

# R\_BÁSICO

## 1. Descripción, historia y utilidades.

## 2. Antes de empezar

2.1. Obtener e instalar R (bajo Windows)

2.2. ¿Cómo funciona R?.

2.3. Abrir y cerrar R

2.4. Operador asignar (<- ó ->)

2.5. Definiciones

2.6. Workspace

2.7. Ayuda en línea

## 3. Manipulaciones simples: Números y Vectores

3.1. Números

3.2. Vectores numéricos

3.3. Aritmética vectorial

3.4. Vector de caracteres

3.5. Vector como factor

3.6. Expresiones lógicas

3.7. Indexar vectores

3.8. Generación de sucesiones

3.9. Atributos de objetos

3.10. Missing values

## 4. Tipos de objetos

4.1. Arrays/Matriz

4.2. Listas

4.3. Hojas de datos (Data frames)

## **5. Importar y Exportar datos**

- 5.1. Importar datos externos
- 5.2. Datos que se cargan con los packages
- 5.3. Exportar datos
- 5.4. Salvar una sesión

## **6. Aumentar las capacidades de R: Instalar y cargar “Add-on packages”**

- 6.1. Standard (Base) packages
- 6.2. Contributed Packages

# R\_BÁSICO

## 1. Descripción, historia y utilidades

La definición más simple de R es: “un lenguaje y entorno para el cálculo estadístico y creación de gráficos”. Es una combinación de un paquete estadístico y un lenguaje de programación con las siguientes características que lo diferencian de otros paquetes estadísticos:

1. R se distribuye gratuitamente bajo licencia GNU General Public Licence: libertad para usar, copiar y distribuir.
2. Hay versiones de R para diferentes plataformas o sistemas operativos: Windows, MacOS, Linux, y Unix.
3. R tiene una gran cantidad de funciones estadísticas escritas y un gran variedad de “add-on packages” (paquetes), que añaden más funciones, como es el caso del FLR. La mayoría de análisis estadísticos convencionales pueden hacerse en R.
4. R trabaja mediante comandos, aunque últimamente se han desarrollado algunas interfaces gráficas (GUI), esto hace que su uso sea más difícil pero que el entorno sea más flexible.



El proyecto R fue iniciado en 1995 por Robert Gentleman y Ross Ihaka (de sus nombres deriva el nombre de “R”) del Statistics Department de la University of Auckland. El lenguaje de programación R es la implementación libre del lenguaje de programación S, que usa el S-PLUS. Aunque existen pequeñas diferencias entre R y el lenguaje S, la mayoría dentro de la interface de gráficos, ambos son esencialmente idénticos.

El software ha ganado un gran número de audiencia entre usuarios y desarrolladores. Actualmente es sostenido por el Grupo Nuclear de Desarrollo de R (R Development Core Team) compuesto por un grupo internacional de desarrolladores voluntarios. La página web del proyecto R es: <http://www.r-project.org>

Este es el principal sitio de información sobre R: software, documentación, FAQs, R wiki, etc... . Los archivos necesarios para las distintas instalaciones pueden descargarse de CRAN, ‘Comprehensive R Archive Network’, junto con las oportunas instrucciones e información para la instalación: <http://cran.us.r-project.org/>. Periódicamente nuevas versiones son colgadas de esta página web.

R posee muchas funciones para análisis estadísticos y creación de gráficos; estos últimos pueden ser visualizados de manera inmediata en su propia ventana y ser guardados en varios formatos (jpg, png, bmp, ps, pdf). Los resultados de análisis estadísticos se muestran en la pantalla, y algunos resultados

intermedios (como valores P-, coeficientes de regresión, residuales, . . . ) se pueden guardar, exportar a un archivo, o ser utilizados en análisis posteriores.

## 2. Antes de empezar

### 2.1. Obtener e instalar R (bajo Windows)

La última versión de R es: 2.4.1 que puede descargarse de la web CRAN (Comprehensive R Archive Network): <http://cran.us.r-project.org/>.

Hasta Noviembre del 2006, el package FLR sólo era compatible con la versión R 2.3.1, por lo que descargaremos esta versión:

- El directorio 'bin/Windows/base/old/R2.3.1' del site CRAN contiene la distribución de R y un amplio número de add-on packages para trabajar bajo Windows 95, 98, NT4, 2000, ME, XP y 2003.
- La instalación es mediante el ejecutable 'R-2.3.1.exe'.
- Doble clic en el icono y seguir instrucciones.
- Se puede desinstalar R desde el Panel de Control como cualquier otro programa.

