Universidad
Carlos III de Madrid

Lenguaje R

José M. Valls Ferrán y Ricardo Aler Mur

Universidad Carlos III de Madrid

Sesión 2: E/S, Matrices, Funciones y Estructuras de control

José M. Valls Ferrán y Ricardo Aler Mur

Entrada / salida básica. scan()

- **scan()** lee datos de consola o de un fichero y genera un vector.
  - Por defecto el tipo de dato es numérico y da error si introducimos caracteres. Por defecto lee de consola
  - Si queremos leer caracteres: **scan(what=character())**
  - Si queremos leer de un fichero: **scan(file="miFich.txt")**

- Ejemplos

```
> cosas <- scan()
1: 5
2: 7
3: 86
4: 3
5: 4
6:
Read 5 items
> cosas
[1] 5 7 86 3 4
```

Entrada / salida básica. scan()

- Escribir un fichero con números y palabras y llamarle miFich.txt (no olvidar situarse en el directorio de trabajo)

```
> cosas<-scan(file="miFich.txt", what=character()) Read 9 items
> cosas
[1] "4" "5" "6" "7" "hola" "3" "4" "5" "56"
```

- ¿Qué ocurre si no usamos la opción what?
  - Por defecto leería datos numéricos y daría error al encontrarse un dato no numérico

- Y si el fichero contiene sólo números?
  - Entonces no daría error, el vector sería numérico

Entrada / salida básica
read.table()

- **read.table()**: lee ficheros de texto en forma de tabla y crea un data frame (las observaciones corresponden a las líneas del fichero y las variables a las columnas o campos de cada línea).
- **read.csv()**: Igual que read.table pero para leer ficheros csv (campos separados por ,)
- Existen los equivalentes para escribir:
  - **write.table()**
  - **write.csv()**

Entrada / salida básica
read.table()

- Ejemplo:
  - **aire<-read.table("CalidadAire.txt")**
- **choose.files ()** permite seleccionar el fichero con ventanas. Devuelve la ruta del fichero y el nombre

```
> choose.files()
[1] "C:\\Users\\jvalls\\Google Drive\\uc3m\\masterActuariales\\NuevaDocumentacion\\ejercicios\\CalidadAire.txt“
```

- aire<-read.table(choose.files())

Factores

- Son vectores que representan datos categóricos
- Se utilizan en análisis de datos como una representación eficiente de valores discretos (categóricos)
- Equivalen a números enteros etiquetados
- Es mejor usar factores que enteros porque se autodescriben. Tener una variable con valores *hombre*, *mujer* es mejor que una variable con los valores 1 y 2.
- Las funciones **factor()** (o as.factor()) convierten un vector con un pequeño número de valores distintos en un factor

```
> v
[1] "hombre" "mujer" "mujer" "hombre" "hombre" "mujer" "hombre" "hombre"
> v<-as.factor(v)
> v
[1] hombre mujer mujer hombre hombre mujer hombre hombre
Levels: hombre mujer
```

Factores

- Puede especificarse un orden al crear un factor. Si no se hace, se supone orden alfabético

```
> v
[1] "f" "m" "m" "f" "m" "m" "m" "f"
> v<-factor(v)
> v
[1] f m m f m m m f
Levels: f m
```

- Ahora con orden m, f

```
> v<-factor(v,levels=c("m","f"))
> v
[1] f m m f m m m f
Levels: m f
```

