



Fastbook 03

Programación en R

Ficheros y control de flujo



03. Ficheros y control de flujo

En el anterior fastbook aprendimos a trabajar con los distintos tipos de datos y variables que existen en R. Como ya comentamos, su contenido es bastante amplio, por lo que no puede ser asimilado de forma automática y rápida, sino con tiempo y práctica. Por dicha razón, aunque en los siguientes fastbooks vayamos a centrarnos en nuevas habilidades, los componentes presentados hasta ahora van a seguir apareciendo, lo que os ayudará en su aprendizaje, repaso y uso.

Concretamente, este tercer fastbook va a cubrir dos elementos. Empezaremos trabajando con **ficheros**, gracias a los cuales podremos importar y exportar datos con facilidad y fluidez. En segundo lugar, estudiaremos aquellas **sentencias** que nos permiten controlar el flujo del programa, pudiendo decidir qué bloque de código se debe ejecutar en cada momento.

¡Comenzamos!

Autor: Juan Jiménez García

Ficheros

Introducción al control del flujo

Condicionales

Loops

Ficheros

X Edix Educación

Vamos a empezar con algunos conceptos básicos.

Los **ficheros** son elementos informáticos identificados por un nombre que almacenan cualquier tipo de información en forma de bytes.

La estructura y codificación con la que se organiza dicha información se denomina **formato**. Gracias a él, los programas son capaces de entenderla y extraerla.

La pregunta que puede surgirnos es: ¿cómo sabe el programa cuál es el formato que debe aplicar para acceder a la información de un determinado fichero? La respuesta nos lleva al concepto de extensión.

La **extensión de un fichero** es la cadena de caracteres precedida por punto que se encuentra al final de su nombre. Su función es la de informar sobre el tipo y formato del archivo de cara a que el software sepa cómo interpretarlo.



Pasemos a lo que nos interesa.

En cualquier proyecto de Analytics necesitamos almacenar y compartir distintos conjuntos de datos, y para eso, entre otras cosas, usamos ficheros.

En R podemos trabajar con una gran cantidad de **formatos** distintos, sin embargo, vamos a centrarnos en los tres que resultan **más importantes y útiles** en base al contenido del programa que estás cursando.

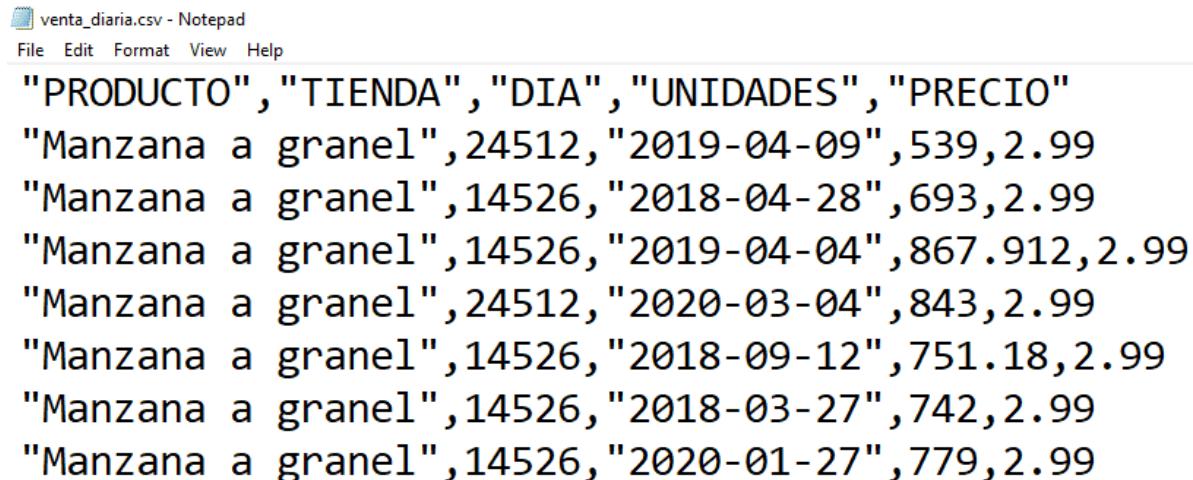
- **Ficheros de texto plano:** son archivos informáticos que únicamente contienen texto formado por caracteres. No se almacena ningún tipo de información relacionada con el aspecto visual de dicho texto.
- **Ficheros Excel:** son los archivos que se generan y editan desde Microsoft Excel.
- **Ficheros R:** son archivos que sirven para almacenar variables del lenguaje de programación R.