**\*\* Es importante tener en cuenta que R va a depender de la configuración de tu ordenador (separadores de listas, símbolo de decimales, etc...).**

Menú y workspace:

El cursor, que por defecto es el símbolo '>', indica que R está listo para recibir un comando. En Windows, algunos comandos pueden ser ejecutados a través de los menús interactivos (por ejemplo: buscar ayuda en línea, abrir archivos, . . . ).

### 2.2. ¿Cómo funciona R?.

↓	R es un lenguaje de programación dirigido a objeto.	↑
	R es un intérprete no un compilador	
	R permite la ejecución de comandos en línea	
	R es un paquete estadístico.	

- R es un lenguaje *Orientado a Objetos*: bajo este término se esconde la simplicidad y flexibilidad de R.

*Orientado a Objetos* significa que las variables, datos, funciones, resultados, etc., 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).

Mientras que programas más clásicos muestran directamente los resultados de un análisis, R guarda estos resultados como un “objeto”, de tal manera que se puede hacer un análisis sin necesidad de mostrar su resultado inmediatamente. Esto puede ser un poco extraño para el usuario, pero esta característica suele ser muy útil.

Otras características de los lenguajes orientados a objetos son la herencia: las subclases heredan las características de las superclases, y el polimorfismo la misma operación aplicada a diferentes objetos resulta en diferentes implementaciones.

- R es un lenguaje interpretado (como Java) y no compilado (como Fortran, Pascal, . . . ), lo cual significa que los comandos escritos en el teclado son ejecutados directamente sin necesidad de construir ejecutables.

### 2.3. Abrir y cerrar R

Una vez instalado el software en el ordenador, para **abrir R** simplemente hacer doble click en el icono. Se abre una consola de línea de comandos, la “Gui” (graphical user interface), con un mensaje de inicio:

```
R version 2.4.1 (2006-12-18)
Copyright (C) 2006 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R es un software libre y viene sin GARANTIA ALGUNA.
Usted puede redistribuirlo bajo ciertas circunstancias.
Escriba 'license()' o 'licence()' para detalles de distribución.

R es un proyecto colaborativo con muchos contribuyentes.
Escriba 'contributors()' para obtener mas informacion y
'citation()' para saber como citar R o paquetes de R en publicaciones.

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.
```

```
[Previously saved workspace restored]
```

```
>
```

El símbolo que aparece abajo (>) se denomina prompt. Después del > prompt, se indica a R lo que se quiere hacer mediante comandos, R hace el trabajo y da la respuesta. Si el comando es demasiado largo para ajustarse a una línea se usa “+” para continuar.

Para **salir de R** se tecldea: `q()` o usa la opción del Menu: `File, Exit`.

## 2.4. Operador asignar (<- ó ->)

El operador “asignar” permite crear objetos en R. El nombre de un objeto debe:

- comenzar con una letra (A-Z y a-z).
- puede incluir letras, dígitos (0-9), y puntos (.).
- R discrimina entre letras mayúsculas y minúsculas para el nombre de un objeto, de tal manera que x y X se refiere a objetos diferentes.
- R no interpreta los espacios excepto para la asignación (bien: <- ,mal: < - ).

La asignación de un valor a un objeto es la forma de crearlo.

Un objeto puede ser creado con el operador “asignar” el cual se denota como una flecha con el signo menos y el símbolo “>” o “<” dependiendo de la dirección en que asigna el objeto:

Crear objeto de valor 15 y nombre n:

```
> n<- 15
> n
[1] 15
```

Crear objeto de valor 5 y nombre n:

```
> 5 -> n
> n
[1] 5
```

Crear objeto de valor 1 y nombre x:

```
> x<- 1
```

Crear objeto de valor 10 y nombre X:

```
> X<- 10
```

Visualizar los objetos x y X:

```
> x
```

```
[1] 1
> x
[1] 10
```

Si el objeto ya existe, su valor anterior es borrado después de la asignación (la modificación afecta solo objetos en memoria, no a los datos en el disco). El valor asignado de esta manera puede ser el resultado de una operación y/o de una función:

```
> n <- 10 + 2
> n
[1] 12
> n <- 3
> n
[1] 3
```

