USO DE HERRAMIENTAS INFORMÁTICAS PARA LA RECOPILACIÓN, ANÁLISIS E INTERPRETACIÓN DE DATOS DE INTERÉS EN LAS CIENCIAS BIOMÉDICAS

Módulo 5
Programación en entorno R

Alfredo Rigalli
Maela Lupo
María Eugenia Chulibert
Mercedes Lombarte
Patricia Lupión

Centro Universitario de Estudios Medioambientales
Facultad de Ciencias Médicas
Universidad Nacional de Rosario

2020

ISBN 978-987-86-3522-4

USO DE HERRAMIENTAS INFORMÁTICAS PARA LA RECOPILACIÓN, ANÁLISIS E INTERPRETACIÓN DE DATOS DE INTERÉS EN LAS CIENCIAS BIOMÉDICAS

Programación en entorno R

Alfredo Rigalli
Maela Lupo
María Eugenia Chulibert
Mercedes Lombarte
Patricia Lupión

Centro Universitario de Estudios Medioambientales
Facultad de Ciencias Médicas
Universidad Nacional de Rosario

Uso de herramientas informáticas para la recopilación, análisis e interpretación de datos de interés en las ciencias biomédicas : programación en entorno R / Alfredo Rigalli ... [et al.]. - 1a edición para el alumno - Rosario : Alfredo Rigalli, 2020.
Libro digital, PDF

Archivo Digital: descarga
ISBN 978-987-86-3522-4

1. Lenguaje de Programación. I. Rigalli, Alfredo.
CDD 005.4

AUTORES

Chulibert, María Eugenia: Licenciada en nutrición. Estudiante del doctorado en Ciencias Biomédicas de la Facultad de Ciencias Médicas de la Universidad Nacional de Rosario. Becaria doctoral del CONICET.

Lombarte, Mercedes: Licenciada en biotecnología y Doctora en Ciencias Biomédicas. Investigadora del Laboratorio de Biología Ósea y del Centro Universitario de Estudios Medioambientales y docente de la cátedra de Química Biológica de la Facultad de Ciencias Médicas de la Universidad Nacional de Rosario.

Lupión, Patricia: Licenciada en biotecnología. Estudiante del doctorado en Ciencias Biomédicas de la Facultad de Ciencias Médicas de la Universidad Nacional de Rosario. Becaria doctoral del CONICET.

Lupo, Maela: Licenciada en biotecnología y Doctora en Ciencias Biomédicas. Investigadora del Laboratorio de Biología Ósea y del Centro Universitario de Estudios Medioambientales y docente de la cátedra de Química Biológica de la Facultad de Ciencias Médicas de la Universidad Nacional de Rosario.

Rigalli, Alfredo: Bioquímico y Doctor en bioquímica. Investigadora del Laboratorio de Biología Ósea y del Centro Universitario de Estudios Medioambientales y docente de la cátedra de Química Biológica de la Facultad de Ciencias Médicas de la Universidad Nacional de Rosario. Investigador independiente del Consejo de Investigaciones de la UNR y del CONICET

Tabla de contenidos

Interpretación de este manual

Mostraremos el script en una tabla de dos columnas.

Cuando en la tabla haya un solo script a la derecha en las líneas correspondientes se harán explicaciones de los comandos.

Cuando se hagan modificaciones a un script se mostrarán en la segunda columna las líneas agregadas en color rojo.

Las líneas que no han cambiado se dejarán una a la par de otra en color negro.

Las líneas eliminadas se colocarán en **azul.**

Recuerde que para crear un script debe copiar o escribir la secuencia de ordenes en un procesador de texto. Luego grabarlo con extensión .txt (documento de texto sin formato) en el mismo directorio en que se halla su espacio de trabajo.

1. Clase 1

1.1. Scripts

Un script es un conjunto de instrucciones que pueden ser ejecutadas con un solo comando. Estas instrucciones son definidas por el usuario de acuerdo a las necesidades de trabajo y tienen grandes ventajas:

- Ganancia de tiempo, especialmente si las instrucciones a ejecutar se realizarán de manera rutinaria.
- Menor posibilidad de cometer errores, ya que una vez revisada y probada la efectividad de las instrucciones, estas serán ejecutadas siempre de la misma manera.
- Automatización de tareas complejas que podrán ser ejecutadas por personas que desconozcan parte de las mismas.
- Ganancia de tiempo, al repetir y ejecutar órdenes en intervalos de tiempo breves.
- Uso de herramientas de R, sin necesidad que el usuario requiera conocer las unciones de R para aplicar dichas herramientas.

En R los scripts se escriben en un editor de texto y se guardan con extensión .txt. Cualquier editor de textos puede ser utilizado, aunque existen algunos que se adaptan a scripts de R, permitiendo diferenciar en colores distintos los objetos, las funciones y los comentarios.

Advertencia: si nunca a realizado tareas de programación puede ser que el primer contacto con los script le resulte dificultoso. La recomendación es avanzar lentamente leyendo el código del script y analizando lo que se observa en la pantalla luego de la ejecución. Rápidamente con el pasaje de los días irá rápidamente entendiendo el tema.

1.2. Primeros pasos

Supongamos que deseamos introducir los datos de la tablaR511 de la planilla de cálculo tablaR5-1.ods/xls. Por lo aprendido anteriormente marcaríamos los datos en la planilla de cálculo y en el espacio de trabajo ejecutaríamos

```
> tablaR511<-read.table("clipboard",header=TRUE,sep="\t",dec=",",encoding="latin1")
```

si quisiéramos verla

```
> tablaR511
   A   B
1 2.1 2.1
2 2.3 5.1
3 2.8 6.1
4 2.9 5.1
5 3.1 5.2
6 4.0 5.0
```

Luego para ver los datos ejecutamos

```
> summary(tablaR511)
       A               B        
 Min.   :2.100   Min.   :2.100  
 1st Qu.:2.425   1st Qu.:5.025  
 Median :2.850   Median :5.100  
 Mean   :2.867   Mean   :4.767  
 3rd Qu.:3.050   3rd Qu.:5.175  
 Max.   :4.000   Max.   :6.100  
```

Si quisiéramos tener una suma de la columna A ejecutaríamos

```
> sum(tablaR511$A)
[1] 17.2
```

hagamos un test de normalidad con la función shapiro.test () para los datos de ambas columnas

```
> shapiro.test(tablaR511$A)

        Shapiro-Wilk normality test

data:  tablaR511$A
W = 0.93956, p-value = 0.6557
```

```
> shapiro.test(tablaR511$B)

        Shapiro-Wilk normality test

data:  tablaR511$B
W = 0.7384, p-value = 0.0153
```

como p-value de los datos de columna B no tienen distribución normal, si A y B fueran datos independientes y quisiéramos comparar las muestras deberíamos hacer un test de Mann Whitney con la función wilcox.test().

```
> wilcox.test(tablaR511$A,tablaR511$B)

        Wilcoxon rank sum test with continuity correction

data:  tablaR511$A and tablaR511$B
W = 5.5, p-value = 0.05382
alternative hypothesis: true location shift is not equal to 0

Warning message:
In wilcox.test.default(tablaR511$A, tablaR511$B) :
```

cannot compute exact p-value with ties

Muy bien. Si repasamos solo las instrucciones ejecutadas tendríamos, las mismas quedarían resumidas como se muestra a continuación

```
> tablaR511<-read.table("clipboard",header=TRUE,sep="\t",dec=",",encoding="latin1")
> tablaR511
> summary(tablaR511)
> sum(tablaR511$A)
> shapiro.test(tablaR511$A)
> shapiro.test(tablaR511$B)
> wilcox.test(tablaR511$A,tablaR511$B)
```

Si todas estas instrucciones las copiamos en un editor de texto tendremos un script que ejecutará automáticamente dichas instrucciones, con algunas limitaciones que iremos solucionando.
Entonces luego de copiar dichas instrucciones en un editor de texto y eliminar los ">" tendremos un script, al que guardamos en el mismo directorio en que estamos ejecutando nuestro espacio de trabajo. En este caso lo guardaremos con el nombre: ordenes1.txt.
En adelante mostraremos los scripts en una tabla con dos columnas. La segunda columna será utilizada cuando comparemos dos scripts.

script original: ordenes1.txt	
#script claseR5-1 tablaR511<- read.table("clipboard",header=TRUE,sep="\ t",dec=",",encoding="latin1") tablaR511 summary(tablaR511) sum(tablaR511$A) shapiro.test(tablaR511$A) shapiro.test(tablaR511$B) wilcox.test(tablaR511$A,tablaR511$B)	

Vayamos ahora a nuestro espacio de trabajo y ejecutemos el script. Primeramente marcaremos los datos de la tablaR511 que deseamos introducir en el espacio de trabajo y la copiamos en el portapapeles, luego ejecutamos el siguiente comando

```
> source("ordenes1")
Warning message:
In wilcox.test.default(tablaR511$A, tablaR511$B) :
  cannot compute exact p-value with ties
```

El Warning message que aparece luego de ejecutar la funcion source() es el mismo apareció cuando ejecutó el último paso, al haber ejercutado la secuencia orden por orden. Que nos está diciendo esto? Que se han ejecutado todas la ordenes, salvo que R no nos muestra nada en la pantalla. No por un error de R, sino porque nosotros no se lo hemos pedido.

1.3. Mostrar pasos y datos intermedios en la consola

Iremos modificando el script y viendo su salida.

1.3.1. La función "print"

print() nos permite ver en pantalla un paso intermedio, ya sea dato o salida de una función. Para comprenderla modifiquemos nuestro script, donde la modificación la resaltaremos en color rojo (solo para distinguirla).
Iremos mostrando un script original a la izquierda y el mismo modificado a la derecha. En la primer fila figura el nombre con el que grabamos el script. Copie la columna de la izquierda, péguela en un procesador de texto y grabe el archivo con el nombre ordenes1.txt. Luego copie la columna de la derecha y péguela en otro documento y grábela con el nombre ordenes2.text

#script original: ordenes1.txt	#script modificado: ordenes2.txt
#script claseR5-1	#script claseR5-1
tablaR511<- read.table("clipboard",header=TRUE,sep="\ t",dec=",",encoding="latin1")	tablaR511<- read.table("clipboard",header=TRUE,sep="\ t",dec=",",encoding="latin1")
tablaR511	print(tablaR511)
summary(tablaR511)	summary(tablaR511)
sum(tablaR511$A)	sum(tablaR511$A)
shapiro.test(tablaR511$A)	shapiro.test(tablaR511$A)
shapiro.test(tablaR511$B)	shapiro.test(tablaR511$B)
wilcox.test(tablaR511$A,tablaR511$B)	wilcox.test(tablaR511$A,tablaR511$B)
	print("usted ha ejecutado su primer script")

Los archivos ordenes1.txt y ordenes2.text, los tiene que haber guardado en el mismo directorio donde se halla su espacio de trabajo.

Ejecutemos el script ordenes2.txt. El comando puede ser source("ordenes2") o source("ordenes2.txt"), dependiendo de las versiones de R o el sistema operativo utilizado. Recuerde que este script introduce datos de una planilla de cálculo, por lo tanto antes de ejecutar la función source() debe marcar los datos de tablaR511 y copiarla al portapapeles

> source("ordenes2")

Luego de ejecutar la línea anterior obtendremos la siguiente salida en la consola

```
  A   B
1 2.1 2.1
2 2.3 5.1
```

3 2.8 6.1

4 2.9 5.1

5 3.1 5.2

6 4.0 5.0

[1] "usted ha ejecutado su primer script"

Warning message:

In wilcox.test.default(tablaR511$A, tablaR511$B) :

cannot compute exact p-value with ties

Algunos comentarios:
1- en la primer línea del script ordenes2.txt, observa #script claseR5-1. Cuando una línea o parte de ella es antecedida por el símbolo #, lo que sigue al símbolo es interpretado como un comentario y no será ejecutado por el script. Por lo tanto el símbolo # puede ser utilizado para introducir comentarios en el script, que luego nos servirán para interpretar que quísimos hacer con el código.
2- se ha utilizado la función print() de dos maneras
2.1- print(tablaR511): esta función nos mostrará la tablaR511.
2.2- print("Usted ha ejecutado su primer script"): mostrará en la consola la frase: **Usted ha ejecutado su primer script**

Supongamos que deseamos ver en la pantalla también los valores del summary() y los shapiro.test().
Modificamos el script ordenes2.txt, generando ordenes3.txt, como se muestra en la tabla siguiente.

#script original: ordenes2.txt	# script modificado: ordenes3.txt
#script claseR5-1	#script claseR5-1
tablaR511<- read.table("clipboard",header=TRUE,sep="\ t",dec=",",encoding="latin1")	tablaR511<- read.table("clipboard",header=TRUE,sep="\ t",dec=",",encoding="latin1")
print(tablaR511)	print(tablaR511)
summary(tablaR511)	print(summary(tablaR511))
sum(tablaR511$A)	sum(tablaR511$A)
shapiro.test(tablaR511$A)	print(shapiro.test(tablaR511$A))
shapiro.test(tablaR511$B)	print(shapiro.test(tablaR511$B))
wilcox.test(tablaR511$A,tablaR511$B)	wilcox.test(tablaR511$A,tablaR511$B)
print("usted ha ejecutado su primer script")	print("usted ha ejecutado su primer script")

Ejecutamos el script ordenes3.txt. Recuerde que el script introduce datos de la tablaR511, por lo que antes de ejecutar el comando debe copiar los datos de la tabla al portapapeles.

```
> source("ordenes3")
   A   B
1 2.1 2.1
2 2.3 5.1
3 2.8 6.1
4 2.9 5.1
```

```
5 3.1 5.2
6 4.0 5.0
       A               B
 Min.   :2.100   Min.   :2.100
 1st Qu.:2.425   1st Qu.:5.025
 Median :2.850   Median :5.100
 Mean   :2.867   Mean   :4.767
 3rd Qu.:3.050   3rd Qu.:5.175
 Max.   :4.000   Max.   :6.100

            Shapiro-Wilk normality test

data:  tablaR511$A
W = 0.93956, p-value = 0.6557

            Shapiro-Wilk normality test

data:  tablaR511$B
W = 0.7384, p-value = 0.0153

[1] "usted ha ejecutado su primer script"
Warning message:
In wilcox.test.default(tablaR511$A, tablaR511$B) :
  cannot compute exact p-value with ties
```

Vemos que R nos mostró la tabla, el summary y los shapiro.test, pero nos está faltando títulos que nos den orden y detalles de que hemos obtenido. Hagamos algunas modificaciones con la función print()

#script original: ordenes3.txt	#script modificado: ordenes4.txt
#script claseR5-1	#script claseR5-1
tablaR511<- read.table("clipboard",header=TRUE,sep="\ t",dec=",",encoding="latin1")	tablaR511<- read.table("clipboard",header=TRUE,sep="\ t",dec=",",encoding="latin1") print("DATOS TablaR511")
print(tablaR511)	print(tablaR511)
	print("Summary DATOS TablaR511")
print(summary(tablaR511))	print(summary(tablaR511))
sum(tablaR511$A)	sum(tablaR511$A)
	print("TEST DE NORMALIDAD PARA DATOS A Y B")
print(shapiro.test(tablaR511$A))	print(shapiro.test(tablaR511$A))
print(shapiro.test(tablaR511$B))	print(shapiro.test(tablaR511$B))
wilcox.test(tablaR511$A,tablaR511$B)	wilcox.test(tablaR511$A,tablaR511$B)
print("usted ha ejecutado su primer script")	print("usted ha ejecutado su primer script")

Cuando ejecutemos ordenes4.txt obtendremos

> source("ordenes4")

[1] "DATOS TablaR511"

```
   A   B
1 2.1 2.1
2 2.3 5.1
3 2.8 6.1
4 2.9 5.1
5 3.1 5.2
6 4.0 5.0
[1] "Summary DATOS TablaR511"
       A                B
 Min.   :2.100     Min.   :2.100
 1st Qu.:2.425     1st Qu.:5.025
 Median :2.850     Median :5.100
 Mean   :2.867     Mean   :4.767
 3rd Qu.:3.050     3rd Qu.:5.175
 Max.   :4.000     Max.   :6.100
[1] "TEST DE NORMALIDAD PARA DATOS A Y B"
        Shapiro-Wilk normality test
data:  tablaR511$A
W = 0.93956, p-value = 0.6557
        Shapiro-Wilk normality test
data:  tablaR511$B
W = 0.7384, p-value = 0.0153

[1] "usted ha ejecutado su primer script"
Warning message:
In wilcox.test.default(tablaR511$A, tablaR511$B) :
  cannot compute exact p-value with ties
```

Veamos ahora como introducir datos sin utilizar "clipboard", sino el nombre del archivo que contiene los datos. Para ello grabe la tablaR511 con formato .csv, utilizando codificación:

Unicode, separador de campos {tab} y delimitador de texto ". En caso que no lo pueda lograr, el archivo será enviado junto con la clase.

#script original: ordenes4.txt	#script modificado: ordenes5.txt
#script claseR5-1	#script claseR5-1
tablaR511<- read.table("clipboard",header=TRUE,sep="\ t",dec=",",encoding="latin1")	tablaR511<- read.table("tablaR511.csv",header=TRUE,sep ="\t",fileEncoding ="Unicode", dec=",")
	print("Los datos fueron importados exitosamente")
print("DATOS TablaR511") print(tablaR511)	print("DATOS TablaR511") print(tablaR511)
print("Summary DATOS TablaR511") print(summary(tablaR511))	print("Summary DATOS TablaR511") print(summary(tablaR511))
sum(tablaR511$A)	sum(tablaR511$A)
print("TEST DE NORMALIDAD PARA DATOS A Y B") print(shapiro.test(tablaR511$A))	print("TEST DE NORMALIDAD PARA DATOS A Y B") print(shapiro.test(tablaR511$A))
print(shapiro.test(tablaR511$B))	print(shapiro.test(tablaR511$B))
wilcox.test(tablaR511$A,tablaR511$B)	wilcox.test(tablaR511$A,tablaR511$B)
print("usted ha ejecutado su primer script")	print("usted ha ejecutado su primer script")

al ejecutar ordenes5.txt obtendrá, sin necesidad de marcar los datos en la planilla de cálculo, la siguiente ejecución de órdenes.

```
> source("ordenes5")
[1] "Los datos fueron importados exitosamente"
[1] "DATOS TablaR511"
   A   B
1 2.1 2.1
2 2.3 5.1
3 2.8 6.1
4 2.9 5.1
5 3.1 5.2
6 4.0 5.0
[1] "Summary DATOS TablaR511"
       A               B
 Min.   :2.100   Min.   :2.100
 1st Qu.:2.425   1st Qu.:5.025
 Median :2.850   Median :5.100
 Mean   :2.867   Mean   :4.767
 3rd Qu.:3.050   3rd Qu.:5.175
 Max.   :4.000   Max.   :6.100
[1] "TEST DE NORMALIDAD PARA DATOS A Y B"

        Shapiro-Wilk normality test

data:  tablaR511$A
W = 0.93956, p-value = 0.6557


        Shapiro-Wilk normality test

data:  tablaR511$B
```

W = 0.7384, p-value = 0.0153

[1] "usted ha ejecutado su primer script"

Warning message:

In wilcox.test.default(tablaR511$A, tablaR511$B) :

cannot compute exact p-value with ties

1.3.2. La función readline

readline: le permite detener el script y tomar una decisión. Si oprime <enter> continuará el scrip, si oprime <control + c> se detendrá

#script original: ordenes5.txt	#script modificado: ordenes6.txt
#script claseR5-1	#script claseR5-1
	readline("comenzará su script, si desea continuar oprima <enter> en caso contrario oprima <control + c>")
tablaR511<- read.table("tablaR511.csv",header=TRUE,sep="\t",fileEncoding ="Unicode", dec=",")	tablaR511<- read.table("tablaR511.csv",header=TRUE,sep="\t",fileEncoding ="Unicode", dec=",")
print("Los datos fueron importados exitosamente")	print("Los datos fueron importados exitosamente")
print("DATOS TablaR511") print(tablaR511)	print("DATOS TablaR511") print(tablaR511)
print("Summary DATOS TablaR511") print(summary(tablaR511))	print("Summary DATOS TablaR511") print(summary(tablaR511))

sum(tablaR511$A) print("TEST DE NORMALIDAD PARA DATOS A Y B") print(shapiro.test(tablaR511$A)) print(shapiro.test(tablaR511$B)) wilcox.test(tablaR511$A,tablaR511$B) print("usted ha ejecutado su primer script")	sum(tablaR511$A) print("TEST DE NORMALIDAD PARA DATOS A Y B") print(shapiro.test(tablaR511$A)) print(shapiro.test(tablaR511$B)) wilcox.test(tablaR511$A,tablaR511$B) print("usted ha ejecutado su primer script")

al ejecutar ordenes6.txt obtendrá

> source("ordenes6")

comenzará su script, si desea continuar oprima <enter> en caso contrario oprima <control + c> de acuerdo a su decisión puede haber salido o continuado con el script

[1] "Los datos fueron importados exitosamente"
[1] "DATOS TablaR511"
  A B
1 2.1 2.1
2 2.3 5.1
3 2.8 6.1
4 2.9 5.1
5 3.1 5.2
6 4.0 5.0
[1] "Summary DATOS TablaR511"

```
       A               B
 Min.   :2.100   Min.   :2.100
 1st Qu.:2.425   1st Qu.:5.025
 Median :2.850   Median :5.100
 Mean   :2.867   Mean   :4.767
 3rd Qu.:3.050   3rd Qu.:5.175
 Max.   :4.000   Max.   :6.100
[1] "TEST DE NORMALIDAD PARA DATOS A Y B"

        Shapiro-Wilk normality test

data:  tablaR511$A
W = 0.93956, p-value = 0.6557


        Shapiro-Wilk normality test

data:  tablaR511$B
W = 0.7384, p-value = 0.0153

[1] "usted ha ejecutado su primer script"
Warning message:
In wilcox.test.default(tablaR511$A, tablaR511$B) :
  cannot compute exact p-value with ties
```

La función readline(), lleva en su interior un texto que muestra en la pantalla y allí se detiene el script. Si no se da enter no avanzará. Si se oprime enter comienzan a ejecutarse las funciones que se hallan a continuación.

2. Clase 2

Un script es un conjunto de instrucciones que pueden ser ejecutadas con un solo comando. Estas instrucciones pueden permitirnos ejecutar funciones de diversas bibliotecas de R así como introducir datos, tomar decisiones respecto de los análisis a realizar, exportar datos e informar resultados entre otras. El script permite ahorrar tiempo ya que las órdenes se ejecutan sin necesidad de tipear la misma, pero simultáneamente permite que pueda ser realizada por personas que desconocen las sintaxis.

Funciones incorporadas en clases anteriores
source("nombre del archivo de texto con el script a ejecutar")
print("texto a mostrar en pantalla")
readline("Muestra en pantalla un texto y detiene el script hasta que el usuario oprime enter")
# los textos antecedidos por # permiten introducir explicaciones en el script pero no se ejecutan.

Contenidos de esta clase

2.1. Definición de variables

En una secuencia de ordenes podemos definir variables a las que se les asignarán valores que pueden mantenerse o modificarse durante la ejecución del script.
Para definir una variable utilizamos el mismo código que hemos utilizado a lo largo del curso. Por ejemplo si tenemos una variable "a" a la que le queremos asignar un valor utilizaremos
a<- valor (que puede se numérico o una cadena de caracteres)

2.2. Función scan()

La función scan() permite introducir un dato desde el teclado.
La función scan() la podemos utilizar para introducir un número o una cadena de caracteres. Los códigos que se dan a continuación sirven para cada una de las situaciones
a- Introducción de un número con el teclado
n<- as.numeric(scan(file = "", what = "",nmax=1))
b-Introducción de una cadena de caracteres
nombre<- scan(file = "", what = "",nmax=1)

Algunos argumentos útiles de la función scan() cuando se utilizan caracteres.
Veamos la diferencia entre dos valores de argumentos
La línea siguiente si la utilizamos para introducir desde el teclado: san josé de la esquina, creará un objeto con el valor "san jose de la esquina"
ejecute la siguiente orden
>palabra<-scan(file = "", what = "",nmax=1,sep="\n",nlines=1)
en pantalla le aparecerá 1: ..... R está esperando que introduzca "san jose de la esquina"
escribimos: san jose de la esquina
si vemos que hay dentro de palabra
> palabra

[1] "san jose de la esquina"
preguntamos a R si palabra es un vector
> is.vector(palabra)
[1] TRUE
y cuantos elementos tiene
> length(palabra)
[1] 1

sin embargo si lo hubiéramos utilizado como
> palabra<-scan(file = "", what = "",nmax=5,sep=" ",nlines=1)
Queda esperando que introduzcamos el dato. Escribimos san josé de la esquina
si vemos que tiene el objeto palabra
> palabra
[1] "san" "jose" "de" "la" "esquina"
> is.vector(palabra)
[1] TRUE
> length(palabra)
[1] 5

habría creado un objeto con "san" "jose" "de" "la" "esquina"

La función scan() será de utilidad para introducir datos al ejecutar un script.

