

Introducción a la programación en R

Sitio: [Plataforma de Formación On line del Instituto Andaluz de Administración Pública](#)

Curso: (I22F-PT05) Entorno de Programación R

Libro: Introducción a la programación en R

Imprimido por: ALFONSO LUIS MONTEJO RAEZ

Día: miércoles, 13 de abril de 2022, 13:19

Tabla de contenidos

1. Introducción

2. Estructuras de programación

2.1. If

2.2. if else y función ifelse()

2.3. Bucle for

2.4. Bucle while

2.5. Bucle repeat

2.6. Detención de bucles

3. Anidar bucles

4. Funciones

5. Ejemplo completo

1. Introducción

Una de las ventajas de R es que, aparte de tener un conjunto de funciones para la carga, tratamiento y análisis estadístico de datos, cuenta con un lenguaje de programación con el que podemos construir scripts o grupos de instrucciones para la realización de tareas, normalmente de tipo repetitivo.

Hay varias cosas que tenemos que tener en cuenta a la hora de programar con R:

- **El lenguaje de R es interpretado**, lo que quiere decir que lo podemos ejecutar directamente si necesidad de compilar el código del programa.
- **No podemos usar las estructuras de programación en modo de trabajo interactivo**. Para poder usarlas tenemos que trabajar desde un script que luego ejecutaremos para ver su resultado.

A lo largo de la unidad conoceremos las principales instrucciones así como la creación de funciones.

2. Estructuras de programación

Dentro de un lenguaje de programación son muy importantes las estructuras que manejan el flujo de ejecución de un programa. Estas estructuras permiten variar el orden de ejecución de las órdenes dándonos más control sobre el código.

Las principales instrucciones para controlar la programación serían:

- **Condicionales.** If, if else.
- **Repeticiones (Bucles):** for, while, repeat.

Vamos a ver cada una de estas estructuras detenidamente y con ejemplos.

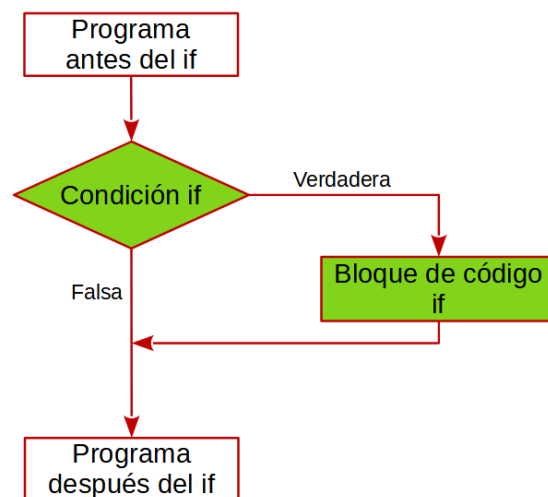
2.1. If

El **if** es una estructura condicional simple que nos permite ejecutar una serie de órdenes si se cumplen unas determinadas condiciones. En caso de no cumplirse esas condiciones el programa continua su ejecución normal.

A tener en cuenta:

- Las órdenes que se ejecutan, en caso de cumplirse la condición, deben ir entre llaves `{}`.
- La condición que pongamos debe ir entre paréntesis `()`.
- Y podemos usar una o varias, enlazadas con `&` (and) o `|` (or).

Las estructura básica sería **if(condición){órdenes}** y gráficamente sería así:



Ejemplo

Vamos a crear un condicional que compruebe si un número generado al azar entre el -100 y el 100 es negativo. En caso de que el número sea negativo escribirá un mensaje por pantalla.

```
# Generamos el valor y lo almacenamos
valor<-sample(-100:100,1)
# Hacemos la comprobación
if(valor<0){print('Este número es negativo')}
# Mostramos el número por pantalla
valor
```

Resultado si se cumple la condición

```
> # Generamos el valor y lo almacenamos
> valor<-sample(-100:100,1)
> # Hacemos la comprobación
> if(valor<0){print('Este número es negativo')}
[1] "Este número es negativo"
> # Mostramos el número por pantalla
> valor
[1] -22
```

Resultado si no se cumple la condición

```
> # Generamos el valor y lo almacenamos  
> valor<-sample(-100:100,1)  
> # Hacemos la comprobación  
> if(valor<0){print('Este número es negativo')}  
> # Mostramos el número por pantalla  
> valor  
[1] 3
```

Si el valor no cumple la condición (ser negativo) el programa se salta la orden de escribir por pantalla.