Se puede escribir una expresión sin asignar su valor a un objeto; en este caso el resultado será visible en la pantalla pero no será guardado en memoria:

```
> (10 + 2) * 5
[1] 60
```

## 2.5. Definiciones

- *Objeto* : En R casi todo es un objeto.
- *Clase*: un objeto que sus datos y sus funciones para manipular esos datos.
- *Función*: un objeto que contiene los argumentos formales y el cuerpo de la función, y que nos devuelve un valor cuando lo llamamos.
- *Método* : Una función genérica.

## 2.6. Workspace

Todos los objetos creados se guardan en el workspace o área de trabajo. Para ver qué objetos están en el workspace se usa la función `ls()`:

La función `ls()` simplemente lista los objetos en memoria, sólo se muestran los nombres de los mismos:

```
> name <- "Carmen"
> ls()
[1] "n" "x" "X" "name"
```

Si se quiere listar sólo aquellos objetos que contengan un carácter en particular, se puede usar la opción `pattern` (que se puede abreviar como `pat`):

```
> ls(pat = "m")
[1] "name"
```

Para borrar objetos en memoria, utilizamos la función `rm()` o `remove()`:

```
> rm(x)
```

elimina el objeto `x`

```
> rm(x, X)
```

elimina ambos objetos `x` y `X`, `y`

```
> rm(list=ls())
```

elimina todos los objetos en memoria.

Al salir de la sesión de R, el software pregunta si quieres salvar la imagen de tu área de trabajo. Si se contesta que sí, todos los objetos (los nuevos creados en la sesión actual y los objetos de sesiones previas) estarán disponibles en la siguiente sesión.

## 2.7. Ayuda en línea

Para obtener ayuda acerca de un comando se emplea la función `help(function)` o el carácter “?” antes del comando. Por ejemplo, para saber más de la función `log`, los siguientes comandos nos darían la misma información:

```
> help(log)
> ?log                                # exclusiva de Windows
```

En Windows se abre una ventana de ayuda que siempre consta, al menos, de las siguientes partes:

**Description:** descripción breve.

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

**Arguments:** para una función, describe en detalle cada uno de sus argumentos.

**Details:** descripción detallada.

**Value:** si se aplica, el tipo de objeto retornado por la función o el operador.

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

**Examples:** algunos ejemplos que generalmente pueden ser ejecutados sin abrir la ayuda con la función `example()`.

Otras secciones que pueden estar presentes son **Note:** (notas adicionales), **References:** (bibliografía que puede ser útil) o **Author(s):** (nombre del autor o autores).

## Logarithms and Exponentials

### Description

`log` computes natural logarithms, `log10` computes common (i.e., base 10) logarithms, and `log2` computes binary (i.e., base 2) logarithms. The general form `logb(x, base)` computes logarithms with base `base`.

`log1p(x)` computes  $\log(1+x)$  accurately also for  $|x| \ll 1$  (and less accurately when  $x$  is approximately  $-1$ ).

`exp` computes the exponential function.

`expm1(x)` computes  $\exp(x) - 1$  accurately also for  $|x| \ll 1$ .

### Usage

```
log(x, base = 10)
logb(x, base = exp(1))
log10(x)
log2(x)
```

```
exp(x)
expm1(x)
```

```
log1p(x)
```

### Arguments

`x` a numeric or complex vector.

`base` positive number. The base with respect to which logarithms are computed. Defaults to  $e = \exp(1)$ .

### Details

`exp` and `log` are generic functions: methods can be defined for them individually or via the `Math` group generic.

`log10` and `log2` are only special cases, but will be computed more efficiently and accurately where supported by the OS.

### Value

A vector of the same length as `x` containing the transformed values. `log(0)` gives `-Inf` (when available).

### Note

`log` and `logb` are the same thing in  $\mathbb{R}$ , but `logb` is preferred if `base` is specified, for S-PLUS compatibility.

## References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole. (for `log`, `\log10` and `exp`.)

Chambers, J. M. (1998) *Programming with Data. A Guide to the S Language*. Springer. (for `logb`.)

## See Also

[Trig](#), [sqrt](#), [Arithmetic](#).