Veamos un script que nos permita sumar dos números. En este script veremos la asignación de valores a variables numéricas a través del teclado.
Mostraremos el script en una tabla de dos columnas.
Cuando se hagan modificaciones a un script se mostrarán en la segunda columna la línea agregadas en color rojo
Las líneas que no han cambiado se dejarán una a la par de otra en color negro
Las líneas eliminadas se colocarán en azul
Para ejecutar el script copie el código en un editor de texto, grabe el archivo en el mismo directorio que ejecutará el script. Luego abra R de manera que su espacio de trabajo quede en el mismo directorio que grabó el script y ejecútelo. Para ello ejecute

> source("script521")

script521.txt	explicacion
#este script permite sumar dos números (n1 y n2) que el usuario fijará por el teclado. #llamaremos n1 y n2 a los números #deseamos que al arrancar el script n1 y n2 tengan el valor 0	
n1<-0	asigna a n1 el valor 0
n2<-0	asigna a n2 el valor 0
readline("si desea realizar una suma oprima <enter> en caso contrario <control + c>")	anuncio en pantalla
print("introduzca el primer número a través de su teclado")	nos pide el primer número
n1<-as.numeric(scan(file = "", what = "",nmax=1))	lo introducimos por el teclado
print("introduzca el segundo número a través de su teclado")	nos pide el segundo número
n2<-as.numeric(scan(file = "", what = "",nmax=1))	lo introducimos por el teclado y el script lo asina a n2
print("numeros introducidos")	nos muestra el texto "números introducidos" nos muestra el texto "n1"
print("n1")	nos muestra el valor de n1
print(n1)	nos muestra el texto "n2"
print("n2")	nos muestra el valor de n2
print(n2)	realiza la suma
sum<-n1 + n2	nos muestra el texto "suma de los números"
print("suma de los números")	nos muestra el valor de la suma
print(sum)	

2.3. Función writeLines

La función writeLines() permite mostrar una cadena de caracteres en la consola, pero no una variable, como vemos en el siguiente ejemplo

```
> writeLines("Esto es una prueba de uso de una función")
Esto es una prueba de uso de una función
```

si definimos una variable

```
> a<-23
> writeLines(a)
Error in writeLines(a) : invalid 'text' argument
```

vemos que nos arroja un error

Sin embargo el código que se da a continuación es general y puede ser modificado para diferentes situaciones, permitiendo mostrar textos y variables simultáneamente.

```
writeLines(paste("texto ", nombrevariable))
```

Por ejemplo

```
> writeLines(paste("El valor de la variable a es: ",a,sep=" "))
El valor de la variable a es:  23
```

Como ya vimos la función print() permite más o menos lo mismo. Pero permite de manera individual mostrar texto o función

```
> print("valor de la función a")
[1] "valor de la función a"
> print(a)
[1] 23
> print(paste('valor de la variable a: ', a))
[1] "valor de la variable a:  23"
```

Veamos el script anterior modificado (script522.txt) utilizando la función writeLines()

script521.txt	script522.txt
#este script permite sumar dos números (n1 y n2) que el usuario fijará por el teclado.	#este script permite sumar dos números (n1 y n2) que el usuario fijará por el teclado.
#llamaremos n1 y n2 a los números	#llamaremos n1 y n2 a los números
#deseamos que al arrancar el script n1 y n2 tengan el valor 0	#deseamos que al arrancar el script n1 y n2 tengan el valor 0
n1<-0	n1<-0
n2<-0	n2<-0
readline("si desea realizar una suma oprima <enter> en caso contrario <control + c>")	readline("si desea realizar una suma oprima <enter> en caso contrario <control + c>")
print("introduzca el primer número a través de su teclado")	print("introduzca el primer número a través de su teclado")
n1<-as.numeric(scan(file = "", what = "",nmax=1))	n1<-as.numeric(scan(file = "", what = "",nmax=1))
print("introduzca el segundo número a través de su teclado")	print("introduzca el segundo número a través de su teclado")
n2<-as.numeric(scan(file = "", what = "",nmax=1))	n2<-as.numeric(scan(file = "", what = "",nmax=1))
print("numeros introducidos")	print("numeros introducidos")
print("n1")	
print(n1)	
print("n2")	
print(n2)	
sum<-n1 + n2	
print("suma de los números")	
print(sum)	
	writeLines(paste("primer número introducido: ", n1))
	writeLines(paste("segundo número introducido: ", n2))
	sum<-n1 + n2
	writeLines(paste("suma: ", sum))

Copia el script522.txt en su editor de texto, guárdelo en el directorio de su espacio de trabajo y ejecútelo con la línea de comando
source("script522")

Analice el código y lo que observa en la pantalla. Compárelo con el scritp521.txt
La función writeLines() nos da una mejor presentación de nuestras salidas de pantalla, pero los

resultados son similares a utilizar print(). La ventaja de writeLines() es que permite mostrar con un solo comando textos y valores de objetos, como en este caso los valores de las variables n1 y n2.

2.4. Función Sys.sleep()

La función Sys.sleep() le permite producir un retardo en la secuencia de eventos. Supongamos que desea ejecutar un script que le permita ir viendo paso a paso su desarrollo, con tiempos que usted desea fijar. La función Sys.sleep() permite esto y su código general es

Sys.sleep(n)

donde n es el número de segundos de retardo entre las órdenes. Durante ese tiempo el teclado es bloqueado y R ejecuta las órdenes según la cantidad de segundos introducidos en el código del script

script522.txt	script523.txt

#este script permite sumar dos números (n1 y n2) que el usuario fijará por el teclado.
#llamaremos n1 y n2 a los números
#deseamos que al arrancar el script n1 y n2 tengan el valor 0
n1<-0
n2<-0
readline("si desea realizar una suma oprima <enter> en caso contrario <control + c>")
print("introduzca el primer número a través de su teclado")
n1<-as.numeric(scan(file = "", what = "",nmax=1))
print("introduzca el segundo número a través de su teclado")
n2<-as.numeric(scan(file = "", what = "",nmax=1))
print("numeros introducidos")
 	
writeLines(paste("primer número introducido: ", n1))
 	
writeLines(paste("segundo número introducido: ", n2))
sum<-n1 + n2
 	
writeLines(paste("suma: ", sum))
	#este script permite sumar dos números (n1 y n2) que el usuario fijará por el teclado y tiene retardos al mostar los resutados
	#llamaremos n1 y n2 a los números
	#deseamos que al arrancar el script n1 y n2 tengan el valor 0
	n1<-0
	n2<-0
	readline("si desea realizar una suma oprima <enter> en caso contrario <control + c>")
	print("introduzca el primer número a través de su teclado")
	n1<-as.numeric(scan(file = "", what = "",nmax=1))
	print("introduzca el segundo número a través de su teclado")
	n2<-as.numeric(scan(file = "", what = "",nmax=1))
	print("numeros introducidos")
	Sys.sleep(2)
	writeLines(paste("primer número introducido: ", n1))
	Sys.sleep(2)
	writeLines(paste("segundo número introducido: ", n2))
	sum<-n1 + n2
	Sys.sleep(2)
	writeLines(paste("suma: ", sum))

Copia el script523.txt en su editor de texto, guárdelo en el directorio de su espacio de trabajo y ejecútelo con la línea de comando. Podrá parecerle que su computadora se tildó, pero no es así está esperando 2 segundos entre las ordenes.
source("script523")

Analice el código y lo que observa en la pantalla. Compárelo con el scritp521.txt

Si la función Sys.sleep() no le funciona. Intente el siguiente código equivalente a Sys.sleep(2)

> system('sleep 2')

2.5. Función alarm()

La función alarm() permite generar un sonido que nos indica cuando un script llegó a un dado punto. Si usted tiene un script con retardos (introducidos con Sys.sleep()), de manera que le permita ir desarrollando alguna actividad y cuando llega a un punto requiere que el programa le avise con un sonido, alarm() es un buen recurso.

script523.txt	script524.txt
#este script permite sumar dos números (n1 y n2) que el usuario fijará por el teclado. y tiene retardos al mostrar los resultados	#este script permite sumar dos números (n1 y n2) que el usuario fijará por el teclado, tiene retardos al mostrar los resultados y nos avisará con un sonido cuando llegó a un punto determinado
#llamaremos n1 y n2 a los números	#llamaremos n1 y n2 a los números
#deseamos que al arrancar el script n1 y n2 tengan el valor 0	#deseamos que al arrancar el script n1 y n2 tengan el valor 0
n1<-0	n1<-0
n2<-0	n2<-0
readline("si desea realizar una suma oprima <enter> en caso contrario <control + c>")	readline("si desea realizar una suma oprima <enter> en caso contrario <control + c>. Al finalizar el script se lo indicará con la repetición de tres beeps")
print("introduzca el primer número a través de su teclado")	print("introduzca el primer número a través de su teclado")
n1<-as.numeric(scan(file = "", what = "",nmax=1))	n1<-as.numeric(scan(file = "", what = "",nmax=1))
print("introduzca el segundo número a través de su teclado")	print("introduzca el segundo número a través de su teclado")
n2<-as.numeric(scan(file = "", what = "",nmax=1))	n2<-as.numeric(scan(file = "", what = "",nmax=1))
print("numeros introducidos")	print("numeros introducidos")
Sys.sleep(2)	Sys.sleep(2)
writeLines(paste("primer número introducido: ", n1))	writeLines(paste("primer número introducido: ", n1))
Sys.sleep(2)	Sys.sleep(2)
writeLines(paste("segundo número introducido: ", n2))	writeLines(paste("segundo número introducido: ", n2))
sum<-n1 + n2	sum<-n1 + n2
Sys.sleep(2)	Sys.sleep(2)
	alarm()
	readline("si desea ver el resultado de la suma oprima <enter>, en caso contrario oprima <control + c>")
	writeLines(paste("suma: ", sum))
writeLines(paste("suma: ", sum))	alarm()
	Sys.sleep(0.2)
	alarm()
	Sys.sleep(0.2)
	alarm()

Si tuviera inconvenientes con la función alarm(), verifique que los parlantes estén prendidos. Si persiste el problema intente con el código
cat('\a')
que es equivalente a alarm()

2.6. Función beep()

Esta función le permite indicar diferentes partes del script con sonidos distintos. Debe instalar el paquete beepr y ejecutar el comando
beep(sound=number)
number: 1 a 10 con diferentes sonidos

script524.txt	script525.txt
#este script permite sumar dos números (n1 y n2) que el usuario fijará por el teclado, tiene retardos al mostrar los resultados y nos avisará con un sonido cuando llegó a un punto determinado	#este script permite sumar dos números (n1 y n2) que el usuario fijará por el teclado, tiene retardos al mostrar los resultados y nos avisará con un sonido cuando llegó a un punto determinado
#llamaremos n1 y n2 a los números	#llamaremos n1 y n2 a los números
#deseamos que al arrancar el script n1 y n2 tengan el valor 0	#deseamos que al arrancar el script n1 y n2 tengan el valor 0
n1<-0	n1<-0
n2<-0	n2<-0
readline("si desea realizar una suma oprima <enter> en caso contrario <control + c>. Al finalizar el script se lo indicará con la repetición de tres beeps")	readline("si desea realizar una suma oprima <enter> en caso contrario <control + c>. Al finalizar el script se lo indicará con la repetición de tres beeps")
print("introduzca el primer número a través de su teclado")	print("introduzca el primer número a través de su teclado")
n1<-as.numeric(scan(file = "", what = "",nmax=1))	n1<-as.numeric(scan(file = "", what = "",nmax=1))
print("introduzca el segundo número a través de su teclado")	print("introduzca el segundo número a través de su teclado")
n2<-as.numeric(scan(file = "", what = "",nmax=1))	n2<-as.numeric(scan(file = "", what = "",nmax=1))
print("numeros introducidos")	print("numeros introducidos")
Sys.sleep(2)	Sys.sleep(2)
writeLines(paste("primer número introducido: ", n1))	writeLines(paste("primer número introducido: ", n1))
Sys.sleep(2)	Sys.sleep(2)
writeLines(paste("segundo número introducido: ", n2))	writeLines(paste("segundo número introducido: ", n2))
sum<-n1 + n2	sum<-n1 + n2
Sys.sleep(2)	Sys.sleep(2)

alarm() readline("si desea ver el resultado de la suma oprima <enter>, en caso contrario oprima <control + c>") writeLines(paste("suma: ", sum)) alarm() Sys.sleep(0.2) alarm() Sys.sleep(0.2) alarm()	alarm() readline("si desea ver el resultado de la suma oprima <enter>, en caso contrario oprima <control + c>") writeLines(paste("suma: ", sum)) alarm() Sys.sleep(0.2) alarm() Sys.sleep(0.2) library(beepr) beep(sound=2)

Puede probar con otros números para el argumento sound.
El uso de la función beep() no es meramente estético. Usted puede asociar cada evento de un script a un sonido en particular. Por ejemplo, si se debe introducir un número telefónico anteceder por un sonido en particular, cuando se solicite el DNI utilizar otro y así sucesivamente. El usuario del script tendrá así un elemento de orientación a la hora de introducir datos. También pueden utilzarse para mostrar ciertas cosas en pantalla.

2.7. Detener un script

Para detener un script en cualquier punto oprima <Control + c>

2.8. Introducción de constantes

Cuando se desea introducir una constante o vector con varios elementos constantes, la definición se realiza utilizando el código aprendido en módulos anteriores
Para una constante con un solo valor, utilice
a<-3
Para un vector con varios valores constantes, en este caso dos valores utilice
b<-c(2,3)
Por supuesto, el nombre de la variable puede elegirlo usted, siempre utilizando algún criterio que resulte claro.

script525.txt	script526.txt
#este script permite sumar dos números (n1 y n2) que el usuario fijará por el teclado, tiene	#este script permite sumar dos números (n1 y n2) que el usuario fijará por el teclado,

```
retardos al mostrar los resultados y nos avisará con un sonido cuando llegó a un punto determinado


#llamaremos n1 y n2 a los números
#deseamos que al arrancar el script n1 y n2 tengan el valor 0




n1<-0
n2<-0
readline("si desea realizar una suma oprima <enter> en caso contrario <control + c>. Al finalizar el script se lo indicará con la repetición de tres beeps")
print("introduzca el primer número a través de su teclado")
n1<-as.numeric(scan(file = "", what = "",nmax=1))
print("introduzca el segundo número a través de su teclado")
n2<-as.numeric(scan(file = "", what = "",nmax=1))
print("numeros introducidos")




Sys.sleep(2)
writeLines(paste("primer número introducido: ", n1))
Sys.sleep(2)
writeLines(paste("segundo número introducido: ", n2))
sum<-n1 + n2
```

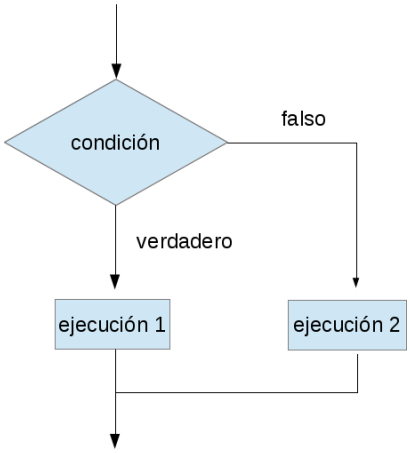
```
multiplicar dicha suma por una constante que ya se halla en el script,  tiene retardos al mostrar los resultados y nos avisará con un sonido cuando llegó a un punto determinado

#llamaremos n1 y n2 a los números
#deseamos que al arrancar el script n1 y n2 tengan el valor 0
# el script asigna a una constante el valor 7. No se podrá modificar por el teclado
constante<-7
n1<-0
n2<-0
readline("si desea realizar una suma oprima <enter> en caso contrario <control + c>. Al finalizar el script se lo indicará con la repetición de tres beeps")
print("introduzca el primer número a través de su teclado")
n1<-as.numeric(scan(file = "", what = "",nmax=1))
print("introduzca el segundo número a través de su teclado")
n2<-as.numeric(scan(file = "", what = "",nmax=1))
print("numeros introducidos")
Sys.sleep(1)
writeLines(paste("valor de la constante interna: ", constante))
Sys.sleep(1)
writeLines(paste("primer número introducido: ", n1))
Sys.sleep(1)
writeLines(paste("segundo número introducido: ", n2))
sum<-n1 + n2
```

2.9. Toma de decisiones

Tomar decisiones implica poder elegir una u otra opción dentro del script. Estas ordenes comienzan a hacer versátil un script, permitiendo ejecutar rutinas diferentes. Siempre en este tipo de rutinas habrá una condición que se evalua y en base a ella se realiza uno u otra cosa, pudiendo ser múltiples las elecciones. Pongamos un ejemplo fuera de la programación, pero que sirve para ilustrar el caso. Supongamos que mañana dispongo del día libre entonces hoy pienso: si llueve iré al cine, si no llueve iré a pescar. Es claro que mañana lo primero que haré es evaluar si llueve. En caso que no lo haga, iré a pescar, pero si llueve, iré al cine. Podría fijar otras condiciones, como por ejemplo, si llego antes de la hora de la película iré a tomar un café y mientras no sea la hora consultaré mis redes sociales en internet. Entonces, arranco el día, veo que llueve. Como consecuencia no voy a pescar y me voy al cine. Llegué media hora antes, entonces me voy a tomar un café mientras consulto las redes sociales. Miro el reloj, faltan 20 minutos para comenzar la función de cine, sigo con las redes sociales. Miro nuevamente, faltan 10 minutos, sigo con redes sociales. Miro el reloj, es hora del cine, dejo las redes y me voy al cine. Cada decisión la tomé en base a la evaluación de una condición establecida.
ESto mismo nos permite los controles de flujo o decisión de un script.
Veremos primero la función if()
El código general de esta lo podemos escribir de la siguiente manera

```
if (condicion) {

ejecución 1

}else{

ejecución 2

}
```

En el esquema siguiente se muestra el funcionamiento de esta función



Cuando el script llegue a "if" evaluará el valor de una "condición" que podrá tomar dos valores. Si condición es verdadera realizará la ejecución 1, en caso contrario, si la condición es falsa, realizará la ejecución 2. En cualquiera de las dos situaciones luego continuará con las ordenes del script.

Veamos el desarrollo del scritp siguiente

script526.txt	script527.txt
#este script permite sumar dos números (n1 y n2) que el usuario fijará por el teclado, multiplicar dicha suma por una constante que ya se halla en el script, tiene retardos al mostrar los resultados y nos avisará con un sonido cuando llegó a un punto determinado #llamaremos n1 y n2 a los números #deseamos que al arrancar el script n1 y n2 tengan el valor 0 # el script asigna a una constante el valor 7. No se podrá modificar por elteclado	#este script permite sumar dos números (n1 y n2) que el usuario fijará por el teclado, multiplicar dicha suma por una constante que ya se halla en el script, luego permitirá ver la suma de los número o el producto de la suma por la constante tiene retardos al mostrar los resultados y nos avisará con un sonido cuando llegó a un punto determinado #llamaremos n1 y n2 a los números #deseamos que al arrancar el script n1 y n2 tengan el valor 0 # el script asigna a una constante el valor 7. No se podrá modificar por elteclado

constante<-7	constante<-7
n1<-0	n1<-0
n2<-0	n2<-0
readline("si desea realizar una suma oprima <enter> en caso contrario <control + c>. Al finalizar el script se lo indicará con la repetición de tres beeps")	readline("si desea realizar una suma oprima <enter> en caso contrario <control + c>. Al finalizar el script se lo indicará con la repetición de tres beeps")
print("introduzca el primer número a través de su teclado")	print("introduzca el primer número a través de su teclado")
n1<-as.numeric(scan(file = "", what = "",nmax=1))	n1<-as.numeric(scan(file = "", what = "",nmax=1))
print("introduzca el segundo número a través de su teclado")	print("introduzca el segundo número a través de su teclado")
n2<-as.numeric(scan(file = "", what = "",nmax=1))	n2<-as.numeric(scan(file = "", what = "",nmax=1))
print("numeros introducidos")	print("numeros introducidos")
Sys.sleep(1)	Sys.sleep(1)
writeLines(paste("valor de la constante interna: ", constante))	writeLines(paste("valor de la constante interna: ", constante))
Sys.sleep(1)	Sys.sleep(2)
writeLines(paste("primer número introducido: ", n1))	writeLines(paste("primer número introducido: ", n1))
Sys.sleep(1)	Sys.sleep(2)
writeLines(paste("segundo número introducido: ", n2))	writeLines(paste("segundo número introducido: ", n2))
sum<-n1 + n2	sum<-n1 + n2
Sys.sleep(1)	Sys.sleep(2)
alarm()	alarm()
readline("si desea ver el resultado de la suma oprima <enter>, en caso contrario oprima <control + c>")	print("si desea ver la suma de los numeros oprima 1, si desea ver el producto oprima cualquier otro número")
	n3<-as.numeric(scan(file = "", what = "",nmax=1))
	if(n3==1){
	writeLines(paste("suma: ", sum))

Como habrá observado, el script le permitió ver la suma o el producto. Usted tuvo que tomar una decisión, la que pudo elegir a través del teclado. Ejecute al menos dos veces el script tomando una u otra decisión.

3. Clase 3

3.1. Funciones incorporadas en clases anteriores

source("nombre del archivo de texto con el script a ejecutar")
print("texto a mostrar en pantalla")
print(valor de un objeto de R)
readline("Muestra en pantalla un texto y espera enter para continuar")
#
los textos antecedidos por # permiten introducir explicaciones en el script pero no se ejecutan.
scan: introduce un número o caracteres en la variable n

n<- as.numeric(scan(file = "", what = "",nmax=1))

n<- scan(file = "", what = "",nmax=1)

writeLines() permite mostrar un texto o texto con variable combinada con la función paste()

writeLines("texto")

writeLines(paste("texto ", nombrevariable))

Sys.sleep(n)

introduce una pausa automática de n segundos de retardo entre las órdenes consecutivas

alarm()

genera un beep por sus parlantes

beep(sound=number)

genera otros sonidos a elección dentro de una biblioteca

Control + c

detiene el script

Controles de flujo del script

if-else

permite elegir dos ejecuciones diferentes según se cumpla o no una condición

3.2. Estructura if()

Una estructura similar ya fue analizada. Nos referimos a la estructura if-else, la que permitía ejecutar dos acciones diferentes luego de evaluar una condición. La estructura if es similar en el sentido que se evalúa una condición pero se ejecuta una orden o no. La forma general de la estructura es
if (condicion) {
ejecución

}

Escrito de esta manera le permite ejecutar una orden si se cumple una condición y, en caso que no se cumpla esa condición sigue el flujo normal del script.

Supongamos que se realizó un script para tomar información en un grupo de personas. La información requerida es solamente la edad de los hombres. La recopilación de información podría hacerse buscando hombres y haciendo la pregunta o bien encuestando a todas las personas, preguntando primero el sexo y luego que solo se pregunte el peso en el caso que sea hombre. El script para realizar esto quedaría con la siguiente estructura (por supuesto no es la única posible).

script531	explicaciones
#este script le permite obtener información sobre la edad solo de hombres, habiendo interrogado a hombres y mujeres	
print("Buen día, tomaremos solo unos segundo de su tiempo")	Imprime en pantalla el mensaje entre comillas. No detiene el script y no exige ninguna intervención del usuario.
Sys.sleep(1)	Introduce un retardo de 1 segundo entre el comando print y alarm()
alarm()	Produce un beep en los parlantes, pero el script continua y no requiere intervención del usuario.
readline("si desea colaborar con el estudio oprima <enter> en caso contrario control c")	Muestra el mensaje y requiere acción desde el teclado
print("si es hombre introduzca la letra h por el teclado, si es mujer m")	
sexo<-scan(file = "", what = "",nmax=1)	Asigna al objeto sexo, el valor introducido a través del teclado
if(sexo=="h"){	Compara el contenido de la variable sexo (que puede ser "h" o "m") con "h". Si sexo=h, ejecutará la orden siguiente. Si sexo=m, saltará directamente a la orden alarm()
print("por favor indique su edad")	
edad<-as.numeric(scan(file = "", what = "",nmax=1))	
}	
alarm()	
print("gracias por colaborar, con el siguiente número participará en el sorteo de una Ferrari")	
sorteo<- trunc(runif(1,0,999))	Asigna a la variable sorteo un valor aleatorio entre 0 y 999 (debido a función runif()), sin decimales (por la función trun())
writeLines(paste("su número para el sorteo es: ", sorteo))	Imprime en pantalla "su número para el sorteo es: " seguido del número que se le asigno a quien se prestó a responder la encuesta

Cada vez que se ejecute el script, la información que se halla en la variable edad y que fue obtenida en la persona anteriormente encuestada, se perderá. Esto ocurre porque cada vez que ejecutamos el script se escribe el valo de la edad sobre la misma variable. Introduzcamos en el script un mecanismo de almacenamiento de la información en un data frame. Podemos guardar en diferentes columnas por ejemplo, el sexo y la edad. Analice el siguiente script que realizada dichas rutinas.