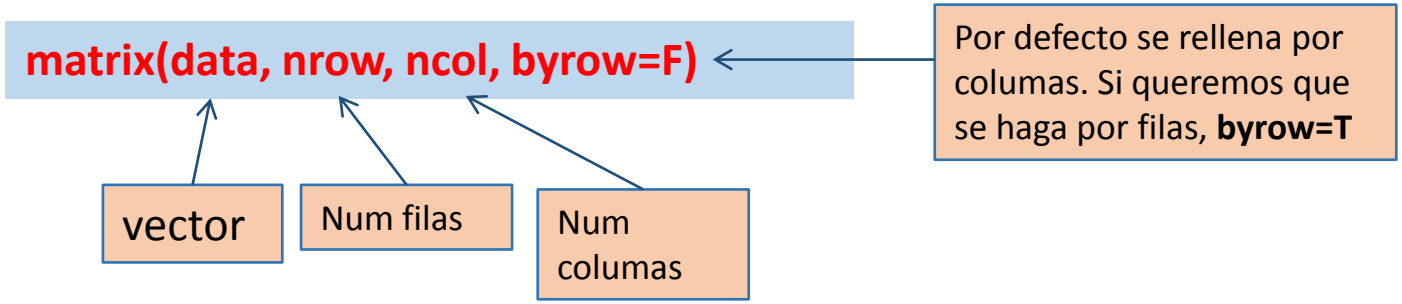
Construyendo Matrices
matrix()

- Son como los vectores pero en dos dimensiones
- Se pueden crear a partir de un vector cambiando la dimensión con **dim()**, o bien con una función específica **matrix()**

```
> m<-c(4,8,2,3,89,5,4,23,4,6,1,2,3,6,1)
> dim(m)<-c(3,5)   ## 3 filas y 5 col
> m [,1] [,2] [,3] [,4] [,5]
[1,]   4    3    4    6    3
[2,]   8   89   23    1    6
[3,]   2    5    4    2    1
```

El vector m tiene 15 elementos
Se le dice que sus dimensiones sean:
3 filas y 5 columnas: se convierte en matriz.
Se rellena la matriz por columnas a partir del vector.
Si (filas x columnas) no coincide con la longitud del vector, da un error

- Generar una matriz m a partir de un vector, con **matrix()**
  - La sintaxis de matrix es:



Construyendo Matrices

- Ejemplos

```
> v<-c(4,8,2,3,89,5,4,23,4,6,1,2,3,6,1)
> m<-matrix(v,3,5)
> m
     [,1] [,2] [,3] [,4] [,5]
[1,]    4    3    4    6    3
[2,]    8   89   23    1    6
[3,]    2    5    4    2    1
```

```
> m<-matrix(c(3,2,6,4,2,1),3,2)
> m
     [,1] [,2]
[1,]    3    4
[2,]    2    2
[3,]    6    1
```

```
> m<-matrix(c(3,2,6,4,2,1),3,2,byrow=T) > m
     [,1] [,2]
[1,]    3    2
[2,]    6    4
[3,]    2    1
```

```
> v<-c(4,8,2,3,89,5,4,23,4,6,1,2,3,6,1)
```

```
> m<-matrix(v,3)
> m
     [,1] [,2] [,3] [,4] [,5]
[1,]    4    3    4    6    3
[2,]    8   89   23    1    6
[3,]    2    5    4    2    1
```

Si sólo se pone una dimensión, se entiende que es **nrow** y calcula las columnas

```
> m<-matrix(v,ncol=3)
> m
     [,1] [,2] [,3]
[1,]    4    5    1
[2,]    8    4    2
[3,]    2   23    3
[4,]    3    4    6
[5,]   89    6    1
```

Si ponemos sólo el núm de columnas, hay que especificar el argumento **ncol**, para que no lo tome por nrow

Número de filas y columnas en Matrices

- nrow() y ncol()

```
> m
     [,1] [,2] [,3] [,4] [,5]
[1,]    4    3    4    6    3
[2,]    8   89   23    1    6
[3,]    2    5    4    2    1
> nrow(m)
[1] 3
> ncol(m)
[1] 5
```

- Para comprobar si el objeto es una matriz, podemos usar class():

```
> class(m)
[1] "matrix"
```

- Si queremos “aplanar” una matriz, es decir, convertirla en un vector, podemos usar as.vector()

```
> m<-as.vector(m)
> class(m)
[1] "numeric"
```

Construyendo Matrices
rbind() y cbind()