## Examples

```
log(exp(3))
log10(1e7)# = 7
```

```
x <- 10^-(1+2*1:9)
cbind(x, log(1+x), log1p(x), exp(x)-1, expm1(x))
```

---

[Package *base* version 2.4.1 [Index](#)]

También se puede buscar ayuda mediante palabra clave usando la función `apropos()`. Un ejemplo sería para la función `log`:

```
> apropos(log)
```

Se obtienen las funciones en las que parece la palabra `log`.

La función `help.search()` permite buscar información que se ajusta a una palabra o cadena de caracteres en el nombre, título, concepto o palabras clave de la documentación.

```
> help.search("log")
```

La ayuda también se puede obtener del menú de la console R y la podemos obtener en modo texto o a través del buscador en la web. Que estén disponibles todas las opciones del menú depende de la configuración de nuestra instalación.

La función `example` escribe en pantalla los ejemplos de la documentación de la función entre paréntesis:

```
> example(log)
```

## 3. Manipulaciones simples: Números y Vectores

### 3.1. Números

En R se pueden hacer operaciones como:

```
> (10 + 2) * 5
```

```
[1] 20
```

A veces se quiere guardar el resultado en una variable (objeto). Esto se hace con el operador de asignación "<- "

```
> n<- (10 + 2) * 5  
[1] 60
```

Se pueden hacer cálculos con objetos, e.g:

```
> n^2  
[1] 3600  
> n + 10  
[1] 70  
> n <- n + n  
[1] 120
```

**\* Observación: si el objeto existe se sobrescribe el valor**

### 3.2. Vectores numéricos

R trabaja con estructuras de datos. La estructura más simple es el vector numérico, que consiste en una colección ordenada de números (1 ó más). Para construir un vector llamado x, compuesto de cinco números: 10.4, 5.6, 3.1, 6.4 y 21.7, se usa el comando c():

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

ó

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

Dentro de los vectores se pueden incluir objetos:

```
> y<- c(56,x ,0)
```

### 3.3. Aritmética vectorial

La mayoría de las operaciones matemáticas de R son vectorizadas, se aplican a cada elemento del vector:

Añadir un número a un vector:

```
> 5 + c(4, 7, 17)  
[1] 9 12 22
```

Multiplicar un vector por un número:

```
> 5 * c(4, 7, 17)  
[1] 20 35 85
```

Sumar dos vectores de la misma longitud:

```
> c(-1, 3, -17) + c(4, 7, 17)  
[1] 3 10 0
```

Aplicar una función a cada uno de los elementos del vector:

```
> c(2, 4, 5)^2  
[1] 4 16 25
```

Si dos vectores de diferente longitud se añaden, restan, el elemento más corto se repite para ajustar las longitudes:

```
> c(1, 2, 3) + c(2, 4, 8, 12, 14, 18, 22)  
[1] 3 6 11 13 16 21 23
```

### 3.4. Vector de caracteres

Un vector también puede ser una colección de caracteres:

```
> vect_car <- c("green", "blue sky", "-77")  
[1] "green" "blue sky" "-77"
```

**\* Observación: "-77" es un carácter, no es un número.**

### 3.5. Vector como factor

Un factor es un carácter especial. Su empleo es importante como niveles en modelos de regresión como variables discretas.

El vector cuenta con el atributo *levels*.

Pasar de un vector de caracteres a factor:

```
> a<- c("green", "blue", "green", "yellow")
[1] "green" "blue" "green" "yellow"

> factor(a)
[1] green blue green yellow
Levels: blue green yellow
```

Pasar un vector numérico en factor:

```
> b<- c(2, 1, 3, 1)
[1] 2 1 3 1

> b<- factor(b)
[1] 2 1 3 1
Levels: 1 2 3
```

### 3.6. Expresiones lógicas

Una expresión lógica es una expresión que es TRUE o FALSE, que en R se puede abreviar en T o F.