Recuerde que la columna de la izquierda contiene el script anterior y a la derecha es script modificado y mejorado.

script531	scrip532
#este script le permite obtener información sobre la edad solo de hombres, habiendo interrogado a hombres y mujeres	#este script le permite obtener información sobre la edad solo de hombres, habiendo interrogado a hombres y mujeres
print("Buen día, tomaremos solo unos segundo de su tiempo")	print("Buen día, tomaremos solo unos segundo de su tiempo")
Sys.sleep(1)	Sys.sleep(1)
alarm()	alarm()
readline("si desea colaborar con el estudio oprima <enter> en caso contrario control c")	readline("si desea colaborar con el estudio oprima <enter> en caso contrario control c")
print("si es hombre introduzca la letra h por el teclado, si es mujer m")	print("si es hombre introduzca la letra h por el teclado, si es mujer m")
sexo<-scan(file = "", what = "",nmax=1)	sexo<-scan(file = "", what = "",nmax=1)
if(sexo=="h"){	if(sexo=="h"){
print("por favor indique su edad")	print("por favor indique su edad")
edad<-as.numeric(scan(file = "", what = "",nmax=1))	edad<-as.numeric(scan(file = "", what = "",nmax=1))
}	}
alarm()	alarm()
print("gracias por colaborar, con el siguiente número participará en el sorteo de una ferrari")	print("gracias por colaborar, con el siguiente número participará en el sorteo de una ferrari")
sorteo<- trunc(runif(1,0,999))	sorteo<- trunc(runif(1,0,999))
writeLines(paste("su número para el sorteo es: ", sorteo))	writeLines(paste("su número para el sorteo es: ", sorteo))
	#creamos un data.frame con columnas: sexo y edad
	datos<-data.frame(sexo=sexo,edad=edad)

veamos las explicaciones del script532

scrip532	explicaciones

#este script le permite obtener información sobre la edad solo de hombres, habiendo interrogado a hombres y mujeres
 	
print("Buen día, tomaremos solo unos segundo de su tiempo")
Sys.sleep(1)
alarm()
readline("si desea colaborar con el estudio oprima <enter> en caso contrario control c")
print("si es hombre introduzca la letra h por el teclado, si es mujer m")
sexo<-scan(file = "", what = "",nmax=1)
if(sexo=="h"){
print("por favor indique su edad")
edad<-as.numeric(scan(file = "", what = "",nmax=1))
}
alarm()
print("gracias por colaborar, con el siguiente número participará en el sorteo de una ferrari")
sorteo<- trunc(runif(1,0,999))
writeLines(paste("su número para el sorteo es: ", sorteo))
#creamos un data.frame con columnas: sexo y edad
datos<-data.frame(sexo=sexo,edad=edad)
	crea el data.frame "datos" y asigna a la primer columna llamada sexo, el valor de la variable sexo ingresada por teclado. También asigna a la columna edad, la edad introducido por teclado.

puede comprobar que en su espacio de trabajo hay un data.frame llamado "datos" con los datos introducidos. Escriba

> datos

y deberá ver el dato introducido.

¿Pero cuál es el el problema de este scritp? Cada vez que lo ejecutaremos creará un data.frame con el nombre "datos" y perderemos los datos anteriores. Para solucionar dicho problema primeramente desarrollaremos otra forma de controlar el flujo de un script.

3.3. Estructura for

Esta estructura permite repetir un dado número de veces una rutina en particular. Su forma general es

```
for (condición) {
acción
}
```

Es decir que la estructura evalúa una condición y si se cumple ejecuta la acción, que se hallará descripta dentro de las {}. Al finalizar la secuencia de órdenes vuelve al principio, evalúa nuevamente la condición, si se sigue cumpliendo la condición, repite las acciones. Si la condición deja de cumplirse, ya no se ejecutará la acción y seguirá el flujo de órdenes del script.

Supongamos que desea generar una cantidad predeterminada de números aleatorios comprendidos entre 0 y 10. La opción más sencilla sería ejecutar el número de veces deseado la instrucción trunc(runif(1,0,10)).

La función runif() genera un número entre 0 y 10 con cifras decimales y luego a dicho valor, la variable trunc() lo corta en el punto decimal, generando un número entero. Es claro que si desea hacer 10 número, ejecutará la orden 10 veces. Si lo desea 50 lo haría 50 veces y así sucesivamente. Surge claramente que cuando el número crece el tiempo invertido se transformará en tiempo perdido. Si bien la función runif() tiene la opción de generar un gran número de valores aleatorios, modificando el primer argumento, lo haremos con la estructura for().

script533	explicaciones
#este script permite generar una vector de números aleatorios entre dos números fijados por el usuario y en la cantidad deseada	este script generará números aleatorios enteros comprendidos entre dos números que llamaremos limitesuperior y limiteinferior, los cuales los puede fijar el usuario. El usuario también puede fijar la cantidad de números comprendidos entre los límites establecidos, la variable que contiene dicho número se llama cantidaddenumeros.
print("Este es un script generador de números aleatorios comprendido en un rango fijado por el usuario")	
print("Introduzca el límite inferior del rango de los números")	
limiteinferior<-as.numeric(scan(file = "", what = "",nmax=1))	asigna a la variable limiteinferior un valor a través del teclado.
print("Introduzca el límite superior del rango de los números")	
limitesuperior<-as.numeric(scan(file = "", what = "",nmax=1))	asigna a la variable limitesuperior un valor introducido por el teclado.
print("Introduzca la cantidad de números que desea generar")	
cantidaddenumeros<-as.numeric(scan(file = "", what = "",nmax=1))	asigna a la variables cantidaddenumeros un valor a través del teclado
vector<-c()	crea un vector sin elementos que almacenará los números aleatorios que serán creado a continuación.
for(i in 1:cantidaddenumeros){	la estructura for, repetirá la orden que se encuentra entre {} una cantidad de veces igual al número que se halle en la variable cantidad de números. Repetira el ciclo for{} por una cantidad de veces igual a cantidaddenumeros
vector<-c(vector,trunc(runif(1,limiteinferior,limitesuperior)))	nos muestra en pantalla el valor introducido para el límite inferior.
}	nos muestra en pantalla el valor del límite superior introducido.
readline("desea ver las condiciones fijadas y el resultado? oprima enter. Si no lo desea oprima control + c")	nos muestra en pantalla la cantidad de números fijados por el usuario.
writeLines(paste("limite inferior: ", limiteinferior))	
writeLines(paste("limite superior: ", limitesuperior))	
writeLines(paste("cantidad de datos: ", cantidaddenumeros))	nos muestra los valores generados. Podrá comprobar que en dicho vector no habrá números menores al valor de limiteinferior, ni superiores al valor de limitesuperior. Si cuenta los elementos del vector, verá que la cantidad coincide con el número introducido en la variable cantidaddenumeros.
print("vector con los datos")	
print(vector)	

En este momento estamos en condiciones de crear un objeto, como por ejemplo un data.frame y cuando ejecutemos el script se cheque si el mismo existe o no. En caso que exista

utilizaremos el mismo data.frame. En caso contrario lo crearemos. Esta toma de decisión la haremos con la estructura if-else y la búsqueda del objeto en el espacio de trabajo la haremos con la estructura for().
El script532
pide que introduzca h o m según sea el sexo del interrogado, hombre o mujer.
Si es hombre le pregunta la edad.
Si es mujer u hombre le agradece la colaboración.
Le otorga un número para un sorteo.
Crea un data.frame llamado "datos" con las columnas sexo y edad.
Archiva el sexo en la fila 1 y columna 1
Archiva la edad en la fila 1 columna 2
El problema radicaba en que al ejecutar nuevamente el script, le sobre-escribe al objeto y perdemos los datos.
Entonces primero introduciremos un código que lee un vector con los nombres de los objetos del espacio de trabajo

scrip532	script534
#este script le permite obtener información sobre la edad solo de hombres, habiendo interrogado a hombres y mujeres	#este script le permite obtener información sobre la edad solo de hombres, habiendo interrogado a hombres y mujeres
print("Buen día, tomaremos solo unos segundo de su tiempo")	print("Buen día, tomaremos solo unos segundo de su tiempo")
Sys.sleep(1)	Sys.sleep(1)
alarm()	alarm()
readline("si desea colaborar con el estudio oprima <enter> en caso contrario control c")	readline("si desea colaborar con el estudio oprima <enter> en caso contrario control c")
print("si es hombre introduzca la letra h por el teclado, si es mujer m")	print("si es hombre introduzca la letra h por el teclado, si es mujer m")
sexo<-scan(file = "", what = "",nmax=1)	sexo<-scan(file = "", what = "",nmax=1)
if(sexo=="h"){	if(sexo=="h"){
print("por favor indique su edad")	print("por favor indique su edad")
edad<-as.numeric(scan(file = "", what = "",nmax=1))	edad<-as.numeric(scan(file = "", what = "",nmax=1))
}	}
alarm()	alarm()
print("gracias por colaborar, con el siguiente número participará en el sorteo de una ferrari")	print("gracias por colaborar, con el siguiente número participará en el sorteo de una ferrari")
sorteo<- trunc(runif(1,0,999))	sorteo<- trunc(runif(1,0,999))
writeLines(paste("su número para el sorteo es: ", sorteo))	writeLines(paste("su número para el sorteo es: ", sorteo))

#creamos un data.frame con columnas: sexo y edad
datos<-data.frame(sexo=sexo,edad=edad)
	#chequeamos si existe el objeto datos y lo creamos o utilizamos
	nombreobjetos<-ls()
	cantidadobjetos<-length(nombreobjetos)
	marcador<-0
	for(i in 1:cantidadobjetos){
	if(nombreobjetos[i]=="datos"){
	print("El data.frame: datos ya se halla en su espacio de trabajo")
	alarm()
	marcador<-1
	}
	}
	if(marcador==0){
	#creamos un data.frame con columnas: sexo y edad
	datos<-data.frame(sexo=sexo,edad=edad)
	} else {
	fila<-nrow(datos)
	#las siguientes línea almacenan los datos en el data.frame
	datos[fila+1,1]<-sexo
	datos[fila+1,2]<-edad
	}

analicemos primero las líneas introducidas en script534

script534	explicaciones
#este script le permite obtener información sobre la edad solo de hombres, habiendo interrogado a hombres y mujeres print("Buen día, tomaremos solo unos segundo de su tiempo") Sys.sleep(1) alarm() readline("si desea colaborar con el estudio oprima <enter> en caso contrario control c")	

print("si es hombre introduzca la letra h por el teclado, si es mujer m")
sexo<-scan(file = "", what = "",nmax=1)
if(sexo=="h"){
print("por favor indique su edad")
edad<-as.numeric(scan(file = "", what = "",nmax=1))
}
alarm()
print("gracias por colaborar, con el siguiente número participará en el sorteo de una ferrari")
sorteo<- trunc(runif(1,0,999))
writeLines(paste("su número para el sorteo es: ", sorteo))
#chequeamos si existe el objeto datos y lo creamos o utilizamos	

nombreobjetos<-ls()
 	
cantidadobjetos<-length(nombreobjetos)
marcador<-0
 	
for(i in 1:cantidadobjetos){
 	
if(nombreobjetos[i]=="datos"){
print("El data.frame: datos ya se halla en su espacio de trabajo")
alarm()
marcador<-1
}
}
 	
if(marcador==0){
#creamos un data.frame con columnas: sexo y edad
datos<-data.frame(sexo=sexo,edad=edad)
	Crea un vector "nombreobjetos" con los nombres de los objetos que tenemos en el espacio de trabajo
	Cuenta cuantos objetos tiene el espacio de trabajo
	Crea una variable "marcador" a la que le asignamos el valor 0. Esta variable nos ayudará en orientar el script. ver más adelante
	La estructura for(i in 1:cantidaddeobjetos) repetirá la rutina una cantidad de veces igual a la cantidad de objetos del espacio de trabajo
	if(nombreobjeto[i]=="datos"), lo que hace es tomar el primer objeto del vector nombreobjetos, cuando i=1, lo compara con la palabra "datos" (que es el nombre de nuestro data.frame). Si coincide, nos avisa que el data.frame ya está en el espacio de trabajo y asigna a la variable marcador el valor 1. En caso que no coincida, el if no hace nada, ya que no tiene else y deja el marcador en el valor 0. Como el if está dentro de la estructura for, ahora i pasará a tomar el valor 2 y nuevamente if comparará el segundo valor del vector nombreobjetos con datos y asi sucesivamente. Si ningún elemento del vector nombreobjetos coincide con "datos", la variable marcador quedará en el valor 0
	Esta estructura if evalúa si el marcador =0.
	Si lo es quiere decir que no halló en el espacio de trabajo al data.frame "datos" y por lo tanto, lo crea.
	En cambio si halló al data.frame "datos" puso a la variable marcador en el valor 1 y por ende la estructura if ingresará en el else{}
	
	a la variable "fila" se le asigna el número de filas del

Ahora el script es más eficiente en el sentido que permite ir agregando a un objeto, en este caso un data.frame, nuevos datos en sucesivas ejecuciones del script. Aun cuando salgamos de nuestro espacio de trabajo, quedará el objeto con los datos y podrá ser utilizado. Sin embargo resulta algo incómodo que debamos ejecutar la orden.

source("scritp534")

cada vez que deseemos tomar un dato.

Le daremos solución a este problema en la próxima clase.

4. Clase 4

Funciones incorporadas en clases anteriores
source("nombre del archivo de texto con el script a ejecutar")
print("texto a mostrar en pantalla")
print(valor de un objeto de R)
readline("Muestra en pantalla un texto y espera enter para continuar")
#
los textos antecedidos por # permiten introducir explicaciones en el script pero no se ejecutan.
scan: introduce un número o caracteres en la variable n

n<- as.numeric(scan(file = "", what = "",nmax=1))

n<- scan(file = "", what = "",nmax=1)

writeLines() permite mostrar un texto o texto con variable combinada con la función paste()

writeLines("texto")

writeLines(paste("texto ", nombrevariable))

Sys.sleep(n)

introduce una pausa automática de n segundos de retardo entre las órdenes consecutivas

alarm()

genera un beep por sus parlantes

beep(sound=number)

genera otros sonidos a elección dentro de una biblioteca

Control + c

detiene el script

Controles de flujo del script

if-else

permite elegir dos ejecuciones diferentes según se cumpla o no una condición

if
permite evaluar una condición y actuar o no según el valor de la misma.
for
permite repetir una secuencia de ordenes un número preestablecido de veces.

En la clase anterior desarrollamos un script que nos permite encuestar a personas, preguntándole si son hombres o mujeres y si son hombres preguntarles la edad. Los datos los vamos almacenando en un data.frame llamado datos. El problema que tenía este script es que cada vez que queremos encuestar a una persona debemos ejecutar el script con la orden

source("script")

En esta clase daremos solución a este problema. En primer lugar desarrollaremos dos estructuras de toma de decisión que nos permitirían solucionar este problema

4.1. Estructura while()

Esta estructura nos permite repetir un procedimiento hasta que una dada condición deje de cumplirse. Haciendo una transposición didáctica para comprender la estructura analicemos el siguiente párrafo: *Comeré de a una porción de torta hasta que me sienta satisfecho.* ¿Cómo comienza el proceso? Si estoy satisfecho no como torta, pero si no estoy satisfecho comeré una porción. Luego de comerla evaluaré: ¿Estoy satisfecho?. Si la respuesta es NO, comeré otra porción. En cambio si la respuesta fuera SI, ya no comeré más. Así, luego de cada porción evaluaré y seguiré comiendo o no. Esto hace la función while()

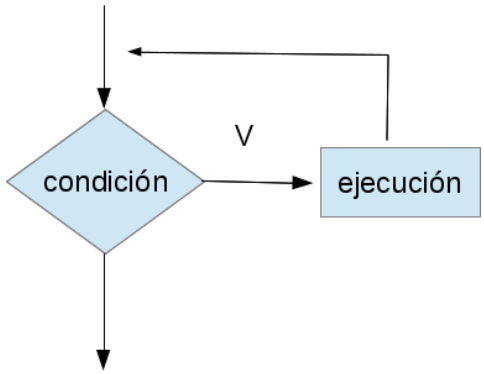
La estructura general es

while (condición) {

acción

}

El esquema siguiente muestra la estructura. Las flechas indican el sentido que tienen o toman las instrucciones del script. Como vemos al llegar a la evaluación de la condición, esta puede ser verdadera (V) o falsa. Si la condición es verdadera, se procede a ejecutar una serie de órdenes, Si no fuera V, el script sale del ciclo while y continua con el script.



Supongamos que desea hacer un script que le permita generar números aleatorios entre -100 y 100 y que dicho proceso lo realice hasta que encuentre un 0.

script541	explicaciones

#este script permite generar una vector de números aleatorios que no contenga el número cero.
 	
print("Este es un script generador de números aleatorios comprendido entre -100 y 100 pero se detiene cuando se genera un 0")
vector<-c()
a<-trunc(runif(1,-100,100))
 	
while(a!=0){
vector<-c(vector,a)
a<-trunc(runif(1,-100,100))
 	
}
 	
print("vector con los datos")
print(vector)
	
	mensaje que aparecerá en pantalla
	crea un vector sin elementos
	crea un número aleatorio entre -100 y 100, lo transforma en entero y lo asigna a la variable a
	Evalúa si a es diferente de cero. Si lo es, agrega al vector creado el valor de la variable a
	gemera un número aleatorio entre -100 y 100, lo transforma en entero y lo asigna a la variable a
	la estructura vuelve a evaluar si a es distinto de cero y repetirá este ciclo mientras a no reciba el valor 0. Cuando a tome el valor cero, saldrá de la estructura while, ya que esta se ejecuta solo si a!=0.
	
	Muestra en pantalla el vector creado

Ahora estamos en condiciones de hacer el script para encuestar sobre la edad de hombres. La pregunta se hace a ambos sexos y según la elección de sexo realizada por el encuestado se le pregunta la edad. El nuevo script no requiere que ejecute la orden source() cada vez que tengo que realizar la encuesta. Analice los cambios introducidos en el script

script534 (pendiente de solución de clase anterior)	script542
#este script le permite obtener información sobre la edad solo de hombres, habiendo interrogado a hombres y mujeres	#este script le permite obtener información sobre la edad solo de hombres, habiendo interrogado a hombres y mujeres

```
print("Buen día, tomaremos solo unos segundo de su tiempo")
Sys.sleep(1)
alarm()

readline("si desea colaborar con el estudio oprima <enter> en caso contrario control c")
print("si es hombre introduzca la letra h por el teclado, si es mujer m")
sexo<-scan(file = "", what = "",nmax=1)
if(sexo=="h"){
print("por favor indique su edad")
edad<-as.numeric(scan(file = "", what = "",nmax=1))
}
alarm()
print("gracias por colaborar, con el siguiente número participará en el sorteo de una ferrari")
sorteo<- trunc(runif(1,0,999))
writeLines(paste("su número para el sorteo es: ", sorteo))
#chequeamos si existe el objeto datos y lo creamos o utilizamos
nombreobjetos<-ls()
cantidadobjetos<-length(nombreobjetos)
marcador<-0
for(i in 1:cantidadobjetos){
if(nombreobjetos[i]=="datos"){
print("El data.frame: datos ya se halla en su espacio de trabajo")
alarm()
marcador<-1
}
}
if(marcador==0){
#creamos un data.frame con columnas: sexo y edad
datos<-data.frame(sexo=sexo,edad=edad)
```

```
print("Buen día, tomaremos solo unos segundo de su tiempo")
Sys.sleep(1)
alarm()
interruptor=1
while(interruptor==1){
readline("si desea colaborar con el estudio oprima <enter> en caso contrario control c")
print("si es hombre introduzca la letra h por el teclado, si es mujer m")
sexo<-scan(file = "", what = "",nmax=1)
if(sexo=="h"){
print("por favor indique su edad")
edad<-as.numeric(scan(file = "", what = "",nmax=1))
}
alarm()
print("gracias por colaborar, con el siguiente número participará en el sorteo de una ferrari")
sorteo<- trunc(runif(1,0,999))
writeLines(paste("su número para el sorteo es: ", sorteo))
#chequeamos si existe el objeto datos y lo creamos o utilizamos
nombreobjetos<-ls()
cantidadobjetos<-length(nombreobjetos)
marcador<-0
for(i in 1:cantidadobjetos){
if(nombreobjetos[i]=="datos"){
print("El data.frame: datos ya se halla en su espacio de trabajo")
alarm()
marcador<-1
}
}
if(marcador==0){
#creamos un data.frame con columnas: sexo y edad
datos<-data.frame(sexo=sexo,edad=edad)
```

Analicemos el script542

script542	
#este script le permite obtener información sobre el peso solo de hombres, habiendo interrogado a hombres y mujeres	
print("Buen día, tomaremos solo unos segundo de su tiempo")	
Sys.sleep(1)	
alarm()	creamos la variable interruptor a la que asignamos el valor 1
interruptor=1	
while(interruptor==1){	mientras interruptor tenga el valor 1, se ejecutarán las ordenes siguientes
readline("si desea colaborar con el estudio oprima <enter> en caso contrario control c")	
print("si es hombre introduzca la letra h por el teclado, si es mujer m")	
sexo<-scan(file = "", what = "",nmax=1)	
if(sexo=="h"){	
print("por favor indique su edad")	
edad<-as.numeric(scan(file = "", what = "",nmax=1))	
}	
alarm()	
print("gracias por colaborar, con el siguiente número participará en el sorteo de una Ferrari")	
sorteo<- trunc(runif(1,0,999))	
writeLines(paste("su número para el sorteo es: ", sorteo))	
#chequeamos si existe el objeto datos y lo creamos o utilizamos	
nombreobjetos<-ls()	
cantidadobjetos<-length(nombreobjetos)	
marcador<-0	
for(i in 1:cantidadobjetos){	
if(nombreobjetos[i]=="datos"){	
print("El data.frame: datos ya se halla en su espacio de	

trabajo")
alarm()
marcador<-1
}
}
if(marcador==0){
#creamos un data.frame con columnas: sexo y edad
datos<-data.frame(sexo=sexo,edad=edad) dat
} else {
fila<-nrow(datos)
#las siguientes línea almacenan los datos en el data.frame
if(sexo=="h"){
datos[fila+1,1]<-sexo
datos[fila+1,2]<-edad
}
}	

print("Si desea encuestar otra persona oprima 1, en caso contrario oprima cualquier numero") interruptor<-as.numeric(scan(file = "", what = "",nmax=1)) }	En este punto el script nos pregunta por la pantalla si deseamos encuestar a otra persona. Si así lo deseamos, le asignamos nuevamente 1 a la variable interruptor En esta llave (que es la que cierra la estructura while) vuelve al while. Como interruptor aun vale 1, ejecuta todas las ordenes nuevamente, permitiéndonos encuestar a otra persona sin ejecutar nuevamente source() Nos muestra el número total de encuestados.
writeLines(paste("Numero total de hombre encuestadas: ",nrow(datos[datos$sexo=="h",]))) writeLines(paste("Numero total de personas encuestadas: " ,nrow(datos))) print(datos)	nos muestra los datos de los encuestados

Obviamente el script aun tiene algunas dificultades y comete una buena cantidad de errores que iremos corrigiendo a medida que avancemos en el tema.

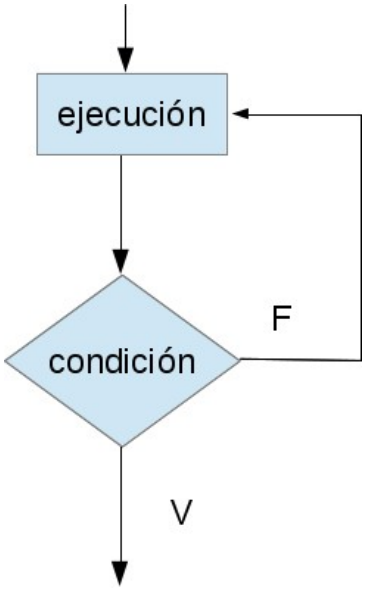
4.2. Estructura repeat()

La estructura repeat() es similar a while(). La diferencia es que while() evalua una condición al principio de las instrucciones mientras que repeat() la evalua al final. Sin embargo, casi siempre se puede utilizar ambas para realizar determinadas rutinas. La estructura repeat() como for() y while() repiten una rutina.
Transposición didáctica. Supongamos que deseo comunicarme telefónicamente con alguien. Estoy decidido a llamar hasta que me atiendan. Es decir repetiré una instrucción un dado número de veces hasta lograr el objetivo.
Con la estructura while() equivale a marcar un número y espero la contestación. La consigna será "llamaré mientras no me atiendan", es decir que si no me atienden llamo nuevamente.
Con la estructura repeat() equivale a marcar un número y espero la contestación. Si me atienden, dejo de llamar. Es decir que si me atienden dejo de llamar. Equivale a "llamar hasta que me atiendan"
Habitualmente cuando llamamos a alguien por teléfono utilizamos una estructura for(). Marco el número y espero. Si me atendieron, no marco más. Si no me atendieron, pruebo dos veces más (por ejemplo) y si no me atienden, probaré en otro momento. Es decir el bucle se termina en un número pre-establecidos de vueltas.

