Introducción a R

Francisco Leiva

3/15/2021

## Comandos para R

### Comandos para interactuar con el interfaz o dentro del script en particular

* Ctrl + shift + N: crea un nuevo script. Sirve también cuando accidentalmente cierren el script, sólo vean el ambiente “consola” del lado izquierdo y no sepan como volver a escribir sobre el script.
* Ctrl + L: limpia la consola (no hace falta posicionarse en ella).
* Ctrl + enter: correr el código seleccionado. En caso de no haber seleccionado nada, ejecutará el código sobre donde se encuentre uno. Por ejemplo, si hacemos clic en la línea 32 en cualquier parte del código, se ejecutará todo el código. Siempre que uno quiera ejecutar una línea en concreto, se recomienda hacer clic (es decir, posicionarse) sobre el final del código para controlar mejor la extensión del mismo. Ctrl + shift + enter: ejecuta todo el script. No se recomienda su uso por razones muy sencillas: 1.¿para qué queremos volver a ejecutar códigos anteriores que sabemos que funcionan y nos devuelven lo que esperábamos que devolvieran? 2. A mayor cantidad de códigos que se escriban en el script y/o mayor cantidad de datos que con los que estemos trabajando, más demora tendrá Rstudio en devolvernos el resultado. Este comando se menciona, mas si el alumno puede olvidarse de haberlo leído, mejor.
* Ctrl + F: activa el modo busqueda y reemplazo.
* Ctrl + +/-: Zoom-in/Zoom-out.
* Ctrl + D: borrar todo lo escrito en la línea.
* Ctrl + I: desplazarse a la línea de abajo (nótese que línea es diferente a renglón).
* Comandos útiles para desplazarse que por lo general sirven en todos los editores de texto.
  + Ctrl + flecha izquierda/derecha: se desplaza hacia la palabra de atrás o hacia la palabra posterior.
  + Tecla Inicio: nos desplazamos hasta el primer caracter de la línea.
  + Tecla Fin: nos desplazamos hasta el último caracter de la línea.
  + Ctrl + Tecla inicio: Nos desplazamos hacia el inicio del documento de texto.
  + Ctrl + Tecla fin: Nos desplazamos hacia el final del documento de texto.
  + Ctrl + shift + flecha derecha (izquierda): seleccionamos toda el texto hasta el próximo (anterior) espacio en blanco, si se vuelve a presionar se añadirá el texto hasta el siguiente (anterior) espacio en blanco y así…
  + Shift + inicio (final): selecciona el texto hasta el comienzo (final) del renglón.
  + Ctrl + tab: nos movemos hacia la pestaña de la derecha (en caso de existir).
  + Ctrl + shift + tab: nos movemos hacia la pestaña de la izquierda (en caso de existir)[^nota1: el alumno puede verificar que también funciona en google chorme].

A veces es útil realizar anotaciones o aclaraciones en el espacio donde se codifica, ya sea para mencionar lo que se está haciendo, esto puede realizarse mediante la utilización del caracter ‘#’. Nótese que, en cuanto uno escriba este caracter e intente correr la línea, la consola no devolverá un resultado, sino que omitirá por completo la línea o imprimirá la línea con un resultado seguido de la anotación. Por ejemplo, intente correr la siguiente oración -sin las comillas: “2+4 #esto es dos más cuatro”, la consola registrará este “pedido” y devolverá simplemente 6. Como se dijo, es útil para escribir aclaraciones en el mismo espacio en el que se codifica.

### Atajos y comentarios de Rmarkdown

* Ctrl + Alt + I: crea un *chunk*. Un *chunk* es, básicamente un espacio dónde hay fragmentos de código, o donde funcionan las mismas “reglas de juego” que en el script de R.
* Nótese que en Rmarkdown no es necesario escribir # para que R no la interprete como código. En este ambiente, si no se escribe un *chunk*, Rmarkdown devolverá todo en forma de texto.