Determina si dos números son iguales (==) o distintos (!=):

```
> 7 == 6
[1] FALSE
> 7 != 6
[1] TRUE
```

Comparación de números:

```
> 7 > 6
[1] TRUE
> 6 <= 7
[1] TRUE
```

Las operaciones lógicas se pueden combinar con OR(|) y AND(&). Por ejemplo:

¿es (7 == 9) o (7 > 0)?

```
> (7 == 9) | (7 > 0)
[1] TRUE
```

¿es 7 == 9 y 7 > 0?

```
> (7 == 9) & (7 > 0)
[1] FALSE
```

Las operaciones lógicas son también vectorizadas

```
> c(13, 4, 9, -7, 18) > 7
[1] TRUE FALSE TRUE FALSE TRUE
> c(T, F, T) == F
[1] FALSE TRUE FALSE
```

### 3.7. Indexar vectores

Para seleccionar elementos de un vector se usan "[ ]":

Seleccionar los elementos 3, 4 y 1 de un vector:

```
> a<- c(13, 4, 9, -7, 18)
> a[c(3, 4, 1)]
[1] 9 -7 13
```

Seleccionar todos los elementos excepto el elemento 2 y el 5:

```
> a[- c(2, 5)]
[1] 13 9 -7
```

### 3.8. Generación de sucesiones

Secuencias regulares:

- Todos los números entre dos números:

```
> v <- 4:8
[1] 4 5 6 7 8
```

- Una secuencia entre dos números a intervalos regulares: números entre 4.7 y 6.1 en steps de 0.4

```
> v<- seq(from = 4.7, to = 6.1, by = 0.4)
[1] 4.7 5.1 5.5 5.9
```

- Repetir la secuencia 1,2,3 cuatro veces:

```
> rep(1:3, times = 4)
[1] 1 2 3 1 2 3 1 2 3 1 2 3
```

### 3.9. Atributos de objetos

R trabaja con objetos los cuales tienen nombre y contenido, pero también *atributos* que especifican el tipo de datos representados por el objeto. Para entender la utilidad de estos atributos, consideremos una variable que toma los valores 1, 2, o 3: tal variable podría ser un número entero (por ejemplo, el número de huevos en un nido), o el código de una variable categórica (por ejemplo, el sexo de los individuos en una población de crustáceos: macho, hembra, o hermafrodita).

Con R, los atributos del objeto proporcionan la información necesaria. En general, y hablando un poco más técnicamente, la acción de una función sobre un objeto depende de los atributos de este último.

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

- El **tipo** se refiere a la clase básica de los elementos en el objeto; existen cuatro tipos principales: numérico, carácter, complejo, y lógico (FALSE [Falso] o TRUE [Verdadero]).
- La **longitud** es simplemente el número de elementos en el objeto.

Para ver el tipo y la longitud de un objeto se pueden usar las funciones `mode` y `length`, respectivamente:

```
> x<- 1
> mode(x)
[1] "numeric"
> length(x)
[1] 1

> A<- "Gomphotherium"; compar <- TRUE; z <- 1i
> mode(A); mode(compar); mode(z)
[1] "character"
[1] "logical"
[1] "complex"
```

### 3.10. Missing values

En ocasiones puede que no todas las componentes de un vector sean conocidas. Cuando falta un elemento, lo que se denomina 'valor faltante', se le asigna un valor especial, NA (*Not Available*). En general, casi cualquier operación donde intervenga un valor NA da por resultado NA.

La función `is.na(x)` crea un vector lógico del tamaño de `x` cuyos elementos sólo valdrían T si el elemento correspondiente de `x` es NA, y F en caso contrario.

```
> z<- c(1:3,NA)
> ind<- is.na(z)
```

Además hay una segunda clase de "missing values", producidos por el cálculo. Son los llamados valores NaN (*Not a Number*).