2.2. if else y función ifelse()

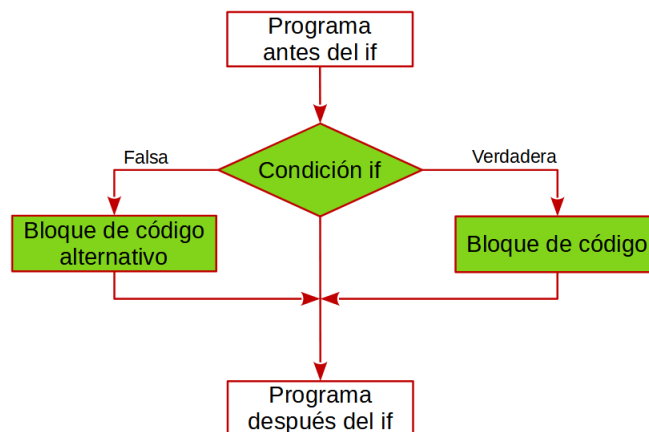
El **if else** es una estructura condicional doble, que permite crear condicionales en las que hay dos acciones alternativas.

Si se cumplen una serie de condiciones, se ejecuta un bloque de instrucciones y si no se cumplen se ejecuta un bloque de instrucciones alternativo.

A tener en cuenta:

- Las órdenes que se ejecutan, en caso de cumplirse la condición, deben ir entre llaves **{}**.
- La condición debe ir entre paréntesis **()**.
- Y pueden ser una o varias, enlazadas con **&** (and) o **|** (or).

La estructura básica sería **if(condición){órdenes} else {órdenes alternativas}** y gráficamente sería así:



Ejemplo 1

Vamos a crear un condicional que compruebe el signo de un número generado al azar entre el -100 y el 100. Nos mostrará en pantalla un mensaje mostrando el signo del número generado.

```
# Generamos el valor y lo almacenamos
valor<-sample(-100:100,1)
# Hacemos la comprobación
if(valor<0){print('Este número es negativo')}
else {print('Este número es positivo')}
```

```
# Mostramos el valor por pantalla
valor
```

Resultado si se cumple la condición

```
> # Generamos el valor y lo almacenamos
> valor<-sample(-100:100,1)
> # Hacemos la comprobación
> if(valor<0){print('Este número es negativo')} else
+ {print('Este número es positivo')}
[1] "Este número es negativo"
> # Mostramos el número por pantalla
> valor
[1] -81
```

Resultado si no se cumple la condición

```
> # Generamos el valor y lo almacenamos
> valor<-sample(-100:100,1)
> # Hacemos la comprobación
> if(valor<0){print('Este número es negativo')} else
+ {print('Este número es positivo')}
[1] "Este número es positivo"
> # Mostramos el número por pantalla
> valor
[1] 11
```

Como hemos dicho antes, todas las estructuras de programación solo se pueden emplear en modo script.

Sin embargo, existe una función que tiene la misma utilidad que la estructura if else, pero puede ser usada en modo interactivo.