La implementación de estos comandos a la hora de escribir ahorran tiempo y mantienen las manos en el espacio de teclado, lo que incrementa también el rendimiento de escritura.

## Consejos útiles

Antes de comenzar, mencionar lo que se cree más importante en el manejo de R, Rstudio y, en general, la práctica de programar: ser ordenados y claros. Los resultados que se desprendan de los códigos pueden reflejar lo que uno busca obtener, pero en cuanto ocurran errores (que efectivamente ocurrirán), será mucho más sencillo encontrarlos. Para ello, se recomienda lo siguiente a la hora de escribir en el script:

* Dejar un renglón en blanco por cada código que se escribe.
* Dejar espacios entre los operadores matemáticos
* Crear la cantidad justa de objetos y eliminar aquellos con los que no se trabaja (uno siempre puede volver a crearlos, se borra el objeto, no el código)
* Muchas veces sucederá que ciertas líneas de código sean similares o se confundan, para combatir esto a veces es útil realizar aclaraciones mediante el caracter #, como si se escribiera una receta con los códigos y al costado (o antes de escribir el código) se especificara lo que uno esta haciendo. Realizar esto no sólo brindará información al usuario (dado que uno puede olvidarse de qué fue lo que quiso obtener cuando escribió el código), sino también que, en caso de compartir el archivo que contiene el código, otra persona puede guiarse de una mejor manera.

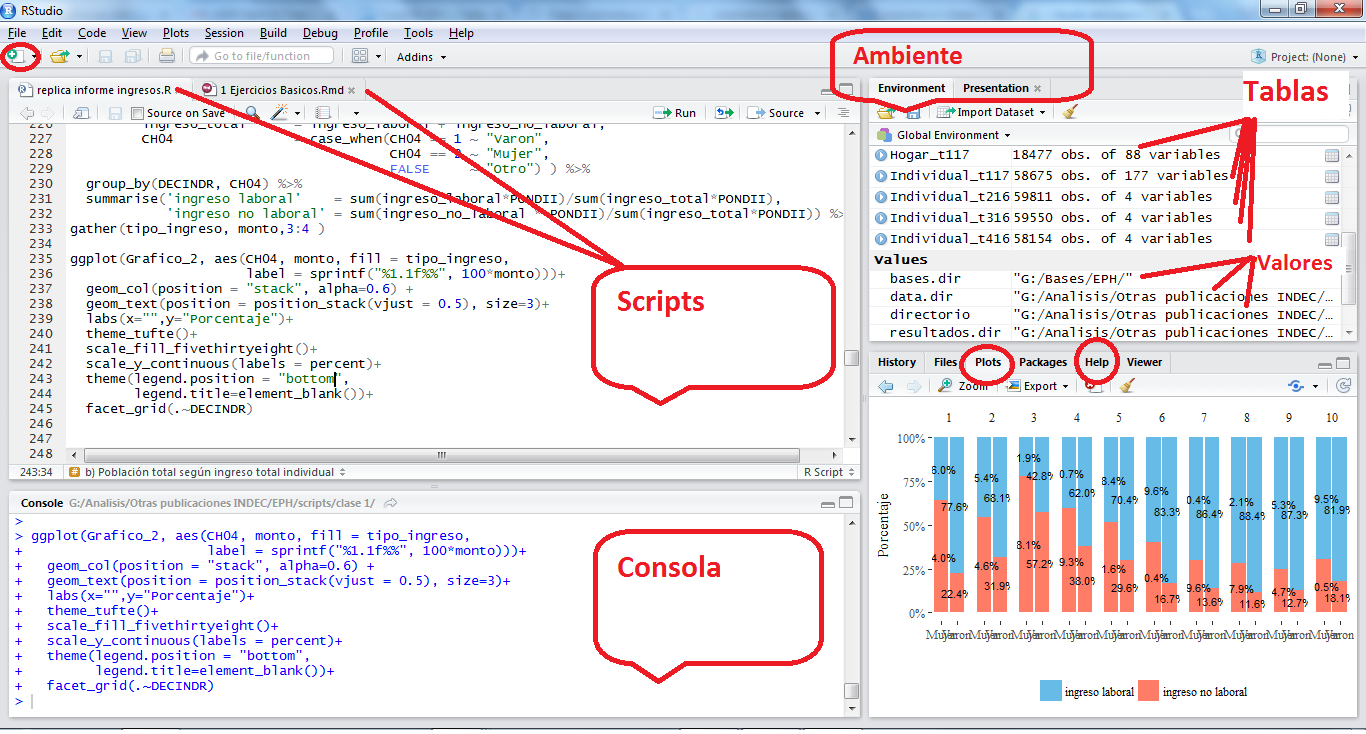
Otra idea que puede llegar a aligerar el proceso no sólo de aprendizaje, sino también de trabajo en R es la siguiente: hay muchas maneras o alternativas para lograr un mismo resultado en R. Sin embargo, es cierto que hay algunas vías más eficientes u óptimas que otras. Esta cuestión tiene especial importancia cuando uno busca comparar con otra persona el procedimiento por el cual llego al resultado y encuentra que la otra persona utilizó vías diferentes. En este sentido, se creer que la comparación de procedimiento a la hora de llegar un resultado enriquece el proceso de aprendizaje.