Ficheros de texto plano

La principal característica de los ficheros de texto es que **no tienen formato**. La información textual (solo caracteres) se encuentra en su interior sin ningún tipo de codificación o distribución compleja, por lo que **no necesitan ser interpretados para leerse**.

A la hora de trabajar con datos, los ficheros de texto plano son usados para almacenar información tabular, es decir, que se encuentra distribuida a lo largo de filas y columnas.

A través de una imagen podremos entenderlo de forma más sencilla.



PRODUCTO	TIENDA	DIA	UNIDADES	PRECIO
"Manzana a granel"	24512	"2019-04-09"	539	2.99
"Manzana a granel"	14526	"2018-04-28"	693	2.99
"Manzana a granel"	14526	"2019-04-04"	867.912	2.99
"Manzana a granel"	24512	"2020-03-04"	843	2.99
"Manzana a granel"	14526	"2018-09-12"	751.18	2.99
"Manzana a granel"	14526	"2018-03-27"	742	2.99
"Manzana a granel"	14526	"2020-01-27"	779	2.99

La imagen muestra el contenido del fichero `venta_diaria.csv`. El programa utilizado para su visualización es Notepad.

Como podéis ver, la primera fila contiene el nombre de cada una de las columnas. A partir de la segunda se encuentra el dato. **Cada fila es una muestra de información**, en este caso, las unidades vendidas de un producto en un determinado día, a un precio y en la correspondiente tienda. Resulta importante notar que cada uno de los campos se encuentra **separado por una coma**. Gracias a dicho delimitador podemos saber qué parte del texto corresponde a cada columna.

1

CSV significa **comma-separated-values**, debido a que, como ya hemos comentado, la separación entre campos se realiza con una coma. En ciertas ocasiones podemos encontrarnos archivos CSV separados por punto y coma debido a que sus siglas coinciden (**colon-separated-values**).

2

Bajo esta misma filosofía se construyen las extensiones **TSV (tab-separated-values)** y **DSV (delimiter-separated-values)**. En la primera de ellos usamos el tabulador como separador, mientras que en la segunda existe libertad para usar el delimitador que nos interese.

3

Por otro lado se encuentra la extensión **TXT**. Es la más famosa y usada a nivel general cuando la información almacenada no responde a una estructura tabular.



En cualquier caso, como ya se ha comentado, la extensión no es más que la etiqueta que colocamos en el archivo. Es decir, podríamos crear y acceder a la información de un archivo CSV aunque usase un tabulador como delimitador. Sin embargo, resulta recomendable respetar este tipo de convenciones y estándares.

Tras la visión teórica toca centrarnos en R. Vamos a empezar con la lectura de este tipo de ficheros haciendo uso de la función `read.table()`. Se encuentra incluida en Rbase, por lo que no es necesario importar librerías externas. Dicha función recibe la ruta del archivo (entre otros parámetros) y **devuelve un dataframe con la información rescatada**.