La estructura de la función repeat() es:

```
repeat {lo que se repite
        if(condición){
                break
                }
        }
```

El esquema siguiente muestra esta estructura



Supongamos que deseamos buscar un número establecido de antemano, y la búsqueda la haremos utilizando un generador de números aleatorios enteros entre 0 y 999 (como ya hemos visto). Lo que deseamos es que mientras se ejecute el script nos indique por que iteración está y que número halló. Cuando encuentre el número, deseamos que finalice el script y nos indique cuantas iteraciones realizó para lograrlo. Para ralentizar el proceso hemos introducido una función Sys.sleep(0.1) esto nos permite ir viendo el proceso de búsqueda. Puede cambiar el tiempo si desea que es proceso transcurra más rápido o más lento.

script543	explicaciones
# buscador de un número utilizando un generador de números aleatorios	
print("introduzca el número que desea buscar")	Inicializamos desde el teclado la variable numeroabuscar
numeroabuscar<-as.numeric(scan(file = "", what = "",nmax=1))	declaramos la variable contador y la inicializamos con el valor 0. Esta variable irá tomando números enteros sucesivos, contando cada bucle de búsqueda
contador<-0	
repeat{	crea un número aleatorio entero entre 0 y 999 y lo asigna a la variable numeroaleatorio
numeroaleatorio<- trunc(runif(1,0,999))	la variable contador toma un valor una unidad mayor que el valor que tenia
contador=contador + 1	Nos muestra el número de iteración y el valor aleatorio hallado
writeLines(paste("numero de iteración: ",contador, " numero hallado: ", numeroaleatorio))	
Sys.sleep(0.1)	pone una pausa para ralentizar el proceso. Si no se coloca Sys.sleep el proceso será instantáneo
if(numeroaleatorio==numeroabuscar){break}	cuando el número aleatorio coincide con el número introducido finaliza el bucle repeat
}	emitirá un sonido para avisarnos que halló el número.
alarm()	
writeLines(paste("Ha hallado el número deseado en: ", contador, " iteraciones"))	

4.3. Bucles y estructuras anidadas

Los procesos que implican estructuras recursivas o repetitivas, así como las tomas de decisión, pueden anidarse. De hecho ya han sido utilizadas en algunos scripts mostrados anteriormente. Estar anidada implica que mientras una se está ejecutando, otra también lo está haciendo. Veamos una transposición didáctica. Podemos plantear en la vida cotidiana comportamientos recursivos anidados.

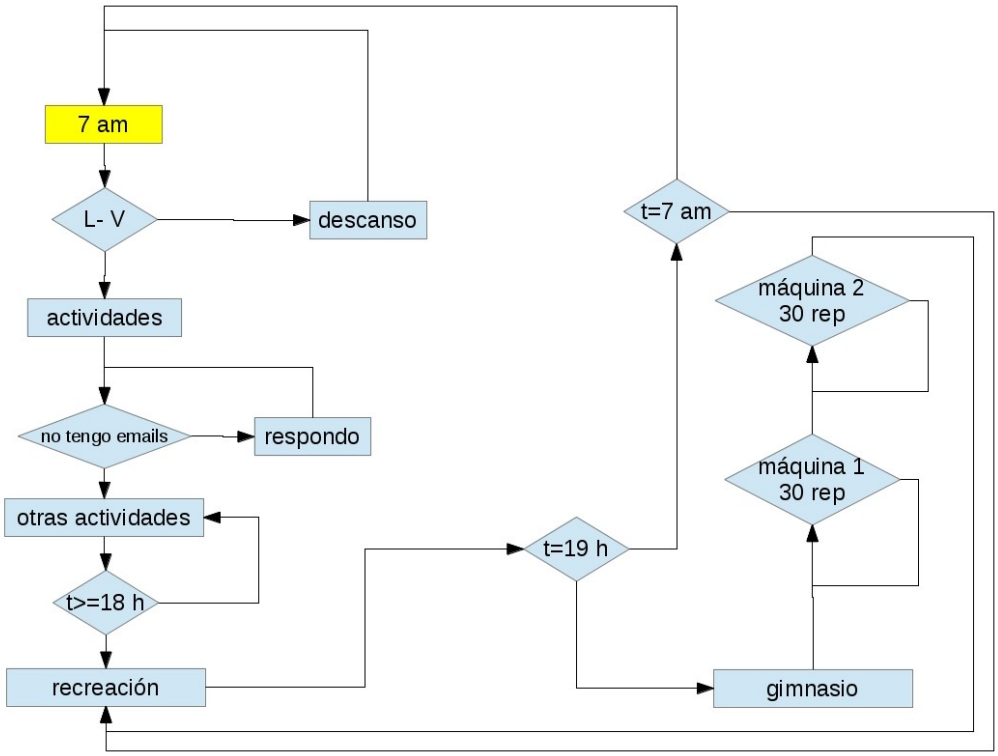
Procedimiento 1: cada día inicio mi trabajo a las 7 am y lo finalizo a las 6 pm. Este proceso lo repito cada 24 h, los 365 días del año de lunes a viernes. Sábados y domingo si bien inicio el día a las 7 am, no realizo las actividades comunes de los lunes a viernes. Este procedimiento lo podemos entender como un procedimiento **for** que se ejecuta de 1 a 365, cada año. Dentro de

este for, existe un **if**: si es día de lunes a viernes realizaré ciertos trabajos, entre ellos responder los mails a las 8 am, en cambio, si es sábado o domingo ejecuto otras actividades.

Procedimiento 2: dentro de cada día de los lunes a viernes, a las 8 am consulto mi correo electrónico. Si no hay emails sigo con las actividades, pero si existen mails, tomo el primer mensaje recibido, lo leo, interpreto y: descarto si es no deseado o respondo si corresponde. Paso al segundo mensaje y procedo de igual manera. Repito este procedimiento hasta llegar al último mensaje y luego salgo de esta actividad recursiva.

Procedimiento 3: ir al gimnasio. Los días de lunes a viernes voy al gimnasio a las 19 h. Supongamos que utilizo solo dos máquinas diferentes en forma secuencial, en los que realizo 30 repeticiones con cada uno. Cuando llego a la repetición 30 en un aparato, dejo ese aparato y sigo con el siguiente. Cuando termino con la rutina regreso a las actividades del horario de descanso

El esquema siguiente nos muestra estas actividades.



Veamos un script con varias estructuras anidadas. En este script se utilizan también otros recursos como son el manejo de vectores y los índices del mismo que será discutido en

profundidad en próximas clases. Concéntrese en el análisis de las estructuras. En el script siguiente hay tres estructuras anidades: repeat (marcada en negrita), if (marcada en cursiva) y for (marcada en azul).

Básicamente, mientras la variable ***a*** sea igual a 1, se repetirá el bucle "repeat". Dentro de repeat hay una estructura if que se ejecutará si la variable ***sexo*** toma el valor h. Dentro de la estructura if hay un bucle for que se repetirá tres veces, preguntando edad, peso corporal y estatura.

script546

```
# El siguiente script tiene estructuras anidadas. En estos casos se recomienda la indentación de las mismas
a<-1
variables<-c("edad","peso corporal","estatura","sexo")
unidades<-c("años","kilogramos","metros")
datos<-c()
repeat{
        if(a!=1){break}
        print("Introduzca a través del teclado h si es hombre y m si es mujer")
        sexo<-scan(file = "", what = "",nmax=1)
                if(sexo=="h"){
                        for(i in 1:(length(variables)-1)){
                        writeLines(paste("Introduzca a través del teclado su ", variables[i], " en la unidad: ", unidades[i]))
                        datos[i]<-as.numeric(scan(file = "", what = "",nmax=1))
                        }
                datos[4]<-sexo
                }
print("Si requiere tomar otro dato introduzca 1, para salir oprima cualquier número")
a<-as.numeric(scan(file = "", what = "",nmax=1))
}
```

4.4. Bucles infinitos

Los bucles infinitos son consecuencias de errores de programación que conducen a que un bucle no pueda finalizar nunca. En el ejemplo siguiente tenemos uno de ellos. Cuando ocurre esto, para detener el scritp oprima <control + c>, ya que no se detendrá por sus propias instrucciones

script545	

# un script que no le alcanzará la vida para ver el mensaje final 	
print("A través del teclado introduzca un número positivo cualquiera")
numero<-as.numeric(scan(file = "", what = "",nmax=1))
a<-0
contador<-0
while(a<numero){
a=a+1
numero=numero+2
 	
contador=contador+1
writeLines(paste("numero de bucle: ", contador))
}
	inicializamos a través del teclado la variable número
	inicializamos la variable a con el valor 0
	inicializamos la variable contador con el valor 0. La variable contador
	mientras la variable ***a*** tenga un valor menor que el número introducido por el teclado, se le sumará 1 al valor de la variable ***a*** y se sumará 2 al valor de la variable ***número***. Es claro que como ***a*** comenzó del valor 0 y en cada bucle se le suma 1, mientras que a la variable ***número*** se le suman 2 unidades en cada bucle, siempre la variable ***a*** será menor que la variable ***numero***
	la variable contador tomará el número de bucle realizado

4.5. Errores en los scripts

Dentro de los scripts tenemos básicamente dos tipos de errores:
1- de sintaxis: son aquellos errores que producirán una falla en el funcionamiento y R nos avisará en la consola, dándonos indicadores claros de donde se halla dicho error.
Ejemplo 1: En el ejemplo siguiente vemos un script que produjo un error de sintaxis. Dicho error es marcado en qué línea se encuentra y precisamente donde está, que queda indicado con "^". Además nos dice que hay un "unexpected }". Esto nos está indicando que hay una "}" de más o nos falta la "{" de apertura

```
> source("script545")
Error in source("script545") : script545:11:1: unexpected '}'
10: writeLines(paste("numero de bucle: ", contador))
11: }
    ^
```

Ejemplo 2. Otro error en la sintaxis. No nos marca la línea pero nos indica que hay una función desconocida: "asnumeric". Claramente escribimos mal la función as.numeric.

> source("script545")

[1] "A través del teclado introduzca un número cualquiera"

Error in eval(expr, envir, enclos) : could not find function "asnumeric"

Ejemplo 3. No reconoce un objeto porque puede no estar definido o estar mal escrito

Error: object 'sexor' not found

En síntesis los errores de sintaxis serán visualizados y no conducirán a resultados a menos que sean solucionados. Los script que tienen errores de sintaxis, no se ejecutan, aunque pueden inicializar varaibles o crear objetos que estén anteriores al sitio del error.

Los errores más peligrosos son aquellos que no impiden la ejecución del script pero que llevan a mal almacenamiento de resultados o datos o falsas conclusiones.

script544	explicación
# Este script realiza una prueba t-student y le arroja las conclusiones	
# este script analiza los datos de una tabla llamada tablaR54.csv que se halla en su espacio de trabajo y realiza la prueba. No prueba supuestos de normalidad ni homocedasticidad	
tablaR54<-read.table("tablaR54.csv",sep="\ t",dec=",",header=TRUE)	introduce en el data.frame: tablaR54 los datos del archivo tablaR54.csv
ts<-t.test(tablaR54$var1[tablaR54$var2=="a"],tablaR54$var1[tablaR54$var2=="b"])	realiza la prueba t de Student y asigna su resultado al objeto ts
	Según el valor de ts$p.value da una u otra respuesta
if(ts$p.value>0.05){	respuesta si p>0.05
writeLines(paste("El valor de var1 difiere entre los tratamientos a y b, p.value= ",ts$p.value))	
}else{	respuesta si p<= 0.05
writeLines(paste("El valor de var1 no difiere entre los tratamientos a y b, p.value= ",ts$p.value))	
}	

Claramente el script comete un error, sin embargo se ejecuta sin indicar el mismo y solo un investigador conocedor de la estadística y despierto se dará cuenta del mismo. Note que si p.value>0.05 el mensaje que verá en pantalla será "El valor de var1 difiere entre los tratamientos a y b" y mostrará el valor de p.value. Sabemos que si p.value>0,05 no hay diferencias. En nuestro caso si ejecuta el script verá que el mensaje sera "El valor de var1 no difiere entre los tratamientos a y b, p.value= 0.00116406277189068 ", lo cual es incongruente, por un lado dice que no hay diferencias cuando el valor de p.value asi lo indica. El error surgió de escribir la estructura if de la siguiente manera

if(ts$p.value>0.05){

writeLines(paste("El valor de var1 difiere entre los tratamientos a y b, p.value= ",ts$p.value))

}else{

writeLines(paste("El valor de var1 no difiere entre los tratamientos a y b, p.value= ",ts$p.value))

}

Cuando debería haber sido

```
if(ts$p.value<0.05){
writeLines(paste("El valor de var1 difiere entre los tratamientos a y b, p.value= ",ts$p.value))
}else{
writeLines(paste("El valor de var1 no difiere entre los tratamientos a y b, p.value= ",ts$p.value))
}
```

Como dijimos, un buen observador y conocedor logrará detectar el error. Pero en otros casos ni aun así será fácil detectarlos. La ejecución de un script permite automatizarnos y no estar tan atentos, lo cual puede tener consecuencias dramáticas.

Un script es como comprarse una automóvil. Con él, usted llegará más rápido y más cómodo a muchos lugares, pero no por ello debe desentenderse del medioambiente y de estar más alerta que si se moviera en un sistema de transporte público.

5. Clase 5

Funciones incorporadas en clases anteriores
source("nombre del archivo de texto con el script a ejecutar")
print("texto a mostrar en pantalla")
print(valor de un objeto de R)
readline("Muestra en pantalla un texto y espera enter para continuar")
#
los textos antecedidos por # permiten introducir explicaciones en el script pero no se ejecutan.
scan: introduce un número o caracteres en la variable n

n<- as.numeric(scan(file = "", what = "",nmax=1))

n<- scan(file = "", what = "",nmax=1)

writeLines() permite mostrar un texto o texto con variable combinada con la función paste()

writeLines("texto")

writeLines(paste("texto ", nombrevariable))

Sys.sleep(n)

introduce una pausa automática de n segundos de retardo entre las órdenes consecutivas

alarm()

genera un beep por sus parlantes

beep(sound=number)

genera otros sonidos a elección dentro de una biblioteca

Control + c

detiene el script

Controles de flujo del script

if-else

permite elegir dos ejecuciones diferentes según se cumpla o no una condición

if
permite evaluar una condición y actuar o no según el valor de la misma.
for
permite repetir una secuencia de ordenes un número preestablecido de veces.
while
permite repetir una rutina hasta que una variable tome un valor predeterminado o se cumpla una dada condición. Antes de ejecutar el bucle while se evalua la condición.
repeat
Permite repetir una rutina hasta que una variable tome un valor predeterminado o se cumpla una dada condición. La condición se evalúa al finalizar el bucle y allí se decide si se continua o

no en el bucle.
Recuerde que para ejecutar un script, el archivo que contiene las instrucciones, lo que llamamos script, debe estar ubicado en la misma carpeta en que se halla nuestro espacio de trabajo.

5.1. Manejo de vectores

Como ya hemos visto un vector es un objeto que contiene elementos de un mismo tipo, aunque puede haber vectores con elementos de diferente tipo. Los elementos más comunes son strings o cadenas de caracteres y números.

Ejemplo de vector formado por strings es el caso del vector al que llamamos vehículos, sus elementos pueden ser ("auto","camión","moto"). Un ejemplo de vector con elementos numéricos, es el vector numeropar, cuyo elementos son (2,4,6).

Es habitual manejar los vectores dentro de scripts y es imprescindible conocer el concepto de índice de un vector. El índice indica la posición del elemento dentro del vector.

Veamos un ejemplo.

Tenemos un vector que inicializamos con los siguientes valores

V<-c(23,34,56)

Para el vector V, los índices pueden valer solo: 1, 2 y 3. El índice 1 corresponde al elemento 23, el índice 2 al 34 y el índice 3 al 56

Nombramos al índice con la letra i. El índice i puede tomar los siguientes valores: 1 ,2, 3.

Si escribimos

V[2]

obtendremos el valor

34

Si colocáramos por ejemplo

V[4]

obtendremos NA, ya que el vector no tiene 4 elementos

podemos saber la longitud y el valor máximo del índice de un vector con la función length()

> length(V)

[1] 3

El siguiente es un script que nos muestra los elementos de un vector y nos indica el índice de cada valor que compone el vector

script551	explicación

# Lector de vectores 	
# el script crea un vector de tres elementos 	
V<-c(23,34,56)
largovector<-length(V)
for (i in 1:largovector){
writeLines(paste("El elemento de ubicación: ", i, "en el vector V es: ", V[i]))
}
 	
	crea el vector con elementos fijos: 23, 34 y 56
	inicializa y asigna a la variable largovector la cantidad de elementos del vector V
	repetirá la rutina entre {} desde que i=1 hasta que i tome el valor de largovector, es decir la cantidad de elementos del vector V
	Imprime en la pantalla una combinación de strings y variables con la forma: "El elemento de ubicación: 1 en el vector V es: 23"

Cuando ejecute el script verá en la pantalla:

El elemento de ubicación: 1 en el vector V es: 23
El elemento de ubicación: 2 en el vector V es: 34
El elemento de ubicación: 3 en el vector V es: 56

El manejo del vector puede hacerse en el orden que se desee. Veamos una lectura en sentido contrario en el script552

script551	script552

# Lector de vectores 	
# el script crea un vector de tres elementos 	
V<-c(23,34,56)
largovector<-length(V)
for (i in 1:largovector){
writeLines(paste("El elemento de ubicación: ", i, "en el vector V es: ", V[i]))
}
	# Lector de vectores
	# el script crea un vector de tres elementos
	V<-c(23,34,56)
	largovector<-length(V)
	for (i in largovector:1){
	writeLines(paste("El elemento de ubicación: ", i, "en el vector V es: ", V[i]))
	}

Este script muestra los elementos del vector desde el último hasta el primer elemento. Usted deberá observar en la pantalla

> source("script552")

El elemento de ubicación: 3 en el vector V es: 56

El elemento de ubicación: 2 en el vector V es: 34

El elemento de ubicación: 1 en el vector V es: 23

Veamos uno que lea los elemento del vector V, pero no incluya el último elemento y comparémoslo con el script551

script551	script553

# Lector de vectores 	
# el script crea un vector de tres elementos 	
V<-c(23,34,56)
largovector<-length(V)
for (i in 1:largovector){
writeLines(paste("El elemento de ubicación: ", i, "en el vector V es: ", V[i]))
}
	# Lector de vectores
	# el script crea un vector de tres elementos
	V<-c(23,34,56)
	largovector<-length(V)-1
	for (i in 1:largovector){
	writeLines(paste("El elemento de ubicación: ", i, "en el vector V es: ", V[i]))
	}

En la pantalla deberemos tener

> source("script553")

El elemento de ubicación: 1 en el vector V es: 23

El elemento de ubicación: 2 en el vector V es: 34

como vemos, no mostró el tercer elemento. Esto se debe a que la variable largovector fue inicializada con un valor igual a la cantidad de elementos del vector, menos 1.

Podemos utilizar estos conceptos para combinar elementos de vectores. Supongamos que tenemos cuatro vectores cuyos nombre son: nombre, apellido, edad y sexo. El vector nombre, contiene el nombre de personas, el vector apellido contiene el apellido de la persona, el vector edad contiene la edad y el vector sexo contiene el sexo. Es una buena práctica para la simplicidad de los scripts que los objetos tengan nombres cortos pero representativos de su contenido. En el ejemplo que estamos planteando es más que evidente esta característica. El script siguiente permite introducir los datos de cada persona y formar los cuatro vectores. Al finalizar nos da un reporte de los individuos introducidos.

script554	explicación

# El siguiente script permite tomar datos de personas	
a<-1	La variable a nos permitirá permanecer o no dentro del bucle repeat
nombre<-c()	Inicializamos un vector que tendrá los nombre de los individuos, en principio sin ningún elemento
apellido<-c()	Inicializamos un vector que tendrá los apellidos de los individuos, en principio sin ningún elemento
edad<-c()	Inicializamos un vector que tendrá la edad de los individuos, en principio sin ningún elemento.
sexo<-c()	Inicializamos un vector que tendrá el sexo de los individuos, en principio sin ningún elemento.
readline("Este script le permite tomar nombre, apellido, edad y sexo de personas. Oprima enter para comenzar")	Nos avisa que vamos a hacer y debemos oprimir enter para comenzar
repeat{	
if(a!=1){break}	Si la variable a tomara un valor distinto de 1, el bucle repeat se interrumpirá
i=length(nombre)+1	Inicializa y asigna a la variable i un valor una unidad mayor que el número de elementos del vector nombre. Todos los elementos tendrán igual longitud. De esta manera los valores que se introduzcan a continuación lo harán en una posición siguiente del vector sin sobreescribir valores.
print("Introduzca el nombre")	Permite introducir por el teclado el nombre.
nombre[i]<-scan(file = "", what = "",nmax=1)	
print("Introduzca el apellido")	Permite introducir por el teclado el apellido.
apellido[i]<-scan(file = "", what = "",nmax=1)	
print("Introduzca la edad en años")	Permite introducir por el teclado el edad.
edad[i]<-scan(file = "", what = "",nmax=1)	
print("Introduzca h si es hombre o m si es mujer")	Permite introducir por el teclado el sexo.
sexo[i]<-scan(file = "", what = "",nmax=1)	Si introducimos 1, la variable a volverá a tomar 1 y se repetirá el bucle repeat. Si introducimos cualquier otro numero, la variable a tomará este valor y se interrumpirá el bucle repeat en la sentencia "if(a!=1){break} "
print("Si requiere tomar otro dato introduzca 1, para salir oprima cualquier número")	
a<-as.numeric(scan(file = "", what = "",nmax=1))	
}	Imprime en pantalla
print("DATOS DISPONIBLES EN SU BASE DE DATOS")	"\n" introduce un retorno de carro o salto de línea.
writeLines(paste("\n"))	Imprime en pantalla las palabra: apellido, nombre, edad y sexo, separadas por una tabulación. "\t" da la orden para tabulador.
writeLines(paste("apellido","\t","nombre","\t","edad","\	Imprimirá en cada línea de la pantalla el apellido,

Lo que nos muestre el script al finalizar a través de la pantalla puede carecer de estética. Lo que usted obtendrá es

[1] "DATOS DISPONIBLES EN SU BASE DE DATOS"

apellido	nombre	edad	sexo
bb	aa	45	h
ee	ee	45	m

Puede que le quede corrido por cuestiones de tabulador. Esto tiene solución con un poco más de trabajo, código e imaginación. La estética la dejaremos para más adelante.

Ya está en condiciones de comenzar a construir sus primeras bases de datos, con un dispositivo de incorporación de datos que le pregunte cada dato a introducir. Pruebe con algunos de sus datos.

5.2. Manejo de data.frames

El manejo de data.frames a través de script es común y muy necesario. Como la lectura y escritura de vectores se puede realizar con bucles for(), salvo que a diferencia de los vectores que utilizan un bucle for() para data.frames es común la utilización de dos bucles.
Veremos un script que nos permite listar los elementos de un data.frame, indicando que elementos se hallan en cada columna, siguiendo cada línea. Utilizaremos para este ejercicio la planilla de cálculo tablaR5-5.ods/xls. Allí hallará la tablaR557, la que debe grabar con formato .csv, con el nombre tablaR557.csv, que introduciremos con el propio script en el espacio de trabajo. Si tiene dificultades para crear el archivo .csv, utilice el que se provee con el material anexo al curso

script557	explicación

# script para leer un data frame 	
 	
datoscoches<-
read.table("tablaR557.csv",header=TRUE,sep="\t")
 	
for (i in 1:nrow(datoscoches)){
 	
writeLines(paste("datos línea ", i))
 	
for (j in 1:ncol(datoscoches)){
 	
writeLines(paste(datoscoches[i,j]))
	crea un data.frame llamado datoscoches
	
	Este bucle for lee cada línea y se repite hasta que i toma el valor del número de líneas del data.frama. "i" va tomando el valor de cada línea. Comienza con 1 (primer línea) y figura con nrow(datoscoches), que es el número de líneas del data.frame.
	