Otro consejo de utilidad puede ser confeccionar una lista de los comandos (códigos, funciones, etc.) que vaya aprendiendo para tener a mano en caso de necesitar realizar una determinada operación.

## ¿Que es R?

* Lenguaje para el procesamiento y análisis estadístico de datos
* Software Libre
* Sintaxis Básica: R base
* Sintaxis incremental: El lenguaje se va ampliando por aportes de Universidades, investigadores y empresas privadas, organizados en librerías (o paquetes)
* Comunidad web muy grande para realizar preguntas y despejar dudas.
* Graficos con calidad de publicación

El *entorno* más cómodo para utilizar el *lenguaje* **R** es el *programa* **R studio**



Pantalla Rstudio

* Rstudio es una empresa que produce productos asociados al lenguaje R, como el programa sobre el que corremos los comandos, y extensiones del lenguaje (librerías).

## Diferencias con STATA y SPSS

* Gratuito
* Únicamente funciona por líneas de código (No hay botones para ejecutar comandos)
* Posibilita trabajar con múltiples bases de microdatos al mismo tiempo, sin mayor dificultad (No requiere abrir cada base, trabajarla por separado y luego cerrarla)
* Más potente
  + Totalmente automatizable
  + Aportes de usuarios
  + Extensible a otros lenguajes y usos (Presentación como esta, diseño de aplicaciones)
* Más veloz

# Lógica sintáctica.

## Definición de objetos[[1]](#footnote-1)

Los **Objetos/Elementos** constituyen la categoría escencial del R. De hecho, todo en R es un objeto, y se almacena con un nombre específico que **no debe poseer espacios**. Un número, un vector, una función, la progresión de letras del abecedario, una base de datos, un gráfico, constituyen para R objetos de distinto tipo. Los objetos que vamos creando a medida que trabajamos pueden visualizarse en la panel derecho superior de la pantalla.

El operador <- sirve para definir un objeto. **A la izquierda** del <- debe ubicarse el nombre que tomará el elemento a crear. **Del lado derecho** debe ir la definición del mismo. De esta manera, si quiere definir al número 1 con el nombre “A”, se escribe en el script lo siguiente:

A <- 1

Al definir un elemento, el mismo queda guardado en el ambiente del programa, y podrá ser utilizado posteriormente para observar su contenido o para realizar una operación con el mismo. Para obsevarlo u operar, se escribe el nombre del objeto (“A”). Al correr una linea con el nombre del objeto, la consola del programa muestra su contenido. Entre corchetes observamos el número de orden del elemento en cuestión[[2]](#footnote-2).