La siguiente tabla muestra sus atributos de entrada más importantes:

Función <code>read.table()</code>		
Parámetro	Clase	Definición
file	character	Ruta del archivo a leer (localización y nombre).
header	boolean	Indica si el fichero incluye o no los nombres de las columnas en la primera fila.
sep	character	Separador utilizado en el archivo.
dec	character	Carácter usado con los números decimales (punto o coma).
col.names	vector de characters	Vector con el nombre de las columnas en caso de que no estén en el archivo o se quieran modificar.
colClasses	vector de characters	Vector indicando la clase de dato de cada columna. (aunque no lo introduzcamos, R ya realiza una detección automática).
na.strings	vector de characters o character	Carácter o vector de caracteres usados para representar los datos perdidos.

Como se puede observar, la función nos brinda una gran cantidad de posibilidades con el objetivo de que podamos adaptarnos a cualquier tipo de estructura.

Con la intención de simplificar el proceso, existen versiones de la función `read.table` donde algunos de sus parámetros ya vienen definidos por defecto.

Por cierto, `\t` es la forma de representar el **tabulador**.

Función	Parámetros por defecto
<code>read.csv()</code>	<code>sep = "," dec = "."</code>
<code>read.csv2()</code>	<code>sep = ";" dec = ","</code>
<code>read.delim()</code>	<code>sep = "\t" dec = "."</code>
<code>read.delim2()</code>	<code>sep = "\t" dec = ","</code>

Las funciones presentadas hasta ahora nos ayudan a llevar la información de nuestros ficheros a dataframes de R. Cuando queremos realizar el proceso contrario, debemos usar las funciones de escritura. Con la misma lógica que la presentada hasta ahora, tenemos la función `write.table()` y sus parámetros de entrada:

Función `write.table()`

Parámetro	Clase	Definición
x	dataframe	Variable dataframe que queremos exportar.
file	character	Ruta del fichero que se va a crear.
sep	character	Separador a utilizar entre campos.
dec	character	Carácter a utilizar con los números decimales.
na	character	Texto que va a representar los valores perdidos.
row.names	boolean	Indica si los nombres de las filas deben introducirse en el fichero.
col.names	boolean	Indica si los nombres de las columnas deben introducirse en el fichero.

Al igual que en la lectura, existen versiones para algunos casos particulares:

Función	Parámetros por defecto
write.csv()	sep = “,” dec = “.”
write.csv2()	sep = “;” dec = “;”

Por último, es interesante que sepáis que, de cara a **conocer los valores por defecto de una función**, tenemos dos opciones. Podemos ir a su documentación (*help*) o ejecutar únicamente el **nombre de la función** (sin paréntesis), lo que automáticamente nos mostrará su código.

Ficheros Excel

Supongo que no es necesario hacer demasiadas presentaciones. Como todos ya sabemos, Excel es el software estándar de Office **para trabajar con datos, tablas, visualizaciones y análisis**.

En muchas ocasiones, la información que queremos trabajar desde R se encuentra alojada en este tipo de ficheros (extensión .xlsx), por lo que debemos conocer las funciones que nos facilitan su lectura y escritura.

En este caso tendremos que instalar e importar el **paquete openxlsx**.

La función **read.xlsx()** recibe la ruta del fichero a leer (entre otros parámetros) y devuelve un dataframe con la información.

Función `read.xlsx()`

Parámetro	Clase	Definición
<code>xlsxFile</code>	<code>character</code>	Ruta del archivo a leer (localización y nombre).
<code>sheet</code>	<code>integer o character</code>	Número o nombre de la hoja Excel que se va a leer (por defecto es la primera).
<code>startRow</code>	<code>integer</code>	Número de fila en el que empieza la tabla que queremos leer (por defecto es la 1).
<code>colNames</code>	<code>boolean</code>	Indica si la tabla incluye o no los nombres de las columnas en la primera fila.

En sentido contrario se encuentra la función `write.xlsx()`, la cual genera un fichero Excel a partir del dataframe recibido.

