Primeros pasos en R.

Al iniciarse **R** (ver Figura 16), **R** espera la entrada de órdenes y presenta un símbolo para indicarlo. El símbolo asignado, como puede observarse al final, es

>

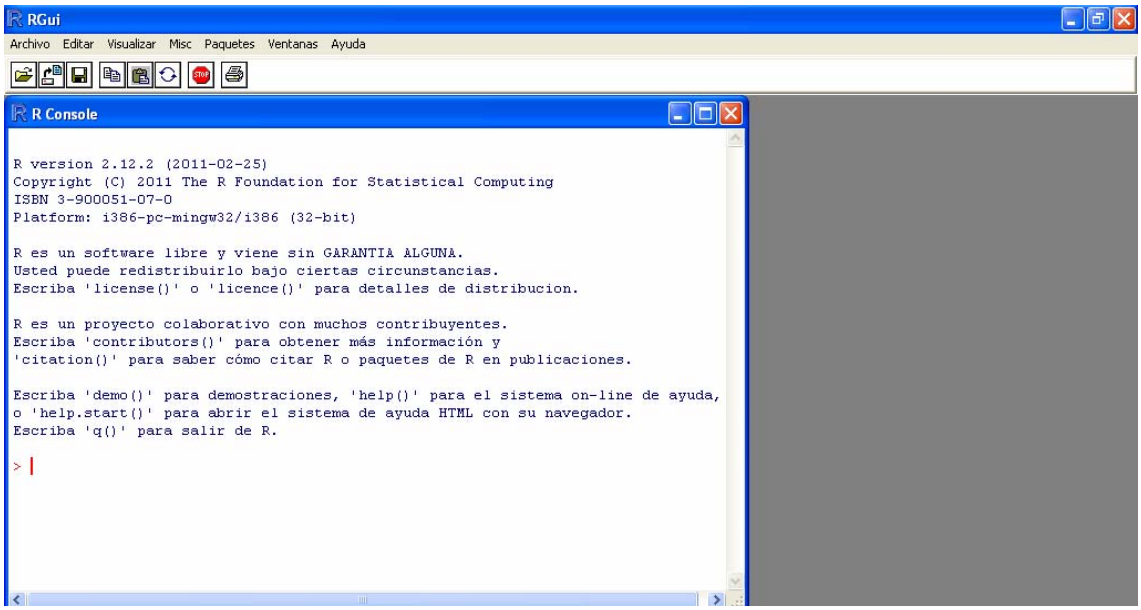


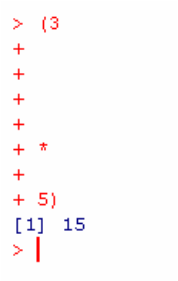
*Figura 16. Pantalla de inicio de **R**.*

Con el fin de trabajar de forma ordenada, es interesante almacenar todo el material creado en un mismo "directorio de trabajo" para que pueda ser usado fácilmente. Obsérvese, de hecho, que el programa pregunta al finalizar cada sesión si se desea salvar los datos. Con este objetivo se crea un directorio de trabajo. Los pasos a seguir son:

1. Creamos un directorio, en el que se almacenaran los distintos archivos de datos y las funciones que crearemos y que usaremos en **R**. Con este objetivo, en el directorio raíz crearemos una carpeta que llamaremos, por ejemplo, *Econometría*.
2. Para especificar que este será nuestro directorio de trabajo, cada vez que se inicie **R**, habrá que pulsar sobre *Archivo* en el menú que aparece en la parte superior del programa, y en la cascada que se despliega, seleccionar *Cambiar dir...*. Entonces, en la nueva pantalla que surge, hay que buscar la carpeta creada en el punto anterior. El camino es "*C:\Econometría*". Una vez seleccionada la carpeta se pulsa en *Aceptar* y ya tenemos establecido nuestro directorio de trabajo.

A continuación se introducen algunos conceptos iniciales que pueden resultar útiles para realizar cualquier operación básica. **R** es un lenguaje interpretado, lo que significa que se pueden escribir órdenes y **R** las ejecuta inmediatamente. Por ejemplo, **R** se puede utilizar como una calculadora, ya que es capaz de realizar fácilmente las operaciones elementales. De este modo puede escribir 3+5 o 6*3 y obtendrá los resultados esperados. La operaciones de suma, resta, multiplicación división, exponenciación, división entera y módulo se realizan, respectivamente, mediante los símbolos **+**, **-**, *****, **/**, **^**, **%/%**, y **%%**.