A

## [1] 1

A+6

## [1] 7

El operador = es **equivalente** a <-, pero en la práctica no se utiliza para la definición de objetos.

B = 2  
B

## [1] 2

<- es un operador **Unidireccional**, es decir que:  
A <- B implica que **A** va tomar como valor el contenido del objeto **B**, y no al revés.

A <- B  
A #Ahora A toma el valor de B, y B continua conservando el mismo valor

## [1] 2

B

## [1] 2

## R base

Con *R base* uno se refiere a los comandos básicos que vienen incorporados en el R, sin necesidad de cargar librerías.

Existen diferentes tipos de datos básicos:

* numeric: valores numéricos, incluye decimales.
* integer: números enteros, no incluye decimales.
* character valores alfanuméricos, es decir, letras, números y signos mezclados.
* logical: valores lógicos, TRUE o FALSE.

## Operadores lógicos:

Para dar un ejemplo de su funcionamiento, observe lo que se escribe en el script y la respuesta que da R. Primero se definen los objetos, luego se realizan las comparaciones lógicas, tal como sigue:

#Se redefinen los valores A y B  
A <- 10  
B <- 20  
#Se realizan comparaciones lógicas  
  
A > B

## [1] FALSE

A >= B

## [1] FALSE

A < B

## [1] TRUE

A <= B

## [1] TRUE

A == B

## [1] FALSE

A != B

## [1] TRUE

C <- A != B  
C

## [1] TRUE

Como muestra el último ejemplo, el resultado de una operación lógica puede almacenarse como el valor de un objeto.

Haga una pausa y ejecute la siguiente orden

class(A)

## [1] "numeric"

class(B)

## [1] "numeric"

class(C)

## [1] "logical"

Aquí le estamos preguntando a R (por medio del comando class()) lo siguiente: dado que se han definido los elementos A, B y C, ¿qué tipo de dato básico son?

## Operadores aritméticos:

#suma  
A <- 5+6  
A

## [1] 11

#Resta  
B <- 6-8  
B

## [1] -2

#cociente  
C <- 6/2.5  
C

## [1] 2.4

#multiplicacion  
D <- 6\*2.5  
D

## [1] 15

#raiz cuadrada  
E <- sqrt(9)  
E

## [1] 3

#potencia  
F <- 4^2  
F

## [1] 16

## Funciones:

Las funciones son series de procedimientos estandarizados, que toman como imput determinados argumentos a fijar por el usuario, y devuelven un resultado acorde a la aplicación de dichos procedimientos. Su lógica de funcionamiento es:  
funcion(argumento1 = arg1, argumento2 = arg2)

Agunos ejemplos para comprender su funcionamiento:

* paste() : concatena una serie de caracteres, indicando por última instancia como separar a cada uno de ellos.

paste("Apruebo", "Econometría", "con", "buena", "nota", sep = ";")

## [1] "Apruebo;Econometría;con;buena;nota"

En caso de que deseemos una separación de un espacio, se deberá escribir sep = " ", o simplemente omitir el argumento sep. El alumno podrá comprobarlo por sí mismo. Nótese que también se podrían definir ciertos objetos y luego realizar la concatenación, tal como sigue:

A <- "Apruebo"  
B <- "Econometría"  
C <- "con"  
D <- "buena"  
E <- "nota"  
  
paste(D, E, A, B, C)

## [1] "buena nota Apruebo Econometría con"

El alumno puede intentar ordenarlos tal como se hizo en la primera concatenación.