Función `write.xlsx()`

Parámetro	Clase	Definición
x	dataframe	Variable dataframe que queremos exportar.
file	character	Ruta y nombre del fichero que se va a crear.
sheetName	character	Nombre de la hoja Excel que se va a crear.
colNames	boolean	Indica si los nombres de las columnas deben introducirse en la tabla.
rowNames	boolean	Indica si los nombres de las filas deben introducirse en la tabla.

Ficheros R

Cuando la información con la que estamos trabajando tiene que ser compartida con otros sistemas o personas, resulta conveniente usar ficheros de texto plano o Excel. Sin embargo, si queremos **guardar ciertas variables para que posteriormente sigan siendo trabajadas desde R**, podemos usar los archivos de almacenamiento RDS y RData, diseñados justamente para eso.

- Los ficheros **RDS** tienen extensión .rds y nos permiten **almacenar una variable R en su interior**.
- Los ficheros **RData** tienen extensión .RData y nos permiten almacenar **más de una variable R en su interior**.

La siguiente tabla presenta todas las funciones que nos interesan:

Función	Definición	Parámetros
readRDS	Lee la variable que se encuentra en el interior de un fichero RDS.	file: ruta del archivo.
saveRDS	Guarda una variable en el interior de un fichero RDS.	object: variable R a guardar. file: ruta del archivo a crear.
load	Carga en el entorno las variables guardadas en un fichero RData.	file: ruta del archivo.
save	Guarda una o más variables en el interior de un fichero RData.	var1, var2...: todas las variables que queramos guardar separadas por coma. file: ruta del archivo a crear.
save.image	Guarda dentro de un fichero RData todas las variables que existen en el entorno.	file: ruta del archivo.

