

Trabajo Práctico 2: Introducción al lenguaje R- continuación

2. Operaciones

Veremos las siguientes funciones, operadores y constantes:

```
+, -, *, /, ^, %*%,  
abs(), as.matrix(), is.matrix, cbind(), rbind(), choose(), cos(),  
cumprod(), cumsum(), det(),  
diff(), dim(), eigen(), exp(), factorial(), gamma(), length(), pi,  
prod(), sin(), solve(), sort(), sqrt(), sum(), t(), tan().
```

2.1 Matemática básica.

Una de las formas más simples, pero muy útil, de utilizar R es como una calculadora poderosa.

El lenguaje R incluye los operadores aritméticos usuales.: +, -, *, /, ^.

Veamos algunos ejemplos:

```
> 2+3  
[1] 5  
> 3/2  
[1] 1.5  
> 2^3 # otra forma 2**3  
[1] 8  
> 4^2-3*2  
[1] 10  
> (56-14)/6 - 4*7*10/(5^2-5)  
[1] -7
```

Las siguientes funciones estándar que se encuentran en la mayoría de las calculadoras están disponibles en R:

Nombre	Operación
<code>sqrt(x)</code>	raíz cuadrada
<code>abs(x)</code>	valor absoluto
<code>sin(x)</code> , <code>cos(x)</code> , <code>tan(x)</code>	funciones trigonométricas (radianes) escriba ? Trig para hallar otras
<code>pi</code>	número $\pi = 3.1415926..$
<code>exp(x)</code> , <code>log(x)</code>	exponencial, logaritmo
<code>gamma(x)</code>	función gamma de Euler
<code>factorial(x)</code>	función factorial
<code>choose(n,k)</code>	función que calcula el número combinatorio

```
> sqrt(2)  
[1] 1.414214  
> abs(2-4)  
[1] 2  
> cos(4*pi)
```

```
[1] 1
> log(0) # no definido
[1] -Inf
> factorial(6) # 6!
[1] 720
> choose(52,5) # esto es 52!/(47!5!)
[1] 2598960
```

2.2 Aritmética Vectorial

Los vectores pueden ser manipulados en forma similar a los escalares utilizando las mismas funciones que presentamos en la última sección. Sin embargo, se debe ser muy cuidadoso/a al sumar o restar vectores de diferentes longitudes ya que se pueden obtener resultados inesperados. Algunos ejemplos de tales operaciones son:

```
> x <- c(1,2,3,4)
> y <- c(5,6,7,8)
> x*y
[1] 5 12 21 32
> y/x
[1] 5.000000 3.000000 2.333333 2.000000
> y-x
[1] 4 4 4 4
> x^y
[1] 1 64 2187 65536
> cos(x*pi) + cos(y*pi)
[1] -2 2 -2 2
>
```

Algunas funciones útiles que atañen vectores son:

Nombre	Operación
<code>length()</code>	devuelve la cantidad de componentes del vector
<code>sum()</code>	calcula la suma de las componentes del vector
<code>prod()</code>	calcula el producto de las componentes del vector
<code>cumsum()</code> , <code>cumprod()</code>	calcula sumas y productos acumulados
<code>sort()</code>	ordena el vector
<code>diff()</code>	calcula diferencias de vectores adecuadamente corridos (por defecto 1)

Algunos ejemplos:

```
> s <- c(1,1,3,4,7,11)
> length(s)
[1] 6
> sum(s) # 1+1+3+4+7+11
[1] 27
> prod(s) # 1*1*3*4*7*11
[1] 924
> cumsum(s)
[1] 1 2 5 9 16 27
> diff(s) # 1-1, 3-1, 4-3, 7-4, 11-7
```

```
[1] 0 2 1 3 4
> diff(s, lag = 2) # 3-1, 4-1, 7-3, 11-4
[1] 2 3 4 7
```

2.3 Operaciones Matriciales

Entre las poderosas características de R se encuentra su habilidad para realizar operaciones matriciales. Como hemos visto en el capítulo anterior se pueden crear objetos matriciales a partir de un vector numérico utilizando el comando `matrix()`:

```
> a <- c(1,2,3,4,5,6,7,8,9,10)
> A <- matrix(a, nrow = 5, ncol = 2) # entra los valores por
columna
> A
[,1] [,2]
[1,] 1 6
[2,] 2 7
[3,] 3 8
[4,] 4 9
[5,] 5 10

> B <- matrix(a, nrow = 5, ncol = 2, byrow = TRUE) # por fila
> B
[,1] [,2]
[1,] 1 2
[2,] 3 4
[3,] 5 6
[4,] 7 8
[5,] 9 10

> C <- matrix(a, nrow = 2, ncol = 5, byrow = TRUE)
> C
[,1] [,2] [,3] [,4] [,5]
[1,] 1 2 3 4 5
[2,] 6 7 8 9 10
>
```