	Cada vez que ingrese al bucle imprimirá "datos linea " seguido de un número que será el valor de i.
	
	Este bucle for lee cada columna y se repetirá tantas veces como columnas tenga el data.frame. Cada vez que se repita escribirá en la pantalla el valor que tiene el data.frame en la fila i y la columna j.

} }	

Cuando ejecute el script obtendrá el listado de cada vehículo, patente, propietario y año.

5.3. Evitar errores

Como vimos en clases anteriores los scripts puede ocasionar errores que detengan el script y por ende acarren pérdida de tiempo o datos. Independientemente de que no se pierda nada, cuando un script se detiene es molesto. Veamos como evitarlo. Para ellos utilizaremos la función try()

El código general de la función es

try(código a ejecutar con posible error,silent=TRUE o FALSE)
Si cometiera un error en el código indicado, el script continuará igualmente. Avisándonos o no del error, dependiente que el argumento silent tome el valor FALSE o TRUE, respectivamente. Pero, no se detendrá el script.
Un sitio común en muchos programas es el error al intentar calcular logaritmos de números menores que 0 o divisiones por 0. En estos casos R no emite un error sino que indica un valor estimado por su límite (tema que no trataremos en este curso). Sin embargo existe riesgo de errores al inicializar o asignar valores a variables. Veamos un caso en que una variable A debería tener valores numéricos.
El script555, pide un número y le suma 2 mostrando el resultado. Luego, utilizando un bucle repeat nos permite repetir el procedimiento. Mientras lo deseemos, deberemos introducir un valor numérico (1 para continuar) o cualquier otro (para detenerlo). Si en este punto introdujéramos por ejemplo "a" que no es numérico, dará un error y se abortará el script.

script555	

#este script permite introducir números y nos muestra su logaritmo
A<-1
repeat{
print("introduzca un número")
numero<-as.numeric(scan(file = "", what = "",nmax=1))
resultado<- 2 + numero
writeLines(paste("Usted introdujo el número: ", numero, ", el resultado es: ", resultado))
print("Si desea introducir otro número oprima 1, si no lo desea oprima 0")	

A<-as.numeric(scan(file = "", what = "",nmax=1))
if(A!=1){break}
 	
}
	Si A toma un valor distinto de 1 se detendrá el script, pero si erróneamente oprimimos una letra en lugar de 1 dará un error y se detendrá cuando no era lo que deseábamos.

Cuando introducimos un caracter que no es numérico nos dará un error, que vemos a continuación

[1] "Si desea introducir otro número oprima 1, si no lo desea oprima cualquier otro número"

1: a

Read 1 item

[1] "introduzca un número"

Error in if (A != 1) { : missing value where TRUE/FALSE needed

In addition: Warning message:

NAs introduced by coercion

y se cortó el script. Resultando bastante molesto.

Utilicemos las función try() para evitar que el script se corte. Si introdujéramos con el teclado un valor no numérico, por ejemplo la letra q, al evaluar la sentencia if(), si bien cometerá un error la saltará y continuará sin avisar que hubo un error. Si hubiera colocado el argumento silent=FALSE. Nos avisaría del error, pero el script seguiría en marcha

script555	script556
#este script permite introducir números y nos muestra su logaritmo	#este script permite introducir números y nos muestra su logaritmo
A<-1	A<-1
repeat{	repeat{
print("introduzca un número")	print("introduzca un número")
numero<-as.numeric(scan(file = "", what = "",nmax=1))	numero<-as.numeric(scan(file = "", what = "",nmax=1))
resultado<- 2 + numero	resultado<- 2 + numero
writeLines(paste("Usted introdujo el número: ", numero, ", el resultado es: ", resultado))	writeLines(paste("Usted introdujo el número: ", numero, ", el resultado es: ", resultado))
print("Si desea introducir otro número oprima 1, si no lo desea oprima 0")	print("Si desea introducir otro número oprima 1, si no lo desea oprima 0")
A<-as.numeric(scan(file = "", what = "",nmax=1))	A<-as.numeric(scan(file = "", what = "",nmax=1))

if(A!=1){break} }	try(if(A!=1){break},silent=TRUE) }

Este script al producirse el error ni nos avisa del error, pero sigue ejecutando el script. Nos pedirá nuevamente el número.

5.4. Warnings en R

Los warnings son alertas que puede darnos R luego de ejecutar un dado comando. Que si bien no es un error, nos está alertando de algo en particular. En algunas situaciones como puede ser dentro de un script, la aparición de un warning puede ser confusa, especialmente si el script lo utiliza una persona que es usuaria pero no experta en el tema como para poder interpretarla.

Puede ser de utilidad en un script invalidar el uso de warnings, permanente o temporariamente.

Suprimir warnings en forma global, es decir para todas las funciones. No es recomendado pero para hacerlo debe escribir

options(warn=-1)

Para reponer la opción que nos muestre los warnings

options(warn=0)

Veamos un ejemplo en un script sin anular los warnings

script558	

# script para sacar logaritmos 	
writeLines('intoduzca un número al que le desea obtener el logaritmo')
 	
a<-as.numeric(scan(file='',what='',nmax=1))
writeLines(paste('El logaritmo del número ', a, 'es ', log(a)))
	nos pide un número. Toda vez que intoduzcamos un número mayor que cero tendremos un resultado numérico. Sin embargo si introducimos un número negativo el resultado sera NaN (not a number) y nos avisará con un warning()

prueba a introducir el número 10 por ejemplo. Obtendrá

> source('script558')

intoduzca un número al que le desea obtener el logaritmo

1: 10

Read 1 item

El logaritmo del número 10 es 2.30258509299405

pero si introduce -3, resultará

> source('script558')

intoduzca un número al que le desea obtener el logaritmo

1: -3

Read 1 item

El logaritmo del número -3 es NaN

Warning message:

In log(a) : NaNs produced

Si especulamos que el usuario del script puede sorprenderse y no saber que hacer luego de ver el Warning, podemos desactivarlo al principio del script y activarlo al finalizar, como podemos ver en el script559

script559	

# script para sacar logaritmos 	
options(warn=-1)
writeLines('introduzca un número al que le desea obtener el logaritmo')
a<-as.numeric(scan(file='',what='',nmax=1))
writeLines(paste('El logaritmo del número ', a, 'es ', log(a)))
 	
options(warn=0)
	
	anula la posibilidad de mostrar warnins
	nos pide un número. Toda vez que intoduzcamos un número mayor que cero tendremos un resultado numérico. Sin embargo si introducimos un número negativo el resultado sera NaN
	Pero no nos mostrará el warning
	Restablece la opción de mostrar warnings. La opción solo está presente durante este script

probemos a introducir 10, que sabemos nos da un número y no warnings.

> source('script559')

introduzca un número al que le desea obtener el logaritmo

1: 10

Read 1 item

El logaritmo del número 10 es 2.30258509299405

Ahora probemos con -3 que dara un NaN y por otra parte tendríamos un Warning.

> source('script559')

intoduzca un número al que le desea obtener el logaritmo

1: -3

Read 1 item

El logaritmo del número -3 es NaN

Como vemos ahora no hemos tenido el warning.

Si desea suprimir los warnings generados por una determinada expresión utilice

suppressWarnings(expr)

Veamos el script5510

script5510	

# script para sacar logaritmos 	
writeLines('intoduzca un número al que le desea obtener el logaritmo')
a<-as.numeric(scan(file='',what='',nmax=1))
 	
suppressWarnings(writeLines(paste('El logaritmo del número ', a, 'es ', log(a))))
	nos pide un número. Toda vez que introduzcamos un número mayor que cero tendremos un resultado numérico. Sin embargo si introducimos un número negativo el resultado sera NaN
	Pero no nos mostrará el warning, ya que el calculo del logaritmo se hace dentro de la función suppressWarnings()

probemos su funcionamiento con el número 10 que sabemos que funciona y el log nos da un número

> source('script5510')

intoduzca un número al que le desea obtener el logaritmo

1: 10

Read 1 item

El logaritmo del número 10 es 2.30258509299405

ahora probemos con el número -3, que sabemos que produce un resultado NaN y además debería dar un Warnings

> source('script5510')

intoduzca un número al que le desea obtener el logaritmo

1: -3

Read 1 item

El logaritmo del número -3 es NaN

como vemos no nos mostró el warning. Sin embargo, con esta forma de proceder la anulación del warning es solo para la expresión y no tengo necesidad de reactivarlo. Podemos comprobar que el sistema de warnings está activo probando obtener el logaritmo de -3

> log(-3)

[1] NaN

Warning message:

In log(-3) : NaNs produced

6. Clase 6

R nos proporciona una gran cantidad de bibliotecas gráficas especializadas y generales. Cuando se realiza la instalación se cuenta con la biblioteca graphics que tiene los recursos básicos e indispensable para una primer representación de nuestros datos, para así proceder luego al análisis. Con esta biblioteca y los recursos analíticos en general es suficiente para realizar el análisis e interpretación de resultados. Algunos paquetes especiales como pROC que hemos abordado en otros módulos de este curso, además nos proveen de gráficas especiales para ciertos análisis de datos. Sin embargo, a la hora de mostrar nuestro resultados en congresos, publicaciones o seminarios podemos requerir o desear una mejor y más clara presentación. R nos provee de herramientas para construir nuestros gráficos a gusto y voluntad.

Estas herramientas prácticamente nos permiten ir dibujando sobre un sistema de ejes coordenados, nuestros datos en el formato gráfico que deseemos. La combinación de estos recursos con los scripts nos permitirán mostrar nuestros resultados con el detalle que deseemos.

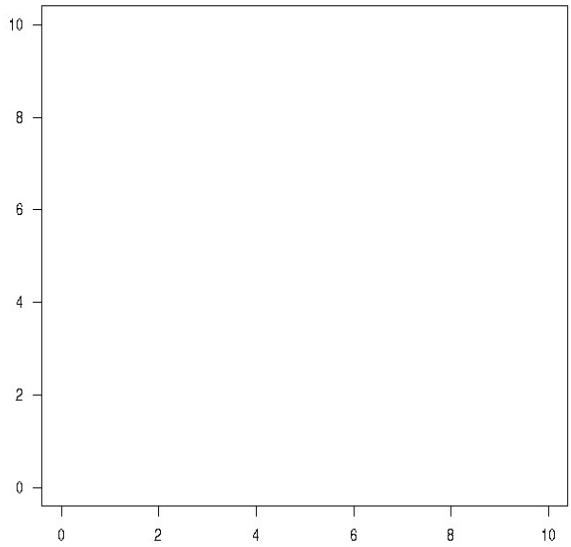
Los elementos básicos que veremos son: rectángulos, polígonos, círculos, líneas y flechas. Con ellos luego usted podrá crear el gráfico deseado.

Elementos básicos de dibujo

6.1. Rectángulos

Si deseamos dibujar un rectángulo lo primero que tenemos que hacer es crear un gráfico vacío, con ciertos límites en sus ejes. Por ejemplo creamos un gráfico vacío, sin denominación de sus ejes (xlab="", ylab=""), sin ningún punto (type="n"), con los rótulos de los ejes en forma horizontal (las=1) y donde ambos ejes tiene una escala de 0 a 10: xlim=c(0,10), ylim=c(0,10)

```
> plot(0,0,type="n",xlim=c(0,10),ylim=c(0,10),las=1,xlab="",ylab="")
```



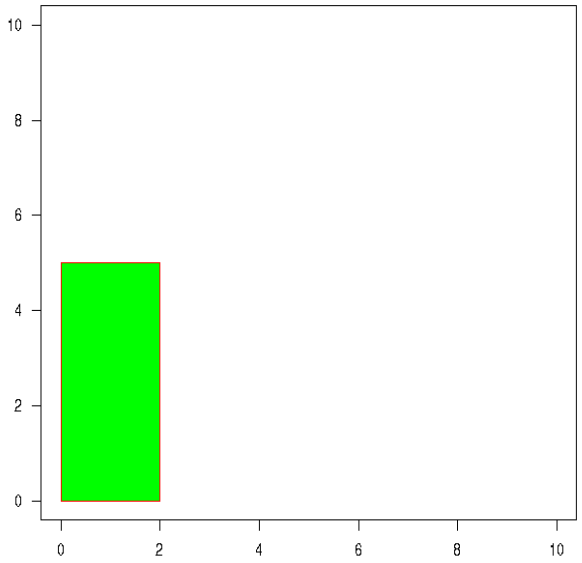
por supuesto que puede agregarle o modificarle los elementos a esta gráfica. Solo tendrá que revisar conceptos de módulos anteriores.

Si queremos dibujar un rectángulo utilizaremos la función rect(). Lo básico de esta función es que le indicamos las coordenadas de sus vértices inferior izquierdo y superior derecho, utilizando los valores de los ejes horizontal y vertical. La forma general de la función es

rect(xleft, ybottom, xright, ytop, density = nº, angle = nº , col = nº , border = nº, lty = nº , lwd = nº)

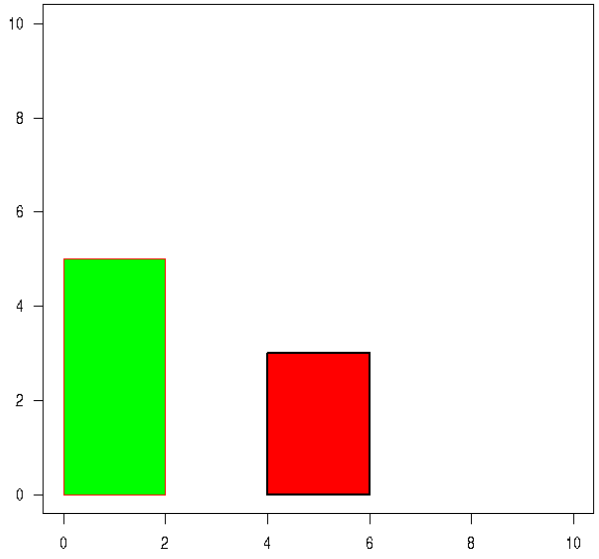
Supongamos que deseamos dibujar un rectángulo donde el vértice inferior-izquierdo esta en el punto (0,0) y el vértice superior-derecho en (2,5), de color verde de relleno y bordes rojos

```
> rect(0,0,2,5,col="green",border="red")
```



Si deseara construir otro rectángulo que se extienda de 4 a 6 de base y hasta la altura tres, de color rojo pero con línea negra y más gruesa, puede utilizar el siguiente código

> rect(4,0,6,3,col="red",border="black",lwd=2)



Más adelante veremos esta función para mostrar nuestros resultados.

6.2. líneas y segmentos

De la misma manera que para hacer rectángulos cuando deseamos hacer líneas debemos crear un gráfico vacío con ciertos argumentos y luego utilizaremos para dibujar la línea la función segments(). Repitamos el anterior

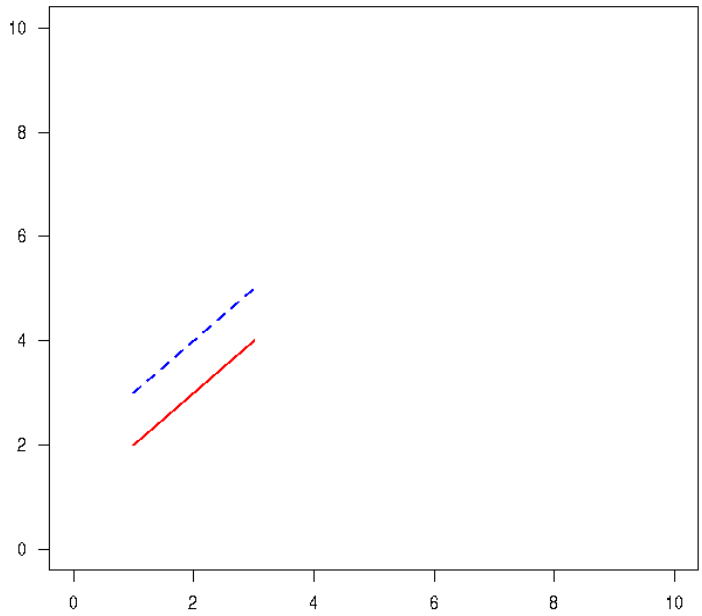
```
> plot(0,0,type="n",xlim=c(0,10),ylim=c(0,10),las=1,xlab="",ylab="")
```

La función general para dibujar un segmento requiere indicar las coordenadas de un extremos: x0, y0 y las coordenadas finales

segments(x0, y0, x1, y1, col = nº, lty = nº , lwd = nº)

El código siguiente dibuja segmento desde los puntos (1,2) al (3,4) de color rojo y el siguiente otro segmento de los puntos (1,3) al punto (3,5), siendo el segundo azul y de línea de puntos

```
> segments(1,2,3,4,col="red", lwd=2)
> segments(1,3,3,5,col="blue", lwd=2,lty=2)
```



Más adelante utilizaremos la función segments() para construir nuestros gráficos a la hora de mostrar los resultados

6.3. flechas

Otro de los recursos gráficos son las flechas, que se dibujan utilizando la función arrows(). Estas se definen indicando los dos puntos entre los cuales se extiende (x0,y0) a (x0,y1).
El formato general es

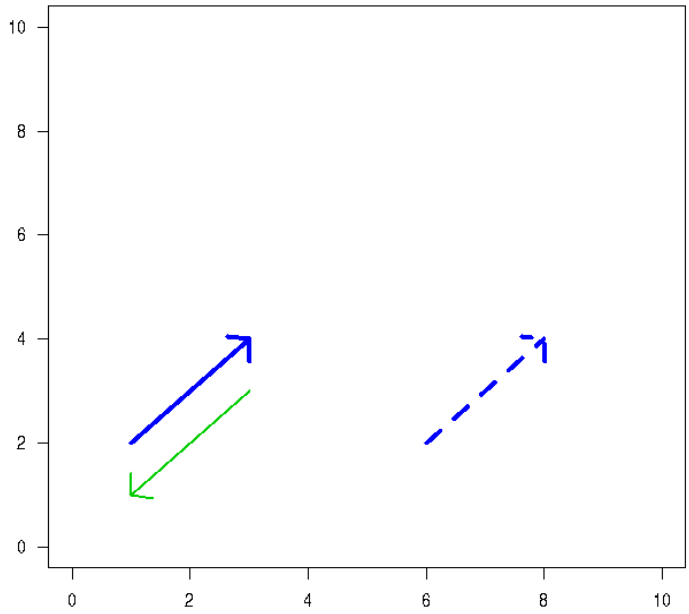
arrows(x0,y0,x1,y1,length= n,angle=n, code=1, lwd=nº, col= nº)

El parámetro length indica la longitud de la punta de flecha con un número "n". El ángulo de la flecha también se indica con un número "n". El parámetro code indica si va del punto (x0,y0) a (x1,y1) o viceversa. Los parámetros lwd, col, lty son parámetros que indican grosor, color y tipo de línea, respectivamente. code=1, la flecha va del punto (x1,y1) al punto (x0,y0). Code=2, la flecha del punto (x0,y0) al (x1,y1). Para recordar con más facilidad, code=1, tiene la punta de la flecha en el primer punto, code=2, tiene la punta de flecha en el segundo punto.
Definamos un gráfico vacío con las mismas condiciones anteriores
y dibujemos las siguientes flechas

```
> plot(0,0,type="n",xlim=c(0,10),ylim=c(0,10),las=1,xlab="",ylab="")
```

> arrows(1,1,3,3,length=0.2,angle=50,code=1,lwd=2,col=3)
> arrows(1,2,3,4,length=0.2,angle=50,code=2,lwd=4,col="blue")
> arrows(6,2,8,4,length=0.2,angle=50,code=2,lwd=4,col="blue",lty=2)

obtenemos el siguiente gráfico



6.4. Círculos

Para dibujar círculos utilizamos la biblioteca tripack
> library(tripack)

La función que utilizamos para hacer un círculo es circles(). El código general de la función indica el centro (x0,y0) y el radio: r, además de algunas características de la línea

circles(x0,y0,r,lwd=n,lty=n,col="color")

Veamos una sucesión de círculos para comprender su utilización

Creamos un gráfico vacío

> plot(0,0,type="n",xlim=c(0,10),ylim=c(0,10),las=1,xlab="",ylab="")

realizamos un círculo cuyo centro está en el punto (1,1) y su radio es 1.

> circles(1,1,1)

realizamos un círculo cuyo centro está en el punto (2,2), con radio 1 y color rojo.

```
> circles(2,2,1,col="red")
```

realizamos un círculo cuyo centro está en el punto (5,5), con radio 1, color rojo y tipo de línea 2.

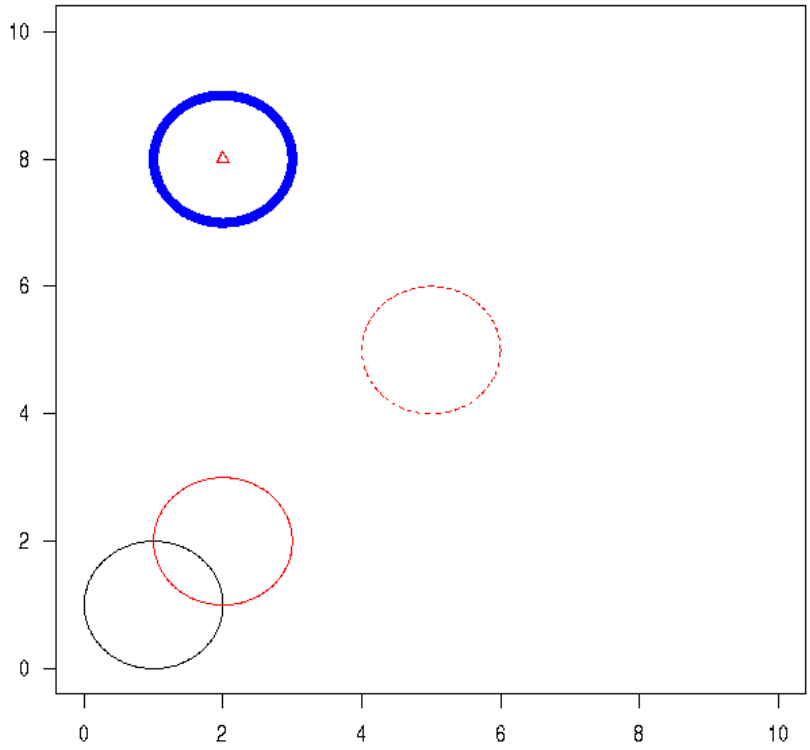
```
> circles(5,5,1,col="red",lty=2)
```

realizamos un círculo cuyo centro está en el punto (2,8), con radio 1, color azul,tipo de línea 1, y grosor 7.

```
> circles(2,8,1,col="blue",lty=1,lwd=7)
```

Podemos si lo deseamos agregar el centro de la última circunferencia, con la función points()

```
> points(2,8,col="red",pch=2)
```



6.5. Polígonos

La función polygon() nos permite definir un polígono de la cantidad de vértices deseados simplemente indicando la posición de los mismos. Además podemos asignarle un relleno de rallas con diferentes inclinaciones y color

El formato general de la función es

polygon(x, y = NULL, density = n, angle = n, border = n, col = "color", lty = n)

x e y vector con los vértices.

density: densidad de líneas de relleno.

angle: la pendiente de las líneas de relleno.

col= color del relleno.

border: color de la línea del perímetro.

Veamos un ejemplo de polígono

Primero creamos un gráfico vacío

```
> plot(0,0,type="n",xlim=c(0,10),ylim=c(0,10),las=1,xlab="",ylab="")
```

creamos un vector con los valores de x de cada uno de los vértices, en este caso haremos un polígono de 4 vértices

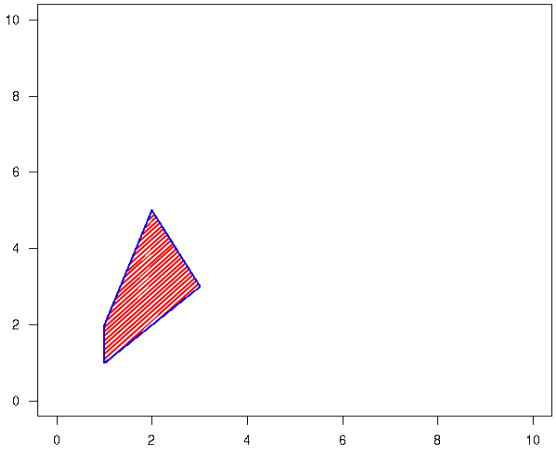
```
> x<-c(1,1,2,3)
```

creamos un vector con los valores de y de cada vértice

```
> y<-c(1,2,5,3)
```

en lugar de dos vectores se podría haber construído un data.frame con una columna con las x y en otra los valores de y de cada vértice.

```
> polygon(x,y,density=25,col="red",border="blue",lwd=2)
```



6.6. Puntos

La colocación de puntos ya lo hemos incorporado en otro módulo y revisamos en este momento. La función general es

points(x0,y0,pch=nº,col=nº,cex=nº....)

donde pch, col y cex indican el tipo de punto, el color y el tamaño respectivamente.