- Podemos añadir columnas a una matriz con **cbind()**
- Podemos añadir filas a una matriz con **rbind()**
- Pueden añadirse vectores o matrices como filas o columnas, siempre que las dimensiones coincidan

```
> x
   [,1] [,2]
[1,]   1   2
[2,]   3   4
> y
   [,1] [,2] [,3] [,4] [,5]
[1,]   10   11   12   13   14
[2,]   15   16   17   18   19
> cbind(x,y)
   [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]   1   2   10   11   12   13   14
[2,]   3   4   15   16   17   18   19
> cbind(y,x)
    [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,] 10   12   14   16   18    1    3
[2,] 11   13   15   17   19    2    4
```

```
> x
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> y
     [,1] [,2] [,3]
[1,]   21   26   31
[2,]   22   27   32
[3,]   23   28   33
[4,]   24   29   34
[5,]   25   30   35
```

```
> rbind(x,y)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
[4,]   21   26   31
[5,]   22   27   32
[6,]   23   28   33
[7,]   24   29   34
[8,]   25   30   35
```

Accesos a matrices. Por valores

- Si queremos acceder a ciertas filas y columnas: **matriz(filas,columnas)**
- **Ejemplos** con la siguiente matriz:

```
> m<-matrix(1:30,5,3)
> m
     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
[4,]    4    9   14
[5,]    5   10   15
```

- Elemento de fila 2 columna 3

```
> m[2,3]
[1] 12
```

- Todos los elementos de la columna 2 excepto el de la fila 3

```
> m[-3,2]
[1]  6  7  9 10
```

Si el resultado corresponde a una fila o columna (o ambas) **devuelve un vector**

- Elementos de la columna 2 y filas 2 y 5:

```
> m[c(2,5),2]
[1]  7 10
```

Accesos a matrices. Por valores

- Elementos de las filas 2 y 5 y las columnas 1 y 3

```
> m[c(2,5),c(1,3)]
     [,1] [,2]
[1,]    2   12
[2,]    5   15
```

- Elementos de la fila 1 y todas las columnas

```
> m[1,]
[1]  1  6 11
```

- Elementos de todas las filas y columna 2

```
> m[,2]
[1]  6  7  8  9 10
```

- Elementos de las filas 1 a la 3 y todas las columnas

```
> m[1:3,]
     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
```

Accesos a matrices. Por valores
Conservar el formato matriz

- Si queremos que conserve el **formato de matriz** aunque sólo corresponda a una fila o columna (o un solo elemento), indicar la opción **drop=F**
- Fila 1 y todas las columnas, con formato matriz:

```
> m[1,,drop=F]
     [,1] [,2] [,3]
[1,]    1    6   11
```

- Elemento de la fila 2, columna 3

```
> m[2,3,drop=F]
     [,1]
[1,]   12
```

```
> v<-m[2,3,drop=F]
> v
     [,1]
[1,]   12
> class(v)
[1] "matrix"
> v<-m[2,3]
> v
[1] 12
> class(v)
[1] "integer"
```

matriz

vector

Accesos a matrices mediante índices booleanos

- Igual que en los vectores, podemos acceder a las matrices con índices booleanos

Extraer todos los elementos que cumplan una condición.

- Ejemplo: comprendidos entre 5 y 10:

```
> m>=5 & m<=10
      [,1] [,2]  [,3]
[1,] FALSE TRUE FALSE
[2,] FALSE TRUE FALSE
[3,] FALSE TRUE FALSE
[4,] FALSE TRUE FALSE
[5,]  TRUE TRUE FALSE
```

```
> m<-matrix(1:30,5,3)
> m
     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
[4,]    4    9   14
[5,]    5   10   15
```

Se aplica la regla del reciclado

- Entonces accedemos a la matriz mediante esta matriz booleana y obtenemos un vector con los elementos que cumplen la condición:

```
> m[m>=5 & m<=10]
[1]  5  6  7  8  9 10
```

Accesos a matrices mediante índices booleanos

```
> m<-matrix(1:30,5,3)
> m
     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
[4,]    4    9   14
[5,]    5   10   15
```