* paste0(): concatena una serie de caracteres sin separar
* sum(): suma de todos los elementos de un vector

A <- 4  
sum(A +3+sqrt(9))

## [1] 10

* mean() promedio aritmético de todos los elementos de un vector

PromAcad <- mean(7, 6, 7, 10, 9, 1) #Nótese que no deben sumarse los elementos, dado que constituyen diferentes argumentos dentro de la función  
PromAcad

## [1] 7

## Caracteres especiales

* R es sensible a mayúsculas y minúsculas, tanto para los nombres de las variables, como para las funciones y parámetros.
* Los **espacios en blanco** y los **carriage return** (*enter*) no son considerados por el lenguaje. Se pueden aprovechar para emprolijar el código y que la lectura sea más simple.
* El **numeral** # se utiliza para hacer comentarios. Todo lo que se escribe después del # no es interpretado por R. Se debe utilizar un # por cada línea de código que se desea anular.
* Los **corchetes** [] se utilizan para acceder a un objeto:
  + en un vector[n° orden]
  + en una tabla[fila, columna]
  + en una lista[n° elemento]
* el signo **$** también es un método de acceso. Particularmente, en los dataframes, permitirá acceder a una determinada columna (o variable) de una tabla
* Los **paréntesis**() se utilizan en las funciones para definir los parámetros.
* Las **comas** , se utilizan para separar los parametros al interior de una función.

## Objetos:

Existen un gran cantidad de objetos distintos en R:

* Valores
* Vectores
* Data Frames
* Listas

### Valores

Los valores y vectores pueden ser, como ya se ha mencionado, a su vez de distintas *clases*:

**Numeric**

A <- 1  
class(A)

## [1] "numeric"

**Character**

A <- paste('Soy', 'una', 'concatenación', 'de', 'caracteres', sep = " ")  
A

## [1] "Soy una concatenación de caracteres"

class(A)

## [1] "character"

**Factor**

A <- factor("Soy un factor, con niveles fijos")  
class(A)

## [1] "factor"

La diferencia entre un *character* y un *factor* es que el último tiene solo algunos valores permitidos (levels), con un orden interno predefinido (el cual ,por ejemplo, se respetará a la hora de realizar un gráfico).

### Vectores

Para crear un **vector** se utilizará el comando c(), de combinar.

C <- c(1, 3, 4)  
C

## [1] 1 3 4

sumarle 2 a cada elemento del **vector** anterior

C <- C + 2  
C

## [1] 3 5 6

sumarle 1 al primer elemento, 2 al segundo, y 3 al tercer elemento del **vector** anterior

D <- C + 1:3 #esto es equivalente a hacer 3+1, 5+2, 6+3  
D

## [1] 4 7 9

#O también  
E <- C + c(1,2,3)  
E

## [1] 4 7 9

1:3 significa que se quieren todos los números enteros desde 1 hasta 3.

A continuación, se crea un **vector** que contenga las palabras: “Carlos”,“Federico”,“Pedro”.

E <- c("Carlos","Federico","Pedro")  
E

## [1] "Carlos" "Federico" "Pedro"

Para acceder a algún elemento del vector, podemos buscarlo (como ya se mencionó en el apartado de ‘Caracteres especiales’) por su número de orden, entre [ ], tal como se observa a continuación:

E[2]

## [1] "Federico"

Si lo que interesa almacenar dicho valor, al buscarlo lo asignamos a un nuevo objeto, dandole el nombre que se desee

elemento2 <- E[2]  
elemento2

## [1] "Federico"

para **borrar** un objeto del ambiente de trabajo, se utilizará el comando *rm()*

rm(elemento2)  
elemento2

## Error in eval(expr, envir, enclos): object 'elemento2' not found

Este comando es muy importante dado que permite mantener ordenado el ambiente, atendiendo a los objetos que se están utilizando o que se quieran utilizar en el momento más próximo.

También se puede cambiar los argumentos de un vector. Por ejemplo, si se quisiera cambiar el argumento 2 del vector E por “Pablo”, se podría hacer de la siguiente manera

E[2] <- "Pablo"  
E

## [1] "Carlos" "Pablo" "Pedro"

O también redefinir el vector E, ahora atendienod a que se trata de “Pablo” y no de “Federico”. Como se dijo anteriormente, el mismo resultado a través de distintas vías.