Veamos un ejemplo

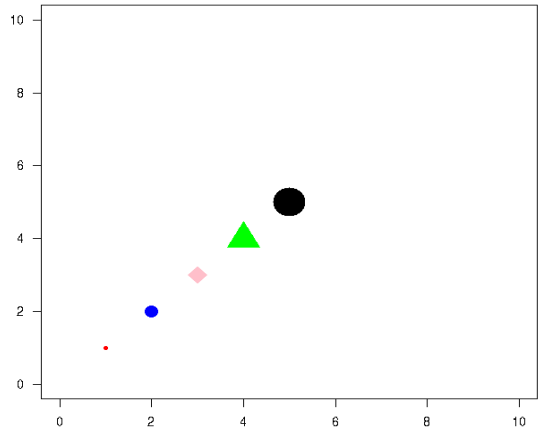
Definimos un gráfico similar a los casos anteriores

```
> plot(0,0,type="n",xlim=c(0,10),ylim=c(0,10),las=1,xlab="",ylab="")
```

y colocamos diferentes puntos para comprender los parámetros

```
> points(1,1,pch=20,col="red",cex=1)
> points(2,2,pch=19,col="blue",cex=2)
> points(3,3,pch=18,col="pink",cex=3)
> points(4,4,pch=17,col="green",cex=4)
> points(5,5,pch=16,col="black",cex=5)
```

luego de ejecutar todos los códigos mencionados obtendremos

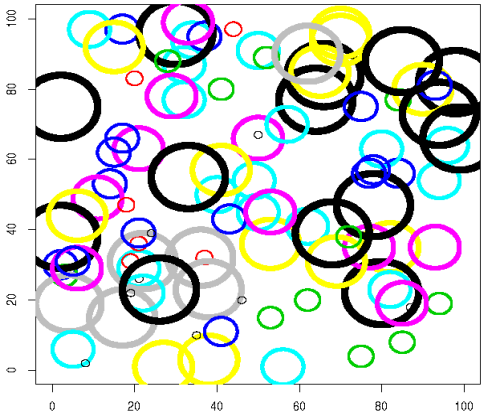


Veamos algunas aplicaciones con gráficos y scripts. En la próxima clase aplicaremos lo aprendido a construir gráficas personalizadas profundizar en la programación.
El siguiente script dibujará círculos de diámetros, centros, colores y grosores distintos de manera aleatoria en un gráfico. Llamaremos a nuestro script círculos danzantes

script561	explicación
#circulos danzantes	
plot(0,0,type="n",xlim=c(0,100),ylim=c(0,100),xlab="",ylab="")	crea un gráfico vacío de límites 0-100 para ambos ejes
n=0	inicializamos una variable n en un valor 0
while(n<100){	bucle while que se repetirá 100 veces
i<-trunc(runif(1,1,10))	en cada bucle asigna a i un valor aleatorio entero entre 1 y 10
print(i)	muestra el valor de i en la pantalla
circles(trunc(runif(1,1,100)),trunc(runif(1,1,100)),i,col=i,lwd=i)	crea un círculo de centro aleatorio entre 1 y 100 para x e y con radio i cuyo color y grosor de línea cambia según el valor de i
n<-n+1	renueva la variable n a una unidad mayor
Sys.sleep(0.2)	retarda la ejecución 0,2 segundos
}	

Se obtendrá la siguiente gráfica final, aunque durante la ejecución irán apareciendo círculos de diferente color, radio, posición y grosor de líneas. La velocidad a la que aparecen los círculos dependera del tiempo que le asignamos

a la función Sys.sleep()

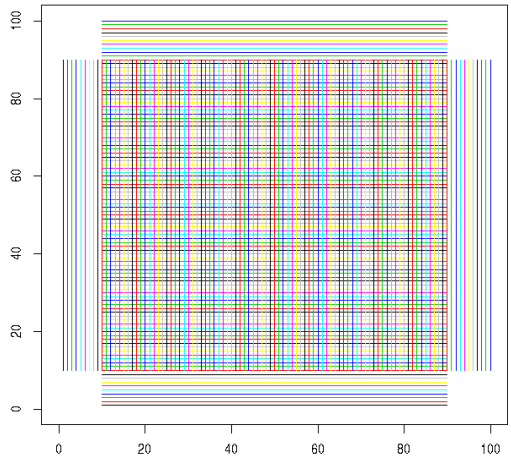


Supongo que ya se imagina que show podrá hacer al mostrar sus resultados. Aunque no es una cosa sencilla, el uso cotidiano lleva a un manejo muy seguro y eficiente de gráficos y scripts. Veamos otro ejemplo para ir comprendiendo el manejo gráfico con scripts. Este script dibujará un segmento ascendente con cambios de colores y luego verticales que se generan de izquierda a derecha.

script562	explicación

#segmentos ascendentes
plot(0,0,type="n",xlim=c(0,100),ylim=c(0,100),xlab="",ylab="")
for (i in 1:100){
segments(10,i,90,i,col=i)
 	
Sys.sleep(0.2)
}
for (i in 1:100){
segments(i,10,i,90,col=i)
 	
Sys.sleep(0.2)
}
	Crea un gráfico x-y vacío de límites 100x100
	Ejecuta buble 100 veces
	dibuja un segmento con extremos en x=10 y 90, pero en el eje y ascendente. Además cambia el color en cada bucle tomando el valor del bucle ejecutado.
	Hace una pausa de 0,2 seg entre segmento ysegmento
	Incia otro bucle que se repetirá 100 veces
	dibuja un segmento con los extremos constantes en y=10 e y=90, pero la posición en x va aumentando al ritmo de i.
	Pausa de 0,2 seg

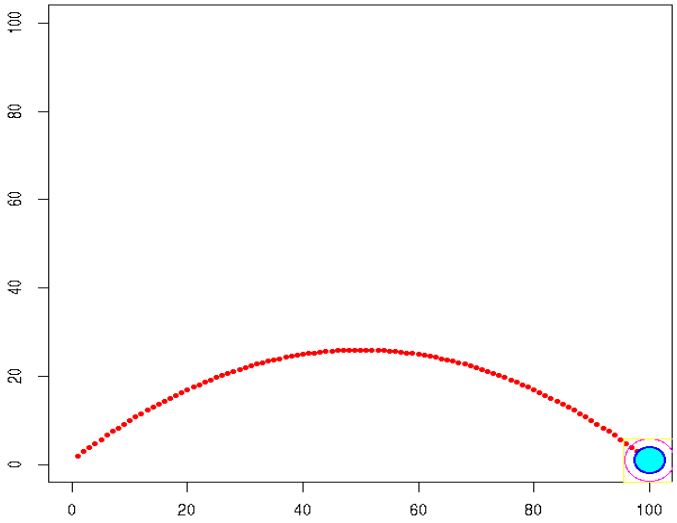
Se obtendrá la siguiente gráfica final, aunque la misma se modificará durante la ejecución



El siguiente script grafica punto a punto una función parabólica e indica el final con efectos especiales. Ideal para simular resultados.

script563	explicación
#la parábola	
f<-function(x){-0.01*x^2+x+1}	definimos una función $y=-0,01*x^2 + x + 1$
plot(0,0,type="n",xlim=c(0,100),ylim=c(0,100),xlab="",ylab="")	REaliza un gráfico vacío de x(0,100) e y(0,100)
for (i in 1:99){	Realiza un bucle 99 veces donde i toma valores de 0 a 99
points(i,f(i),pch=20,col="red")	grafica puntos en color rojo, donde las coordenadas de cada punto son (i,f(i)), es decir cada punto tiene como abscisa al valor i y como ordenada al valor de la función reemplazando en x el valor i.
Sys.sleep(0.1)	Hace una pausa de 0,2 seg lo que le da el aspecto de desarrollo de proceso
}	
for(i in 1:7){	Realiza otro bucle que se repite 7 veces donde grafica siempre el punto en el valor x=100 e y= f(100), pero el punto cambia en cada ciclo asi como el color y el tamaño.
points(100,f(100),pch=15+i,col=i,cex=i)	cada ciclo da dos alarmas
alarm()	Da un espacio de 0,2 segundos entre cada punto.
alarm()	
Sys.sleep(0.2)	
}	

La gráfica final del proceso es


0
20
40
60
80
100
0
20
40
60
80
100

7. Clase 7

Si bien R proporciona muchas bibliotecas que permiten hacer gráficos, la construcción de estos a través de scripts permiten crearlos a voluntad. En principio, podrá resultarle el mecanismo, un tanto complejo y sin futuro, pero la persistencia en el trabajo le darán grandes beneficios. La suma de recursos a cada script permitirá ir haciendo a los mismos más versátiles. Por otra parte el uso de scripts se irá haciendo común y no solo lo aplicará a gráficos sino también al procesamiento, análisis, presentación y almacenamiento de sus datos.
En esta clase veremos la construcción de dos tipos de gráficas: de barras y de puntos apareados.
Analizaremos en esta clase dos tablas de datos que se adjuntan entre material de esta clase, la planilla de cálculo tablaR5-7, donde hallará dos tablas graficobarras y glucemia. Introdúzcalas en su espacio de trabajo con los mismos nombres, utilizando el código
graficobarras<-read.table('clipboard',header=T,dec=',',sep='\t',encoding='latin1')
comprobamos el ingreso de datos

```
> summary(graficobarras)
 tratamiento    variable
 A:10         Min.   :2.000
 B:10         1st Qu.:4.000
 C:10         Median :6.000
              Mean   :5.867
              3rd Qu.:8.000
              Max.   :9.000
```

glucemia<-read.table('clipboard',header=T,dec=',',sep='\t',encoding='latin1')
comprobamos el ingreso de los datos

```
> summary(glucemia)
       n             trat        t              glucemia          rata
 Min.   : 1.00   g:10   Min.   : 0.0   Min.   :0.5300   Min.   : 1.0
 1st Qu.: 5.75   s:10   1st Qu.: 0.0   1st Qu.:0.7000   1st Qu.: 3.0
 Median :10.50          Median : 7.5   Median :0.7400   Median : 5.5
 Mean   :10.50          Mean   : 7.5   Mean   :0.7475   Mean   : 5.5
 3rd Qu.:15.25          3rd Qu.:15.0   3rd Qu.:0.8125   3rd Qu.: 8.0
 Max.   :20.00          Max.   :15.0   Max.   :0.8800   Max.   :10.0
```

La tabla graficobarras contiene datos de una medición llamada "variable" en 30 unidades experimentales, sometidas a tres tratamientos distintos: A, B y C. De la descripción anterior se deduce que no hay ningún otro factor en juego y que los valores de "variable" con un tratamiento son independientes de los valores de dicha variable con otro tratamiento. Con ella intentaremos hacer en base a un script un gráfico sencillo para comprender los recursos básicos, de allí para adelante estará en su deseo e imaginación hasta donde llegar.
La tabla glucemia contiene mediciones de la glucemia en 10 animales, sometidos a dos tratamientos (columna trat): s y g. Por otra parte a cada animal se midió la glucemia a tiempo cero y a los 15 minutos. Se trata de un estudio de datos apareados donde la glucemia a los 15

minutos no es independiente del valor a los 0 minutos y la forma de mostrarlo en gráficos es determinante. Como se puede ver en esta tabla para la rata nº 1 el valor a tiempo 0 se halla en la primer fila y el valor a los 15 min en la fila 11. De la misma manera están distribuidos los valores de glucemia para las otras ratas.
En primer lugar comprenda el experimento que dio origen a cada tabla, luego analice cada paso del script. Para comprender el funcionamiento del script es recomendable hacer cambios de a un argumento y observar la modificación que se observa en el gráfico.
Recuerde, la programación y el manejo de script requiere un tiempo de dedicación hasta sobrepasar cierto nivel desde donde todo fluye con facilidad!! Una vez alcanzado ese punto los beneficios de manejar este tipo de herramientas marca una gran diferencia en el manejo de datos.

7.1. Gráficos de barras

A continuación se muestra un código para la construcción de un gráfico de una variable con tres tratamientos. En la columna de la derecha se harán los comentarios. Copie el contenido de la segunda celda de la columna script571 (resaltada en amarillo) en un procesador de texto. Guarde el archivo con el nombre script571.txt, en el mismo directorio en que se hallará su espacio de trabajo

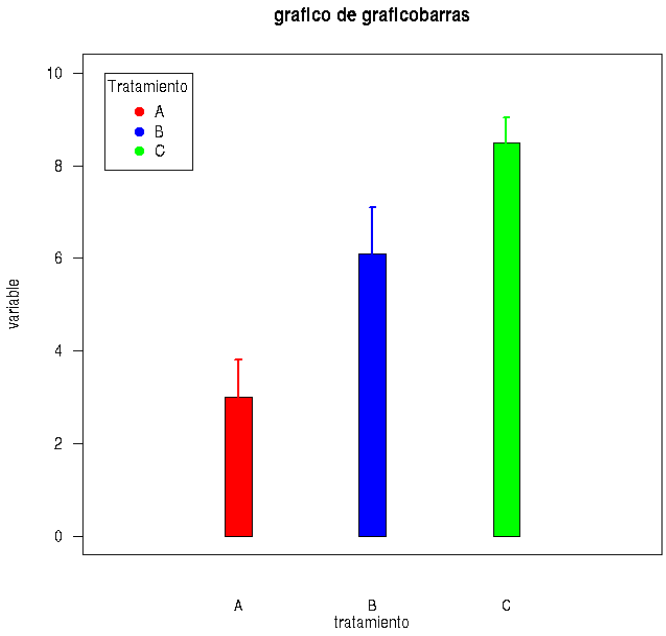
script571	explicación
#este script es útil para realizar gráficos de barras de una variable para varios tratamientos #si usted tiene una data.frame llamado "graficobarras" con valores de una variable llamada "variable" y una columna con tratamientos, llamada "tratamiento" realizará la grafica de las medias y los sd de x para cada tratamiento	
mean<-tapply(graficobarras$variable,graficobarras$tratamiento,mean)	Creamos un objeto llamado "mean" que tiene las medias de la variable para cada tratamiento
sd<-tapply(graficobarras$variable,graficobarras$tratamiento,sd)	Creamos un objeto llamado "sd" que tiene los desvíos estándar de la variable para cada tratamiento
color=c("red","blue","green")	El objeto color, tiene los colores en que queremos se vea cada barra
tratamiento<-levels(graficobarras$tratamiento)	El objeto tratamiento llevará los nombre de los niveles de la variable tratamiento del data.frame
d=0.2	La variable d, tomará el valor que deseamos de ancho de la línea que dibuja el desvío estándar
a=1	La variable ***a*** tomará el valor del ancho que le queremos dar a cada barra
par(xpd=TRUE)	Cambiamos el valor del parámetro xpd para poder dibujar fuera del gráfico.
plot(0,0,type="n",xlim=c(0,20),ylim=c(0,10),xlab="tratamiento",ylab="variable",xaxt="n",las=1, main="Gráfico de graficobarras")	Definimos un gráfico vacío con límites deseados para los ejes vertical y horizontal

legend(0,10,pch=c(19,19,19),legend=levels(graficobarras$tratamiento) ,col=color,title="Tratamiento",horiz=F)	Ubica la leyenda en posición 0,10 (pero podría cambiarla). Con pch fija el tipo símbolo para mostrar las series, y asigna con col los colores que tiene que objeto color definido anteriormente
j=5	La variable j indica en que valor del eje x se colocará cada rotulo. En el ciclo for siguiente se va cambiando
for (i in 1:3){	la función text coloca el rotulo de cada tratamiento en el eje horizontal
text(j,-1.5,labels=tratamiento[i],cex=0.9)	Asignamos nuevamente a j el valor 5
j=j+5	
}	De acá en adelante hace los puntos, los desvíos y las líneas de unión
j=5	realiza 3 bucles y en cada uno grafica
for (h in 1:3){	un rectángulo con máximo en el valor de la media del grupo
rect(j-a/2,0,j+a/2,mean[h],col=color[h],)	grafica un segmento del largo del sd para arriba
segments(j,mean[h],j,mean[h]+sd[h],col=color[h],lwd=1.7)	grafica un segmento horizontal
segments(j-d/2,mean[h]+sd[h],j+d/2,mean[h]+sd[h],col=color[h],lwd=1.7)	grafica otro segmento horizontal
segments(j-d/2,mean[h]-sd[h],j+d/2,mean[h]-sd[h],col=color[h],lwd=1.7)	
j=j+5	
}	

Ejecute el script

> source('script571')

como resultado de la ejecución del script, deberá obtener el siguiente gráfico



Regrese sobre el script, ponga la atención en el nombre de los objetos definidos: mean, sd, color, tratamiento, d, a y j. Intente repetir el script cambiando d de 0.2 a 0.5, vea el efecto.

Cambie luego el vector color, por ejemplo, reemplace 'green' por 'black'

7.2. Gráfico de puntos aparedos

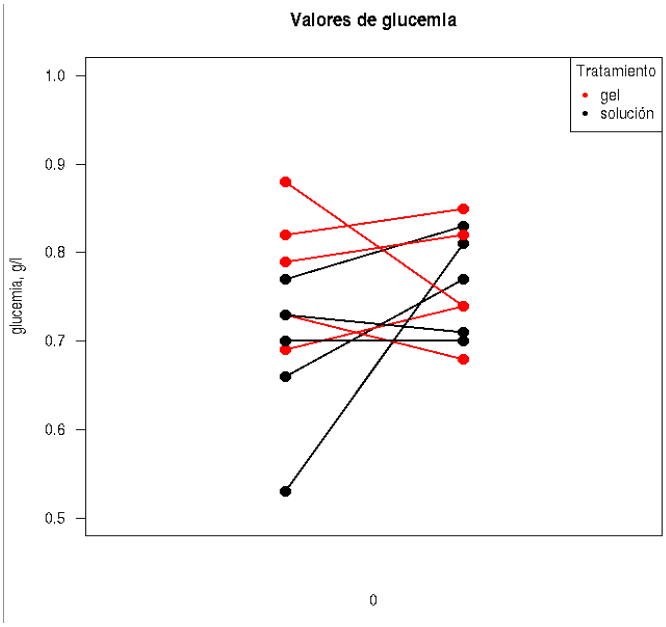
Copie el contenido de la segunda celda de la columna script572 en un archivo de texto, guárdelo en en el directorio donde se encuentra su espacio de trabajo con el nombre script572.txt

script572	explicacion
#script para graficar datos de glucemia con dos tratamientos a dos tiempos	
trat<-levels(glucemia$trat)	Creo un objeto trat con los niveles de la columna trat del data frame glucemia
plot(0,0,ylim=c(0.5,1),xlim=c(0,15),xaxt="n",ylab="glucemia, g/l",las=1,main="Valores de glucemia")	Creamos un gráfico sin datos, con límites de los ejes x e y adecuados
for(i in 1:5){	Se realizan cinco bucles en cada uno de ellos se grafica
points(5,glucemia[i,4],pch=20,col="black",cex=2)	el valor de la primer rata del primer trat a tiempo 0 en negro
points(10,glucemia[i+10,4],pch=20,col="black",cex=2)	el valor de la misma rata pero a tiempo 15
segments(5,glucemia[i,4],10,glucemia[i+10,4],col="black",lwd=2)	se unen ambos valores con un segmento
points(5,glucemia[i+5,4],pch=20,col="red",cex=2)	el valor de la primer rata del segundo trat a tiempo 0 en rojo
points(10,glucemia[i+15,4],pch=20, col="red",cex=2)	el valor de la misma rata a tiempo 15
segments(5,glucemia[i+5,4],10,glucemia[i+15,4],col="red",lwd=2)	se unen con un segmento
}	
legend("topright",legend=c("gel","solución"),pch=c(20,20),col=c("red","black"),title="Tratamiento",horiz=F)	coloca la leyenda

Ejecute el script572

source('script572')

como resultado de la ejecución del deberá obtener este gráfico.



Lea las instrucciones del script y sus explicaciones en la tabla anterior. Haga pequeñas modificaciones al código, cambiando parámetros como col, cex, etc. Luego anímese con cosas más complejas.

7.3. Graficos con retardo de tiempo

Puede introducir retrasos de tiempo con la función Sys.sleep(). Analice el siguiente script que es igual al script571.txt, pero introduciendo esta función.

script571sleep	explicacion
#este script es útil para realizar gráficos de barras de una variable para varios tratamientos #si usted tiene una data.frame llamado "graficobarras" con valores de una variable llamada "variable" y una columna con tratamientos, llamada "tratamiento" realizará la grafica de las medias y los sd de x para cada tratamiento mean<- tapply(graficobarras$variable,graficobarras$tratamiento,mean) sd<-tapply(graficobarras$variable,graficobarras$tratamiento,sd)	es igual al script571.txt, pero introduciendo Sys.sleep() en puntos específicos

```
color=c("red","blue","green")

tratamiento<-levels(graficobarras$tratamiento)

d=0.2

a=1
par(xpd=TRUE)

plot(0,0,type="n",xlim=c(0,20),ylim=c(0,10),xlab="tratamiento"
,ylab="variable",xaxt="n",las=1, main="grafico de
graficobarras")

Sys.sleep(1)
legend(0,10,pch=c(19,19,19),legend=levels(graficobarras$trata
miento)
,col=color,title="Tratamiento",horiz=F)

# e es una variable que permite desplazar cada punto para que no
se superpongan. comienza con cero pero se va desfazando un
valor, ver mas adelante, aca esta fijado en 0.2
#j es la variable que indica donde se pone la marca y el valor del
tiempo
j=5
for (i in 1:3){
#pone el nombre del tratamiento en el eje
text(j,-1,labels=tratamiento[i],cex=0.9)
j=j+5
Sys.sleep(1)
}
# de aca en adelante hace los puntos, los desvios y las lineas de
unión
j=5
for (h in 1:3){
```

al ejecutar

```
> source('script571sleep.txt')
```

obtendrá la misma gráfica, pero con efecto de retardo de tiempo.

8. Clase 8

Utilización de un script para almacenar, administrar, analizar e informar datos relacionados a un trabajo de investigación.
En el Centro Universitario de Estudios Medioambientales de la Facultad de Medicina de la UNR se lleva adelante una investigación que involucra muestras de aguas de consumo de diversas partes del país. Estas muestras son recibidas, clasificadas y enviadas al laboratorio para el análisis de diversas sustancias. En el proyecto participan numerosas personas encargadas de diferentes análisis. La administración de este trabajo es gestionado a través de un script que permite atender a todas las actividades involucradas.
Brevemente, el script desarrollado permite el ingreso de la muestras y la tabulación de los datos básicos de la misma. El script le asigna a la muestra un código único, con el que todos los involucrados en el proyecto se manejarán, desconociendo su origen y características.
Cada persona luego de realizar los análisis correspondientes ingresará al script, utilizando una clave única de acceso. Allí cargará los datos en el sector correspondiente al código del agua analizada.
El script tiene forma de corregir errores en el supuesto que sean detectados.
Un sector del script permite generar un informe de una muestra en particular, informe que será grabado en un archivo txt, que quedará almacenado en el mismo directorio en que se encuentra el script.
Además el script cuenta con un sector que permite obtener estadísticas básicas del proceso de almacenamiento y medición, indicando para cada variable los valores aceptados o sugeridos por legislaciones aplicables a las aguas de consumo.
Los usuarios disponen de una clave única de acceso al script y al hacerlo queda grabado en un data.frame la fecha y hora de ingreso, así como el nombre del usuario y los sectores recorridos. De no disponerse de clave, no se puede acceder a la base de datos.

Analicemos los sectores y códigos del script en su versión 1.0 que tiene 190 líneas que se muestran a continuación. Se muestra solo una parte del mismo con fines de comprender el uso que se le puede dar a los scripts en el trabajo cotidiano de investigación y desarrollo. La complejidad que puede alcanzarse es ilimitada. Resaltaremos en diferentes colores los sectores que analizaremos luego por separado

Antes de comenzar introduzca en su espacio de trabajo la tablaR581 de la planilla de cálculo tablaR58.xls/ods. La misma ingrésela creando el objeto datosaguas, con el siguiente código
> datosaguas<-read.table('clipboard',header=TRUE,sep='\t',dec=',',encoding='latin1')
esto creará un data.frame necesario para el funcionamiento del script.
A continuación se muestra todo el script, que luego se discutirá por sectores.