Acceder a elementos que cumplan una condición en una fila o columna particular

- **Ejemplo**: extraer las filas cuyo valor en la segunda columna sea menor que 8
  - La condición será:

```
> m[,2]<8
[1]  TRUE  TRUE FALSE FALSE FALSE
```

  - Entonces aplicamos ese vector booleano a las filas:

```
> m[m[,2]<8,]
     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
```

Queremos sólo las filas que cumplan la condición, y de esas filas, queremos todas las columnas

MUY IMPORTANTE

- Ejemplo: extraer la columna 1 de las filas cuyo valor en la columna 3 sea par

```
> m[m[,3]%%2==0,1]
[1] 2 4
```

Matrices. Nombres en filas y columnas

- Podemos identificar con nombres las filas y las columnas haciendo nuestros datos más cómodos de manejar
- Ejemplo: supongamos la siguiente matriz con las notas de 4 alumnos en 3 asignaturas.
- Para dar nombre a las columnas: **colnames()**
- Para dar nombre a las filas: **rownames()**

```
> m
     [,1] [,2] [,3]
[1,]    4  5.6  3.0
[2,]    8  9.0  7.5
[3,]    7  6.0  5.5
[4,]    3  2.0  4.0
```

```
> colnames(m)<-c('Matem','Fisica','Ingles')
> rownames(m)<-c('Jose','Luis','Beatriz','Maria')
> m
        Matem Fisica Ingles
Jose        4    5.6    3.0
Luis        8    9.0    7.5
Beatriz     7    6.0    5.5
Maria       3    2.0    4.0
```

- Ahora podemos acceder por nombre

```
> m['Beatriz',]
 Matem Fisica Ingles
   7.0    6.0    5.5
```

```
> m[,'Fisica']
   Jose    Luis Beatriz   Maria
    5.6     9.0     6.0     2.0
> m[c('Beatriz','Jose'),c('Matem','Ingles')]
        Matem Ingles
Beatriz     7    5.5
Jose        4    3.0
```

Aritmética de matrices

- %*%: **producto matricial** entre dos matrices

```
> m1
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> m2<-matrix(1:12,4,3)
> m2
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
> m1%*%m2
     [,1] [,2] [,3]
[1,]   50  114  178
[2,]   60  140  220
> m2%*%m1
Error in m2 %*% m1 : non-conformable arguments
```

Matriz 2 x 4

Matriz 4 x 3

Válido

Resultado: Matriz 2 x 3

Inválido

Aritmética de matrices

- +, -, *, /, ^: Se aplican componente a componente.
- **Regla del reciclado**:
  - Si **x** e **y** son matrices, tienen que tener el mismo número de componentes
  - Si **x** es matriz e **y** es escalar, entonces se aplica la regla del reciclado
  - Si x es matriz e y es vector, se considera que ambos son columnas y se aplica reciclado
  - t(): transpone la matriz. Si se aplica a un vector se considera que este es **columna**
- Se puede utilizar log(), sqrt(), …, sobre matrices

```
> x
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> x+1
     [,1] [,2] [,3]
[1,]    2    3    4
[2,]    5    6    7
> x^2
     [,1] [,2] [,3]
[1,]    1    4    9
[2,]   16   25   36
```

```
> t(c(1,2,3))
     [,1] [,2] [,3]
[1,]    1    2    3
> t(t(c(1,2,3)))
     [,1]
[1,]    1
[2,]    2
[3,]    3
```

Ejercicios de matrices

1. Crear una matriz de 10x10 valores numéricos y poner a 0 los valores que estén en las filas 3 a 5 y en las columnas 5 a 8.
2. Crear una matriz de 10x10 valores numéricos y poner a 0 aquellos valores que tengan fila par y columna impar.
3. Crear una matriz de 10x10 valores numéricos (+ y -) y poner a 0 los valores negativos (por ejemplo, se puede hacer con la secuencia -50:49 desordenada con sample)
4. Crear una matriz de 10x5 valores numéricos (+ y -) y seleccionar las filas que tengan valores negativos en la cuarta columna

