

# Empezar a trabajar con R

Sitio: [Plataforma de Formación On line del Instituto Andaluz de  
Administración Pública](#)  
Curso: (I22F-PT05) Entorno de Programación R  
Libro: Empezar a trabajar con R

Imprimido por: ALFONSO LUIS MONTEJO RAEZ  
Día: viernes, 1 de abril de 2022, 08:30

# Tabla de contenidos

## **1. Arrancando R**

- 1.1. Barra de Menu
- 1.2. Barra de herramientas
- 1.3. Consola

## **2. Formas de trabajo con R**

- 2.1. Forma interactiva
- 2.2. Mediante scripts

## **3. Consideraciones al programar con R**

## **4. Creando objetos**

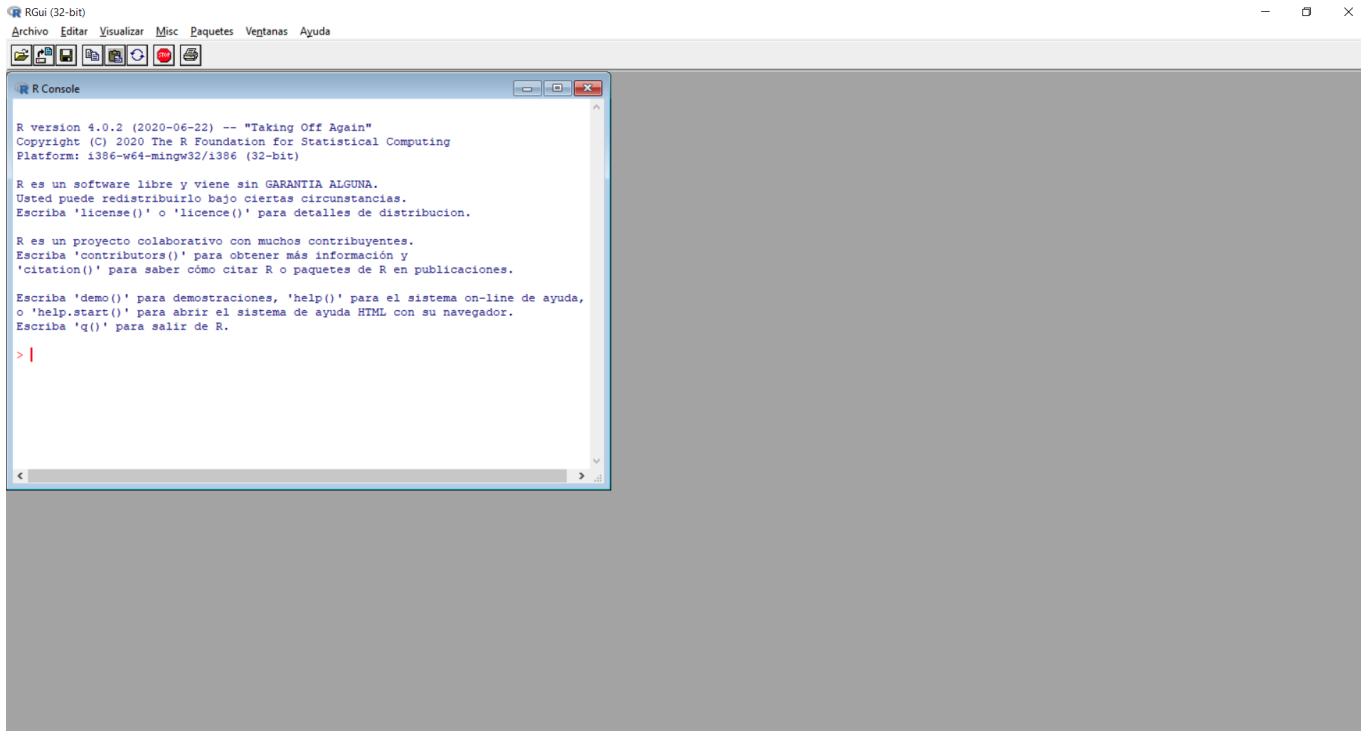
- 4.1. Trabajando con objetos
- 4.2. Como guardar objetos y recuperarlos

## **5. Ayuda en R**

## **6. Resumiendo**

# 1. Arrancando R

Ya hemos instalado el programa y lo hemos ejecutado para empezar a trabajar. Lo que nos encontramos es esto:



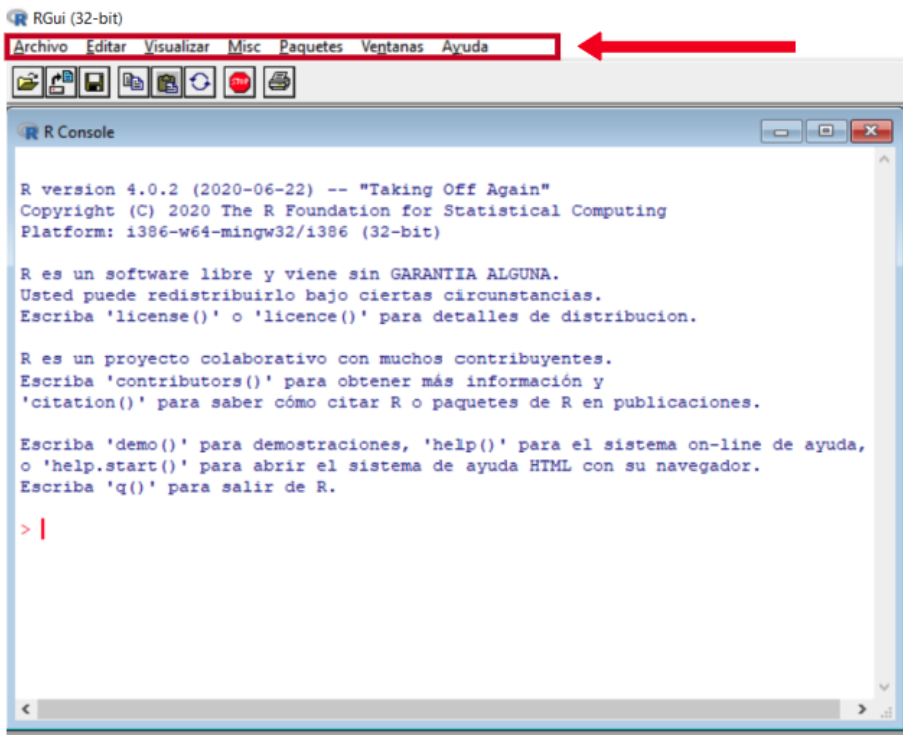
Una pantalla en blanco y una marca roja parpadeando esperando a que le demos órdenes.

**¡¡¡Y ahora que hacemos!!!**

Vamos a empezar describiendo las pantallas del programa antes de lanzarnos a programar.

## 1.1. Barra de Menu

Como en cualquier programa con un entorno gráfico, la **barra de menú** contiene todas las opciones con las que se puede trabajar en el programa ordenadas por tipo de funcionalidad.

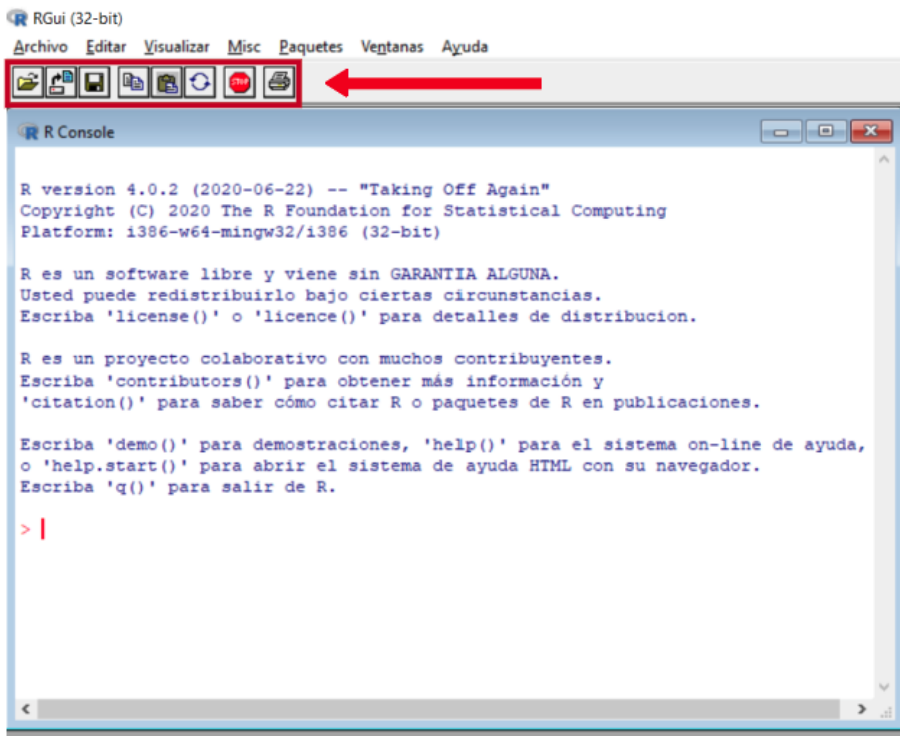


Las opciones más importantes son:

- **Archivo:** Donde podremos abrir y guardar archivos en los que escribir nuestros programas (llamados scripts) así como seleccionar el área de trabajo.
- **Paquetes:** Desde donde podremos gestionar todo lo referente a los paquetes o librerías que vamos a usar para ampliar las capacidades del programa.
- **Ayuda:** Menú donde podremos acceder a información disponible sobre funciones, manuales de uso, preguntas frecuentes, etc.