R no evalúa una expresión hasta que tiene la certeza de que se ha terminado su escritura, es decir, si una expresión comienza con un paréntesis (o una llave), se podrán utilizar varias líneas para su escritura, tal como se observa en la siguiente salida



Cuando las expresiones ocupan varias líneas o son bastantes largas, se recomienda el uso de un editor de textos, tal como el Bloc de notas o Wordpad. Otra opción es utilizar la aplicación Commander que incorpora **R**.

R también permite realizar tareas a través de funciones que tiene implementadas. El primer lugar indicamos que aunque se explicarán de forma adecuada todas las herramientas utilizadas, puede ocurrir que surja la necesidad por parte del alumno de realizar un aprendizaje autónomo sobre algunas cuestiones no explicadas con la suficiente claridad. En tales casos se recomienda

- Usar el sistema de ayuda que ofrece **R**, el cual consiste en ejecutar el comando ***help*** sobre la función que se necesita ayuda. Otra opción puede ser solicitar ejemplos de uso sin más que usar el comando ***example***.
- Si no se conoce el nombre exacto de una función, puede pedirse ayuda usando la función ***apropos***, que localiza objetos cuyo nombre coincide parcialmente con el argumento que se suministra.
- En la página web del programa **R** existen distintos manuales. Concretamente, para unos primeros pasos, es interesante el manual titulado ***An Introduction to R.*** Estos manuales se encuentran en el menú superior al seleccionar *Ayuda* (siempre que haya sido solicitado en la instalación del programa).

A continuación se muestran ejemplos de las funciones anteriormente comentadas. Notamos que en **R** se utiliza el símbolo **#** para realizar comentarios.

```
> # Ejemplo del uso de la función "example" en la función "mean"
> example("mean")

mean> x <- c(0:10, 50)

mean> xm <- mean(x)

mean> c(xm, mean(x, trim = 0.10))
[1] 8.75 5.50

mean> mean(USArrests, trim = 0.2)
  Murder  Assault UrbanPop     Rape
    7.42   167.60    66.20    20.16
>
> # Ejemplo para buscar funciones que contengan "mean" mediante la función "apropos"
> apropos("mean")
 [1] "colMeans"        "kmeans"          "mean"            "mean.data.frame" "mean.Date"       "mean.default"
 [7] "mean.difftime"   "mean.POSIXct"    "mean.POSIXlt"    "rowMeans"        "weighted.mean"
>
```

En **R** se puede redireccionar la salida de una expresión de forma que aparezca escrita en pantalla, sino que vaya dirigida a un objeto con el nombre que indiquemos. Por ejemplo, si se escribe **`RESULTADO <- 5+3`**, no se obtendrá ningún resultado en pantalla, sino que éste se almacena en el objeto de nombre **`RESULTADO`**. Posteriormente, se podrá recuperar este objeto y utilizarlo en cualquier momento sin más que escribir su nombre. Así, **`RESULTADO`** devuelve el valor 8 y **`RESULTADO^2`** devolverá el valor 64. El redireccionamiento se puede escribir en la dirección contraria, es decir escribir **`5+3 ->RESULTADO`**, o bien utilizar el símbolo **=**. Recomendamos el uso de mayúsculas para los objetos creados por el usuario, ya que, en general, **R** utiliza las minúsculas para los objetos que tiene implementados. De esta forma se evitaría que el usuario utilice el nombre de un objeto ya existente en **R**, ya que en caso de que esto ocurra no se podría utilizar el objeto de **R** que ha sido redireccionado por el usuario.

A continuación se describirán algunas clases de objetos comunes que se pueden utilizar en **R**. Estos objetos son los vectores, las matrices y las listas. Recordamos que en el manual ***An Introduction to R*** se puede consultar una descripción más detallada de estos y otros objetos en **R**.