Lección 2 de 4

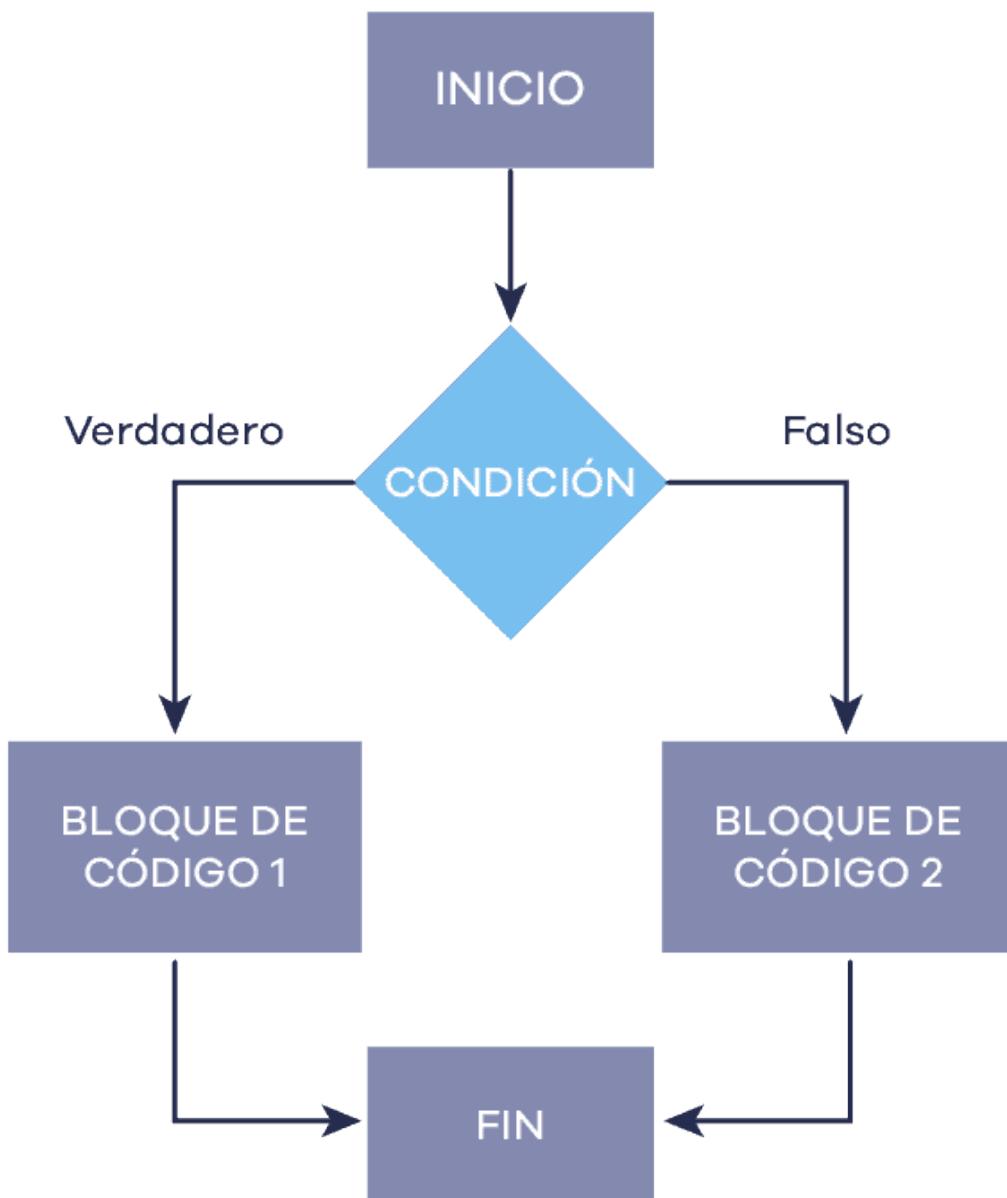
Introducción al control del flujo

X Edix Educación

El **control del flujo** nos permite manejar con libertad el proceso de ejecución de nuestro programa.

Haciendo uso de diferentes sentencias, podremos decidir bajo qué condiciones o cuántas veces se debe ejecutar un determinado bloque de código.

Existen dos grupos de elementos que nos permiten controlar el flujo: los **condicionales** y los **loops**.

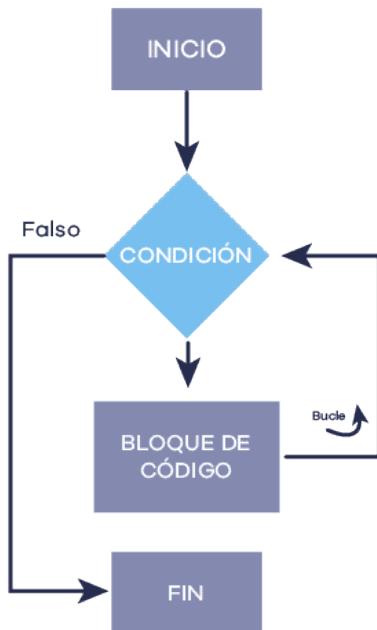


El primero de ellos usa condiciones para decidir **cuál es el camino que debe seguir el código**, como se observa en el gráfico.

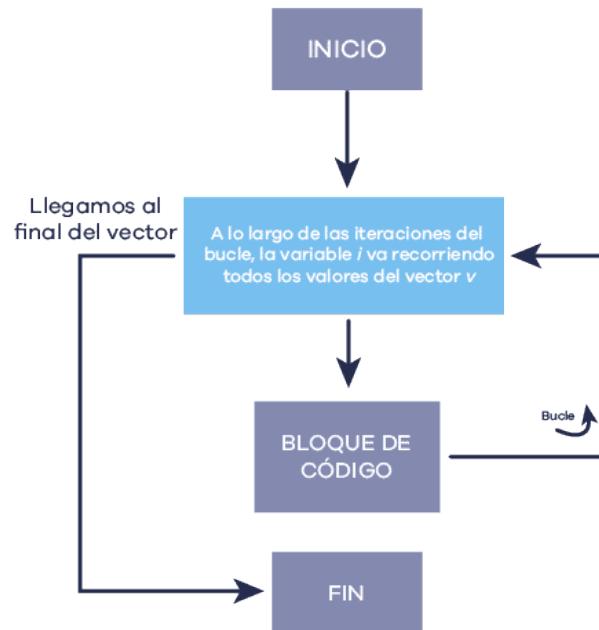
Cuando hablamos de **condicionales** nos referimos a cualquier tipo de **comprobación o comparativa** que nos interese realizar para decidir qué líneas de código hay que ejecutar. Por ejemplo, podríamos comprobar si las ventas de un año superan los 100.000€, si una determinada carpeta almacena el fichero que estamos buscando o si nuestro dataframe contiene el número de columnas correcto.