## 1.2. Barra de herramientas

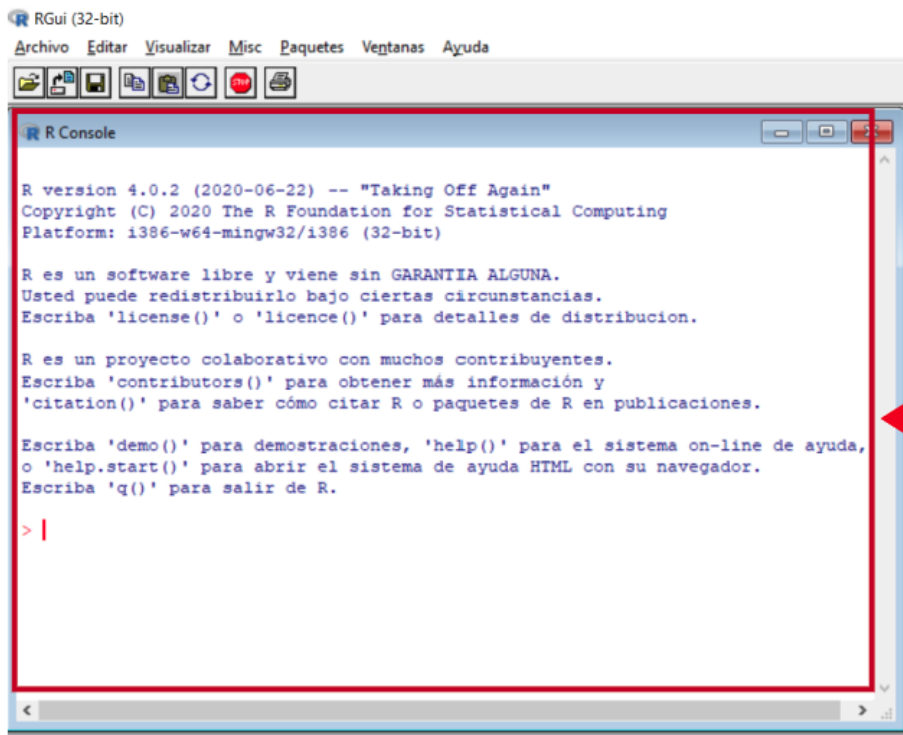
Otra cosa común a todos los programas con entornos gráficos es la **barra de herramientas**. En ella podemos encontrar accesos directos a las opciones más usadas del programa sin tener que estar buscando dentro de los menús.



Uno de los más importantes cuando empezemos a trabajar es el botón de **STOP**. Este paraliza la ejecución de un programa, por ejemplo en el caso de que se quede colgado o cuando estimemos que está tardando mucho en responder.

## 1.3. Consola

La **consola de trabajo** es la más importante ya que es donde vamos a ir escribiendo las órdenes al programa y donde irán saliendo los resultados de lo que vayamos pidiendo.



```
RGui (32-bit)
Archivo  Editar  Visualizar  Misc  Paquetes  Ventanas  Ayuda

R Console

R version 4.0.2 (2020-06-22) -- "Taking Off Again"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

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

R es un proyecto colaborativo con muchos contribuyentes.
Escriba 'contributors()' para obtener más información y
'citation()' para saber cómo 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.

> |
```

Las **órdenes** que le demos al programa se escribirán **en color rojo**, mientras que los **resultados** saldrán **en color azul**.

## 2. Formas de trabajo con R

Cuando trabajamos con R hay dos formas de proceder:

- **Interactiva**
- **Mediante scripts**

A continuación, vamos a explicar cada una de estas formas en detalle.

## 2.1. Forma interactiva

La primera forma de relacionarnos con R es escribir órdenes directamente en la consola.

Esta sería la **forma interactiva de trabajar**, ya que escribimos una orden y, pulsando ENTER, se ejecuta y nos devuelve el resultado también en la consola.

R es un lenguaje interpretado, es decir, sus órdenes se ejecutan si necesidad de compilar.

A screenshot of the R Console window. The title bar reads "R Console". The console shows the command `> sum(4, 5, 6, 7)` entered in red text. Below it, the output `[1] 22` is displayed in blue text. A red prompt character `>` and a vertical cursor are on the next line. The window has standard OS controls (minimize, maximize, close) in the top right corner and a scrollbar on the right side.

```
> sum(4, 5, 6, 7)
[1] 22
> |
```

**sum()** sería la orden para sumar números. Con el comando que he escrito, se suman los números del 4 al 7 y, al pulsar ENTER, me devuelve el resultado que es 22.



## 2.2. Mediante scripts

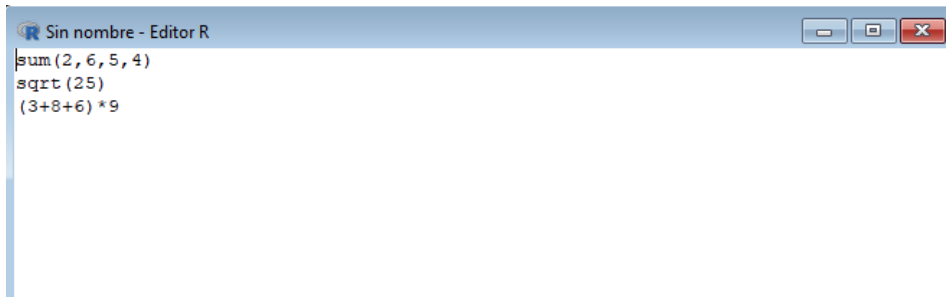
La segunda forma de trabajo sería mediante scripts.

Un **script** es un documento donde se van escribiendo órdenes pero sin ejecutarse. Se suele usar cuando aplicamos un proceso con varios pasos a un conjunto de datos.