Existen distintas formas de crear un vector. Una forma es utilizar el símbolo de dos puntos situado entre dos números, lo cual genera un vector en orden ascendente o descendente según indiquemos. Algunos ejemplos de este comando son los siguientes:

```
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
> 10:1
 [1] 10  9  8  7  6  5  4  3  2  1
>
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
>
> 10:1
 [1] 10  9  8  7  6  5  4  3  2  1
>
> 1:3.5
[1] 1 2 3
>
> 1.8:4.6
[1] 1.8 2.8 3.8
>
```

El comando anterior es un caso particular de la función **seq**, que permite construir vectores que son sucesiones equiespaciadas. Consultando la ayuda de este comando, **help(seq)**, puede comprobarse que este comando posee los siguientes argumentos:

- **from**: indica el valor inicial de la sucesión
- **to**: indica el valor final de la sucesión
- **by**: indica el espaciado entre los valores
- **length**: indica la longitud del vector resultante
- **along**: un objeto cuya longitud se usará para el objeto a construir.

No es necesario utilizar todos los argumentos existentes en una función. Sólo se pueden dar los argumentos necesarios, y el resto se calculan a partir de los dados o bien, si no se indica ninguno, tomarán el valor que tengan por defecto. En este caso, como puede observarse en la ayuda de esta función, tales argumentos toman el valor 1. Tampoco es necesario indicar el argumento que queremos utilizar, puesto que **R** interpreta la información que le estamos dando en el orden que están colocados sus argumentos, y hará falta indicar los argumentos en el caso de que no sigamos el orden en el que están colocados. Por último indicamos que tampoco es necesario escribir el nombre completo de los argumentos, sino que indicando una parte de ellos sería suficiente. Algunos ejemplos de este comando y las observaciones comentadas son los siguientes:

```
> seq(from=1, to=10, by=2)
[1] 1 3 5 7 9
>
> seq(1, 10, 2)
[1] 1 3 5 7 9
>
> seq(10,-6,-2)
[1] 10  8  6  4  2  0 -2 -4 -6
>
> seq(from=1, to=11, length=6)
[1]  1  3  5  7  9 11
>
> seq(1, 11, len=6)
[1]  1  3  5  7  9 11
>
```

Otra forma de definir vectores es mediante las funciones **rep** y **double**. La función **rep** permite crear un vector de forma que se repita un número tantas veces como se indique. Por su parte, la función **double** hace lo mismo, sólo que el número que se repite es el 0. Algunos ejemplos son los siguientes:

```
>
>
> rep(3,5)
[1] 3 3 3 3 3
>
> double(5)
[1] 0 0 0 0 0
```

Para definir o construir vectores que no sean sucesiones equiespaciadas de números, se puede utilizar la función de concatenación **c**, la cual no se limita exclusivamente a números y permite definir cadenas de caracteres. Además puede admitir como argumento varios vectores y lo concatena en uno solo. Algunos ejemplos son los siguientes

```
> c(2,4,5,6,-1)
[1]  2  4  5  6 -1
>
> c(1:5,2,2,seq(7,1,by=-1))
 [1] 1 2 3 4 5 2 2 7 6 5 4 3 2 1
>
> VECTOR1 <- c(1,1,2,2)
> VECTOR2 <- seq(4,8, by=2)
> c(VECTOR1, VECTOR2)
[1] 1 1 2 2 4 6 8
>
> DIAS <- c("Lunes", "Martes", "Miercoles", "Jueves", "Viernes", "Sabado", "Domingo")
> DIAS
[1] "Lunes"     "Martes"    "Miercoles" "Jueves"    "Viernes"   "Sabado"    "Domingo"
>
> c(DIAS, VECTOR1)
 [1] "Lunes"     "Martes"    "Miercoles" "Jueves"    "Viernes"   "Sabado"    "Domingo"   "1"         "1"         "2"
[11] "2"
>
```

Además de concatenar, **R** permite realizar otras operaciones básicas con vectores, como la suma, la multiplicación, etc. A continuación se muestran algunos ejemplos de multiplicación, los cuales se pueden trasladar a otras operaciones básicas como la suma, la resta o la división.

```
>
> (1:10)*2
 [1]  2  4  6  8 10 12 14 16 18 20
>
> (1:10)*(1:5)
 [1]  1  4  9 16 25  6 14 24 36 50
>
> (1:10)*(1:10)
 [1]   1   4   9  16  25  36  49  64  81 100
>
> (1:10)*(2:5)
 [1]  2  6 12 20 10 18 28 40 18 30
Mensajes de aviso perdidos
In (1:10) * (2:5) :
  longitud de objeto mayor no es múltiplo de la longitud de uno menor
>
```