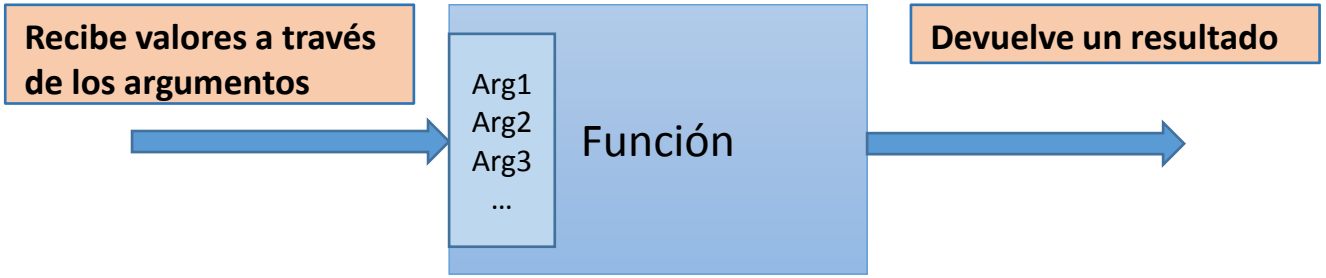
Ejercicios de matrices
Ejercicios evaluables 1

Leer el fichero *CalidadAire.txt* de Aula Global y guardarlo en una variable m que será un data.frame (se verá en la siguiente clase). Lo convertiremos a matriz con: m<-as.matrix(m). Si lo visualizamos observaremos que las filas y columnas tienen nombre.

1. Averiguar el número de filas y de columnas de m
2. Comprobar que m es una matriz. Visualizar los nombres de las columnas
3. Visualizar la columna *Solar.R* con índices de valores y con nombre. Guardar el resultado en *v*
4. ¿Qué es v? ¿coincide su longitud con el número de filas de m?
5. ¿En cuántos días no hay datos de radiación solar?
6. Visualizar las filas que corresponden al mes de agosto
7. Visualizar la cantidad de ozono de cada día del mes de agosto
8. ¿En cuántos días no hay medidas de ozono en el mes de agosto?
9. ¿Qué días son?
10. Calcular la media del Ozono en agosto (¿sale NA?, ¿por qué?, ¿qué hay que hacer para no tener en cuenta los NA?)

Funciones

- Las funciones se tratan como otros objetos en R
- Se pueden pasar como argumentos a otras funciones




```
f <- function (lista de argumentos formales) {
        ## líneas que hacen algo interesante
        ## la última expresión evaluada es el valor devuelto
}
```

- Se puede devolver el valor con return(valor)
- Si no hay return(), la última expresión es el valor que se devuelve

Funciones

- Ejemplo: función f que recibe dos números y eleva el primero al segundo como exponente

Argumentos formales

```
f<-function(a,b){
    #funcion que devuelve a elevado a b
    return(a^b)

}
```

```
f<-function(a,b){
    #funcion que devuelve a ^ b
    a^b

}
```

Definición de la función

- Ahora podemos **llamar** a esta función como a cualquier otra función de R

```
> f(2,3)
[1] 8
> f(2,300)
[1] 2.037036e+90
```

Llamada a la función

Matching por posición: el 2 pasa a **a** y el 3 a **b**

- Como utiliza los dos argumentos, si falta alguno en la llamada da un error

```
> f(2)
Error in f(2) : argument "b" is missing, with no default
```

Argumentos de funciones

- Las funciones tienen argumentos con nombre que pueden tener valores por omisión (default values)

```
f1<-function(a,b=1){
    #funcion que devuelve a ^ b
    # b vale 1 por omisión
    a^b

}
```

- En este caso, si no se especifica el valor de b, valdrá 1

```
> f1(3)
[1] 3
> f1(3,4)
[1] 81
```

b valdrá 1	**Matching** por posición: el 3 pasa a **a** y a **b** no se le pasa ningún valor, por lo tanto tomará el valor 1
b valdrá 4	**Matching** por posición: el 3 pasa a **a** y a **b** se le pasa el valor 4