Para usar un script en R hay que seguir la siguiente ruta:

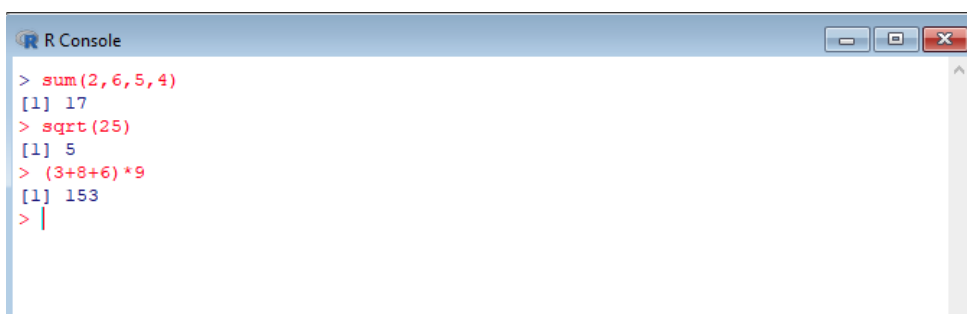
**Archivo → Nuevo script**

En este caso se abre un editor de R donde podemos escribir las órdenes que queramos.



```
Sin nombre - Editor R
sum(2, 6, 5, 4)
sqrt(25)
(3+8+6) * 9
```

Para que se ejecute la acción escrita en el script hay que pulsar **CTRL+R** sobre cada la línea y éstas se ejecutaran de manera secuencial devolviendo el resultado uno debajo de otro.



```
R Console
> sum(2, 6, 5, 4)
[1] 17
> sqrt(25)
[1] 5
> (3+8+6) * 9
[1] 153
> |
```

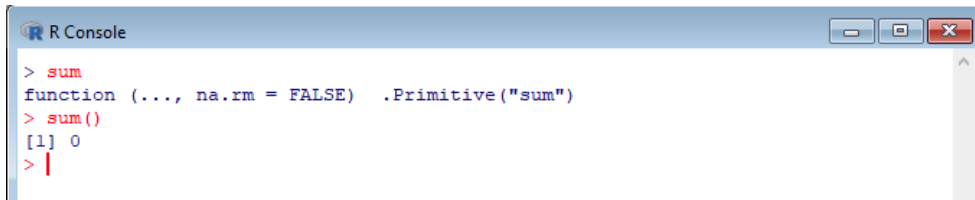
En este caso hemos usado la función **sum()** para sumar un conjunto de números, la función **sqrt()** que realiza la raíz cuadrada de un número y luego hemos usado los operadores básicos de suma (+) y multiplicación (\*).

### 3. Consideraciones al programar con R

Como todo lenguaje, R requiere que las órdenes se escriban correctamente y que la sintaxis sea la adecuada en cada caso.

Vamos a ofrecer una serie de cuestiones que hay que tener en cuenta a la hora de programar en R.

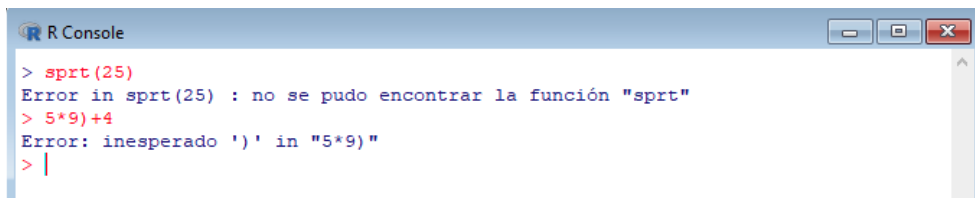
- **Cuando escribamos una función debe tener los argumentos entre paréntesis**



```
> sum
function (..., na.rm = FALSE) .Primitive("sum")
> sum()
[1] 0
> |
```

Si usamos una función sin el paréntesis lo que nos devuelve es el código de como está programada. Esto es útil cuando queremos ver que hace una función (ventajas del código abierto que hemos comentado antes).

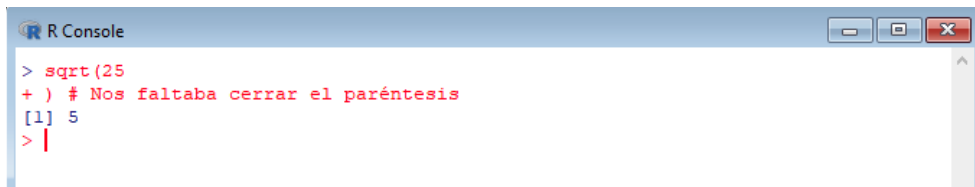
- **Hay que escribir bien las funciones**



```
> sprt(25)
Error in sprt(25) : no se pudo encontrar la función "sprt"
> 5*9)+4
Error: inesperado ')' in "5*9)"
> |
```

Si algo está mal escrito mostrará un mensaje de error y no se ejecutará la acción.

- **Hay que escribir la orden completamente**



```
> sqrt(25
+ ) # Nos faltaba cerrar el paréntesis
[1] 5
> |
```

Si en la expresión que se introduce falta algo, R no la evaluará y en la línea siguiente esperará que se complete. Lo indicará con la marca de continuación que, por defecto, es un signo +.

En el código se pueden añadir comentarios usando el carácter #. Todo lo que vaya después el programa no lo lee.