La función se llama **ifelse()** y su forma básica sería:

ifelse(condicion,acción si condición es verdadera, acción si condición falsa)

Ejemplo 2

Vamos a generar una muestra de 55 números elegidos entre el -100 y el 100 y vamos a ver el signo que tienen.

```
muestra<-sample(-100:100,55)
ifelse(muestra<0,'negativo','positivo')
```

```
> muestra
[1] -91 63 -73 -24 38 40 20 -57 30 52 72 -65 69 -79 -32 84 -59 61 43 2 71 37 7 -70 -39 89 -89 81
[29] 64 -51 -44 -81 -56 -63 56 -49 -42 -30 -53 55 28 60 78 -25 9 -43 -7 14 -26 -18 85 22 76 -97 -46
> ifelse(muestra<0,'negativo','positivo')
[1] "negativo" "positivo" "negativo" "negativo" "positivo" "positivo" "positivo" "negativo" "positivo" "positivo"
[11] "positivo" "negativo" "positivo" "negativo" "negativo" "positivo" "negativo" "positivo" "positivo" "positivo"
[21] "positivo" "positivo" "positivo" "negativo" "negativo" "positivo" "negativo" "positivo" "positivo" "negativo"
[31] "negativo" "negativo" "negativo" "negativo" "positivo" "negativo" "negativo" "negativo" "negativo" "positivo"
[41] "positivo" "positivo" "positivo" "negativo" "positivo" "negativo" "negativo" "positivo" "negativo" "negativo"
[51] "positivo" "positivo" "positivo" "negativo" "negativo"
```


2.3. Bucle for

El bucle **for** es una estructura repetitiva que realiza una o varias órdenes un número determinado de veces.

A tener en cuenta:

- Las órdenes que se ejecutan deben ir entre llaves **{}**.
- La secuencia debe ir entre paréntesis **()**.
- La secuencia puede ser numérica (ej. del 1 al 5) o un conjunto de valores.
- Normalmente se usa como indicador de la secuencia la letra i, pero podemos usar cualquiera.

La forma básica del bucle es **for(i in secuencia){órdenes}**.

Ejemplo 1

Vamos a mostrar por pantalla los números del 1 al 5.

```
for(i in 1:5){print(i)}
```

En que secuencia Acción a ejecutar

```
> for(i in 1:5){print(i)}  
[1] 1  
[1] 2  
[1] 3  
[1] 4  
[1] 5
```

Ejemplo 2

Vamos a crear un vector que almacene la tabla de multiplicar del 9

```
> a<-0  
> for (i in 1:10){a[i]<-i*9}  
> a  
[1] 9 18 27 36 45 54 63 72 81 90
```

2.4. Bucle while

El bucle **while** es una estructura repetitiva que realiza una o varias órdenes mientras que se cumpla una condición; cuando la condición deja de cumplirse se detiene la ejecución del bucle.

A tener en cuenta:

- Las órdenes que se ejecutan mientras se cumpla la condición deben ir entre llaves **{}**.
- La condición debe ir entre paréntesis **()**.
- La condición a cumplir puede ser una o varias enlazadas con **&** (and) o **|** (or).
- Debemos tener cuidado al definir la condición. Hay que elegir una que pueda dejarse de cumplir ya que, si siempre se cumple, se genera un bucle infinito que no se puede detener.

La forma básica del bucle es **while(condición){órdenes}**.

Ejemplo

Vamos a generar un número que se vaya incrementando desde el 0 al 10 y que se muestre por pantalla siempre que sea menor a 10.

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

Condición

Acción a ejecutar

```
> n<-0
> while(n<10){n<-n+1; print(n)}
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

2.5. Bucle repeat

El bucle **repeat** es una estructura repetitiva que realiza una o varias órdenes hasta que se cumpla una condición establecida para que el bucle se pare.

A tener en cuenta:

- Las órdenes que se ejecutan deben ir entre llaves **{}**.
- Debemos definir una condición para detener la ejecución del bucle ya que si no, se genera un bucle infinito.
- La condición debe ir entre paréntesis **()**.
- Y puede ser una o varias enlazadas con **&** (and) o **|** (or).

La forma básica del bucle es **repeat{órdenes; if(condición){break}}**.

Ejemplo

Vamos a generar un número que se vaya incrementando desde el 1 y que se muestre por pantalla. Cuando el número sea 6 se detiene la ejecución del bucle.