Otra forma de generar matrices a partir de vectores es utilizando las funciones `cbind()` y `rbind()`

```
> x1<- c(1,2,3)
> x2<- c(4,5,6)
> x3<- c(0,0,0)
> XC<- cbind(x1,x2,x3)

> XC
      x1 x2 x3
[1,]  1  4  0
[2,]  2  5  0
[3,]  3  6  0
```

La función `cbind()` pega sus argumentos por columnas y en este caso, como hemos pegado vectores, obtenemos un objeto de clase `matrix`.

La función `is.matrix` determina si su argumento es un objeto de clase `matrix`.

```
> is.matrix(XC)
[1] TRUE
```

La función `rbind()` pega sus argumentos por filas. Obtenemos también en este caso un objeto de clase `matrix`.

```
> XR<- rbind(x1,x2,x3)

> XR
      [,1] [,2] [,3]
col1     1     2     3
col2     4     5     6
col3     0     0     0

> is.matrix(XR)
[1] TRUE
```

Las operaciones matriciales (multiplicación, traspuesta, etc.) pueden ser realizadas fácilmente en R utilizando funciones simples:

Nombre	Operación
<code>dim()</code>	dimensión de la matriz (cantidad de filas y columnas)
<code>as.matrix()</code>	fuerza un argumento en un objeto tipo matriz
<code>%*%</code>	multiplicación matricial
<code>t()</code>	traspuesta
<code>det()</code>	determinante de una matriz cuadrada
<code>solve()</code>	inversa de una matriz. También resuelve un sistema de ecuaciones lineales
<code>eigen()</code>	calcula autovalores y autovectores

Podemos jugar un poco al álgebra lineal con las matrices **A**, **B** y **C** que hemos creado:

```
> t(C) # esta es la misma que A!!
[,1] [,2]
[1,] 1 6
[2,] 2 7
[3,] 3 8
[4,] 4 9
[5,] 5 10

> B%*%C
[,1] [,2] [,3] [,4] [,5]
[1,] 13 16 19 22 25
[2,] 27 34 41 48 55
[3,] 41 52 63 74 85
[4,] 55 70 85 100 115
[5,] 69 88 107 126 145
> D <- C%*%B
> D
[,1] [,2]
[1,] 95 110
[2,] 220 260
> det(D)
[1] 500
```

```
> solve(D) # esto es D-1
[,1] [,2]
[1,] 0.52 -0.22
[2,] -0.44 0.19
>
```

2.4 Ejercicios

Use R para realizar los siguientes cálculos:

1. $|2^2 - 3^3|$
2. e^e
3. $(2.3)^8 + \ln(7.5) - \cos(\pi/\sqrt{2})$

4. Sean $A = \begin{bmatrix} 1 & 2 & 3 & 2 \\ 2 & 1 & 6 & 4 \\ 4 & 7 & 2 & 5 \end{bmatrix}$, $B = \begin{bmatrix} 1 & 3 & 5 & 2 \\ 0 & 1 & 3 & 4 \\ 2 & 4 & 7 & 3 \\ 1 & 5 & 1 & 2 \end{bmatrix}$. Halle los productos matriciales AB^{-1} and BA^T .

5. Halle el producto, elemento a elemento de los siguientes vectores
 - i) $[2, 5, 6, 7]$, $[-1, 3, -1, -1]$.
 - ii) $[2, 5, 6]$, $[-1, 3, -1, -1]$. ¿Qué realiza R en este caso?
6. Explique qué obtiene al sumar un vector y un escalar.

3. Cargando datos en R

Veremos las siguientes funciones y operadores: `$`, `:`, `attach()`, `attributes()`, `data()`, `data.frame()`, `edit()`, `file.choose()`, `fix()`, `read.table()`, `rep()`, `scan()`, `search()`, `seq()`, `names()`, `row.names()`

Hemos visto cómo la función concatenar `c()` en R puede construir un vector de valores numéricos. Esta función también permite construir vectores de valores de texto:

```
> mishijos <- c("Esteban", "Christian") # el texto entre comillas
> mishijos
[1] "Esteban" "Christian"
```

Veremos que hay muchas otras formas de crear en R vectores y conjuntos de datos.

