R**.



# Ventajas de R

- Capacidad de combinar, manipular, modificar datos y funciones.
- Gráficos de alta calidad
- La comunidad de R es muy dinámica, con gran crecimiento del número de paquetes.
- Hay extensiones específicas a nuevas áreas como BioInformática, geoestadística y modelos gráficos.
- Es un lenguaje orientado a objetos.



# Instalación de R

La versión más actualizada de *R* se puede descargar gratis de su sitio oficial *R- CRAN* (abreviación de *The Comprehensive R Archive Network*) en <http://cran.r-project.org/>.



The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (Friday 2017-04-21, You Stupid Darkness) [R-3.4.0.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.
- Source code of older versions of R is [available here](#).
- Contributed extension [packages](#)

Questions About R

CRAN  
[Mirrors](#)  
[What's new?](#)  
[Task Views](#)  
[Search](#)

About R  
[R Homepage](#)  
[The R Journal](#)

Software  
[R Sources](#)  
[R Binaries](#)  
[Packages](#)  
[Other](#)

Documentation  
[Manuals](#)  
[FAQs](#)  
[Contributed](#)



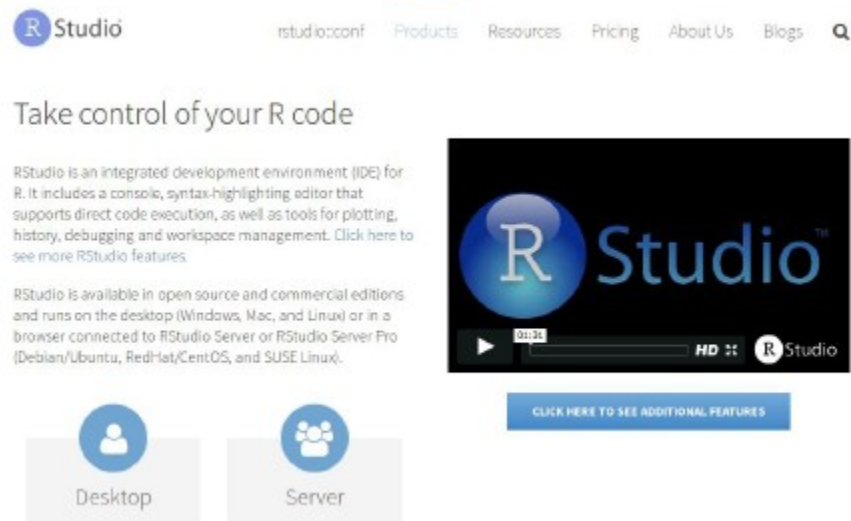


# Instalación de R-Studio

El **RStudio** es un editor que permite escribir líneas de instrucciones (“script”) en lenguaje R para ser ejecutadas por el intérprete de R. Su uso reporta una serie de ventajas como la de poder guardar y editar el script y reconocer las sintaxis de programación en **R**.

El **RStudio** identifica en colores la sintaxis conocida y brinda otras comodidades que hacen más sencillo el trabajo con el programa.

**RStudio** se puede descargar en forma gratuita en <http://rstudio.org>



The screenshot shows the RStudio website homepage. At the top is the RStudio logo and a navigation menu with links: rstudio:conf, Products, Resources, Pricing, About Us, Blogs, and a search icon. Below the navigation is the heading "Take control of your R code". Under this heading is a paragraph describing RStudio as an integrated development environment (IDE) for R, mentioning features like a console, syntax-highlighting editor, direct code execution, plotting tools, history, debugging, and workspace management. Below this is another paragraph stating that RStudio is available in open source and commercial editions and runs on Windows, Mac, Linux, or in a browser connected to RStudio Server or RStudio Server Pro. To the right of the text is a video player showing the RStudio logo and the text "R Studio". Below the video player is a blue button that says "CLICK HERE TO SEE ADDITIONAL FEATURES". At the bottom of the page are two buttons: "Desktop" with a user icon and "Server" with a group of people icon.