Argumentos de funciones

- ¿Qué ocurre si el valor por omisión es el primero y falta un argumento en la llamada?

```
f2<-function(a=1,b){
    #funcion que devuelve a ^ b
    # a vale 1 por omisión
    a^b

}

> f2(3,4)
[1] 81
> f2(3)
Error in f2(3) : argument "b" is missing, with no default
```

Matching por posición: el 3 pasa a **a** y a **b** no se le pasa ningún valor, por lo dará error

Argumentos de funciones

- Como vemos, el matching o correspondencia de argumentos se hace **por posición**
- También puede hacerse **por nombre**
  - Especificamos el nombre del argumento en la llamada
  - Pueden mezclarse argumentos con nombre con otros sin nombre
  - Cuando hay una lista larga de argumentos es conveniente usar el nombre
  - Primero se resuelven los que tienen nombre y luego los demás se asignan por posición

```
f1<-function(a,b=1){
    a^b
}
```

```
> f1(b=3,a=2)
[1] 8

> f1(a=2,3)
[1] 8
> f1(b=3,2)
[1] 8
```

2^3

```
> f1(a=2)
[1] 2
> f1(b=2)
Error in f1(b = 2)
```

2^1

- La función args (f) nos da la lista de argumentos de cualquier función f

Argumentos de funciones

- Es importante entender el funcionamiento de la asignación de argumentos, no sólo cuando llamamos a nuestras funciones sino también a cualquier función de R
- Ejemplo: función sd()

```
> args(sd)
function (x, na.rm = FALSE)
```

  - Los argumentos son:
  - x es un vector numérico y ra.rm un argumento lógico, por omisión FALSE
  - Supongamos un vector v<-c(3,5,2)
  - Las siguientes llamadas son equivalentes, aunque no es recomendable alterar el orden, por claridad.

```
> sd(v)
[1] 1.527525
> sd(x=v, na.rm=F)
[1] 1.527525
> sd(na.rm=F,v)
[1] 1.527525
> sd(F,x=v)
[1] 1.527525
```

Cuidado:

```
> sd(x,na.rm=F)
Error in is.data.frame(x) : object 'x' not found
```

¿Qué ocurre aquí?: el objeto x no existe en nuestro entorno, sólo dentro de la función. Si existiera otro vector x en nuestro entorno, se calcularía la *sd* de este x

Ejercicios con funciones

1. Hacer una función **genera.vector(n,numNa=0)** que genere un vector de ***n*** números enteros entre 1 y 100 , mezclados con ***numNa*** Nas. Si no indicamos el número de NAs, será por omisión 0.
2. Mejorar la función anterior añadiendo los argumentos ***min*** y ***max***, que indican el número mínimo y máximo de la secuencia aleatoria (en lugar de 1 y 100)

Ejercicios con funciones

1. La función puede ser la siguiente

```
genera.vector<-function(n,numNa=0){
  #función que genera un vector de n números entre 1 y 100 con numNa valores NA.
  v<-sample(1:100,n,rep=T)    # también v<-round(runif(n,min=1,max=100))
  vna<-vector("numeric",numNa)    #ha creado un vector con numNA posiciones a 0
  vna[]<-NA   #asignamos NA al vector completo
  #ahora fundimos los dos vectores y los permutamos aleatoriamente
  v<-c(v,vna)
  v<-sample(v)   #esta ultima expresión es el vector que se devuelve
  v          #  si no se pone, no se visualiza el vector
}
```

2. La función mejorada:

```
genera.vector1<-function(n,numNa=0,min=1,max=100){
  #función que genera un vector de n números entre 1 y 100 con numNa valores NA.
  v<-sample(min:max,n,rep=T)
  vna<-vector("numeric",numNa)    #ha creado un vector con numNA posiciones a 0
  vna[]<-NA   #asignamos NA al vector completo
  #ahora fundimos los dos vectores y los permutamos aleatoriamente
  v<-c(v,vna)
  v<-sample(v)   #esta ultima expresión es el vector que se devuelve
  v
}
```