- **Se pueden unir y anidar funciones para su ejecución**

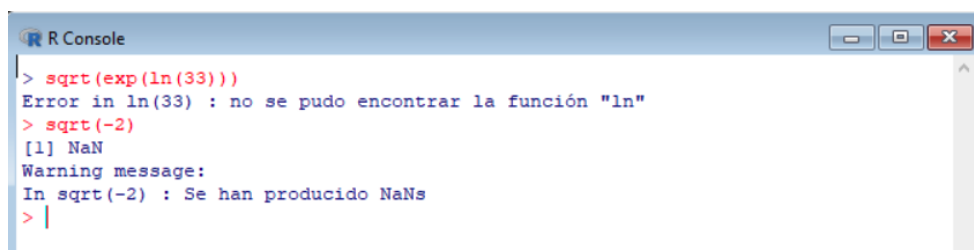


```
> sqrt(14.7); 5*(9+47); exp(25)
[1] 3.834058
[1] 280
[1] 72004899337
> sqrt(cos(log(2.4)))
[1] 0.800398
> |
```

Se pueden ejecutar varias acciones al mismo tiempo si las separamos con un punto y coma (;). Las acciones se ejecutan de manera secuencial empezando por la orden que está más a la izquierda.

También se pueden anidar varias funciones incluyendo unas dentro de otras. En este caso se ejecutan desde la que está más en el interior hacia fuera.

- Distinguir entre errores y avisos



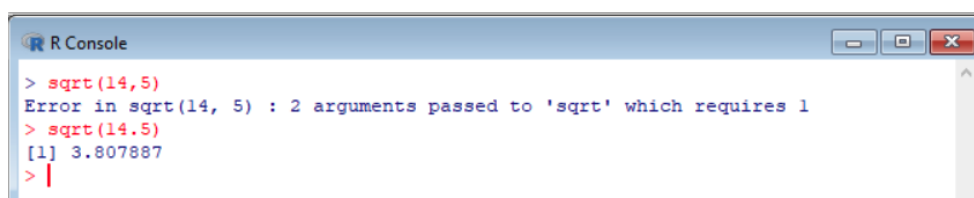
```
R Console
> sqrt(exp(ln(33)))
Error in ln(33) : no se pudo encontrar la función "ln"
> sqrt(-2)
[1] NaN
Warning message:
In sqrt(-2) : Se han producido NaNs
> |
```

Cuando programamos en R es muy frecuente que aparezcan mensajes de error y avisos. Hay que aprender a reconocerlos porque tienen diferentes repercusiones.

Cuando se produce un **error** (primera línea de la imagen) se detiene la ejecución de la orden y nos dice qué ha pasado (en este caso nos dice que la función `ln()` no existe).

Cuando se produce un **aviso** (tercera línea de la imagen) la orden se ejecuta y nos devuelve un mensaje avisando de que el resultado es NaN (Not a Number o resultado no numérico).

- El separador decimal es el punto (.) no la coma



```
R Console
> sqrt(14,5)
Error in sqrt(14, 5) : 2 arguments passed to 'sqrt' which requires 1
> sqrt(14.5)
[1] 3.807887
> |
```

Cuando queremos escribir un número decimal (como por ejemplo 14,5) tenemos que acordarnos de poner 14.5 En R la coma (,) simplemente es una separación entre cosas (argumentos, listas, etc.)

## 4. Creando objetos

Otras de las características de **R** es que **suporta el trabajo con objetos**.

En R podemos crear objetos que van a ser cajas donde podemos almacenar muchos tipos de cosas (datos, resultados de análisis, gráficos, mapas, etc.)

La creación de los objetos es muy sencilla; simplemente tenemos que ponerle un nombre al objeto y asignarle lo que queramos almacenar con  
`<-` `=` o `->` o `con =`



```
R Console
> Z<-36
> 'prueba' -> z
> b=18
> ls()
[1] "b" "z" "Z"
> |
```

En este ejemplo hemos creado tres objetos. El objeto **Z** que contiene el número 36, el **z** que contiene la cadena de texto prueba y el objeto **b** que contiene el número 18.


Varias cosas a tener en cuenta.

- Se pueden usar indistintamente los operadores de asignación `<-` `=` `->` o `con =`.
- **R es sensible a mayúsculas y minúsculas**, por lo que **Z** y **z** son objetos distintos.
- Cuando usemos **cadenas de texto hay que entrecomillarlas**. Podemos usar tanto las dobles comillas como las comillas simples (" o ' ).

Cuando trabajamos con objetos, la orden **ls()** nos permite obtener un listado de todos los objetos que hemos creado.

## 4.1. Trabajando con objetos

Una vez que tenemos los objetos creados podemos trabajar con ellos y usarlos en operaciones, en funciones, etc. Simplemente tenemos que usar el nombre del objeto y R operará su contenido.



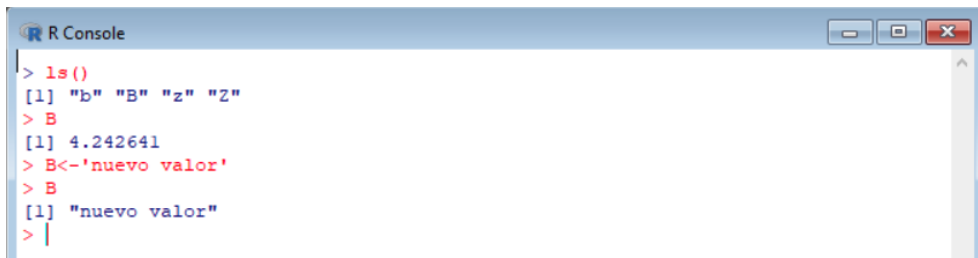
```
R Console
> log(Z)
[1] 3.583519
> mean(c(Z,b))
[1] 27
> print(z)
[1] "prueba"
> sqrt(Z-b)->B
> B
[1] 4.242641
> ls()
[1] "b" "B" "z" "Z"
> |
```

Aquí calculamos el logaritmo del objeto Z (recordar que tenía almacenado el número 36), calculamos la media entre Z y b con la orden **mean()** y por último escribimos el valor de z con la orden **print()**. En el caso de la función **mean()** los objetos que ponemos dentro para hacer la media los debemos concatenar con la orden **c()** ya que, si ponemos solo los nombres, nos dá error.