### Data Frames

Un Data Frame es una tabla de datos, donde cada columna representa una variable, y cada fila una observación.

Este objeto suele ser central en el proceso de trabajo, y suele ser la forma en que se cargan datos externos para trabajar en el ambiente de R, y en que se exportan los resultados de nuestros trabajo.

También Se puede crear como la combinación de N vectores de igual tamaño. Por ejemplo, se toman algunos valores del Indice de salarios proporcionado por el INDEC (2016).

INDICE <- c(100, 100, 100,  
 101.8, 101.2, 100.73,  
 102.9, 102.4, 103.2)  
  
FECHA <- c("Oct-16", "Oct-16", "Oct-16",  
 "Nov-16", "Nov-16", "Nov-16",  
 "Dic-16", "Dic-16", "Dic-16")  
  
  
GRUPO <- c("Privado\_Registrado","Público","Privado\_No\_Registrado",  
 "Privado\_Registrado","Público","Privado\_No\_Registrado",  
 "Privado\_Registrado","Público","Privado\_No\_Registrado")

Hasta aquí se tienen tres vectores, uno que contiene valores númericos (el alumno comprobará que en caso de que al menos uno de los elementos -todos numéricos, se modifica por un elemento del tipo “caracter”, en ese la respuesta que dará R será distinta a la que sigue a continuación)

class(INDICE)

## [1] "numeric"

Y otros dos que tienen caracteres en los argumentos

class(FECHA)

## [1] "character"

class(GRUPO)

## [1] "character"

Ahora bien, en caso de que querramos crear el data frame, se hará de la siguiente mnaera:

Datos <- data.frame(INDICE, FECHA, GRUPO)  
Datos

## INDICE FECHA GRUPO  
## 1 100.00 Oct-16 Privado\_Registrado  
## 2 100.00 Oct-16 Público  
## 3 100.00 Oct-16 Privado\_No\_Registrado  
## 4 101.80 Nov-16 Privado\_Registrado  
## 5 101.20 Nov-16 Público  
## 6 100.73 Nov-16 Privado\_No\_Registrado  
## 7 102.90 Dic-16 Privado\_Registrado  
## 8 102.40 Dic-16 Público  
## 9 103.20 Dic-16 Privado\_No\_Registrado

class(Datos)

## [1] "data.frame"

El alumno tiene que pensar esta operación como si cada vector se insertara como columna. De hecho, si no se especifica otra cosa, el nombre de las columnas será el mismo que el texto con el que se definió.

Tal como en un **vector**, se ubica a los elementos del dataframe mediante [ ], pero tiene una diferencia: se ha dicho que el dataframe contiene filas y columnas, de esta manera, si queremos encontrar el elemento 1 de un dataframe, R nos responderá con todo el vector (o columna) “INDICE”. Si la intención era saber el elemento 1 del vector “INDICE”, en ese caso deberá buscárselo de la siguiente manera:

**[fila, columna]**

Datos[1,1]

## [1] 100

Otra opción es especificar la columna, mediante el operador **$**, y luego seleccionar dentro de esa columna el registro deseado mediante el número de orden (recomendado).

Datos$INDICE

## [1] 100.00 100.00 100.00 101.80 101.20 100.73 102.90 102.40 103.20

Datos$INDICE[1]

## [1] 100

¿que pasa si se hace Datos$FECHA[3,2] ?

Datos$FECHA[3,2]

## Error in `[.default`(Datos$FECHA, 3, 2): incorrect number of dimensions

Nótese que el último comando tiene un número incorrecto de dimensiones, porque se está refiriéndose 2 veces a la columna FECHA.