A partir de las instrucciones anteriores podemos realizar algunas observaciones. En primer lugar, se recomienda el uso de paréntesis para garantizar que las operaciones se realicen en el orden deseado. Cuando dos vectores tienen la misma longitud, observamos que la multiplicación se hace componente a componente. Cuando los vectores no tienen la misma longitud, observamos que **R** lo que es hace es replicar el vector de menor longitud tantas veces como haga falta hasta que los dos vectores tengan la misma longitud, y posteriormente realiza la multiplicación componente a componente. En el caso de la longitud de los vectores no coincida después de realizar la replicación, observamos que **R** nos da un mensaje donde nos indica que la longitud de un vector no es múltiplo de la longitud del otro. Para conocer la longitud de un vector se utiliza la función **`length`**. Esta función también nos permite aumentar o disminuir la longitud de un vector. En caso de aumentar la longitud, se completará con valores NA, es decir, con valores faltantes. Algunos ejemplos de esta función son los siguientes

```
>
> VECTOR1 <- c(1,2,3,4,5)
> length(VECTOR1)
[1] 5
>
> length(VECTOR1)<-10
> VECTOR1
 [1]  1  2  3  4  5 NA NA NA NA NA
>
> length(VECTOR1)<-3
> VECTOR1
[1] 1 2 3
>
```

Para referirse al elemento que ocupa la posición i de un vector x se utiliza el comando **`x[i]`**. También es posible referirse a un subconjunto de elementos de vector x. Para ello reemplazaremos el índice i por un vector de índices. Recordamos que además de números, **R** permite trabajar con cadenas de caracteres. Algunos ejemplos son los siguientes:

```
>
>
> X <-11:20
> X
 [1] 11 12 13 14 15 16 17 18 19 20
>
> # Esto es para referirse al elemento 3 del vector X:
> X[3]
[1] 13
>
> # Esto es para referirse a los elementos impares del vector X:
> X[seq(1,9,by=2)]
[1] 11 13 15 17 19
>
> # Otra forma de referirse a los elementos impares del vector X:
> X[1:10%%2==1]
[1] 11 13 15 17 19
>
> NOMBRES <- c("Pepe", "María", "Juan", "Ana", "Lola", "Paco")
> SEXO <-c("H", "M", "H", "M", "M", "H")
> SEXO=="H"
[1]  TRUE FALSE  TRUE FALSE FALSE  TRUE
> NOMBRES[SEXO=="H"]
[1] "Pepe" "Juan" "Paco"
>
```

El siguiente tipo de objeto que se puede utilizar es la matriz, considerada como una variable indexada mediante dos índices. Para crearlas se puede utilizar la función **`matrix`**. Los parámetros de esta función son:

- **data**: Vector que contiene los valores que formarán la matriz. Se debe de tener en cuenta que si este vector no es suficientemente grande, se repetirá las veces que sea necesario.
- **nrow**: Número de filas de la matriz.
- **ncol**: Número de columnas de la matriz.
- **byrow**: Variable lógica que indica si la matriz debe construirse por filas o por columnas. El valor predeterminado es F, es decir, la matriz se construirá por columnas.
- **dimnames**: Lista de longitud 2 con los nombres de las filas y las columnas

Algunos ejemplos de construcción de matrices son los siguientes:

```
>
>
> matrix(data=1:5, nrow=4, ncol=5)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    4    3    2
[2,]    2    1    5    4    3
[3,]    3    2    1    5    4
[4,]    4    3    2    1    5
>
> matrix(1:5, 4, 5)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    4    3    2
[2,]    2    1    5    4    3
[3,]    3    2    1    5    4
[4,]    4    3    2    1    5
>
> matrix(1:5, 4, 5, byrow="T")
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    1    2    3    4    5
[3,]    1    2    3    4    5
[4,]    1    2    3    4    5
>
> DATOS <- c(
+ 74, 1.68, 22,
+ 85, 1.83, 25,
+ 58, 1.58, 21,
+ 80, 1.72, 20)
> matrix(DATOS, 4, 3, by="T", dim=list(c(), c("PESO", "ESTATURA", "EDAD")))
     PESO ESTATURA EDAD
[1,]   74     1.68   22
[2,]   85     1.83   25
[3,]   58     1.58   21
[4,]   80     1.72   20
>
```