```
repeat {print(x);x = x+1;if (x == 6){break}}
```

Acciones a ejecutar

Condición de salida

```
> x <- 1
> repeat {print(x);x = x+1;if (x == 6){break}}
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

2.6. Detención de bucles

Hay ocasiones que necesitamos detener la ejecución de un bucle. En estos casos, hay dos instrucciones que pueden emplearse con todos los bucles para detener la acción que se está ejecutando o bien saltar al siguiente valor de una sucesión.

Estas órdenes son:

- **next** que interrumpe la ejecución de un bucle devolviéndolo al siguiente paso.
- **break** que detiene por completo la acción de un bucle cuando se cumpla una condición, por lo que siempre debe ir acompañada de una condición if.

Ejemplo 1

Vamos a usar un bucle for y vamos a detenerlo cuando se cumpla una condición con **break**.

```
for (i in c('AL', 'CA', 'CO', 'GR')){  
  print(i)  
  if(i=='CO') break  
}  
  
> for (i in c('AL', 'CA', 'CO', 'GR')){  
+   print(i)  
+   if(i=='CO') break  
+ }  
[1] "AL "  
[1] "CA "  
[1] "CO "
```

Este bucle va a mostrar por pantalla las iniciales de las provincias de Andalucía hasta llegar a la provincia de Córdoba que detiene definitivamente la ejecución del bucle.

Ejemplo 2

Vamos a usar un bucle for y vamos a saltar a otro valor cuando se cumpla una condición con **next**.

```
for(i in 1:20){  
  if (i%%5==0) next  
  print(i)  
}
```

```
> for (i in 1:20){  
+   if(i%%5==0) next  
+   print(i)  
+ }  
[1] 1  
[1] 2  
[1] 3  
[1] 4  
[1] 6  
[1] 7  
[1] 8  
[1] 9  
[1] 11  
[1] 12  
[1] 13  
[1] 14  
[1] 16  
[1] 17  
[1] 18  
[1] 19
```

Este bucle va a mostrar por pantalla los número del 1 al 20, pero saltando los que sean divisibles por 5 que no aparecerán. La función `%%` es el resto de una división.

3. Anidar bucles

Otra de las ventajas de las estructuras de control de flujo en un programa es la posibilidad de **anidar bucles**, es decir, utilizar conjuntamente varios bucles en una misma acción.

Las estructuras más usadas conjuntamente son el bucle **for** y el condicional **if else**. En este caso se realizan una serie de acciones de manera repetida y dentro del bucle for se añade un condicional que permite la ejecución de dos bloques de código alternativos.

Al anidar bucles tenemos que mantener las normas de uso de cada uno de ellos de manera separada.

Ejemplo

Vamos a usar un bucle for anidado con un condicional if else para clasificar los cincuenta primeros números como pares o impares.

```
n<-50
pares<-c(); impares<-c()

for(i in 1:n){
  if(i%%2==0) {pares<-c(pares,i)} else {impares<-c(impares,i)}
}

> pares
[1]  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50
> impares
[1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
```

Para ello usamos el bucle for para ir recorriendo los números del 1 al 50 y dentro del for usamos el condicional if else para preguntar si el número es par o no. En el caso de ser par el resto de la división del número (función `%%`) entre 2 debe ser cero.

El resultado son dos vectores cada uno conteniendo los valores pares e impares del 1 al 50.

4. Funciones

Otra característica de R es que nos permite crear nuevas órdenes o funciones que realicen acciones que no están contempladas en el programa original.

Las **funciones** son un conjunto de órdenes que tienen unos parámetros de entrada (argumentos) con los que se van a realizar acciones que van a devolver un resultado.

A tener en cuenta:

- Dentro de la función podemos usar cualquier orden ya definida en R y también las estructuras de control como bucles y condicionales.
- Se crean usando la orden **function()**.
- Deben ser asignadas a un objeto para poder usarlas.
- Los parámetros de entrada de la función se deben poner entre paréntesis ().
- El conjunto de órdenes de la función tienen que ir entre llaves {}.