#SECTOR INGRESO USUARIO

```
print("SOFTWARE DE ADMINISTRACIÓN, ANÁLISIS E INTERPRETACIÓN DE ANÁLISIS DE AGUAS Y SUELOS")
print("BUEN DÍA, INTRODUZCA SU CLAVE")
clave<-scan(file="",what="",nmax=1)
```

```
if(clave=="a1b2c3d4"|clave=="aconcagua"){
nusuario<-nrow(usuario)
if(clave=="a1b2c3d4"){
usuario[nusuario+1,1]=date()
usuario[nusuario+1,2]="Alfredo Rigalli"}
if(clave=="aconcagua"){
usuario[nusuario+1,1]=date()
usuario[nusuario+1,2]="Victor Perez"}
 #SECTOR SELECTOR DE ACTIVIDAD A REALIZAR
selector="NA"
while(selector!=7){
        repeat{
        print("MENU")
        print("Ingresar nueva muestra de agua, oprima 1")
        print("Ingresar valores de mediciones, oprima 2")
        print("Obtener informe de una medición, oprima 3")
        print("Observar datos tabulados, oprima 4")
        print("Modificar datos,oprima 5")
        print("Obtener estadística básica sobre numero de muestras, tipos,etc. Oprima 6")
        print("Salir, oprima 7")
        #evita se cuelgue por introducción de letras en lugar de números
        try(selector<-as.numeric(scan(file = "", what = "",nmax=1)),silent=TRUE)
        #evita se cuelgue por ingreso de letras en lugar de números
        try(if(selector==1|selector==1|selector==2|selector==3|selector==4|selector==5|
selector==6|selector==7){break})
        }


# ////////////////////////////////////////////////
# SECTOR INGRESO DE MUESTRAS
if (selector==1){
#lee numero de filas del data.frame: datosaguas
nr<-nrow(datosaguas)
print("ingrese su nombre y apellido ej: Maela Lupo")
datosaguas[(nr+1),10]<-scan(file = "", what = "",nmax=1,sep="\n",nlines=1)
#indica cual fue el último ingreso
writeLines(paste("Ultima muestra ingresada: ", datosaguas[nr,1]))
#crea un nuevo código automáticamente
datosaguas[(nr+1),1]<-paste("A",sep="",(nr+1))
writeLines(paste("La muestra que está ingresando llevar el código: ", datosaguas[nr+1,1]))
# graba la fecha de ingreso
datosaguas[nr+1,2]<-date()
print("ingrese la localidad donde se obtuvo la muestra, ej: Casilda")
datosaguas[nr+1,3]<-scan(file = "", what = "",nmax=1,sep="\n",nlines=1)
```

```
print("ingrese la provincia donde se obtuvo la muestra, ej: Santa Fe")
datosaguas[nr+1,4]<-scan(file = "", what = "",nmax=1,sep="\n",nlines=1)
print("ingrese la dirección de donde se obtuvo la muestra, ej: Bolivia 312")
datosaguas[nr+1,5]<-scan(file = "", what = "",nmax=1,sep="\n",nlines=1)
print("ingrese el telefono de contacto, ej: Santa Fe")
datosaguas[nr+1,6]<-scan(file = "", what = "",nmax=1,sep="\n",nlines=1)
print("ingrese el email del contacto, ej: Santa Fe")
datosaguas[nr+1,7]<-scan(file = "", what = "",nmax=1,sep="\n",nlines=1)
print("ingrese nombre y apellido del contacto, ej: Juan Perez")
datosaguas[nr+1,8]<-scan(file = "", what = "",nmax=1,sep="\n",nlines=1)
print("ingrese tipo de agua, pozo - red - bidón - etc")
datosaguas[nr+1,9]<-scan(file = "", what = "",nmax=1,sep="\n",nlines=1)
for(i in 11:22){
#datosaguas[nr+1,i]=NA
}
print("LOS DATOS INGRESADOS SON")
columnas<-names(datosaguas[,c(1:10)])
for(i in 1:10){
writeLines(paste(columnas[1]," :", datosaguas[nr+1,i]))
}
print("RECUERDE DE ROTULAR BOTELLA Y TUBOS")
}


# ////////////////////////////////////////////////
# SECTOR INGRESO DE DATOS
if (selector==2|selector==5){
	salirsector<-"i"
	while(salirsector=="i"){
		for(i in 1:nrow(datosaguas)){
		writeLines(paste(datosaguas[i,1],"\t"))
		}
	print("Ingrese el código del agua respetando mayusculas y números")
	busqueda<-scan(file = "", what = "",nmax=1)
	filaEditar<-row.names(datosaguas[datosaguas$codigo==busqueda,])
	datosaguas[filaEditar,]<-edit(datosaguas[filaEditar,])
	print("si desea ingresar datos de otra muestra oprima i, sino oprima cualquier techa")
	salirsector<-scan(file="",what="",nmax=1,sep="\n")
	}
}


# ////////////////////////////////////////////////
# SECTOR OBTENCION INFORME
```

```
if (selector==3){
        salirsector<-"i"
        while(salirsector=="i"){
                for(i in 1:nrow(datosaguas)){
                writeLines(paste(datosaguas[i,1],"\t"))
                }
        print("Ingrese el código del agua a informarrespetando mayusculas y números")
        busqueda<-scan(file = "", what = "",nmax=1)
        filaEditar<-row.names(datosaguas[datosaguas$codigo==busqueda,])
        writeLines(paste("INFORME MUESTRA: ", busqueda))
    writeLines(paste("CÓDIGO: ", datosaguas[filaEditar,1]))
        writeLines(paste("FECHA INGRESO: ", datosaguas[filaEditar,2]))
        writeLines(paste("LOCALIDAD: ", datosaguas[filaEditar,3]))
        writeLines(paste("PROVINCIA: ", datosaguas[filaEditar,4]))
        writeLines(paste("DIRECCIÓN : ", datosaguas[filaEditar,5]))
        writeLines(paste("TELÉFONO : ", datosaguas[filaEditar,6]))
        writeLines(paste("EMAIL : ", datosaguas[filaEditar,7]))
        writeLines(paste("NOMBRE Y APELLIDO CONTACTO : ",
datosaguas[filaEditar,8]))
        writeLines(paste("TIPO AGUA : ", datosaguas[filaEditar,9]))
        writeLines(paste("RESPONSABLE INGRESO : ", datosaguas[filaEditar,10]))
        writeLines(paste("pH : ", datosaguas[filaEditar,11]))
        writeLines(paste("CONDUCTIVIDAD : ", datosaguas[filaEditar,12]," uS/cm"))
        writeLines(paste("CLORURO : ", datosaguas[filaEditar,12], "ppm", "\t valor
recomendado: menor a 300 ppm"))
        informe[1,1]<-paste("INFORME MUESTRA: ", busqueda)
    informe[2,1]<-paste("CÓDIGO: ", datosaguas[filaEditar,1])
        informe[3,1]<-paste("FECHA INGRESO: ", datosaguas[filaEditar,2])
        informe[4,1]<-paste("LOCALIDAD: ", datosaguas[filaEditar,3])
        informe[5,1]<-paste("PROVINCIA: ", datosaguas[filaEditar,4])
        informe[6,1]<-paste("DIRECCIÓN : ", datosaguas[filaEditar,5])
        informe[7,1]<-paste("TELÉFONO : ", datosaguas[filaEditar,6])
        informe[8,1]<-paste("EMAIL : ", datosaguas[filaEditar,7])
        informe[9,1]<-paste("NOMBRE Y APELLIDO CONTACTO : ",
datosaguas[filaEditar,8])
        informe[10,1]<-paste("TIPO AGUA : ", datosaguas[filaEditar,9])
        informe[11,1]<-paste("RESPONSABLE INGRESO : ", datosaguas[filaEditar,10])
        informe[12,1]<-paste("pH : ", datosaguas[filaEditar,11])
        informe[13,1]<-paste("CONDUCTIVIDAD : ", datosaguas[filaEditar,12]," uS/cm")
        informe[14,1]<-paste("CLORURO : ", datosaguas[filaEditar,12], "ppm", "\t valor
recomendado: menor a 300 ppm")
        write.table(informe,paste("informe
muestra",busqueda,".txt"),row.names=FALSE,col.names=FALSE)
        writeLines(paste("En el directorio de trabajo hallará el archivo con el nombre:
```

```
informe muestra ",busqueda))
            print("si desea obtener informe de otra muestra oprima i, sino oprima cualquier
techa")
            salirsector<-scan(file="",what="",nmax=1,sep="\n")
            }
}

# ////////////////////////////////////////////////
# SECTOR OBSERVACIÓN DATOS
if (selector==4){
edit(datosaguas)

}


# ////////////////////////////////////////////////
# SECTOR ESTADÍSTICA BÁSICA DATOS
if (selector==6){
print("INFORMACIÓN BÁSICA")
writeLines(paste("Total de muestras: ", nrow(datosaguas)))
print("Tipos de agua recibidas")
print(summary(datosaguas$tipoagua))
print("Provincias de origen de las muestras")
print(summary(datosaguas$provincia))
readline("OPRIMA ENTER PARA CONTINUAR")
layout(matrix(1:12,4,3))
try(boxplot(datosaguas$ph,las=1,main="pH"))
try(boxplot(datosaguas$conductividad,las=1,main="Conductividad"))
try(boxplot(datosaguas$cloruro,las=1,main="Cloruro"))
try(segments(0,300,20,300,col= "red",lty=2,lwd=2),silent=TRUE)
try(boxplot(datosaguas$carbonato,las=1,main="Carbonato"))
try(boxplot(datosaguas$bicarbonato,las=1,main="Bicarbonato"))
try(boxplot(datosaguas$solidostotales,las=1,main="Sólidos Totales"))
try(segments(0,1000,20,1000,col= "red",lty=2,lwd=2),silent=TRUE)
try(boxplot(datosaguas$fosfato,las=1,main="Fosfato"))
try(segments(0,0.4,20,0.40,col= "red",lty=2,lwd=2),silent=TRUE)
try(boxplot(datosaguas$nitrato,las=1,main="Nitrato"))
try(segments(0,25,20,25,col= "red",lty=2,lwd=2),silent=TRUE)
try(boxplot(datosaguas$nitrito,las=1,main="Nitrito"))
try(boxplot(datosaguas$fluoruro,las=1,main="Fluoruro"))
try(segments(0,1.5,20,1.5,col= "red",lty=2,lwd=2),silent=TRUE)
try(boxplot(datosaguas$arsenico,las=1,main="Arsénico"))
try(segments(0,50,20,50,col= "red",lty=2,lwd=2),silent=TRUE)
try(boxplot(datosaguas$amonio,las=1,main="Amonio"))
```

```
readline("Oprima enter para cerrar el gráfico")
graphics.off()
}

#////////////////////////////////////
#SECTOR DE SALIDA
#SECTOR DE SALIDA
if (selector==7){
print("HA FINALIZADO EL USO DEL PROGRAMA")
Sys.sleep(1)
print("NO OLVIDE NADA FUERA DE LUGAR")
Sys.sleep(1)
print("APAGUE LA COMPUTADORA")
alarm()
}
```

Analicemos cada sector.

8.1. Sector ingreso al script

el análisis por sectores tiene algunas modificaciones respecto del script completo 'scriptaguas' para que se puedan ejecutar por separado del resto de las instrucciones

sector: ingreso del usuario	explicacion

print("SOFTWARE DE ADMINISTRACIÓN, ANÁLISIS E INTERPRETACIÓN DE ANÁLISIS DE AGUAS Y SUELOS")
print("BUEN DÍA, INTRODUZCA SU CLAVE")
clave<-scan(file="",what="",nmax=1)
if(clave=="a1b2c3d4"|clave=="aconcagua"){
 	
nusuario<-nrow(usuario)
 	
if(clave=="a1b2c3d4"){
usuario[nusuario+1,1]=date()
usuario[nusuario+1,2]="Alfredo Rigalli"}
if(clave=="aconcagua"){
usuario[nusuario+1,1]=date()
usuario[nusuario+1,2]="Victor Perez"}
print('el ingreso fue exitoso y salimos del script')
}else{
print("USTED NO TIENE ACCESO A ESTA BASE")}
	Muestra un título al inicio
	
	nos saluda y pide la clave personal
	Introducimos nuestra clave
	Comprueba si la clave introducida es alguna de las que ya existen. Hasta el momento tiene dos usuarios con claves: a1b2c3d4 y aconcagua
	cuenta el numero de filas del data.frame usuario que es donde se almacena quien ingreso a la base y cuando y asigna el numero de filas a la variable nusuario
	Chequea si es el usuario1 con clave a1b2c3d4
	en caso de serlo en la primer columna del data frame coloca la fecha y en la segunda el nombre del usuario: Alfredo Rigalli
	Chequea si es el usuario2 con clave aconcagua
	en caso de serlo en la primer columna del data frame coloca la fecha y en la segunda el nombre del usuario: Victor Perez.
	En caso de ingresar alguien con clave que no es ninguna de las dos anteriores el script pasa a la línea final y cierra el script.

Para probar el funcionamiento del script, copie la columna de la izquierda, péguela en un procesador de texto, grábela con el nombre script581.txt en el directorio donde tendrá su espacio de trabajo. En caso que no le funcione trate con el archivo enviado en la clase que tiene el mismo nombre. Además cree un data frame que llame usuario, en el mismo espacio de trabajo
usuario<-data.frame()
Luego desde su espacio de trabajo ejecute
source('script581') o source('script581.txt'), dependiendo su entorno de trabajo. Interactue con el script colocando claves adecuadas a un usuari0 (aconcagua o a1b2c3d4) y vea el resultado. Pruebe cualquier otra clave.

Veamos ahora la segunda parte, que corresponde al selector de actividades

8.2. Sector selector de actividades

sector: selector de actividades	explicacion

selector="NA"
 	
while(selector!=7){
 	
repeat{
print("MENU")
print("Ingresar nueva muestra de agua, oprima 1")
print("Ingresar valores de mediciones, oprima 2")
print("Obtener informe de una medición, oprima 3")
print("Observar datos tabulados, oprima 4")
print("Modificar datos,oprima 5")
print("Obtener estadística básica sobre numero de muestras, tipos,etc. Oprima 6")
print("Salir, oprima 7")
#evita se cuelgue por introducción de letras en lugar de números
try(selector<-as.numeric(scan(file = "", what = "",nmax=1)),silent=TRUE)
#evita se cuelgue por ingreso de letras en lugar de números
try(if(selector==1|selector==1|selector==2|selector==3|selector==4|selector==5|selector==6|selector==7){break})
}
}
	creamos una variable "selector". Esta variable puede tomar valores entre 1 y 7
	
	Una vez terminada cada actividad volverá al menú, esto se logra con while(selector!=7). El valor 7 es para salir
	
	repeat, repetirá el menú mientras se oprima una tecla distinta de 1 a 7. No permite colocar cualquier número. Si se coloca un número entre 1 y 7 lo llevará a diferentes sectores que se indican en las sucesivas líneas.
	
	se asigna a la variables selector el número del sector a trabajar
	
	Si selector tomó un valor entre 1 y 7 saldrá del bucle repeat e irá a cada sector.
	
	Con cualquier valor del selector que esté entre 1 y 7 se sale del bucle repeat. Pero, como este bucle está anidado con el bucle while, al terminar cada actividad se vuelve a while con la posibilidad de elegir otra actividad. Solo si se eligió el valor 7 no se volverá, ya que 7 es el valor del selector para salir del script.

Para probar el funcionamiento del script, copie la columna de la izquierda, péguela en un procesador de texto, grábela con el nombre script582.txt en el directorio donde tendrá su espacio de trabajo.
Luego desde su espacio de trabajo ejecute

source('script582') o source('script582.txt'), dependiendo su entorno de trabajo. Interactúe con el script para entender su funcionamiento. Este menú le permitirá dirigirse a otros scripts o bien aun sector del script donde se ejecuten las ordenes referidas a cada tarea.

Veamos ahora el sector de ingreso de muestras

8.3. Sector ingreso de muestras

Sector: ingreso de muestras	explicación
# SECTOR INGRESO DE MUESTRAS	
if (selector==1){	solo ingresa en este sector si asignamos en la sección anterior el valor 1 a la variable selector
nr<-nrow(datosaguas)	asigna a la variable nr el número de filas del data.frame "datosaguas" donde se almacena toda la información
print("ingrese su nombre y apellido ej: Maela Lupo")	Nos pide el nombre del usuario que ingresa la muestra
datosaguas[(nr+1),10]<-scan(file = "", what = "",nmax=1,sep="\n",nlines=1)	asigna el nombre del usuario a la fila siguiente vacía en la columna 10
writeLines(paste("Ultima muestra ingresada: ", datosaguas[nr,1]))	Nos muestra el código de la última muestra ingresada
datosaguas[(nr+1),1]<-paste("A",sep="",(nr+1))	Crea el código inmediato siguiente. En nuestro caso las muestras tiene un código que consiste en una A seguida de números enteros consecutivos
writeLines(paste("La muestra que está ingresando llevar el código: ", datosaguas[nr+1,1]))	Nos indica que código se asigno, para que podamos rotular el recipiente, fraccionarlar y enviarla a laboratorio
datosaguas[nr+1,2]<-date()	Siempre en la misma línea sigue agregando datos. Automaticamnte coloca en columna 2 la fecha
print("ingrese la localidad donde se obtuvo la muestra, ej: Casilda")	Nos pide la localidad, que debemos ingresar por teclado la asigna a la columna 3
datosaguas[nr+1,3]<-scan(file = "", what = "",nmax=1,sep="\n",nlines=1)	nos pide provincia la ingresa en columna 4
print("ingrese la provincia donde se obtuvo la muestra, ej: Santa Fe")	nos pide dirección
datosaguas[nr+1,4]<-scan(file = "", what = "",nmax=1,sep="\n",nlines=1)	la ingresa en columna 5 nos pide teléfono
print("ingrese la dirección de donde se obtuvo la muestra, ej: Bolivia 312")	lo ingresa en columna 6
datosaguas[nr+1,5]<-scan(file = "", what = "",nmax=1,sep="\n",nlines=1)	nos pide email de contacto con la persona que la envió lo ingresa en columna 7

print("ingrese el telefono de contacto, ej: Santa Fe")
datosaguas[nr+1,6]<-scan(file = "", what = "",nmax=1,sep="\n",nlines=1)
print("ingrese el email del contacto, ej: Santa Fe")
datosaguas[nr+1,7]<-scan(file = "", what = "",nmax=1,sep="\n",nlines=1)
print("ingrese nombre y apellido del contacto, ej: Juan Perez")
datosaguas[nr+1,8]<-scan(file = "", what = "",nmax=1,sep="\n",nlines=1)
print("ingrese tipo de agua, pozo - red - bidón - etc")
datosaguas[nr+1,9]<-scan(file = "", what = "",nmax=1,sep="\n",nlines=1)
for(i in 11:22){
#datosaguas[nr+1,i]=NA
}
print("LOS DATOS INGRESADOS SON")
columnas<-names(datosaguas[,c(1:10)])
for(i in 1:10){
writeLines(paste(columnas[1]," :", datosaguas[nr+1,i]))
}
print("RECUERDE DE ROTULAR BOTELLA Y TUBOS")
}
	nos pide nombre y apellido de quien trajo la muestra
	lo ingresa en columan 8
	nos pide que indiquemos el tipo de muestra: pozo, red, etc
	ingresa en dato en columna 9
	Asigna NA al resto de las columnas donde irán las mediciones
	nos muestra los datos ingresados para que podamos comprobar si fueron correctamente ingresados
	Nos recuerda algo muy importante para el trabajo posterior
	regresa al menú

Para probar el funcionamiento del script, copie la columna de la izquierda, péguela en un procesador de texto, grábela con el nombre script583.txt en el directorio donde tendrá su espacio de trabajo.
Dado que este sector requiere que se haya seleccionado para la variable selector el valor 1, realice esta asignación
selector<-1
Ejecute el script582 y oprima 1 para seleccionar la opción (Ingresar nueva muestra de agua".
La primer instrucción evalua si la variable selector tiene el valor 1. Si no se hubiera realizado la elección de este sector no se ejecutaría ninguna de las instrucciones. Tenga en cuenta que estamos ejecutando un gran script por sectores como para ir comprendiendo el funcionamiento.
Luego desde su espacio de trabajo ejecute el script.
source('script583') o source('script583.txt'), dependiendo su entorno de trabajo. Interactúe con el script para entender su funcionamiento.

Veamos ahora el nuevo sector que es ingreso de datos

8.4. Sector ingreso de mediciones

sector ingreso de mediciones	explicación

# SECTOR INGRESO DE DATOS 	
if (selector==2|selector==5){
salirsector<-"i"
while(salirsector=="i"){
for(i in 1:nrow(datosaguas)){
writeLines(paste(datosaguas[i,1],"\t"))
}
print("Ingrese el código del agua respetando mayusculas y números")
busqueda<-scan(file = "", what = "",nmax=1)
filaEditar<-row.names(datosaguas[datosaguas$codigo==busqueda,])
datosaguas[filaEditar,]<-edit(datosaguas[filaEditar,])
print("si desea ingresar datos de otra muestra oprima i, sino oprima cualquier tecla")
salirsector<-scan(file="",what="",nmax=1,sep="\n")
}
}
	ingresa en este sector tanto si queremos colocar nuevos datos como si queremos corregir o modificar: de allí selector 2 o 5
	inicializamos una variable "salirsector" con i
	mientras salirsector valga i nos quedaremos en este sector
	recorre toda la base y nos muestra los códigos de todas las muestras
	nos pide que ingresemos el código que deseamos modificar
	lo ingresamos por teclado y asignamos a la variable "busqueda"
	la variable filaEditar toma el valor de la fila con código que introdujimos con la variable busqueda
	nos abre el data.frame solo en la fila del código deseado
	Una vez que salimos nos pide si deseamos seguir agregando datos. Si oprimimos "i" seguiremos dentro del bucle while
	si oprimimos cualquier otra letra saldremos de while y volveremos la menu.