También podemos almacenar resultados de otras operaciones (creamos el objeto B donde almacenamos la raíz cuadrada de Z-b (que eran 36 y 18)).

### Aviso importante: Cuidado con el nombre de los objetos

Hay que tener en cuenta una cosa importante cuando creamos objetos. Si usamos el nombre de un objeto que ya existe lo estamos machacando y el programa no avisa de eso. Ojo con esto porque podemos perder información y horas de trabajo.

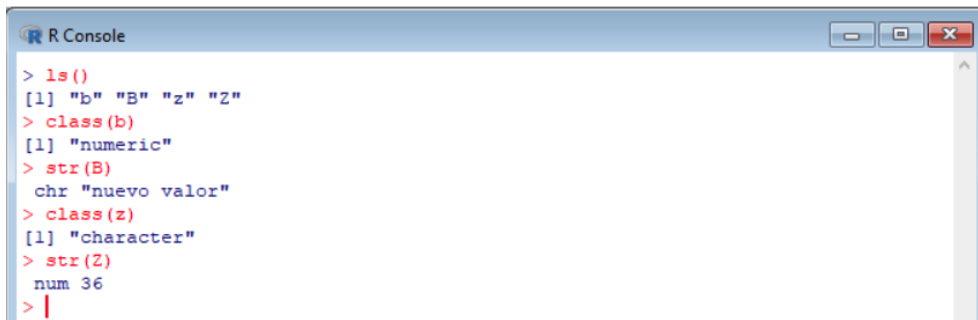


```
R Console
> ls()
[1] "b" "B" "z" "Z"
> B
[1] 4.242641
> B<-'nuevo valor'
> B
[1] "nuevo valor"
> |
```

Por ejemplo, tenemos un objeto que se llama B que contiene el valor 4,242641 y creamos otro objeto B donde almacenamos el texto 'nuevo valor'. Lo que estamos haciendo es machacar el objeto B y cambiando lo que tenía dentro.

Para acceder al contenido de un objeto simplemente tenemos que poner su nombre y pulsar ENTER y R nos devuelve el valor.

Hay dos órdenes muy útiles para trabajar con objetos: **class()** y **str()**.



```
R Console
> ls()
[1] "b" "B" "z" "Z"
> class(b)
[1] "numeric"
> str(B)
chr "nuevo valor"
> class(z)
[1] "character"
> str(Z)
num 36
> |
```

La orden **class()** nos ofrece una descripción básica del contenido del objeto. En nuestro caso, nos dice que el objeto b es de tipo numeric (almacena números) y el objeto z es de tipo character (almacena una cadena de texto).

La orden **str()** nos ofrece una descripción más detallada del objeto ofreciendo el tipo y los primeros valores que almacena el objeto. En nuestro caso, nos dice que el objeto B es de tipo chr (almacena una cadena de texto) y el valor que tiene es 'nuevo valor' y el objeto Z es de tipo num (almacena números) y el valor que tiene es el 36.

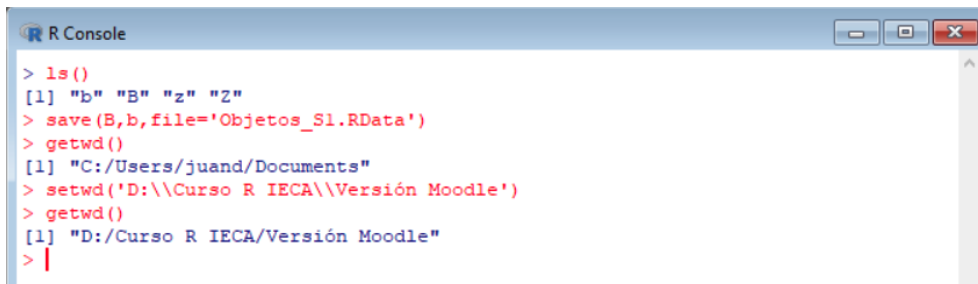


## 4.2. Como guardar objetos y recuperarlos

Los objetos creados en R se almacenan de manera virtual en el área de trabajo. Si se cierra la sesión o el programa sin guardar los objetos, estos se pierden.

### ¿Cómo se pueden guardar de forma permanente?

Los objetos se pueden guardar con la orden **save()** y recuperar posteriormente con la orden **load()**.



```
> ls()
[1] "b" "B" "z" "Z"
> save(B,b,file='Objetos_S1.RData')
> getwd()
[1] "C:/Users/juand/Documents"
> setwd('D:\\Curso R IECA\\Versión Moodle')
> getwd()
[1] "D:/Curso R IECA/Versión Moodle"
> |
```

Hemos creado los objetos b, B, z y Z y vamos a salvar algunos de ellos con la orden **save()**.