# Funcionamiento de R

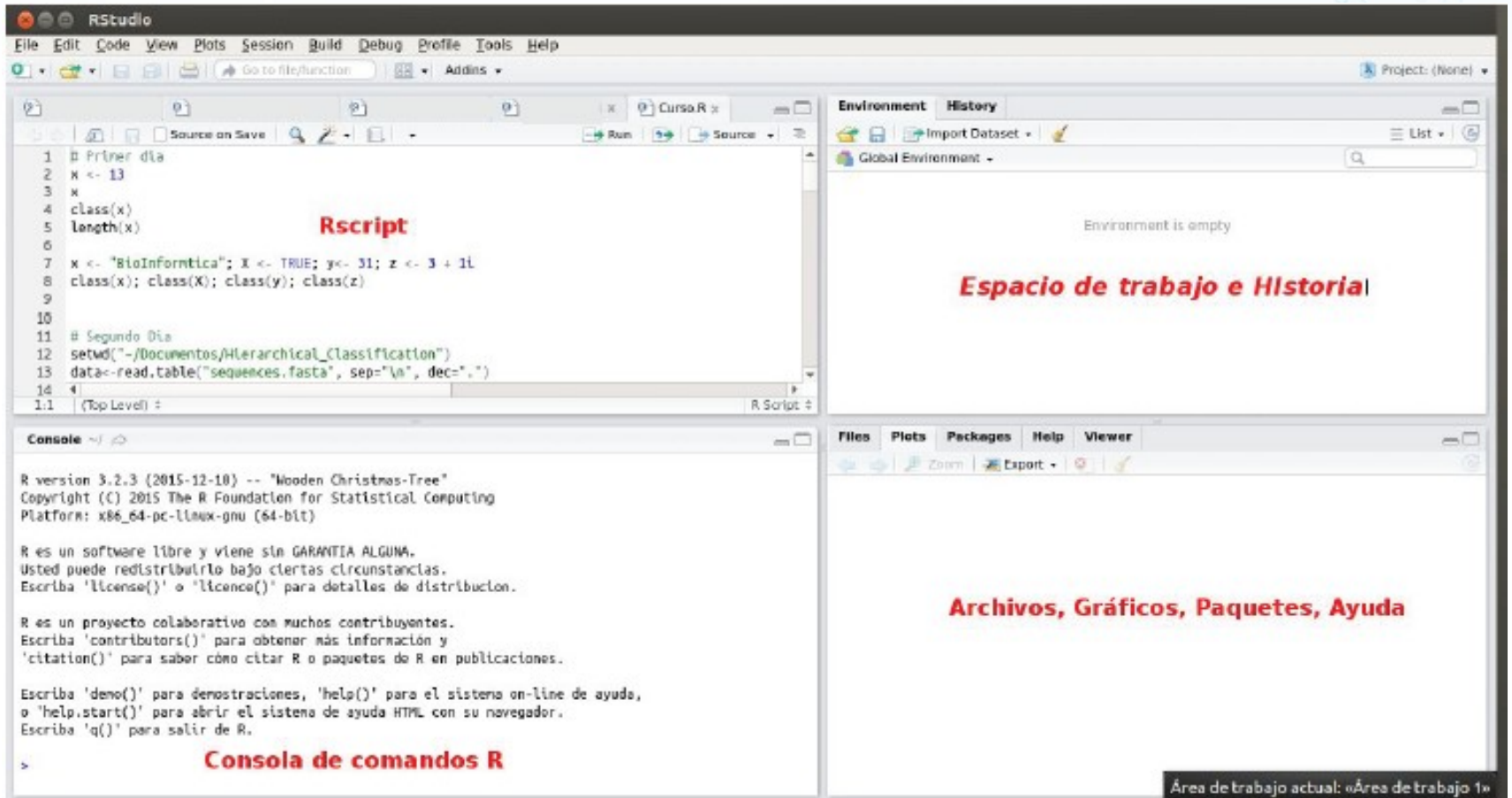
**R** es un lenguaje de programación ***interpretado orientado a objetos***.

*Un lenguaje interpretado significa que las instrucciones escritas en el teclado son ejecutados directamente sin necesidad de construir nada más.*

*Orientado a Objetos significa que las variables, datos, funciones, resultados, se guardan en la memoria activa del computador en forma de objetos con un nombre específico.*

El usuario puede modificar o manipular estos objetos con operadores (aritméticos, lógicos, y comparativos) y funciones (que a su vez son objetos).