Acorde a lo visto anteriormente, el acceso a los **dataframes** mediante [ ], puede utilizarse para realizar filtros sobre la base, especificando una condición para las filas. Por ejemplo, se pueden utilizar los [ ] para conservar del **dataframe** Datos unicamente los registros con fecha de Diciembre 2016:

Datos[Datos$FECHA=="Dic-16",]

## INDICE FECHA GRUPO  
## 7 102.9 Dic-16 Privado\_Registrado  
## 8 102.4 Dic-16 Público  
## 9 103.2 Dic-16 Privado\_No\_Registrado

Obsérvese que se utilizan dos = y no uno. La lógica del paso anterior sería: se accede al dataframe Datos, pidiendo únicamente conservar las filas (por eso la condición se ubica a la *izquierda* de la ,) que cumplan el requisito de pertenecer a la categoría **“Dic-16”** de la variable **FECHA**.

Aún más, podría aplicar el filtro y al mismo tiempo identificar una variable de interés para luego realizar un cálculo sobre aquella. Por ejemplo, podría calcular la media de los indices en el mes de Diciembre.

###Por separado  
Indices\_Dic <- Datos$INDICE[Datos$FECHA=="Dic-16"]  
Indices\_Dic

## [1] 102.9 102.4 103.2

mean(Indices\_Dic)

## [1] 102.8333

### Todo junto  
mean(Datos$INDICE[Datos$FECHA=="Dic-16"])

## [1] 102.8333

La lógica de esta sintaxis sería: “Me quedó con la variable **INDICE**, cuando la variable FECHA sea igual a **"Dic-16"**, luego calculo la media de dichos valores”

Se recomienda no continuar hasta haber entendido bien estos dos últimos códigos y su respuesta, así como también jugar un poco con el dataframe y filtrarlo de otras maneras no sin antes dar una advertencia. Mientras se filtre el dataframe Datos, sin redefinirlo con el mismo nombre, el filtrad sólo aparecerá como resultado mientras que el dataframe se conservará inalterado. Si se escribe Datos <- y luego se lo filtra, el dataframe Datos se habrá alterado. He aquí la importancia de definir bien los objetos antes de continuar. Existen algunas recomendaciones para no equivocarse frente a este error introductorio, una de ellas tiene que ver con no definir ningún objeto hasta saber que el filtrado es efectivamente el que se quiso hacer. Otra alternativa es definir un objeto de manera temporal con el propósito de *chequear* si, empleándolo posteriormente, resulta lo que quiso hacer en un principio.

### Listas

Contienen una concatenación de objetos de cualquier tipo. Así como un vector contiene valores, un dataframe contiene vectores, una lista puede contener dataframes, pero también vectores, o valores, y *todo ello a la vez*

superlista <- list(A,B,C,D,E,FECHA, DF = Datos, INDICE, GRUPO)  
superlista

## [[1]]  
## [1] Soy un factor, con niveles fijos  
## Levels: Soy un factor, con niveles fijos  
##   
## [[2]]  
## [1] "Econometría"  
##   
## [[3]]  
## [1] 3 5 6  
##   
## [[4]]  
## [1] 4 7 9  
##   
## [[5]]  
## [1] "Carlos" "Pablo" "Pedro"   
##   
## [[6]]  
## [1] "Oct-16" "Oct-16" "Oct-16" "Nov-16" "Nov-16" "Nov-16" "Dic-16" "Dic-16"  
## [9] "Dic-16"  
##   
## $DF  
## INDICE FECHA GRUPO  
## 1 100.00 Oct-16 Privado\_Registrado  
## 2 100.00 Oct-16 Público  
## 3 100.00 Oct-16 Privado\_No\_Registrado  
## 4 101.80 Nov-16 Privado\_Registrado  
## 5 101.20 Nov-16 Público  
## 6 100.73 Nov-16 Privado\_No\_Registrado  
## 7 102.90 Dic-16 Privado\_Registrado  
## 8 102.40 Dic-16 Público  
## 9 103.20 Dic-16 Privado\_No\_Registrado  
##   
## [[8]]  
## [1] 100.00 100.00 100.00 101.80 101.20 100.73 102.90 102.40 103.20  
##   
## [[9]]  
## [1] "Privado\_Registrado" "Público" "Privado\_No\_Registrado"  
## [4] "Privado\_Registrado" "Público" "Privado\_No\_Registrado"  
## [7] "Privado\_Registrado" "Público" "Privado\_No\_Registrado"