Por otro lado, se encuentran los **loops o bucles**. Su función es la de **repetir el mismo bloque de código** un número determinado de veces. En base a cómo se definan, podemos diferenciar entre dos tipos: los que **recorren un vector en cada una de sus iteraciones**, y los que **se mantienen en repetición hasta el momento en el que no se cumpla una determinada condición**.

Loop basado en condición



Loop basado en vector



Condicionales

X Edix Educación

Para generar condicionales en R tenemos que usar las sentencias **if** y **else**.

```
#Imaginemos un código inicial en el que el usuario introduce por teclado su contraseña.  
#Dicho texto lo asignamos a la variable password. Su contraseña es TOW2415R82.  
  
#Podemos aplicar condicionales para comprobar si es correcta  
  
if(password == "TOW2415R82") {  
    #Si la condición vale TRUE, se ejecuta este bloque de código  
    print("Contraseña correcta")  
}
```

En el caso de que la condición encerrada por **if** devuelva TRUE, el código encerrado por sus corchetes será ejecutado.

Con **else** podemos añadir un bloque de código alternativo que se ejecutará si la condición es FALSE.

```
#Imaginemos un código inicial en el que el usuario introduce por teclado su contraseña.  
#Dicho texto lo asignamos a la variable password. Su contraseña es TOW2415R82.  
  
#Podemos aplicar condicionales para comprobar si es correcta o errónea  
  
if(password == "TOW2415R82") {  
  
    #Si la condición vale TRUE, se ejecuta este bloque de código  
    print("Contraseña correcta")  
  
} else {  
  
    #Si la condición vale FALSE, se ejecuta este bloque de código  
    print("Contraseña errónea")  
  
}
```

Además, podemos realizar **anidaciones** en el caso de que queramos albergar más de dos posibilidades.

```
#Imaginemos un código inicial en el que el usuario introduce por teclado su contraseña.  
#Dicho texto lo asignamos a la variable password. Su contraseña actual es TOW2415R82  
#mientras que la antigua es QWE063RTK9.  
  
#Podemos aplicar condicionales para albergar las 3 posibilidades  
  
if(password == "TOW2415R82") {  
  
    #Si la condición vale TRUE, se ejecuta este bloque de código  
    print("Contraseña correcta")  
  
} else if(password == "QWE063RTK9") {  
  
    #Si la primera condición vale FALSE y la segunda TRUE, se ejecuta este código  
    print("Contraseña errónea. Has introducido tu antigua contraseña.")  
  
} else {  
  
    #Si ambas condiciones valen FALSE, se ejecuta este bloque de código  
    print("Contraseña errónea")
```

Loops

X Edix Educación

Como ya hemos comentado, existen dos tipos de loops que pueden ser implementados: los basados en un vector y los basados en una condición.

En la presente sección vamos a estudiar tres sentencias R que nos permiten generar bucles:

- **for** (basada en vector).
- **while** (basada en condición).
- **repeat** (también basada en condición).

En la práctica vais a daros cuenta de que cualquier tarea relacionada con la repetición de código puede ser resuelta con las tres metodologías. Sus diferencias simplemente radican en el enfoque y la estructura del código.