Estructuras de control

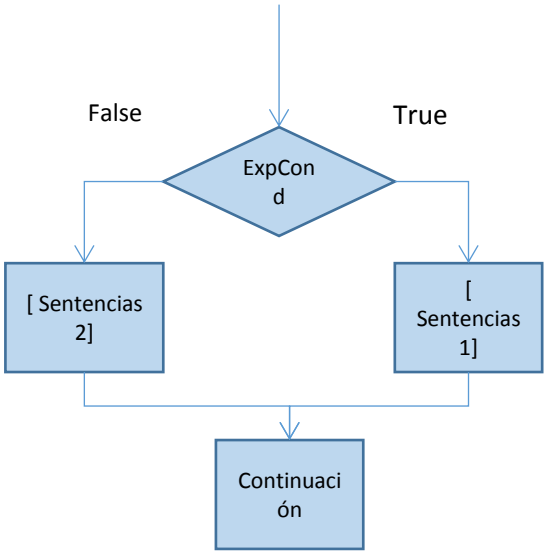
- Las estructuras de control permiten controlar el flujo de ejecución del programa dependiendo de condiciones que se den en el momento de su ejecución
- Estructuras de control más comunes
  - Estructura condicional: **if, else**
  - Estructuras repetitivas (bucles):
    - **for**
    - **while**
    - **repeat**
- Normalmente en sesiones interactivas no se usa, pero sí cuando se escriben funciones o programas

Sentencia if

- Si se cumple la condición se ejecuta el bloque de sentencias1, y si no, se ejecuta el bloque de sentencias2

```
If (condicion) {
            ## sentencias1
 } else {
            ## sentencias2
}
```

- El bloque else es opcional



Sentencia if

- Ejemplo

```
if (x > 3) {
            y<- 10
 } else {
            y<- 0
}
```

- Ejemplo: crear un vector v, y calcular su suma si tiene menos de 10 elementos y su media en caso contrario

```
v<-genera.vector(900)   #aprovechamos la función que hicimos antes
if (length(v)<10){
   resultado <- sum(v)
}else  {
   resultado<-mean(v)
}
resultado                  # visualizamos el resultado
```

Sentencia if

- Ejemplo: Preguntar al usuario si quiere realizar la suma o la media. Hacer la operación correspondiente con el vector

```
# preguntar si quiere media o suma
print("qué cálculo quiere hacer, suma o media? (s/m)")
r=scan(what="character")
if (r=="s"){
   resultado <- sum(v)
}else  {
   resultado<-mean(v)
}
resultado
# escribir el código en un script y ejecutarlo con source
```

Sentencia for

- Es muy utilizada para realizar acciones sobre los elementos de un objeto (recorrer el objeto)
- Tiene una variable de control que va tomando sucesivos valores de una secuencia
- Ejemplo:
  - Imprimir los cuadrados de los 10 primeros números

```
for(i in 1:10) {
                print (i^2)
 }
```

  - Imprimir los números pares de un vector

```
v<-c(3,6,4,9,2,1,4)
for (i in 1:length(v)){
   if (v[i]%%2 == 0){     # si es par
      print(v[i])
   }
}
```

Sentencia for

- Podemos anidar varias sentencias for (bucles anidados)
- Ejemplo: Recorrer cada elemento individual de una matriz, por filas o por columnas

```
#bucles anidados
#recorrer una matriz por filas/columnas
for (i in 1:nrow(m)){
   for (j in 1:ncol(n)){
      print(paste("Elemento ",i,",",j," : ",m[i,j]))
   }
}
```