Al igual que en el caso de vectores, es posible hacer referencia a un elemento de la matriz, o bien a una submatriz. Algunos ejemplos son los siguientes:

```
> MIS.DATOS<-matrix(DATOS,4,3,by="T",dim=list(c("María","Pepe","Ana","Paco"),c("Peso","Estatura","Edad")))
> MIS.DATOS
      Peso Estatura Edad
María   74     1.68   22
Pepe    85     1.83   25
Ana     58     1.58   21
Paco    80     1.72   20
>
> # Esto muestra los pesos (columna 1)
> MIS.DATOS[,1]
María  Pepe   Ana  Paco
   74    85    58    80
>
> # Esto muestra los datos de Paco
> MIS.DATOS[4,]
    Peso Estatura     Edad
   80.00     1.72    20.00
>
> # Esto muestra la estatura de Ana
> MIS.DATOS[3,2]
[1] 1.58
>
> # Otra forma de referirnos a la estatura de Ana
> MIS.DATOS["Ana", "Estatura"]
[1] 1.58
>
> # Esto nos muestra los datos de Pepe y Paco
> MIS.DATOS[c(2,4),]
     Peso Estatura Edad
Pepe   85     1.83   25
Paco   80     1.72   20
>
> # Esto nos muestra el peso y la edad de Pepe y Paco
> MIS.DATOS[c(2,4),c(1,3)]
     Peso Edad
Pepe   85   25
Paco   80   20
>
```

Se pueden realizar operaciones con matrices mediante operadores aritméticos, pero debemos tener en cuenta que, al igual que en el caso de vectores, estas operaciones se realizan componente a componente, como puede observarse a continuación.

```
>
> A1 <- matrix(1:6,2,3)
> A1
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> A2 <- matrix(2:7,2,3)
> A2
     [,1] [,2] [,3]
[1,]    2    4    6
[2,]    3    5    7
> A1+A2
     [,1] [,2] [,3]
[1,]    3    7   11
[2,]    5    9   13
> A1*A2
     [,1] [,2] [,3]
[1,]    2   12   30
[2,]    6   20   42
>
```

El producto matricial entre dos matrices se realiza mediante el operador **%*%**. El operador **t** calcula la matriz opuesta. La función **crossprod** devuelve el producto matricial cruzado de dos matrices, es decir, la traspuesta de la primera matriz multiplicada por la segunda. Si no se especifica la segunda matriz, **R** la toma igual que a segunda. Algunos ejemplos son los siguientes:

```
>
> t(A2)
     [,1] [,2]
[1,]    2    3
[2,]    4    5
[3,]    6    7
> A1%*%t(A2)
     [,1] [,2]
[1,]   44   53
[2,]   56   68
> t(A1)%*%A1
     [,1] [,2] [,3]
[1,]    5   11   17
[2,]   11   25   39
[3,]   17   39   61
> crossprod(A1)
     [,1] [,2] [,3]
[1,]    5   11   17
[2,]   11   25   39
[3,]   17   39   61
>
```

La función **solve** permite obtener la inversa de una matriz cuando sólo se la da un argumento, y permite resolver sistemas de ecuaciones lineales cuando se le dan dos argumentos. El determinante de una matriz se obtiene mediante la función **det**. En el siguiente ejemplo, además de obtener la inversa y calcular un determinante, se resuelve el sistema de ecuaciones

$$\begin{cases} 3x + 2y = 5 \\ x - y = 0 \end{cases}$$

que en forma matricial puede expresarse como

$$\begin{pmatrix} 3 & 2 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 5 \\ 0 \end{pmatrix}$$

```
>
>
> X <- matrix(1:10, ncol=2)
> X
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
> solve(crossprod(X))
       [,1]   [,2]
[1,]  0.264 -0.104
[2,] -0.104  0.044
> det(crossprod(X))
[1] 1250
>
> DATOS1<- matrix(c(3,2,1,-1),ncol=2,by=T)
> DATOS2<- c(5,0)
> solve(DATOS1,DATOS2)
[1] 1 1
>
```