De forma adicional, mostraremos el funcionamiento de otros dos elementos valiosos cuando trabajamos con loops: **next** y **break**, que pueden ser usados con cualquier de las tres sentencias que acabamos de mencionar.

Bucle **for**

El bucle **for** es el más conocido y usado de los tres que vamos a ver. Centrándonos en su funcionamiento, facilita que una variable recorra los elementos de un vector a lo largo de las diferentes iteraciones del loop. En la iteración 1, la variable adquirirá el valor del vector que se encuentra en la posición 1; en la iteración 2, el del vector que se encuentra en la posición 2, y así sucesivamente.

```
#Vamos a definir un vector numérico que contiene la venta mensual de una tienda en el
#año #2020

venta_mensual <- c(4313,5313,3431,4515,3640,NA,3655,4124,5940,5678,4980,6758)

#Recorremos todos los elementos de un vector e imprimimos sus valores por pantalla
for (venta in venta_mensual){

  print(venta)

}

#Podemos hacer lo mismo con un vector numérico que nos permita acceder a cada una de las
#posiciones. Para eso creamos una secuencia que va desde 1 hasta la longitud del vector,
#12 en este caso

for ( posicion in seq(1,length(venta_mensual)) ){

  print(venta_mensual[posicion])

}

#También podemos modificar un determinado valor si resulta necesario. Por ejemplo,
cuando #tengamos un valor perdido le asignamos venta 0

for ( posicion in seq(1,length(venta_mensual)) ){

  if( is.na(venta_mensual[posicion]) ){

    venta_mensual[posicion] <- 0

  }

}
```

También conviene hablar de las sentencias *next()* y *break()*. Como ya hemos adelantado, aunque las presentemos en esta sección, pueden ser usadas en bucles *for*, *while* y *repeat*.

- **next():** saltamos automáticamente a la siguiente iteración del bucle.
- **break():** salimos automáticamente del bucle.

Bucle **while**

El bucle *while* de R trabaja bajo una condición. Mientras esta se cumpla (devuelva TRUE), el loop seguirá iterando ininterrumpidamente. Cuando la condición deje de cumplirse (devuelva FALSE), la ejecución saldrá del bucle.

A continuación podemos ver un ejemplo de uso.

```
#Vamos a definir un vector numérico que contiene la venta mensual de una tienda en el
#año #2020

venta_mensual <- c(4313,5313,3431,4515,3640,NA,3655,4124,5940,5678,4980,6758)

#Recorremos todos los elementos de un vector e imprimimos sus valores por pantalla

#Para ello creamos una variable llamada posicion que va a ir aumentando en cada
#iteración. #Cuando la posicion sea mayor que la longitud del vector el bucle debe acabar
#porque no hay más valores que mostrar

posicion <- 1

while( posicion <= length(venta_mensual) ){

  print(venta_mensual[posicion])

  posicion <- posicion + 1

}
```

Bucle `repeat`

En este caso, el bloque de código se repite de forma infinita hasta que la sentencia `break()` es ejecutada. Lo normal es que dicha sentencia venga encerrada por un `if`, por eso lo consideramos un bucle de tipo condicional.

```
#Vamos a definir un vector numérico que contiene la venta mensual de una tienda en el año #2020

venta_mensual <- c(4313,5313,3431,4515,3640,NA,3655,4124,5940,5678,4980,6758)

#Recorremos todos los elementos de un vector e imprimimos sus valores por pantalla

#Para ello creamos una variable llamada posicion que va a ir aumentando en cada iteración. #Cuando la posicion sea mayor que la longitud del vector el bucle debe lanzar el break() #porque no hay más valores que mostrar

posicion <- 1

repeat{

    print(venta_mensual[posicion])

    posicion <- posicion + 1

    if( posicion > length(venta_mensual) ){

        break()

    }

}
```

¡Enhорabuena! Fastbook superado

edix

Creamos Digital Workers