Simplemente le tenemos que indicar el nombre de los objetos que queremos salvar (serán los objetos B y b) y cómo se va a llamar el fichero donde se guardan (con el parámetro file). Entre comillas indicamos el nombre del fichero y la extensión que, para los datos en R, sería **.RData**.

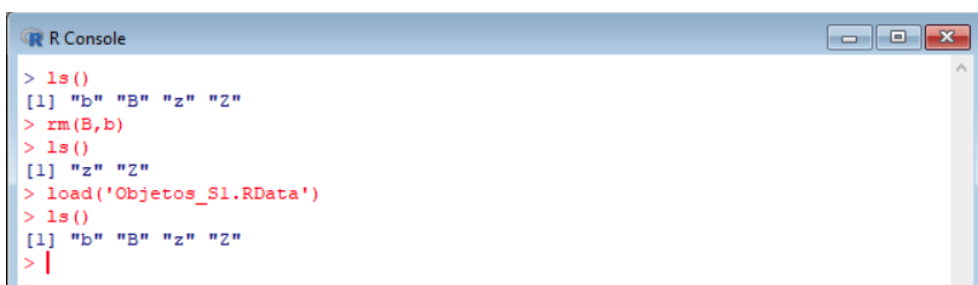
### Y ¿dónde se guardan los ficheros?

R trabaja en el **directorio de trabajo** por defecto asignado cuando empieza la sesión. Para conocer cuál es nuestro directorio de trabajo, usamos la orden **getwd()**. En nuestro caso, estamos trabajando en la carpeta de Documentos del usuario juand. Todo lo que guardemos se almacenará en ese sitio.

Si quisiéramos cambiar la carpeta donde se almacenan las cosas en R podemos usar la orden **setwd()** indicando, entre comillas, la ruta a donde queremos cambiar el directorio de trabajo.

Cuando cambiamos la ruta hay que tener cuidado porque tenemos que escribirla con dobles \ o con /, es decir, `setwd('D:\\Curso R IECA\\Versión Moodle')` o `setwd('D:/Curso R IECA/Versión Moodle')`

### Recuperación de datos



```
> ls()
[1] "b" "B" "z" "Z"
> rm(B,b)
> ls()
[1] "z" "Z"
> load('Objetos_S1.RData')
> ls()
[1] "b" "B" "z" "Z"
> |
```

Cuando hemos salvado los datos, luego podemos recuperarlos con la orden **load()**. Simplemente tenemos que indicar, entre comillas, el nombre del fichero que queremos recuperar.

En nuestro ejemplo, borramos los objetos b y B que ya teníamos guardados en el archivo `Objetos_S1.RData`. Para cargar de nuevo los datos, usamos la orden `load('Objetos_S1.RData')` y los objetos b y B vuelven a estar disponibles.

Hay que tener en cuenta que tenemos que respetar las mayúsculas y minúsculas en el nombre del fichero y que R va a ir a buscarlo en el directorio de trabajo que hemos comentado antes.

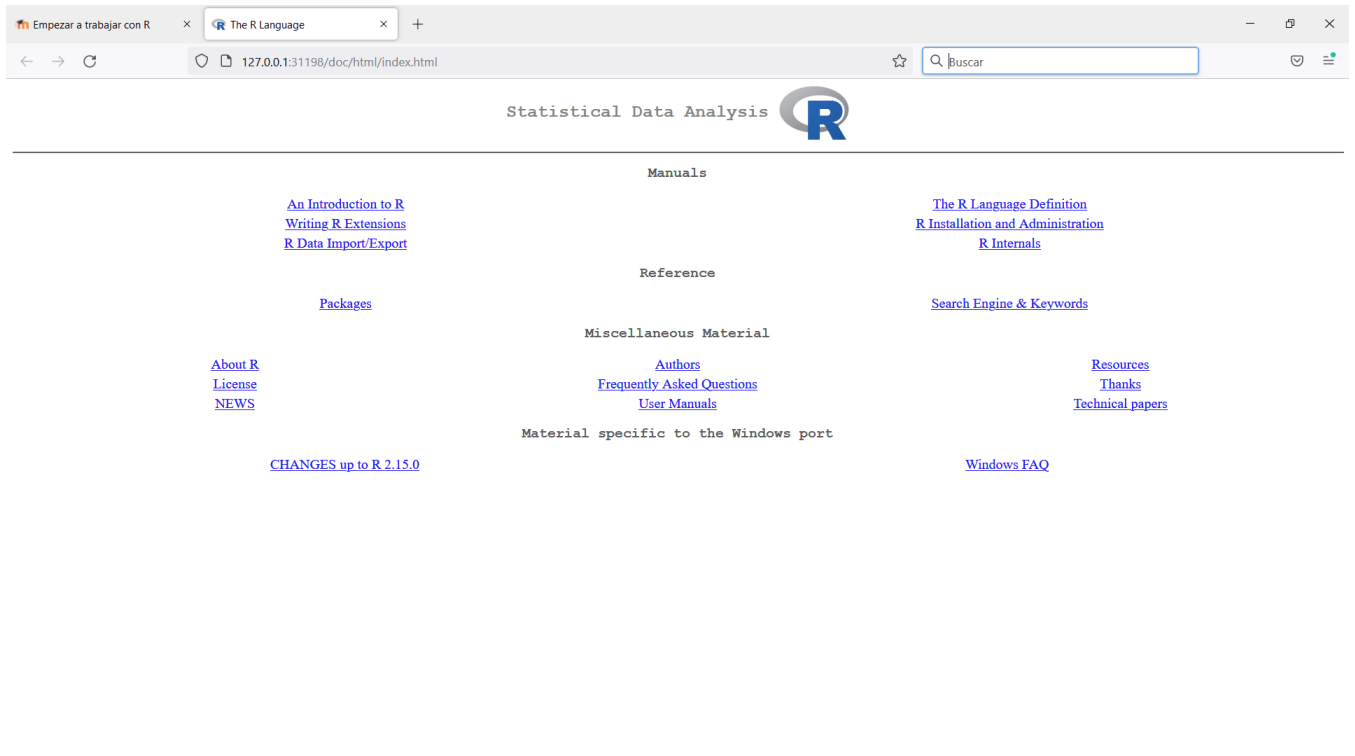
Por cierto, la orden **rm()** nos sirve para borrar los objetos que queramos.