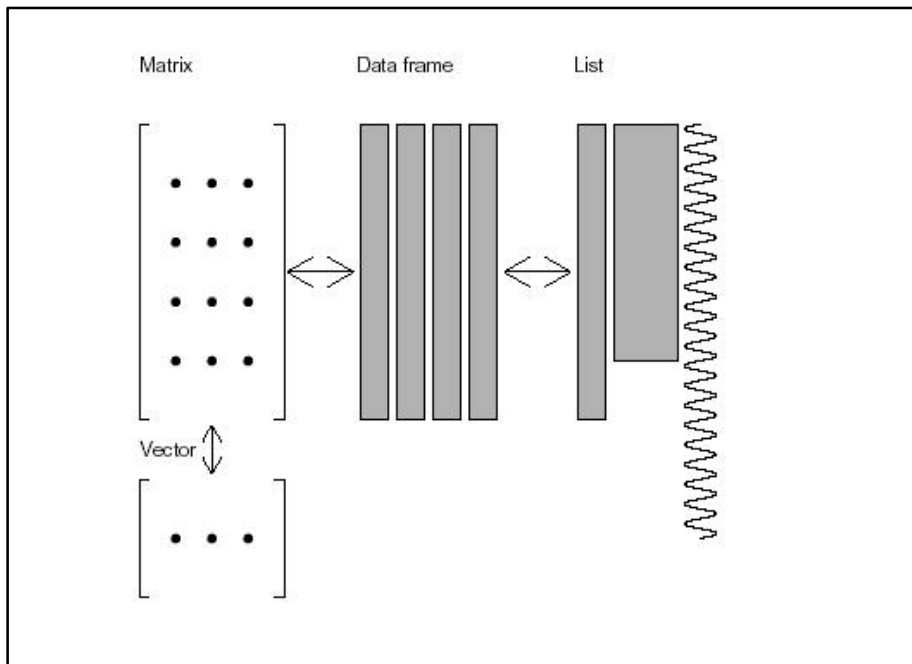
```
> 0/0
[1] NaN
> Inf - Inf
[1] NaN
```

## 4. Tipos de objetos

Casi todo en R es un objeto, los vectores son el tipo básico de objeto en R, pero existen más tipos, los más importantes son:

- Las **matrices** (*matrix*) o, más generalmente, variables indexadas (**arrays**) son generalizaciones multidimensionales de los vectores. De hecho, son vectores indexados por dos o más índices.
- Las **listas** (*list*) son una forma generalizada de vector en las cuales los elementos no tienen por qué ser del mismo tipo y a menudo son a su vez vectores o listas. Las listas permiten devolver los resultados de los cálculos estadísticos de un modo conveniente.
- Las **hojas de datos** (*data frames*) son estructuras similares a una matriz, en que cada columna puede ser de un tipo distinto a las otras. Las hojas de datos son apropiadas para describir 'matrices de datos' donde cada fila representa a un individuo y cada columna una variable, cuyas variables pueden ser numéricas o categóricas. Muchos experimentos se describen muy apropiadamente con hojas de datos: los tratamientos son categóricos pero la respuesta es numérica.
- **Factores** un tipo de vector para datos categóricos.
- Las **funciones** son también objetos de R que pueden almacenarse en el espacio de trabajo.

## Relación entre diferentes clases de estructuras de datos en R



#### 4.1. Arrays/Matrix

Un array es una colección de datos, por ejemplo numéricos, indexada por varios índices. Una matriz es una array de dos dimensiones. Las funciones `matrix()` y `array()`, que permiten asignaciones más sencillas

```

> x <- array(1:20,dim=c(4,5))           # Genera una variable indexada (col 5).
> x
  [,1] [,2] [,3] [,4] [,5]
[1,]  1  5  9 13 17
[2,]  2  6 10 14 18
[3,]  3  7 11 15 19
[4,]  4  8 12 16 20

> m<- matrix(1:20,ncol=2)
> m
  [,1] [,2]
[1,]  1 11
[2,]  2 12
[3,]  3 13
[4,]  4 14
[5,]  5 15
[6,]  6 16
[7,]  7 17
[8,]  8 18
[9,]  9 19
[10,] 10 20

> m<-1:20
> m
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

```

```

> p<-matrix(m,ncol=5)
> p
      [,1] [,2] [,3] [,4] [,5]
[1,]  1   5   9  13  17
[2,]  2   6  10  14  18
[3,]  3   7  11  15  19
[4,]  4   8  12  16  20

> is.matrix(p)
[1] TRUE

```

## 4.2. Listas

En R, una lista es un objeto consistente en una colección ordenada de objetos, conocidos como componentes. No es necesario que los componentes sean del mismo modo, así una lista puede estar compuesta de, por ejemplo, un vector numérico, un valor lógico, una matriz y una función.

La función `list()` permite crear listas a partir de objetos ya existentes.

Ejemplo de una lista:

```
> Lst<- list(nombre="Pedro", esposa="María",edad.hijos=c(4,7,9))
```

Los componentes siempre están numerados y pueden ser referidos por dicho número.