La forma básica de una función será:

nombre_función←function(parámetros de entrada){acciones}

Ejemplo

Vamos a construir una función que calcule el coeficiente de variación. El coeficiente de variación de una variable se calcula como la desviación típica dividida entre el valor absoluto de la media.

A la función la vamos a llamar **cv** y tendrá un parámetro de entrada, que será la variable o conjunto de datos a los que queremos calcularles el coeficiente de variación.

```
cv<-function(x){  
  if (length(x)<=1){'No se puede calcular el coeficiente de variación'} else {  
    sd(x)/abs(mean(x))}  
}
```

Dentro de las órdenes que tenemos que ejecutar vamos a introducir un condicional que controle que haya más de un dato en la variable, ya que no tiene sentido calcular el coeficiente de variación de solo un dato. Para ello usamos la función **length()** que nos da el número de elementos de un objeto.

Para el cálculo usamos las funciones **sd()**, **abs()** y **mean()**.

El resultado sería el siguiente:

```
> cv(1)  
[1] "No se puede calcular el coeficiente de variación"  
> cv(c(1,2,3))  
[1] 0.5  
> cv(sample(1:1000,200))  
[1] 0.533128
```

5. Ejemplo completo

Para terminar esta unidad, vamos a desarrollar un ejemplo completo de uso de una estructura de programación dentro de un data frame.

Para ello vamos a usar los datos de migraciones por municipios de Andalucía que se encuentran en el fichero `migracion_mun.RData` que contiene dos data frames:

- **emigraciones.** Que contiene las salidas de personas de Andalucía por municipios.
- **inmigraciones.** Que contiene las entradas de personas a Andalucía por municipios.

Los dos data frames tienen las variables de Municipio (Nombre del municipio), `Cod_mun` (Código INE del municipio), Sexo y Movimiento (Número de personas).

Antes de empezar con el ejercicio quisiera hacer un pequeño comentario para quien no conozca la codificación de municipios del INE.

El Instituto Nacional de Estadística (INE) tiene una clasificación oficial con todos los municipios de España. Este código está formado por cinco caracteres siendo los dos primeros los correspondientes a la provincia y los tres siguientes al municipio. Para asignar los códigos a las provincias (los dos primeros dígitos) se ordenan alfabéticamente por el nombre de la provincia y se numeran desde el 1.

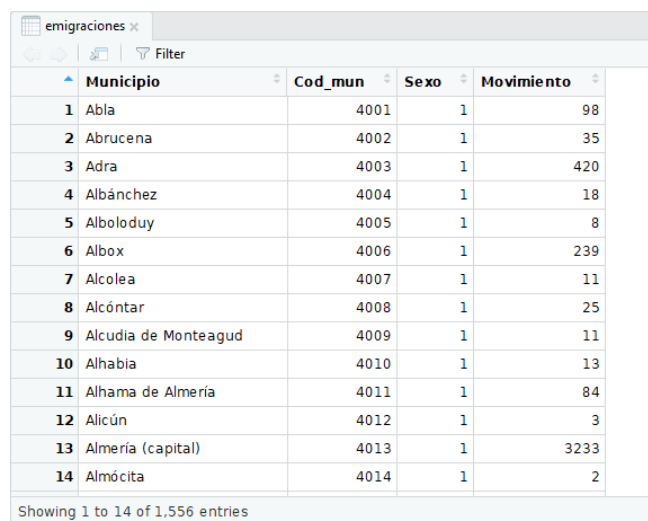
Las provincias que tienen código desde el 1 al 9, para completar los cinco caracteres que tiene que tener el código completo, se rellenan con un cero al principio. Por ejemplo, el código INE del municipio de Almería sería 04013 (04 de la provincia de Almería y 013 por el municipio), el de Granada sería 18087, el de Sevilla sería 41091.