3.1 Sucesiones

Algunas veces necesitamos construir una cadena de caracteres que tiene un patrón regular. En vez de escribir la secuencia podemos definir el patrón utilizando algunos operadores y funciones especiales.

- El operador dos puntos :

El operador dos puntos crea un vector de números con un incremento de una unidad (entre dos números especificados):

```
> 1:9
[1] 1 2 3 4 5 6 7 8 9
> 1.5:10 # aquí no llegaremos al 10
[1] 1.5 2.5 3.5 4.5 5.5 6.5 7.5 8.5 9.5
> c(1.5:10,10) # podemos agregar el 10 al final
[1] 1.5 2.5 3.5 4.5 5.5 6.5 7.5 8.5 9.5 10.0
> prod(1:8) # es igual a factorial(8)
[1] 40320
```

- La función secuencia `seq()`

La función secuencia permite crear una cadena de valores con cualquier incremento que se desee. Se puede especificar el *valor incremental* o la *longitud* deseada:

```
> seq(1,5) # es igual a 1:5
[1] 1 2 3 4 5
> seq(1,5,by=.5) # incrementa por 0.5
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0

> seq(1,5,length=7) # calcula el incremento para obtener esta
longitud del vector
[1] 1.00000 1.66667 2.33333 3.00000 3.66667 4.33333 5.00000
```

- La función replicar `rep()`

La función replicar permite repetir un valor o una secuencia de valores una cantidad especificada de veces:

```
> rep(10,10) # repite el valor 10 diez veces
[1] 10 10 10 10 10 10 10 10 10 10
> rep(c("A","B","C","D"),2) # repite la cadena A,B,C,D dos veces
[1] "A" "B" "C" "D" "A" "B" "C" "D"
> matrix(rep(0,16),nrow=4) # una matriz de ceros de 4x4

[,1] [,2] [,3] [,4]
[1,] 0 0 0 0
[2,] 0 0 0 0
[3,] 0 0 0 0
[4,] 0 0 0 0
>
```

3.2 Lectura de datos: Vectores

La función `c()` es incómoda para entrar una cantidad grande de datos. Alternativamente, los datos pueden ingresarse mediante el teclado utilizando la función `scan()`. Es más útil ya que los datos se ingresan separados únicamente mediante un espacio en blanco, aunque esto puede modificarse en los argumentos de la función. Por defecto la función espera valores *numéricos*, pero se pueden especificar otros usando la opción “*what =*” option. La sintaxis básica es:

```
> x <- scan() # guardar lo que se escribe en la variable x
```

Cuando se escribe lo anterior en una sesión de R, aparece un prompt que significa que R espera que se escriban valores que se entrarán al vector **x**. El prompt está formado por un número y dos puntos, es el valor del índice en el vector que está esperando para recibir los datos. R dejará de agregar datos cuando se entre una línea en blanco.

Después de entrar la línea en blanco, R indicará la cantidad de valores ingresados.

Supongamos que contamos la cantidad de pasajeros, excluido el conductor, en los próximos 30 automóviles que pasan por una esquina:

```
> pasajeros <- scan()
1: 2 4 0 1 1 2 3 1 0 0 3 2 1 2 1 0 2 1 1 2 0 0 # de enter
23: 1 3 2 2 3 1 0 3 # de enter nuevamente
31: # de enter una última vez
Read 30 items
> pasajeros # muestra los valores ingresados
[1] 2 4 0 1 1 2 3 1 0 0 3 2 1 2 1 0 2 1 1 2 0 0 1 3 2 2 3 1 0 3
```

También es posible ingresar datos a un vector a partir de datos guardados en un archivo (tipo texto ASCII) utilizando la función **scan()**. Para ello simplemente se pasa el nombre del archivo entre comillas como argumento de la función **scan()**. Por ejemplo, supongamos que los datos anteriores se encontraban grabados originalmente en el archivo **pasajeros.txt** ubicado en un disco duro F:. Para leer estos datos, simplemente se escribe

```
> pasajeros <- scan("F:/ pasajeros.txt")
Read 30 items
>
```

Notas:

- SIEMPRE mire el archivo de texto antes de ingresarlo a R para asegurarse que es lo que interesa ingresar y tiene el formato adecuado.
- Recordemos que para representar directorios o subdirectorios, se utiliza la barra de dividir (/), no la barra hacia atrás (\) en el path del nombre de un archivo.
- Si la computadora está conectada a internet los datos guardados en un archivo texto pueden leerse de una URL utilizando la función **scan()** function. La sintaxis básica está dada por: **> dat <- scan("http://www...")**
- Si el nombre del directorio o del archivo tiene espacios hay que tener cuidado con el caracter espacio. Esto se realiza incluyendo una barra hacia atrás (\) antes de tipear el espacio.
- Otra opción es utilizar la función **file.choose()** en el lugar del nombre del archivo. Al hacerlo se abre una ventana tipo explorador y el archivo puede seleccionarse interactivamente.

```
> pasajeros <- scan(file.choose())
Read 30 items
>
```

Más sobre lectura de datos en la próxima sección.

3.3 Data Frames

3.3.1 Creando Marcos de Datos (Data Frames)

Un conjunto de datos en estadística generalmente contendrá el registro de más de una variable de un experimento.

Por ejemplo, en el experimento de los automóviles de la sección anterior se podrían haber registrado otras variables como el *tipo de automóvil* (auto, camioneta, 4x4, etc.) y el *uso del cinturón de seguridad* (S, N). Un *data frame* es el mejor tipo de objeto de R para guardar este tipo de datos. Las variables se incluyen como columnas con nombres que las identifican. Todas las columnas deben tener la misma longitud.

Observación: En un *data frame* cada columna es un vector, y éstos pueden ser de distinto tipo: numérico, entero, carácter o lógico.

Los datos pueden ingresarse directamente en el data frame, mediante un editor. Esta es una forma interactiva de entrada de datos que se parece a una hoja de cálculo.

Se puede acceder al editor utilizando los comandos `edit()` o `fix()`:

```
> datos.nuevos <- data.frame() # crea data frame vacío
> datos.nuevos <- edit(datos.nuevos) # se pide que los cambios se
                                     # escriban en el data frame

0

> datos.nuevos <- data.frame() # crea data frame vacío
> fix(datos.nuevos) # guarda los cambios automáticamente
```

El editor de datos permite agregar todas las variables (columnas) que se desee. Los nombres de las columnas pueden cambiarse de los valores por defecto `var1`, `var2`, etc. clickeando en el encabezado de la columna. En este punto el tipo de variable (numérica ó carácter) puede ser especificado. Al cerrar el editor el marco (frame) es salvado.

También se pueden crear data frames a partir de variables preexistentes en el espacio de trabajo. Supongamos que en el último experimento se registró también el uso del cinturón de seguridad por el conductor: "S" si el conductor usaba el cinturón y "N" si no lo usaba. Estos datos se ingresan mediante la función `c` (recordar que estos datos son texto, deben ir entre comillas):

```
> cinturon <- c("S","N","S","S","S","S","S","S","S","S", # enter
+ "N","S","S","S","S","S","S","S","S","S", # enter
+ "S","S","N","S","S","S","S")
>
```

Podemos combinar estos datos en un único data frame

```
> datos.autos <- data.frame(pasajeros,cinturon)
```


Al mirarlo, un data frame tiene el aspecto de una matriz:

```
> datos.autos
  pasajeros cinturón
1          2      S
2          4      N
3          0      S
4          1      S
5          1      S
6          2      S
7          3      S
8          1      S
. . . . .
```

Los valores de la izquierda son simplemente los nombres de las filas. La función `summary()` de un data.frame devuelve un objeto de clase tabla que contiene el mínimo, el máximo, la media, la mediana y los cuartiles para las variables numéricas y la cantidad de datos en cada clase para las categóricas:

```
> summary(datos.autos)
  pasajeros      cinturón
Min.   :0.000    N: 3
1st Qu.:1.000    S:27
Median :1.000
Mean   :1.467
3rd Qu.:2.000
Max.   :4.000

> class(summary(datos.autos))
[1] "table"
```

La función `names()` nos da los nombres de las variables (columnas) del data frame:

```
> names(datos.autos)
[1] "pasajeros" "cinturón"
```

Podemos acceder al nombre de las filas:

```
> row.names(datos.autos)
[1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11" "12"
[13] "13" "14" "15"
[16] "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27"
[28] "28" "29" "30"
```

y también cambiarle el nombre, lo haremos sólo para la primera:

```
> row.names(datos.autos)[1]<- "nombre prueba"
> row.names(datos.autos)
[1] "nombre prueba" "2"          "3"          "4"
[5] "5"             "6"          "7"          "8"
[9] "9"             "10"         "11"         "12"
[13] "13"            "14"         "15"         "16"
[17] "17"            "18"         "19"         "20"
[21] "21"            "22"         "23"         "24"
[25] "25"            "26"         "27"         "28"
[29] "29"            "30"
```