Para acceder a los componentes se usa `[[ ]]` :

```

> Lst[[1]]
[1] "Pedro"
> Lst[[2]]
[1] "María"
> Lst[[3]]
[1] 4 7 9

```

O, si le hemos dado nombre a los elementos :

```

> Lst$nombre
[1] "Pedro"
> Lst$esposa
[1] "María"
> Lst$edad.hijos
[1] 4 7 9

```

Para acceder a un subelemento, como por ejemplo la edad 7:

```

> Lst[[3]][2]
[1] 7

```

## 4.3. Hojas de datos (Data frames)

Una hoja de datos es una lista que pertenece a la clase "data.frame". Los componentes deben ser vectores (numéricos, cadenas de caracteres, o lógicos), matrices numéricas, listas u otras hojas de datos.

A diferencia de las listas, los vectores que constituyen la hoja de datos deben tener todos la misma longitud, y las matrices deben tener el mismo tamaño.

La manera más sencilla de construir una hoja de datos es utilizar la función `read.table()` (o `read.csv()`) para leerla desde un archivo del sistema operativo.

```
>a<- list(fish="toto",age=1:3, good.shape=FALSE)
>a
```

Extraer datos de un data.frame:

```
>b$age
>b[,2]
>b[2:3,]
```

\* Las clases S4 usadas en FLR :

- Son colecciones de objetos organizadas en slots, a los que se accede con el símbolo @: X@a, X@b..
- Los Slots pueden ser de diferentes clases: numeric, character, array etc

## 5. Importar y Exportar datos

### 5.1. Importar datos externos

R utiliza el directorio de trabajo para leer y escribir archivos. Para saber cual es este directorio se puede utilizar el comando:

```
> getwd() # (get working directory)
```

Para cambiar el directorio de trabajo, se utiliza la función `setwd()`:

```
> setwd("C:/Archivos de programa/R/R-2.4.1/Data")
```

Es necesario proporcionar la dirección ('path') completa del archivo si este no se encuentra en el directorio de trabajo.

Archivos txt: `read.table()`

Por lo general, los datos los tenemos en forma de tablas en las que encontramos una serie de variables de diferentes tipos dispuestas en columnas; y una serie de registros que aparecen ocupando las filas de la tabla.

Sea cual sea la aplicación en la que tengamos la información, siempre podremos exportar la tabla en forma de fichero de texto. Esta es una de las formas más fáciles de trasladar datos desde cualquier aplicación hacia R. Tal vez lo más frecuente sea exportar las tablas a ficheros de texto en los que cada campo (columna, variable, como queramos llamarle) quede limitado por tabuladores (*tab delimited* en inglés). Si tenemos una tabla como la siguiente:

Longitud	Sexo	Peso
23	M	300
34	F	456
56	F	

y la exportamos (desde la aplicación en la que la hemos creado) con el nombre *datos.txt* al directorio de trabajo de R, podremos incorporarla al espacio de trabajo con la siguiente orden

```
> datos<- read.table(file="datos.txt", header=T, sep="\t")
```

en ella se dice que el fichero (que debe ir entrecomillado) tiene una primera fila en la que van los nombres de las variables (**header=T**) y que el separador de columnas es el tabulador (**sep="\t"**).

Archivos csv: `read.csv()`

Para archivos separados con comas, como los que se puede generar desde EXCEL, se usa la función `read.csv()`:

```
> datos<- read.csv(file="datos.csv", header=T)
```

En ambos casos el objeto `datos` será un `data.frame`, la hoja de datos de R

**\*Observación:** `read.csv` usa el punto como indicador de decimal y la coma como separador. La función `read.csv2` se acomoda a la convención de países como España, en los que la coma indica los decimales.

Continuamente se crean nuevas formas de importación de datos a R, para archivos EXCEL:

Desde la hoja de datos de EXCEL se selecciona el área deseada se copia en el clipboard usando `ctrl+c` y se teclea en R:

```
> datos<- read.delim("clipboard")
```

Otras formatos que puede leer el R: EpilInfo, Minitab, S-PLUS, SAS, SPSS, Stata, Systat.

## 5.2. Datos que se cargan con los packages

Para ver los sets de datos disponibles en R se teclea:

```
> data()
```

Son accesibles los sets de datos de los packages que están cargados en ese momento.