Ejemplo

Con el fichero de `migraciones_mun.RData`, vamos a arreglar el data frame `emigraciones` realizando las siguientes acciones:

1. Arreglar la codificación de los municipios de Almería (que les falta el cero inicial).
2. Separar de la variable `Cod_mun` los datos de provincia y municipio creando dos nuevas variables para guardar los dos datos por separado.

Lo primero que tenemos que hacer es cargar el fichero `migraciones_mun.RData` y visualizar el data frame `emigraciones`.



	Municipio	Cod_mun	Sexo	Movimiento
1	Abla	4001	1	98
2	Abrucena	4002	1	35
3	Adra	4003	1	420
4	Albánchez	4004	1	18
5	Alboloduy	4005	1	8
6	Albox	4006	1	239
7	Alcolea	4007	1	11
8	Alcántar	4008	1	25
9	Alcudia de Monteagud	4009	1	11
10	Alhabia	4010	1	13
11	Alhama de Almería	4011	1	84
12	Alicún	4012	1	3
13	Almería (capital)	4013	1	3233
14	Almócita	4014	1	2

Si vemos la columna de la variable `Cod_mun`, a los municipios de Almería les falta el cero inicial.

Vamos a realizar las dos acciones (arreglar el código y dividir en dos variables) usando un bucle que recorra todo el data frame.


```

for(i in 1:nrow(emigraciones)){

  # Primer paso añadirle el cero a los códigos de Almería
  if(nchar(emigraciones$Cod_mun[i])<5){

    paste0('0',emigraciones$Cod_mun[i])->emigraciones$Cod_mun[i]

  }

  # Segundo paso crear las variables Prov y Mun
  substr(emigraciones$Cod_mun[i],1,2)->emigraciones$Prov[i]
  substr(emigraciones$Cod_mun[i],3,5)->emigraciones$Mun[i]

}

```

Con el programa lo que hacemos es recorrer todas las filas del data frame migraciones (el valor de i me indica en que fila estoy del data frame).

Dentro del bucle hemos metido una primera acción es con un condicional if() arregla solo el código de los municipios de Almería, dejando al resto como estaban y después creamos las otras dos variables particionando la variable Cod_mun.

	Municipio	Cod_mun	Sexo	Movimiento	Prov	Mun
1	Abla	04001	1	98	04	001
2	Abrucena	04002	1	35	04	002
3	Adra	04003	1	420	04	003
4	Albánchez	04004	1	18	04	004
5	Alboloduy	04005	1	8	04	005
6	Albox	04006	1	239	04	006
7	Alcolea	04007	1	11	04	007
8	Alcántar	04008	1	25	04	008
9	Alcudia de Monteagud	04009	1	11	04	009
10	Alhabia	04010	1	13	04	010
11	Alhama de Almería	04011	1	84	04	011
12	Alicún	04012	1	3	04	012
13	Almería (capital)	04013	1	3233	04	013
14	Almócita	04014	1	2	04	014

Showing 1 to 14 of 1,556 entries

El resultado es el mismo data frame (emigraciones), pero con el código arreglado y las dos variables creadas (Prov y Mun).

Estas acciones también podríamos haberlas hecho directamente con órdenes, sin tener que recurrir a un bucle.

```

# Primer paso añadirle el cero a los códigos de Almería
ifelse(nchar(emigraciones$Cod_mun)<5,
       paste0('0',emigraciones$Cod_mun),emigraciones$Cod_mun)->emigraciones$Cod_mun

# Segundo paso crear las variables Prov y Mun
substr(emigraciones$Cod_mun,1,2)->emigraciones$Prov
substr(emigraciones$Cod_mun,3,5)->emigraciones$Mun

```

Normalmente, es preferible hacer las acciones mediante órdenes directamente ya que los bucles suelen ser más lentos. En un data frame pequeño, la diferencia entre usar uno u otro método no se aprecia pero si el data frame tiene un número importante de registros (>100.000) el bucle siempre es la opción más lenta.