## 5. Ayuda en R

Cuando trabajamos con R siempre vamos a tener a nuestra disposición una ayuda para conocer lo que hacen las funciones que estamos usando y poder resolver las dudas que se nos presenten con su uso.

Desde de R hay varias formas de acceder a esta ayuda.

De manera genérica podemos usar la orden **help.start()**. Con esta orden arrancamos una página web en nuestro navegador donde podemos acceder a toda la ayuda que nos ofrece el programa R (paquetes, funciones, manuales, preguntas frecuentes, etc.)



También podemos preguntar una duda concreta sobre una función con las órdenes **help(load)** o **?load**.

Con **help** o **?** se nos abre una página web en un navegador donde nos explica la función que queramos dándonos todas sus opciones y ejemplos de como se usa.



Empezar a trabajar con R

R: Reload Saved Datasets

127.0.0.1:31198/library/base/html/load.html

Buscar

load {base}

Reload Saved Datasets

R Documentation

Description

Reload datasets written with the function `save`.

Usage

```
load(file, envir = parent.frame(), verbose = FALSE)
```

Arguments

file

a (readable binary-mode) [connection](#) or a character string giving the name of the file to load (when [tilde expansion](#) is done).

envir

the environment where the data should be loaded.

verbose

should item names be printed during loading?

Details

`load` can load R objects saved in the current or any earlier format. It can read a compressed file (see [save](#)) directly from a file or from a suitable connection (including a call to [url](#)).

A not-open connection will be opened in mode "rb" and closed after use. Any connection other than a [gzfile](#) or [gzcon](#) connection will be wrapped in [gzcon](#) to allow compressed saves to be handled: note that this leaves the connection in an altered state (in particular, binary-only), and that it needs to be closed explicitly (it will not be garbage-collected).

Only R objects saved in the current format (used since R 1.4.0) can be read from a connection. If no input is available on a connection a warning will be given, but any input not in the current format will result in an error.

Loading from an earlier version will give a warning about the 'magic number': magic numbers 1971:1977 are from R < 0.99.0, and `RD[ABX]1` from R 0.99.0 to R 1.3.1. These are all obsolete, and you are strongly recommended to re-save such files in a current format.

The `verbose` argument is mainly intended for debugging. If it is `TRUE`, then as objects from the file are loaded, their names will be printed to the console. If `verbose` is set to an integer value greater than one, additional names corresponding to attributes and other parts of individual objects will also be printed. Larger values will print names to a greater depth.

Objects can be saved with references to namespaces, usually as part of the environment of a function or formula. Such objects can be loaded even if the namespace is not available: it is replaced by a reference to the global environment with a warning. The warning identifies the first object with such a reference (but there may be more than one).

Value

## 6. Resumiendo

Para resumir las principales órdenes que hemos estado viendo os dejo un pequeño listado:

- ls()**. Muestra todos los objetos creados.
- class()**. Muestra el tipo de un objeto.
- str()**. Muestra el tipo de un objeto con más información.
- rm()**. Borrar un objeto.
- print()**. Muestra por pantalla el contenido de un objeto.
- save()**. Guardar un objeto en un fichero de datos RData.
- load()**. Cargar un fichero de datos.
- getwd()**. Saber en que directorio estamos trabajando.
- setwd()**. Modificar el directorio de trabajo.
- help()**. Pedir ayuda sobre un comando.

Y las principales funciones matemáticas que podemos usar en R.

### ■ Signos de operaciones aritméticas:

Operación	Suma	Resta	Multiplicación	División	Potencia	Cociente entero	Resto div. entera
Signo	+	-	*	/	^	%/ %	%%

### ■ Funciones numéricas:

Función	$\sqrt{x}$	$e^x$	$\ln(x)$	$\log_{10}(x)$	$\log_a(x)$	$n!$	$\binom{n}{m}$
Signo	sqrt	exp	log	log10	log( , a)	factorial	choose
Función	$\sin(x)$	$\cos(x)$	$\tan(x)$	$\arcsin(x)$	$\arccos(x)$	$\arctan(x)$	$ x $
Signo	sin	cos	tan	asin	acos	atan	abs

fuelle: UIB. Universitat de les Illes Balears