## 5.3. Exportar datos

La función `write.table` guarda el contenido de un objeto en un archivo. El objeto es típicamente un marco de datos ('`data.frame`'), pero puede ser cualquier otro tipo de objeto (vector, matriz, . . .).

Los argumentos y opciones son:

`write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ", eol = "\n", na = "NA", dec = ".", row.names = TRUE, col.names = TRUE, qmethod = c("escape", "double"))`

<code>x</code>	el nombre del objeto a exportar
<code>file</code>	el nombre del archivo (por defecto, el objeto se muestra en la pantalla)
<code>append</code>	si es TRUE anexa los datos al archivo sin borrar datos ya existentes en el mismo
<code>quote</code>	lógico o numérico : si es TRUE variables de tipo caracter y factores se escriben entre ' ; si es un vector numérico, este indica el número de las variables a ser mostradas entre (en ambos casos los nombres de las variables se escriben entre pero no si <code>quote = FALSE</code> )
<code>sep</code>	el separador de campo utilizado en el archivo
<code>eol</code>	el caracter que indica el final de línea (" \n " es 'retorno')
<code>na</code>	el caracter a usarse para datos faltantes
<code>dec</code>	el caracter usado para el punto decimal
<code>row.names</code>	una opción lógica que indica si los nombres de las líneas se escriben en el archivo
<code>col.names</code>	identificación para los nombres de las columnas
<code>qmethod</code>	si es <code>quote=TRUE</code> , especifica la manera como se debe tratar las comillas dobles "en variables tipo caracter: si es "escape"(o "e", por defecto) cada "es reemplazada por \; si es "dçada"es reemplazada por

Para guardar un grupo de objetos de cualquier tipo se puede usar el comando:

```
> save(x, y, z, file= "xyz.RData").
```

#### 5.4. Salvar una sesión

Qué	Acción	Función	Menú
Objetos y funciones	Salvar	save save ("mi_archivo.RData")	File Save Workspace
	Recuperar	load load (mi_archivo.RData)	File Load Workspace
Comandos de la sesión	Salvar	savehistory: savehistory ("mi_archivo.RHistory")	File Save History
	Recuperar (se accede flecha movimiento)	loadhistory loadhistory ("mi_archivo.RHistory")	File Load History
	Recuperar (como texto en ventana aparte)		File, Display file
Objetos, funciones y comandos	Salvar	Ventana de confirmación salir de sesión: <b>Save workspace image? Yes</b> (archivos por defecto: ".RData" y "RHistory" en dir de trabajo)	
	por Recuperar	Se recupera automáticamente al abrir una nueva sesión	

## 6. Aumentar las capacidades de R: Instalar y cargar "Add-on packages"

Las funciones y los sets de datos están almacenados en *packages* (paquetes). El contenido de los paquetes está disponible sólo cuando el paquetes es cargado en la sesión de R.

### 6.1. Standard (Base) packages

Los paquetes base se consideran parte del código de R y contienen las funciones básicas que permiten trabajar a R y los datos y funciones gráficas utilizados en los tutoriales.

Estos paquetes están disponibles automáticamente con cualquier instalación de R.

### 6.2. Contributed Packages

Hay una gran cantidad de paquetes para R escritos por diferentes autores y con diferentes fines (unos implementan métodos estadísticos específicos, dan acceso a datos o hardware) que es posible obtener.

#### Instalar paquetes

Para ver qué paquetes están instalados en tu ordenador se usa el comando:

```
> library()
```

sin argumentos.

A través de internet el usuario puede instalar y actualizar paquetes. En Windows existen diferentes maneras de instalar paquetes, la más sencilla es a través del menú:

### ***Packages***

#### ***Install package(s)***

***CRAN Mirror*** (una vez por sesión)

*escoger el mirror más cercano: **Spain***

#### ***Packages***

*escoger paquete y presionar **ok***

### **Cargar Paquetes**

Para ver qué paquetes están actualmente cargados se usa:

```
> search()
```

Una vez instalados los paquetes, para que estén disponibles hay que cargarlos. Para ello se usa el comando:

```
> library(boot)
```

o el comando

```
> require(boot)
```

o por el Menu:

### ***Packages***

#### ***Load package***

### **Borrar paquetes cargados:**

```
> detach("package:boot")
```

**Ayuda** sobre un **package** en concreto:

```
> library(help=boot)
```