Para probar el funcionamiento del script, copie la columna de la izquierda, péguela en un procesador de texto, grábela con el nombre script584.txt en el directorio donde tendrá su espacio de trabajo.
Previamente deberá escribir la línea
selector<-2
o
selector<-5
ya que este sector se ejecutará si la variable selector toma alguno de esos dos valores. Si no ejecutara la línea mencionada no se ejecutaría ninguna de las instrucciones. Tenga en cuenta que estamos ejecutando un gran script por sectores como para ir comprendiendo el funcionamiento.
Luego desde su espacio de trabajo ejecute el script.
source('script584') o source('script584.txt'), dependiendo su entorno de trabajo. Interactúe con el script para entender su funcionamiento.

veamos ahora el sector informe

8.5. Sector generación de informe

Sector obtención de informe	explicación

# SECTOR OBTENCION INFORME 	
if (selector==3){
salirsector<-"i"
while(salirsector=="i"){
 	
for(i in 1:nrow(datosaguas)){
writeLines(paste(datosaguas[i,1],"\t"))
}
print("Ingrese el código del agua a informarrespetando mayusculas y números")
busqueda<-scan(file = "", what = "",nmax=1)
filaEditar<-row.names(datosaguas[datosaguas$codigo==busqueda,])
 	
writeLines(paste("INFORME MUESTRA: ", busqueda))
writeLines(paste("CÓDIGO: ", datosaguas[filaEditar,1]))
writeLines(paste("FECHA INGRESO: ", datosaguas[filaEditar,2]))
writeLines(paste("LOCALIDAD: ", datosaguas[filaEditar,3]))
writeLines(paste("PROVINCIA: ", datosaguas[filaEditar,4]))
writeLines(paste("DIRECCIÓN : ", datosaguas[filaEditar,5]))
writeLines(paste("TELÉFONO : ", datosaguas[filaEditar,6]))
writeLines(paste("EMAIL : ", datosaguas[filaEditar,7]))
writeLines(paste("NOMBRE Y APELLIDO CONTACTO : ", datosaguas[filaEditar,8]))
writeLines(paste("TIPO AGUA : ", datosaguas[filaEditar,9]))
writeLines(paste("RESPONSABLE INGRESO : ", datosaguas[filaEditar,10]))
writeLines(paste("pH : ", datosaguas[filaEditar,11]))
writeLines(paste("CONDUCTIVIDAD : ", datosaguas[filaEditar,12]," uS/cm"))
writeLines(paste("CLORURO : ", datosaguas[filaEditar,12], "ppm", "\t valor recomendado: menor a 300 ppm"))
informe[1,1]<-paste("INFORME MUESTRA: ", busqueda)
 	
informe[2,1]<-paste("CÓDIGO: ", datosaguas[filaEditar,1])
informe[3,1]<-paste("FECHA INGRESO: ", datosaguas[filaEditar,2])
informe[4,1]<-paste("LOCALIDAD: ", datosaguas[filaEditar,3])
informe[5,1]<-paste("PROVINCIA: ", datosaguas[filaEditar,4])
informe[6,1]<-paste("DIRECCIÓN : ", datosaguas[filaEditar,5])
	Si se selecciona la opción 3, lleva a este sector
	
	la variable salirsector toma el valor "i" y mientras no se cambia se puede permanecer en este sector generando informes de diferentes muestras
	El bucle for imprime en pantalla el código de todas las muestras ingresadas para que podamos seleccionar la deseada.
	Nos pide el código de la muestra
	ingresamos el código en la variable "busqueda"
	en la variable filaEditar coloca el número de la fila del data.frame
	En pantalla nos muestra la muestra seleccionada
	nos indica en pantalla el código de la muestra
	nos indica la fecha de ingreso
	la localidad
	la provincia
	la dirección
	el teléfono del contacto
	el email del contacot
	el nombre y apellido del contacto
	
	el tipo de agua
	el responsable del ingreso de la muestra
	el ph
	la conductividad
	
	concentración de cloruro

informe[7,1]<-paste("TELÉFONO : ", datosaguas[filaEditar,6]) informe[8,1]<-paste("EMAIL : ", datosaguas[filaEditar,7]) informe[9,1]<-paste("NOMBRE Y APELLIDO CONTACTO : ", datosaguas[filaEditar,8]) informe[10,1]<-paste("TIPO AGUA : ", datosaguas[filaEditar,9]) informe[11,1]<-paste("RESPONSABLE INGRESO : ", datosaguas[filaEditar,10]) informe[12,1]<-paste("pH : ", datosaguas[filaEditar,11]) informe[13,1]<-paste("CONDUCTIVIDAD : ", datosaguas[filaEditar,12]," uS/cm") informe[14,1]<-paste("CLORURO : ", datosaguas[filaEditar,12], "ppm", "\t valor recomendado: menor a 300 ppm") write.table(informe,paste("informe muestra",busqueda,".txt"),row.names=FALSE,col.names=FALSE) writeLines(paste("En el directorio de trabajo hallará el archivo con el nombre: informe muestra ",busqueda)) print("si desea obtener informe de otra muestra oprima i, sino oprima cualquier techa") salirsector<-scan(file="",what="",nmax=1,sep="\n") } }	en el data.frame "informe" coloca en linea 1 columna 1 "INFORME MUESTRA ....." en linea 2 el código línea 3 fecha de ingreso localidad provincia dirección teléfono email nombre y apellido del contacto tipo de agua responsable del ingreso pH conductividad cloruro exporta el data.frame "informe" con el nombre "informe muestra .....txt" nos informa que hallaremos en el directorio de trabajo el archivo de texto con ese nombre nos pregunta si queremos realizar otro informe, de ser así introducimos "i" de manera que no salimos del bucle while si oprimimos cualquier otra letra volvemos al menú

Para probar el funcionamiento del script, copie la columna de la izquierda, péguela en un procesador de texto, grábela con el nombre script585.txt en el directorio donde tendrá su espacio de trabajo.
Previamente deberá escribir la línea
selector<-3

ya que este sector se ejecutará si la variable selector toma alguno de esos dos valores. Si no ejecutara la línea mencionada no se ejecutaría ninguna de las instrucciones. Tenga en cuenta

que estamos ejecutando un gran script por sectores como para ir comprendiendo el funcionamiento.
Luego desde su espacio de trabajo ejecute el script.
source('script585') o source('script585.txt'), dependiendo su entorno de trabajo. Interactúe con el script para entender su funcionamiento

Veamos el sector observación de datos

8.6. Sector observación de datos

Sector: observación de datos	Explicación

# SECTOR OBSERVACIÓN DATOS 	
if (selector==4){
edit(datosaguas)
 	
}
	si selector tomó el valor 4
	
	nos muestra el data.frame "datosaguas" con la función edit. En este sector podemos modificar, pero no guardará los cambios. Este sector es solo para observación.

Para probar el funcionamiento del script, copie la columna de la izquierda, péguela en un procesador de texto, grábela con el nombre script586.txt en el directorio donde tendrá su espacio de trabajo.
Previamente deberá escribir la línea
selector<-4
ya que este sector se ejecutará si la variable selector toma alguno de esos dos valores. Si no ejecutara la línea mencionada no se ejecutaría ninguna de las instrucciones. Tenga en cuenta que estamos ejecutando un gran script por sectores como para ir comprendiendo el funcionamiento.
Luego desde su espacio de trabajo ejecute el script.
source('script586') o source('script586.txt'), dependiendo su entorno de trabajo. Interactúe con el script para entender su funcionamiento

veamos ahora el sector de estadística básica

8.7. Sector estadística básica

Sector: estadística básica	Explicación

if (selector==6){
print("INFORMACIÓN BÁSICA")
writeLines(paste("Total de muestras: ", nrow(datosaguas)))
print("Tipos de agua recibidas")
print(summary(datosaguas$tipoagua))
print("Provincias de origen de las muestras")
print(summary(datosaguas$provincia))
readline("OPRIMA ENTER PARA CONTINUAR")
layout(matrix(1:12,4,3))
try(boxplot(datosaguas$ph,las=1,main="pH"))
try(boxplot(datosaguas$conductividad,las=1,main="Conductividad"))
try(boxplot(datosaguas$cloruro,las=1,main="Cloruro"))
try(segments(0,300,20,300,col= "red",lty=2,lwd=2),silent=TRUE)
 	
try(boxplot(datosaguas$carbonato,las=1,main="Carbonato"))
try(boxplot(datosaguas$bicarbonato,las=1,main="Bicarbonato"))
try(boxplot(datosaguas$solidostotales,las=1,main="Sólidos Totales"))
try(segments(0,1000,20,1000,col= "red",lty=2,lwd=2),silent=TRUE)
try(boxplot(datosaguas$fosfato,las=1,main="Fosfato"))
try(segments(0,0.4,20,0.40,col= "red",lty=2,lwd=2),silent=TRUE)
try(boxplot(datosaguas$nitrato,las=1,main="Nitrato"))
try(segments(0,25,20,25,col= "red",lty=2,lwd=2),silent=TRUE)
try(boxplot(datosaguas$nitrito,las=1,main="Nitrito"))
try(boxplot(datosaguas$fluoruro,las=1,main="Fluoruro"))
try(segments(0,1.5,20,1.5,col= "red",lty=2,lwd=2),silent=TRUE)
try(boxplot(datosaguas$arsenico,las=1,main="Arsénico"))
try(segments(0,50,20,50,col= "red",lty=2,lwd=2),silent=TRUE)
try(boxplot(datosaguas$amonio,las=1,main="Amonio"))
readline("Oprima enter para cerrar el gráfico")
graphics.off()
}
	Si en el menú inicial se seleccionó 6 se llega a este sector
	
	nos indica el número total de muestras ingresadas
	
	nos indica cuantas aguas son de cada tipo: pozo, red, etc
	
	cuantas aguas pertenecen a cada provincia
	
	Nos mostrará un gráfico múltiple con 12 graficos distribuidos en 4 filas y 3 columnas
	
	boxplot del pH
	
	boxplot de conductividad
	
	boxplot de cloruro
	
	coloca en el grafico de cloruro en rojo una linea indicando el valor máximo recomendado por legislaciones
	
	de igual manera para las otras variables medidas

	nos pide enter para cerrar el gráfico lo cierra regresa al menú original

Para probar el funcionamiento del script, copie la columna de la izquierda, péguela en un procesador de texto, grábela con el nombre script587.txt en el directorio donde tendrá su espacio de trabajo.
Previamente deberá escribir la línea
selector<-6
ya que este sector se ejecutará si la variable selector toma alguno de esos dos valores. Si no ejecutara la línea mencionada no se ejecutaría ninguna de las instrucciones. Tenga en cuenta que estamos ejecutando un gran script por sectores como para ir comprendiendo el funcionamiento.
Luego desde su espacio de trabajo ejecute el script.
source('script587') o source('script587.txt'), dependiendo su entorno de trabajo. Interactúe con el script para entender su funcionamiento

veamos el último sector del software

8.8. Sector salida del script

el análisis por sectores tiene algunas modificaciones respecto del script completo 'scriptaguas' para que se puedan ejecutar por separado del resto de las instrucciones

Sector: salida	explicación

#SECTOR DE SALIDA
if (selector==7){
print("HA FINALIZADO EL USO DEL PROGRAMA")
Sys.sleep(1)
print("NO OLVIDE NADA FUERA DE LUGAR")
Sys.sleep(1)
print("APAGUE LA COMPUTADORA")
alarm()
}
	Si en el menú inicial eligió 7 llegará a este sector.
	nos indica que finalizó
	nos recuerda que ordenemos el laboratorio
	que apaguemos la computadora

Para probar el funcionamiento del script, copie la zona coloreada de la columna de la izquierda, péguela en un procesador de texto, grábela con el nombre script588.txt en el directorio donde tendrá su espacio de trabajo.
asigne a la variable selector el valor 7
> selector<-7
Luego desde su espacio de trabajo ejecute el script.
source('script588') o source('script588.txt'), dependiendo su entorno de trabajo. Interactúe con el script para entender su funcionamiento

El desarrollo es solo una muestra de un camino fructífero en la organización del manejo de datos, análisis y auditoría, permitiendo un trabajo multiusuario. El crecimiento del script es indefinido y podrá darle usted la forma adecuada a su trabajo, grupo humano y variables utilizadas.

8.9. Prueba del script completo

ejecute
source('scriptaguas')
que corresponde a todos los sectores juntos. Podrá ver su funcionamiento. Por supuesto, tendrá problemas en más de un sector por falta de información adicional para su uso.

9. Clase 9

9.1. Uso de scripts por no-usuarios de R

Puede ocurrir que en su trabajo deba realizar tareas rutinarias que requieren la intervención de R, pero el personal involucrado no lo maneja. Por supuesto que será beneficioso que dichos colaboradores comprendan el lenguaje y su código, pero en algunos casos puede resultar imposible ya que para el personal no está dentro de sus obligaciones. Esto suele ocurrir en casos técnicos, donde las personas ejecutan tareas muy complejas pero automatizadas.

A tal fin se pueden crear scripts que se inicien desde un icono en la pantalla, que la persona no tenga que utilizar ningún comando de R, ni siquiera tipear "R" para iniciar, ni q() para finalizar una sesión de trabajo.

Veamos un ejemplo.

Supongamos que a su trabajo diariamente ingresan datos de medidas de pesos de animales de experimentación durante 10 días. Estos ingresan por ejemplo en un formato .txt. Entre las instrucciones que usted impartió es que cada archivo que recibe tenga el siguiente nombre "ratanumero.txt". Dicho archivo es preparado directamente colocando en un archivo de texto la siguiente información: numero de rata, edad en días, tratamiento, peso corporal a los diferentes días.

El archivo vacío tiene la siguiente forma

numero

edad

tratamiento

peso

t peso

Se trata de una archivo de texto en que quien lo llena debe respetar las líneas y tener un espacio entre los campos de información en cada línea

La siguiente es una planilla de datos, la misma se adjunta para realizar esta clase como rata1.txt.

numero 1

edad 21

tratamiento control

peso 35

t peso

21 35

22 37

23 40

24 43

25 47

26 49

27 51

28 55

29 59

30 63

El personal a cargo del primer procesamiento ingresará a través del script la información y el script solo realizará un gráfico que llevará el mismo nombre que el archivo de texto, el que tendrá la información del archivo en forma gráfica, además de la recta de regresión y modificará el archivo de texto de manera que el mismo además de tener los datos enviados, tenga la pendiente de la recta de regresión, que nos indicará la velocidad de crecimiento.

9.2. Arranque automático de R

Se puede realizar un script que permita ejecutar R directamente sin necesidad de hacerlo como lo hemos hecho hasta ahora. De esta manera el usuario haciendo doble click sobre un icono del escritorio accederá directamente a R sin preocuparse de donde está el espacio de trabajo, cuales son los objetos, etc.

Para ello escribimos un script y lo colocamos en algún sitio de acceso que el usuario simplemente tenga que hacer click.

Explicaremos como hacerlo en linux

En un editor de texto escribimos el siguiente código

```
#!/bin/bash
cd
cd alf/cursos/R/modulo5/clase5-9/tablas
R
```

la primer línea indica que es un script

cd, nos envía al directorio raíz

la tercer línea nos posiciona en el directorio donde tenemos el espacio de trabajo, las planillas de datos descriptas anteriormente, el script y donde quedarán los gráficos una vez realizadas

la cuarta línea arranca R

Se deben hacer algunos ajustes para que pueda ejecutarse con un doble click. Pero logrado esto, al hacer doble click en el icono, se abrirá automáticamente R en el directorio donde está el script y todo lo necesario

Acá se hará cargo del resto la función .First de R que terminará de posicionarnos en el el script.

Como puede ver, el usuario solo necesita conocer cual es el ícono a ejecutar.

En cada sistema operativo y versiones de este deberá experimentarse como construir icones de inicio, accesos directos o lanzadores que permitan iniciar R directamente desde estos recursos de pantalla.

9.3. Arranque automático del script

Para que el script arranque cuando R arranca, se puede incluir la función .First en el espacio de trabaja que será previamente creado por usted. Con el código siguiente puede hacer una muy sencilla función que solo carga una biblioteca (utils) que se halla entre las provistas por R, pero con .First suele dar error si no se carga previamente. Y luego el código de arranque del script, que escribe como sigue

> .First<-function(){library(utils)

source("script591")}

En su consola se verá de la siguiente manera

```
> .First<-function(){library(utils)
+ source("procesodato")}
```

los signos "+" no los debe colocar, cuando de enter en cada línea lo agregará R

por supuesto podría hacer funciones .First más sofisticadas que incluyan retardo, sonidos y mensajes, por ejemplo

.First<-function(){library(utils)

Sys.sleep(1)

print("Espere por favor, cargando bibliotecas y script")

Sys.sleep(1)

source("script591")}

También puede ocurrir que requiera cargar otras bibliotecas. Esto lo detectará fácilmente porque al requerir una función que está en una biblioteca, si ésta no está cargada nos dirá función inexistente.

Utilizando

help(nombre de la funcion),

vemos en que biblioteca se halla

y agregamos la biblioteca en .First

Para el caso desarrollado en esta clase la función .First se introducirá con el siguiente código. De esta manera además de ejecutar automaticamente el script591, previamente cargará varias bibliotecas.

.First <-function{

library(stats)

library(utils)

library(graphics)

library(grDevices)

source('script591')

}

Análisis del script que se describe a continuación

script	explicación

#script de proceso de carga de datos sin conocimientos de R
system("clear")
print("Buen día, comenzará a procesar los datos, siga las instrucciones de la pantalla")
readline("Oprima enter para continuar")
	Limpia la pantalla, equivalente a <control + l>

permanecer<-1	Declaramos e inicializamos la variable que mientras tome el valor 1 permaneceremos en el siguiente bucle while
#lo resaltado en amarillo muestra hasta donde llega el bucle while	
while(permanecer==1){	Bucle while en el que permaneceremos mientras permanecer tome el valor 1
system("clear")	Limpia la pantalla
print("Los archivos que debe procesar son aquellos que figuran como rata seguido de un número. Ejemplo rata1, rata2, etc")	Explicaciones en pantalla de la tarea a realizar
readline("Oprima enter para ver los archivos, ignore los que no tiene el nombre indicado")	
print(dir())	nos muestra los archivos en el directorio en que está el espacio de trabajo
print("ingrese el número del nombre del archivo. Si dice rata1.txt, solo ingrese el número 1")	Nos ordena introducir el número correspondiente al archivo que deseamos procesar.
numero<-as.numeric(scan(file="",what="",nmax=1))	Lo introducimos por teclado y asignamos a la variable "numero"
datostruncados<-read.table(paste("rata",sep="",numero),skip=5)	Introducimos en el data.frame datos truncados los valores del archivo con el nombre rata.... seguido del número que acabamos de introducir. El argumento skip=5 indica que del archivo saltea las 5 primeras filas, ya que solo deseamos el tiempo y los pesos
rl<-lm(V2~V1,datostruncados)	Realiza la regresión lineal entre el peso (V2) y el tiempo (V1). Si al introducir una tabla no se especifica el nombre de las variables. Las llama V1, V2, etc. La regresión lineal la asigna a la variable rl
pendiente<-rl$coefficients[2]	Asigna a la variable pendiente el valor de la pendiente que se halla en el coefficients[2] del objeto rl.
nombregrafica<-paste("rata",sep="",numero,".jpg")	Crea un objeto con el nombre que llevará la grafica
jpeg(filename = nombregrafica)	abre la función jpeg para realizar una gráfica
plot(datostruncados$V1,datostruncados$V2,las=1,pch=19,ylim=c(20,max(datostruncados$V2)),xlab="días",ylab="peso,gramos",main=paste("rata ",numero))	Grafica los puntos del data.frame peso vs tiempo
t<-seq(21,30,0.1)	Crea un vector de valores de 21 a 30 cada 0,1, para realizar la línea de regresión
pesoajustado=rl$coefficients[1]+rl$coefficients[2]*t	crea un vector con los pesos ajustados por la función de regresión
lines(spline(t,pesoajustado),col="red",lwd=2)	En la misma gráfica coloca la línea de regresión Coloca en la gráfica un texto con el valor de la

9.4. Ejecución del script sin la función .First

Procedamos entonces en primer lugar ejecutemos el script591 sin la función .First. Para ello tengamos en el directorio donde se halla nuestro espacio de trabajo los archivos rata1 y rata2 (provistos en la clase) con datos generados por ejemplo en el criadero de animales de su laboratorio. Además debe tener el archivo correspondiente al scritp con el nombre script591.txt, provisto con la clase.

Si no está generada la función .First, cuando el usuario ingrese al espacio de trabajo deberá ejecutar el script, con la siguiente orden

> source('script591')

como resultado tendrá el siguiente mensaje

[1] "Buen día, comenzará a procesar los datos, siga las instrucciones de la pantalla"

Oprima enter para continuar

luego que oprima enter observará

[1] "Los archivos que debe procesar son aquellos que figuran como rata seguido de un número. Ejemplo rata1, rata2, etc"

Oprima enter para ver los archivos, ignore los que no tiene el nombre indicado

al dar enter le mostrará todo el directorio, puede obtener algo así en su computadora. Observará los nombre de todos los archivos que tenga en su directorio, pero nos interesan particularmente los archivos rata1 y rata2, que son los que tienen datos y queremos que se generen gráficas de datos y la regresión lineal

```
 [1] "grafinteractivo~"      "portadas videos R.jpg" "procesodato~"
 [4] "R-5-9.odp"             "R5-9.odt"              "rata-01~"
 [7] "rata1"                 "rata-1~"               "rata1~"
[10] "rata2"                 "script591"             "SoftwareAguas.sh~"
[13] "tablas"
```

[1] "ingrese el número del nombre del archivo. Si dice rata1.txt, solo ingrese el número 1"

ingresamos el número 1 por el teclado, con esto nos realizará automáticamente la gráfica, la regresión lineal y guardará la gráfica en el mismo espacio de trabajo

observaremos luego

Read 1 item

[1] "GRAFICO GENERADO"

[1] "Si desea procesar otro archivo oprima 1, sino oprima cualquier numero"

si deseamos realizar la gráfica del archivo rata2, oprimimos el número 1, sino cualquier número y finalizará el script y se cerrara R. Note que el usuario no puede cometer errores en el guardado del espacio de trabajo, ya que el script lo realiza con la última línea.

Usted podrá comprobar que se han creado las gráficas correspondientes a cada archivo de datos. Sin duda, el uso de R en esta modalidad permite que un usuario cualquiera realice tareas demandantes de R sin ningún conocimiento, más que leer y seguir claramente las instrucciones.

9.5. Ejecución con función .First

Si usted ya creó la función .First, que repetimos a continuación

```
.First <-function{
library(stats)
library(utils)
library(graphics)
library(grDevices)
source('script591')
}
```

cuando el usuario inicie la sesión, por ejemplo escribiendo R en la consola o ejecute R desde un icono, automáticamente se ejecutará el script, observando en la pantalla:

[1] "Buen día, comenzará a procesar los datos, siga las instrucciones de la pantalla"

Oprima enter para continuar

De allí en adelante, el usuario deberá seguir las órdenes escritas en el script.

El perfeccionamiento de este tipo de scripts permite manejar un grupo de trabajo donde existan usuarios iniciados, intermedios y avanzados de R.

9.6. Apertura de scripts desde un script

Los script pueden ejecutarse desde la línea de comando de R, o bien desde otro script. Este recurso permite realizar tareas de programación más ordenadas y por sobre todo permiten identificar y aislar errores con más facilidad.

Supongamos un sencillo script que consta de un script principal (script5921) que incluye un menú desde donde se ejecutan otros scripts con diferentes acciones. Este script tiene una sección que permite sumar dos números (script5922), otra que permite multiplicar dos números (script5923), otra dividir dos números (script5924) y una cuarta que nos permite salir (script5925).

A continuación se muestran los script y su descripción a la derecha

script5921	descripción
#script principal	
readline('Buen día, oprima enter para comenzar')	nos da una bienvenida y debemos dar enter para continuar
print('ELIJA LA OPCIÓN COLOCANDO EL NÚMERO CORRESPONDIENTE')	nos confronta con un menú, del que elegiremos tipeando números de 1 a 4
print('1- sumar dos números')	
print('2- multiplicar dos números')	
print('3- dividir dos números')	
print('4- salir')	
selector<-as.numeric(scan(file='',what='',nmax=1))	permite asignar la elección
if(selector==1){	si elegimos 1, se ejecutará el script 5922
source('script5922')	
}	
if(selector==2){	si elegimos 2, se ejecutará el script 5923
source('script5923')	
}	
if(selector==3){	si elegimos 3, se ejecutará el script 5924
source('script5924')	
}	
if(selector==4){	si elegimos 4, se ejecutará el script 5925
source('script5925')	
}	

script 5922	

#script de suma
print('introduzca uno de los numeros')
a<-as.numeric(scan(file='',what='',nmax=1))
print('introduzca el otro numero')
b<-as.numeric(scan(file='',what='',nmax=1))
readline('Oprima enter para obtener la suma')
writeLines(paste('La suma es: ', a+b))
readline('oprima enter para continuar')
system('clear')
source('script5921')
	Nos pide que introduzcamos uno de los números
	lo hacemos en la variable a
	nos pide el segundo número
	lo introducimos en la variable b
	Nos da tiempo y pide que oprimamos enter para ver el resultado
	lo muestra
	nos pide continuar con un enter
	limpia la pantalla
	rearranca el script que tiene el menu: script5921

script5923	

#script de producto
print('introduzca uno de los numeros')
a<-as.numeric(scan(file='',what='',nmax=1))
print('introduzca el otro numero')
b<-as.numeric(scan(file='',what='',nmax=1))
readline('Oprima enter para obtener la suma')
writeLines(paste('El producto es: ', a*b))
readline('oprima enter para continuar')
system('clear')
source('script5921')
	Nos pide que introduzcamos uno de los números
	lo hacemos en la variable a
	nos pide el segundo número
	lo introducimos en la variable b
	Nos da tiempo y pide que oprimamos enter para ver el resultado
	lo muestra
	nos pide continuar con un enter
	limpia la pantalla
	rearranca el script que tiene el menu: script5921

script5924	

#script de division
print('introduzca uno de los numeros')
a<-as.numeric(scan(file='',what='',nmax=1))
print('introduzca el otro numero, que debe ser diferente de cero')
b<-as.numeric(scan(file='',what='',nmax=1))
readline('Oprima enter para obtener el resultado')
writeLines(paste('El cociente es: ', a/b))
readline('oprima enter para continuar')
system('clear')
source('script5921')
	
	Nos pide que introduzcamos uno de los números
	lo hacemos en la variable a
	nos pide el segundo número
	lo introducimos en la variable b
	Nos da tiempo y pide que oprimamos enter para ver el resultado
	lo muestra
	nos pide continuar con un enter
	limpia la pantalla
	rearranca el script que tiene el menu: script5921

script5925	

#script de salida
system('clear')
print('Salimos del script')
	
	limpia la pantalla
	termina el script y nos da un mensaje de salida de este script.

Coloque todos los script en su espacio de trabajo. Ejecute

source('script5921')

siga las instrucciones.

La ventaje de este tipo de trabajo es que si le surgiera un error en un script, cosa que es habitual, puede anular esa opción. Por ejemplo supongamos que el script5923 da un error que no logramos reparar. Podemos reescribir el script5921 como

```
#script principal
readline('Buen día, oprima enter para comenzar')
print('ELIJA LA OPCIÓN COLOCANDO EL NÚMERO CORRESPONDIENTE')
print('1- sumar dos números')
print('2- multiplicar dos números')
print('3- dividir dos números')
print('4- salir')
selector<-as.numeric(scan(file='',what='',nmax=1))
if(selector==1){
```

```
source('script5922')
}
#if(selector==2){
#source('script5923')
#}
if(selector==3){
source('script5924')
}
if(selector==4){
source('script5925')
}
```

donde vemos que hemos anulado las líneas correspondientes al arranque del script8923. Aunque usted haga la selección oprimiendo 2. Este no se ejecutará.