# R y RStudio



The image shows the RStudio interface with a blue clock background. The interface is divided into four main panes:

- Source Editor (Top Left):** Contains an R script with the following code:

```
1 # Primer día
2 x <- 13
3 x
4 class(x)
5 length(x)
6
7 x <- "BioInformtica"; X <- TRUE; y<- 31; z <- 3 + 11
8 class(x); class(X); class(y); class(z)
9
10
11 # Segundo Día
12 setwd("~/Documentos/Hierarchical_classification")
13 data<-read.table("sequences.fasta", sep="\n", dec=",")
14
15 (TopLevel) +
```
- Environment (Top Right):** Shows the Global Environment, which is empty. The text "Espacio de trabajo e Historia" is displayed in red.
- Console (Bottom Left):** Displays the R version 3.2.3 (2015-12-10) and copyright information. It also includes instructions for using R, such as "R es un software libre y viene sin GARANTIA ALGUNA." and "Escriba 'demo()' para demostraciones, 'help()' para el sistema on-line de ayuda, o 'help.start()' para abrir el sistema de ayuda HTML con su navegador. Escriba 'q()' para salir de R." The text "Consola de comandos R" is displayed in red.
- Files, Plots, Packages, Help, Viewer (Bottom Right):** These panes are currently empty. The text "Archivos, Gráficos, Paquetes, Ayuda" is displayed in red.

The status bar at the bottom right indicates the current workspace: "Área de trabajo actual: «Área de trabajo 1»".



# ***Manejo de Objetos en R***

Los objetos en **R** tienen un nombre, un contenido y atributos que especifican el tipo de dato que presentan.

Todo **objeto** tiene dos atributos intrínsecos: **tipo** y **longitud**.

El **tipo** se refiere a la clase básica de los elementos en el objeto: numérico, carácter, complejo y lógico.

La **longitud** es simplemente el número de elementos en el objeto.

Para asignar un valor a un objeto se utiliza la flecha hacia la izquierda

***variable <- valor***

Para ver el tipo y la longitud de un objeto se pueden usar las funciones *class* y *length* respectivamente.





# ***Manejo de Objetos en R***

- Para saber los objetos que tenemos en el espacio de trabajo se utiliza `ls()`.
- Escribir el nombre de un objeto muestra su contenido: `mean`.
- Para guardar el contenido del espacio de trabajo se pueden utilizar las funciones `save.image()` y `save(<objetos>,file="nombre.RData")`.
- Para acceder a objetos de la carpeta de trabajo se pueden adjuntar `attach("misdatos.RData")`

# Los objetos numéricos: enteros, real, complejos

```
x <- 13
```

```
x
```

```
[1] 13
```

```
class(x)
```

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

```
length(x)
```

```
[1] 1
```

```
z <- 3 + 1i
```

```
class(z)
```

```
[1] "complex"
```

En forma exponencial:

```
x <- 2.1e23
```

```
x
```

```
[1] 2.1e+23
```

Valores numéricos finitos:

```
x <- 5/0
```

```
x
```

```
[1] Inf
```

```
exp(x)
```

```
[1] Inf
```

```
exp(-x)
```

```
[1] 0
```

```
x - 1e15
```

```
[1] Inf
```

Valores no numéricos

```
y <- x - x
```

```
y
```

```
[1] NaN
```



# Los objetos de caracteres y lógicos

```
x <- "BioInformática"; X <- TRUE;
```

```
class(x); class(X)
```

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

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

## Operadores

Aritmético		Lógicos		Comparativos	
+	Suma	!	Not	>	Mayor
-	Resta	&	And	<	Menor
*	Multiplicación	&&	And	>=	Mayor o igual
/	División		Or	<=	Menor o igual
^	Potencia		Or	!=	Distinto
%%	Modulo			==	Igual
%/% División de enteros					