Observación: *R* reconoce acentos en los nombres de sus objetos! Construyamos la nueva variable **cinturón** (con acento)

```
> cinturón <- c(" sí tiene", "no tiene", "no tiene")
> cinturón
[1] " sí tiene" "no tiene"  "no tiene"
```

3.3.2 Conjuntos de Datos Incluidos con *R*

R contiene muchos conjuntos de datos incorporados al software. Estos datos están guardados como data frames. Para ver la lista de conjuntos de datos tipee

```
> data()
```

Se abrirá una ventana en la que se listan los conjuntos de datos disponibles (hay muchos otros en los paquetes escritos por usuarios). Para saber algo más sobre algún conjunto de datos disponible tipee

```
> help(nombre)
```

Para abrir un conjunto de datos (dataset), por ejemplo **trees**, simplemente se escribe

```
> data(trees)
```

Luego el data frame **trees** se encuentra en su espacio de trabajo.

3.3.3 Acceso a partes de un conjunto de datos

Se puede acceder a variables individuales dentro de un data frame utilizando el argumento **\$**. Por ejemplo, vemos que el dataset **trees** tiene tres variables:

```
> trees
  Girth Height Volume
1   8.3    70   10.3
2   8.6    65   10.3
3   8.8    63   10.2
4  10.5    72   16.4
5  10.7    81   18.8
6  10.8    83   19.7
. . . . .
```

Para acceder a las variables individuales en un data frame, se usa un **\$** entre el nombre del data frame y el nombre de la columna:

```
> trees$Height
[1] 70 65 63 72 81 83 66 75 80 75 79 76 76 69 75 74 85 86 71 64 78
[22] 80 74 72 77 81 82 80 80 80 87
> sum(trees$Height)           # suma estos valores
[1] 2356
>
```

También se puede acceder a un elemento individual o a una fila especificando su posición (en un formato fila columna) entre corchetes a continuación del nombre del data frame:

```
> trees[4,3]          # entrada en la fila cuatro y tercera
columna
[1] 16.4
> trees[4,]           # selecciona la fila
  Girth Height Volume
4  10.5     72   16.4
>
```

Habitualmente queremos acceder a las variables en un data frame pero utilizar el argumento `$` puede tornarse incómodo. Afortunadamente, se puede lograr que R encuentre variables que están dentro de un data frame agregándolo a su camino de búsqueda (*search path*). Para incluir las variables en el data frame `trees` en el search path, tipee

```
> attach(trees)
```

Ahora las variables en el data frame `trees` están accesibles sin la notación de `$`:

```
> Height
[1] 70 65 63 72 81 83 66 75 80 75 79 76 76 69 75 74 85 86 71 64
[21] 78 80 74 72 77 81 82 80 80 80 87
```

Para ver exactamente que está ocurriendo visualicemos el camino de búsqueda utilizando el comando `search()`:

```
> search()
[1] ".GlobalEnv"          "trees"              "package:methods"
[4] "package:stats"        "package:graphics"   "package:grDevices"
[7] "package:utils"        "package:datasets"   "Autoloads"
[10] "package:base"
```

Note que el data frame `trees` está colocado en el segundo lugar en el search path. Este es el orden en que R busca los objetos cuando se escriben los comandos, `.GlobalEnv` es su directorio de trabajo y los `package` son bibliotecas (libraries) que contienen, entre otras cosas, las funciones y los conjuntos de datos que estamos viendo en estas guías.

Para remover un objeto del search path, use el comando `detach()` de la misma manera que ha utilizado `attach()`. Note, sin embargo, que cualquier objeto agregado al search path es eliminado de él de cualquier manera al salir de R.

Para listar las características de cualquier objeto en R, ya sea un vector, data frame, etc. use la función `attributes()`. Por ejemplo:

```
> attributes(trees)
$names
```

```
[1] "Girth" "Height" "Volume"
$row.names
[1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11"
[12] "12" "13" "14" "15" "16" "17" "18" "19" "20" "21" "22"
[23] "23" "24" "25" "26" "27" "28" "29" "30" "31"
$class
[1] "data.frame"
>
```

Aquí vemos que el objeto **trees** es un data frame con 31 filas y cuyas variables tienen nombres correspondientes a las mediciones realizadas sobre cada árbol.