El siguiente objeto a estudiar son las listas. Los vectores y las matrices contienen variables de un solo tipo. Mediante la función **`list`** se puede construir una lista, la cual puede disponer de variables de varios tipos. Esta función es la más utilizada para devolver el resultado de una función creada por el usuario.

```
>
>
> MI.LISTA <- list(A=1:10, B=DIAS)
> MI.LISTA
$A
 [1]  1  2  3  4  5  6  7  8  9 10

$B
[1] "Lunes"     "Martes"    "Miercoles" "Jueves"    "Viernes"   "Sabado"    "Domingo"
```

Existen dos formas de referirse a los elementos de una lista. La primera consiste en utilizar el nombre de la lista precedido por el símbolo **$**. La segunda forma consiste en utilizar dobles corchetes, como se muestra a continuación

```
>
> MI.LISTA$A
 [1]  1  2  3  4  5  6  7  8  9 10
>
> MI.LISTA$B
[1] "Lunes"     "Martes"    "Miercoles" "Jueves"    "Viernes"   "Sabado"    "Domingo"
>
> MI.LISTA$B[2]
[1] "Martes"
>
> MI.LISTA[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10
>
> MI.LISTA[[2]]
[1] "Lunes"     "Martes"    "Miercoles" "Jueves"    "Viernes"   "Sabado"    "Domingo"
>
> MI.LISTA[[2]][2]
[1] "Martes"
>
```

Otras funciones comúnmente utilizadas son las se muestran en la siguiente tabla

abs	Valor absoluto
ceiling	Entero mayor o igual
cos	Coseno
dim	Dimensión de una matriz
exp	Exponencial
floor	Entero menor o igual
log10	Logaritmo decimal
log	Logaritmo neperiano
max	Máximo
mean	Media
median	Mediana
min	Mínimo
order	Vector entero que contiene la permutación que ordenaría el argumento en orden creciente
prod	Producto
range	Rango
rev	Los mismos elementos en orden inverso
sample	Generar muestras aleatorias
sin	Seno
sort	Los mismo elementos, ordenados ascendentemente
sqrt	Raíz cuadrada
summary	Resumen estadístico de un objeto
sum	Suma
tan	Tangente
trunc	Entero más próximo a cero
var	Varianza de un vector, matriz de covarianzas o correlaciones de una matriz o covarianzas entre matrices o vectores

El usuario puede crear sus propias funciones y asignarle los argumentos que desee. Una función se define asignando a un objeto la palabra **function** seguida de los argumentos que se desee dar a la función, escritos entre paréntesis y separados por comas, seguida de la orden, la cual deberá estar entre llaves si son varias ordenes. A continuación se define la función **F1**, la cual devuelve la suma de dos argumentos. Observamos que si no se utiliza ningún argumento, la función por defecto utilizará los valores 1 y 2. También observamos que esta función no está limitada al uso de números, sino que se pueden utilizar otros objetos como vectores, matrices, etc.

```
>
> F1 <- function(A=1,B=2) A+B
>
> F1()
[1] 3
> F1(5)
[1] 7
> F1(5,2)
[1] 7
> F1(B=1)
[1] 2
> F1(1:10,2)
 [1]  3  4  5  6  7  8  9 10 11 12
> F1(1:10,1:5)
 [1]  2  4  6  8 10  7  9 11 13 15
>
> M1<-matrix(1:6,nr=3)
> M1
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> F1(M1,4)
     [,1] [,2]
[1,]    5    8
[2,]    6    9
[3,]    7   10
>
```

La función **F2** que se describe a continuación devuelve más de un objeto, en concreto la suma y el producto de los dos argumentos que tiene la función. Para ello, tal como indicamos anteriormente, se utilizará la función **list**. En este caso, si sólo se desease que la función devolviese el producto de los dos argumentos, bastaría con escribir al final de la función **F2** (antes de cerrar la llave) el objeto **PROD**.

```
> F2<- function (A=1, B=2)
+ {
+ SUM <- A+B
+ PROD<- A*B
+ list(SUMA=SUM, PRODUCTO=PROD)
+ }
>
> F2()
$SUMA
[1] 3

$PRODUCTO
[1] 2

> F2(5,3)
$SUMA
[1] 8

$PRODUCTO
[1] 15

> F2(M1,4)
$SUMA
     [,1] [,2]
[1,]    5    8
[2,]    6    9
[3,]    7   10

$PRODUCTO
     [,1] [,2]
[1,]    4   16
[2,]    8   20
[3,]   12   24

>
```