# Conjuntos de datos en R

**vector:** colección ordenada de elementos del mismo tipo.

`V ← c(2,4,5,78,-1)` crea un vector de 5 elementos enteros

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

**arreglo:** generalización multidimensional del vector donde los elementos son del mismo tipo.

`array(data = NA, dim = length(data), dimnames = NULL)`

*data* un vector de datos

*dim* vector que especifica las dimensiones

*dimnames* vector que especifica los nombres





# Conjuntos de datos en R

**matriz:** un arreglo bidimensional.

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
```

*data* un vector opcional de datos

*nrow* especifica la cantidad de filas

*ncol* especifica la cantidad de columnas

*byrow* valor lógico que indica si los valores en *data* se deben rellenar por columnas (por defecto *False*) o por filas (*True*).

*dimnames* permite asignar nombres a las filas y columnas.

```
matrix(data=5, nr=2, nc=2)
```

```
[,1] [,2]  
[1,] 5 5  
[2,] 5 5
```

```
matrix(1:6, 2, 3)
```

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

```
matrix(1:6, 2, 3, byrow=TRUE)
```

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



# Conjuntos de datos en R

**factor:** tipo de vector para datos cualitativos.

```
x <- factor(c(1, 2, 2, 1, 1, 2, 1, 2, 1))
```

**data frame:** similar a un arreglo pero con columnas de diferentes tipos. Es el objeto mas habitual para los datos experimentales.

```
data.exp <- data.frame(ID=c("gen0", "genB", "genZ"), subj1 = c(10, 25, 33), subj2 = c(NA, 34, 15), oncogen = c(TRUE, TRUE, FALSE), loc = c(1,30, 125))
```

**expresión:** serie de caracteres. Cuando se escribe un comando directamente en el teclado, este es evaluado por R y ejecutado si es válido. En muchos casos, es útil construir una expresión (*expression*) *sin evaluarla y se puede evaluar con eval*.

```
media <- 3; varianza <- 0.5; x <- 1  
f_gausiana <- expression (exp(-(x-media)^2/2*varianza))  
eval (f_gausiana)  
[1] 0.3678794
```



# Conjuntos de datos en R

**lista:** colección ordenada de objetos. Los elementos pueden o no ser del mismo tipo de dato. Así una lista puede estar compuesta por un vector numérico, un valor lógico, una matriz y una función.

*list (nombre<sub>1</sub> = objeto<sub>1</sub>, nombre<sub>2</sub> = objeto<sub>2</sub>, ... , nombre<sub>m</sub> = objeto<sub>m</sub>)*

```
X <- c("F", "L", "A", "V", "I", "O")
y <- c("E", "Z", "E", "Q", "U", "I", "E", "L")
z <- c("Spetale")
L1 <- list(x, y, z)
L1
[[1]]
[1] "F" "L" "A" "V" "I" "O"
[[2]]
[1] "E" "Z" "E" "Q" "U" "I" "E" "L"
[[3]]
[1] "Spetale"
```

```
names(L1)
NULL
```

```
L2 <- list(Primer_Nombre=x, Segundo_Nombre=y, Apellido=z)
L2
$Primer_Nombre
[1] "F" "L" "A" "V" "I" "O"
$Segundo_Nombre
[1] "E" "Z" "E" "Q" "U" "I" "E" "L"
$Apellido
[1] "Spetale"
```

```
names(L2)
[1] "Primer_Nombre" "Segundo_Nombre" "Apellido"
```

# Conjuntos de datos en R

**Serie de tiempo:** un objeto creado por la función `ts` a partir de un vector (serie de tiempo única) o una matriz (serie multivariada).

```
ts(data = NA, start = 1, end = numeric(), frequency = 1, deltat = 1, ts.eps =  
getOption("ts.eps"), class = , names = )
```

```
ts(1:10, start = 1959)
```

Time Series:

Start = 1959

End = 1968

Frequency = 1

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

```
ts(1:33, frequency = 12, start = c(1982, 4))
```

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1982				1	2	3	4	5	6	7	8	9
1983	10	11	12	13	14	15	16	17	18	19	20	21
1984	22	23	24	25	26	27	28	29	30	31	32	33





# Ejercicios

Realizar las siguientes instrucciones y analizar su respuesta.