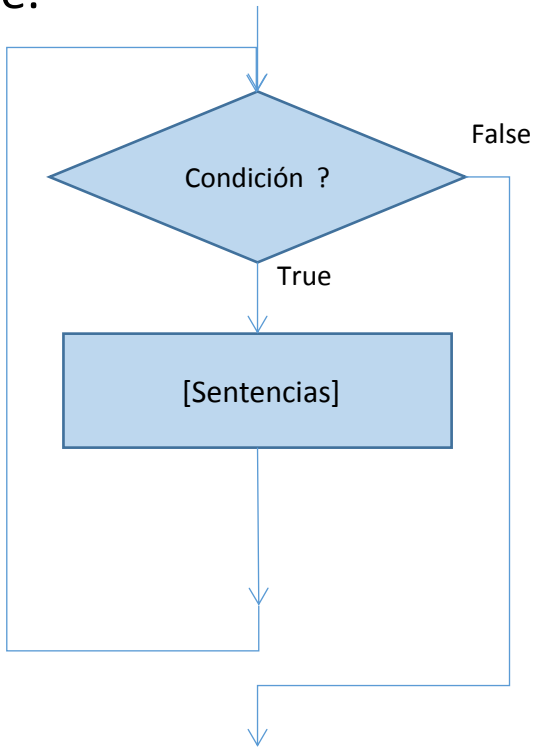
```
> m
     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
[4,]    4    9   14
[5,]    5   10   15
```

```
[1] "Elemento  1 , 1  :  1"
[1] "Elemento  1 , 2  :  6"
[1] "Elemento  1 , 3  :  11"
[1] "Elemento  2 , 1  :  2"
[1] "Elemento  2 , 2  :  7"
[1] "Elemento  2 , 3  :  12"
[1] "Elemento  3 , 1  :  3"
[1] "Elemento  3 , 2  :  8"
[1] "Elemento  3 , 3  :  13"
[1] "Elemento  4 , 1  :  4"
[1] "Elemento  4 , 2  :  9"
[1] "Elemento  4 , 3  :  14"
[1] "Elemento  5 , 1  :  5"
[1] "Elemento  5 , 2  :  10"
[1] "Elemento  5 , 3  :  15"
```

Sentencia while

- Mientras se cumpla la condición se ejecutan las sentencias
- Si en el ejemplo nos olvidamos de la segunda sentencia, siempre se cumplirá la condición y nunca saldríamos del bucle.
  Sería un bucle infinito (cuidado!)

```
contador <- 1
while (contador < 10) {
        print(contador)
        contador <- contador + 1
}
```



Ejercicios de funciones y estructuras de control
Ejercicios evaluables 1

Realizar una función que reciba una matriz de 9x9, que representa un sudoku, y devuelva un valor lógico TRUE si el sudoku está bien formado, o FALSE en caso contrario.

- Consideraremos que el sudoku está bien formado si todas las filas y columnas tienen todos los números del 1 al 9, es decir, no se repite ningún valor
- No consideraremos la condición de que en las submatrices de 3x3 no se repita ningún valor

- PISTAS
  - Comprobar que hay 9 filas y columnas y que los valores están comprendidos entre 1 y 9
  - Recorrer las filas con un bucle for y comprobar con unique() que los vectores tienen 9 elementos.
    - unique(v) devuelve un vector eliminando los elementos repetidos de v
  - Recorrer las columnas y hacer lo mismo
  - Cuando alguna condición no se cumpla, salir de la función con *return (FALSE)*
  - Cuando lleguemos al final, salir con return(TRUE)

9	4	8	5	6	2	7	3	1
6	2	3	1	8	7	4	5	9
5	7	1	4	3	9	2	8	6
3	9	4	2	7	6	8	1	5
7	1	5	8	4	3	9	6	2
2	8	6	9	5	1	3	7	4
1	6	7	3	9	4	5	2	8
4	5	2	7	1	8	6	9	3
8	3	9	6	2	5	1	4	7

Ejercicios de funciones y estructuras de control
Ejercicios evaluables 1

Ejercicio opcional

- Hacer el mismo ejercicio pero considerando además la condición de que no puede haber elementos repetidos en las submatrices.

- Habría que recorrer cada una de las nueve submatrices, convertirlas en vector y comprobar que no hay elementos repetidos. Desde dentro de esta función se puede llamar a la función anterior para no repetir todo el código.