Para acceder un elemento de una lista, podemos utilizar el operador **$**, que se puede usar a su vez de forma iterativa

superlista$DF$FECHA[2]

## [1] Oct-16  
## Levels: Dic-16 Nov-16 Oct-16

# Instalación de paquetes complementarios al R Base

Hasta aquí se han visto múltiples funciones que están contenidas dentro del lenguaje básico de R. Ahora bien, al tratarse de un software libre, los usuarios de R con más experiencia contribuyen sistemáticamente a expandir este lenguaje mediante la creación y actualización de **paquetes** complementarios. Lógicamente, los mismos no están incluidos en la instalación inicial del programa, pero se pueden descargar e instalarlos al mismo tiempo con el siguiente comando:

install.packages("nombre\_del\_paquete")

Resulta recomendable **ejecutar este comando desde la consola** ya que solo se necesita correr una vez en la computadora. Al ejecutar el mismo, se descargarán de la pagina de [CRAN](file:///C:\Users\franl\OneDrive\Documentos\Documentos\www.cran.r-project.org) los archivos correspondientes al paquete hacia el directorio en donde hayamos instalado el programa. Típicamente los archivos se encontrarán en \*\*C:Files-3.5.0\*\*, siempre con la versión del programa correspondiente.

Una vez instalado el paquete, cada vez que abramos una nueva sesión de R y querramos utilizar el mismo debemos **cargarlo al ambiente de trabajo** mediante la siguiente función:

library(nombre\_del\_paquete)

Nótese que al cargar/activar el paquete no son necesarias las comillas.

# Lectura y escritura de archivos

## .csv y .txt

Hay **muchas** funciones para leer archivos de tipo *.txt* y *.csv*. La mayoría sólo cambia los parámetros que vienen por default.

Es importante tener en cuenta que una base de datos que proviene de archivos *.txt*, o *.csv* puede presentar diferencias en cuanto a los siguientes parametros:

* encabezado
* delimitador (,, tab, ;)
* separador decimal

Es por ello que siempre que se cargue un archivo *.txt* o *.csv*, deberá abrírselo anteriormente para observar cómo esta estructurado y, en base a eso, construir y ejectuar el código para cargarlo en el ambiente de trabajo. Resulta razonable añadir que, cuando se desee cargar un archivo de esta especie, por lo general, es recomendable asociarlo a un objeto, *i.e.*, definirlo. En caso de que no se lo haga de esta manera, la manipulación de los archivos se hará sumamente tediosa, tanto más cuanto más grande o extenso sea el archivo en términos de datos.

# Bibliografía

* Kozlowski D., Weksler G. y Shokida N. (2018) Curso R EPH.
* Heiss, F. (2020) - **Using R for Introductory Econometrics**. Florian Heiss

1. De ahora en más, se observará un rectangulo más oscuro que el otro que dan cuenta de lo que se escribe en el script y lo que devuelve R, respectivamente. [↑](#footnote-ref-1)
2. De momento no se profundizará, pero está claro que el objeto “A” es un valor numérico, un objeto que da cuenta de un número. El hecho de que lo definamos de esta manera le permite a R leerlo de tal manera (podría decirse que le estamos diciendo a R cómo leerlo). Se sugiere que en lo que sigue, el alumno vaya prestando atención a la manera en que “piensa y razona” R, tanto más fácil será codificar cuanto más se entendida esta lógica de razonamiento del programa R. [↑](#footnote-ref-2)