```
m1 <- matrix(1, nr = 2, nc = 2)
m2 <- matrix(letters, nr = 2, nc = 2)
rbind(m1, m2)
```

```
cbind(m1, m2)
```

```
diag(m1)
```

```
diag(m1) <- 20
```

```
m2 <- matrix(1:4, nr = 2, nc = 2)
m1 * m2
```

```
crossprod(m1,m2)
```

```
m1 <- matrix(1, nr = 2, nc = 3)
m2 <- matrix(1:6, nr = 3, nc = 2)
m3<- m1 %*% m2
```

```
m3<- t(m1) %*% t(m2)
```



# Ejercicios

Realizar las siguientes instrucciones y analizar su respuesta.

```
bebidas <- factor(c("jugo","jugo","vino","agua", "agua"))
```

```
p <- c(3,7,18)
```

```
q <- c("F", "E", "S")
```

```
x <- list(num=p, letra=q)
```

```
x$letra
```

```
x[2]
```

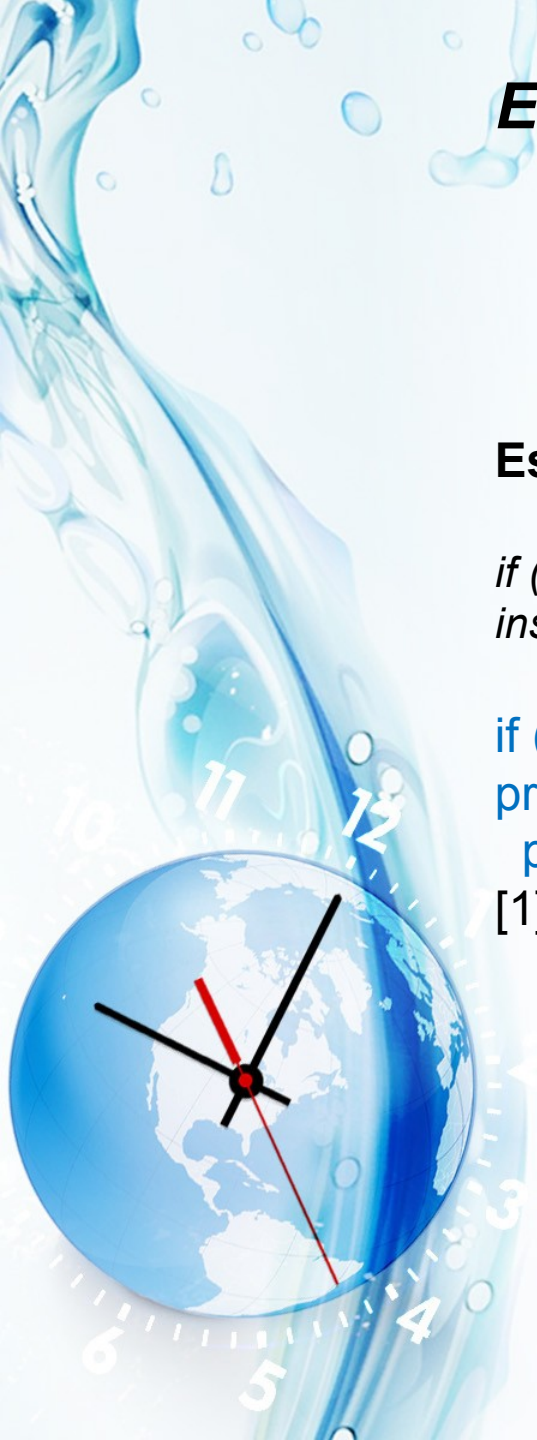


# ***Estructuras de control***

## **Estructura condicional *if***

*if* (expresión) {secuencia de instrucciones 1} else {secuencia de instrucciones 2}

```
if (10 > x) {  
  print("x es MENOR")} else {  
    print("x es MAYOR")}  
[1] "x es MAYOR"
```



# Estructuras de control

## Control condicional *switch*

*switch* (*expresión*, nombre<sub>1</sub>=*{secuencia de instrucciones}*, nombre<sub>2</sub>=*{secuencia de instrucciones}*, ..., nombre<sub>n</sub>=*{secuencia de instrucciones}*)

Se evalúa una *expresión* y dependiendo de su respuesta, se ejecutará la acción que coincida con dicha respuesta.