3.3.4 Lectura de datos externos

La forma más fácil de ingresar los datos en R es haciéndolo primero en un archivo excel y exportándolo como archivo texto (por ejemplo, delimitado por tabulaciones - tab delimited) Puede leerse mediante la función **read.table()**, que carga el conjunto de datos en un data frame. Si la primera línea del archivo texto contiene los nombres de las variables (que es lo habitual), R puede tomar esos nombres como los nombres de las variables en el data frame. Esto es especificado con la opción **header = T** al llamar a la función. Si el archivo no tiene una fila de encabezados se puede ignorar esta opción y R utilizará los nombres por defecto de un data frame. Los nombres de los archivos se especifican de igual forma que en la función **scan()** o se utiliza la función **file.choose()** para seleccionar el archivo en forma interactiva. Por ejemplo, mediante las instrucciones:

```
> pepe <- read.table(file.choose(), header=T)
```

se crea un data frame con el nombre **pepe** con los datos de un archivo de texto especificado por el usuario que tiene en la primera línea los nombres de las variables.

3.3.5 Opciones más avanzadas de importación de datos

Desde el Excel, exporte los datos en formato texto *delimitado por tabulaciones* (tab delimited) o formato separado por comas y utilice las funciones **read.delim** o **read.csv** para importar los datos a R.

También se puede utilizar **odbcConnectExcel** en el paquete **RODBC**. Esto permite seleccionar filas y columnas de cualquiera de las hojas de una planilla Excel.

3.3.6 Exportar Datos

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

x: el nombre del objeto a ser escrito, preferiblemente una matriz o un data frame.

file: una cadena de caracteres con el nombre de un archivo. " " indica que la salida es a la consola

3.4 Listas

Las listas son estructuras muy flexibles de R. Permiten juntar información de diferentes tipos de estructuras (matrices, vectores, funciones, listas, etc).

```
> primeraLista <- list(c(1, 2, 3), c("d", "e", "f", "g"), mean,  
matrix(rep(0,16),nrow=4))
```

La lista creada tiene 4 componentes indexados por dobles corchetes: `[[1]]`, `[[2]]`, `[[3]]` y `[[4]]`

```
> primeraLista[[2]]      # Muestra el segundo objeto de la lista  
[1] "d" "e" "f" "g"  
  
> primeraLista          # Muestra todos los objetos de la lista  
[[1]]  
[1] 1 2 3  
  
[[2]]  
[1] "d" "e" "f" "g"  
  
[[3]]  
function (x, ...)   
UseMethod("mean")  
<environment: namespace:base>  
  
[[4]]  
      [,1] [,2] [,3] [,4]  
[1,]    0    0    0    0  
[2,]    0    0    0    0  
[3,]    0    0    0    0  
[4,]    0    0    0    0  
  
>
```

Asignamos nombres a las componentes de **primeraLista**.

```
> names(primeraLista) <- c("números", "letras", "función",  
"matriz")  
> primeraLista  
$números  
[1] 1 2 3  
  
$letras  
[1] "d" "e" "f" "g"  
  
$función  
function (x, ...)   
UseMethod("mean")  
<environment: namespace:base>  
  
$matriz  
      [,1] [,2] [,3] [,4]
```

```
[1,] 0 0 0 0  
[2,] 0 0 0 0  
[3,] 0 0 0 0  
[4,] 0 0 0 0
```

Podemos acceder a las componentes de la lista por su nombre

```
> primeraLista$letras  
[1] "d" "e" "f" "g"
```

3.5 Ejercicios

1. Genere las siguientes secuencias en R:
 - a. 1 2 3 1 2 3 1 2 3 1 2 3
 - b. 10.00000 10.04545 10.09091 10.13636 10.18182 10.22727
10.27273 10.31818 10.36364 10.40909 10.45455 10.50000
 - c. "1" "2" "3" "banana" "1" "2" "3" "banana"
2. Cree un data frame llamado **cronograma** con las variables:
nombre, días, cantidad de horas semanales
3. Genere un archivo de datos utilizando una planilla de cálculo que contenga las variables: **edad, género, cantidad de materias aprobadas** de un curso hipotético con 15 alumnos. Incluya el nombre de las variables en la primera fila. Guárdelo en formato texto (por ejemplo, delimitado por tabulaciones) e impórtelo a un data frame del R.
4. Guarde en un data frame llamado **tempacid** las variables **Water.Temp** y **Acid.Conc.** del dataset **stackloss** de R.