```
x<-"A"  
switch(x,  
  'A'=print("Primera condición"),  
  'B'=print("Segunda condición"),  
  'C'=print("Tercera condición"))
```

```
[1] "Primera condición"
```





# ***Estructuras de control***

## **Ciclos de repetición**

### **Repeticiones por un número conocido de veces *for***

for (variable\_for in expresión) {secuencia de instrucciones}

```
num_complex <- c( 3 + 2i, 6 + 12i, 9 - 3i, pi)
```

```
for (j in 1:4) {  
  print(num_complex[j]) }
```

```
[1] 3+2i  
[1] 6+12i  
[1] 9-3i  
[1] 3.141593+0i
```



# Estructuras de control

## Ciclos de repetición

### Repeticiones mientras se cumple una condición *while*

`while (condición) {secuencia de instrucciones}`

La secuencia de instrucciones se evalúa repetidamente hasta que la *condición* sea falsa.

```
i <- 6  
while (i >= 6 & i <= 11) {  
  print(LETTERS[i])  
  i <- i + 1  
}
```

```
[1] "F"  
[1] "G"  
[1] "H"  
[1] "I"  
[1] "J"  
[1] "K"
```



# Estructuras de control

## Ciclos de repetición

### Repeticiones infinitas *repeat ... break*

```
repeat {secuencia de instrucciones  
  break}
```

La secuencia de instrucciones se evalúa repetidamente hasta que se encuentra la instrucción *break* que interrumpe el ciclo.

```
i <- 1  
repeat {  
  print(letters[i])  
  i <- i + 1  
  if (i > 6) break}
```

```
[1] "a"  
[1] "b"  
[1] "c"  
[1] "d"  
[1] "e"  
[1] "f"
```



## Ejercicios

Realizar las siguientes instrucciones y analizar su respuesta.

```
x<-colours()  
mode(x)  
length(x)
```

```
if (x=="lightskyblue") {  
  print("El color pertenece a la paleta de colores de R")} else {  
  print("El color no pertenece a la paleta de colores de R")}
```

```
if (x[1]=="lightskyblue") {  
  print("El color pertenece a la paleta de colores de R")} else {  
  print("El color no pertenece a la paleta de colores de R")}
```

```
if (x[430]=="lightskyblue") {  
  print("El color pertenece a la paleta de colores de R")} else {  
  print("El color no pertenece a la paleta de colores de R")}
```





## Ejercicios

Realizar las siguientes instrucciones y analizar su respuesta.

```
x<- c("M", "D", "S", "E", "D", "M", "G", "S", "G", "R", "L", "G", "Q",  
      "I", "L", "K", "Q", "I", "W", "R", "Q", "N", "L", "M", "L", "K", "Q", "A",  
      "L", "L", "G", "D", "N")
```

```
numb_M <-0  
for (i in 1:length(x)){  
  if (x[i]=="M") numb_M<—numb_M+1}
```

```
numb_W<-0  
i<-0  
while(i<length(x)){  
  i<-i+1  
  if (x[i]=="W") numb_W<-numb_W+1}
```



# Funciones

```
NombreDeFuncion <- function(arg1, arg2, ..., argn) {  
    secuencia de instrucciones}
```

```
hipotenusa <- function(x, y){  
    sqrt(x^2 + y^2)}
```

```
class(hipotenusa)  
[1] "function"
```

```
hipotenusa <- function(x=3, y=4) { # valores por ausencia  
    return( sqrt( x^2 + y^2 ) )}
```

```
# Llamamos a la función con argumentos "ausentes"  
hipotenusa()  
[1] 5
```



# Ejercicios

Utilizar las siguientes funciones y analizar sus respuestas

```
print("Función para mostrar en pantalla objetos de R")
```

```
x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
```

```
cat("Función para mostrar en pantalla una serie de  
objetos concatenados", x)
```

```
sum(x)
```

```
which.max(x)
```

```
which.min(x)
```

```
mean(x)
```

```
median(x)
```

```
prod(x)
```

```
sort(x)
```

```
rnorm(n=10, mean=0, sd=1)
```

```
runif(n=10, min=0, max=1)
```

```
rbinom(n=10, size=5, prob=0.5)
```



# Ejemplo de función sample

```
function (x, size, replace = FALSE, prob = NULL)
{
  if (length(x) == 1L && is.numeric(x) && x >= 1) {
    if (missing(size))
      size <- x
    sample.int(x, size, replace, prob)
  }
  else {
    if (missing(size))
      size <- length(x)
    x[sample.int(length(x), size, replace, prob)]
  }
}
```





# Ayuda en línea (Help)

La primera línea indica la función u operador elegido y entre llaves el paquete que lo contiene y la segunda línea nos proporciona el título.

**Description:** Una breve descripción de su función.

**Usage:** Para una función proporciona el nombre de la misma con todos sus argumentos y los posibles valores por defecto (opciones) mientras que para un operador describe su uso típico.

**Arguments:** Solo aplicable para funciones. Aquí se describe en detalle cada uno de sus argumentos.

**Details:** Una descripción más detallada sobre cómo utilizar la función u operador y que obtendría como respuesta.

**Value:** Si aplica el tipo de objeto retornado por la función o el operador puesto que no siempre debe devolver algo.

**Reference:** Papers que dieron origen a estas funciones u operadores.

**See Also:** Otras páginas de ayuda con funciones u operadores similares.

**Examples:** Ejemplos de aplicación de la funciones u operadores.



## Ejercicio

Importar los datos desde el archivo `data_example.csv` y aplicar una de las funciones antes vista, `mediana`, `media`, `suma`, ..., a cada columna de los datos leídos. Puede usar también la función `apply`.

*`apply(X, MARGIN, FUN, ...)`*

Ejemplo: `mediana <- apply(datos, 2, median)`



# Lectura de datos

**Scan:** Lee vectores de datos que tienen el mismo modo. Si el primer argumento a escanear es un archivo se lee de ese archivo; en cambio si el primer argumento está vacío se lee los datos de la terminal finalizando cuando se lee una línea en blanco (enter).

```
scan(file = "", what = double(), nmax = -1, n = -1, sep = "", quote =  
if(identical(sep, "\n")) "" else "\"", dec = ".", skip = 0, nlines = 0,  
na.strings = "NA", flush = FALSE, fill = FALSE, strip.white = FALSE,  
quiet = FALSE, blank.lines.skip = TRUE, multi.line = TRUE,  
comment.char = "", allowEscapes = FALSE, fileEncoding = "", encoding  
= "unknown", text, skipNul = FALSE)
```

```
matrix.numeric <- matrix(scan(file="", what = numeric()),ncol=4)
```



# Lectura de datos

**Read.table:** Lee datos desde un archivo y crea un marco de datos ('data frame') para acceder a ellos.

```
read.table( file, header = FALSE, sep = "", quote = "\"\"", dec = ".",  
numerals = c("allow.loss", "warn.loss", "no.loss"), row.names,  
col.names, as.is = !stringsAsFactors, na.strings = "NA", colClasses =  
NA, nrows = -1, skip = 0, check.names = TRUE, fill = !blank.lines.skip,  
strip.white = FALSE, blank.lines.skip = TRUE, comment.char = "#",  
allowEscapes = FALSE, flush = FALSE, stringsAsFactors =  
default.stringsAsFactors(), fileEncoding = "", encoding = "unknown",  
text, skipNul = FALSE)
```

```
data<-read.table("sequences.fasta", sep="\n", dec=".")
```

## Ejercicio

Descargar desde Uniprot 10 secuencias de proteínas y utilizar la función `read.table`.





# Escritura de datos

**Write:** guarda el contenido de un objeto en un archivo. Este objeto puede ser cualquier tipo de datos.

```
write(data, file="sequences.fasta", sep="\n")
```

**Write.table:** guarda el contenido de un objeto en un archivo. Este objeto es típicamente un data.frame pero puede ser cualquier otro tipo de objeto.

