

Fastbook 02

Programación en R

Trabajando con variables



02. Trabajando con variables

En el fastbook anterior llevamos a cabo la instalación de R y RStudio, entendimos sus componentes más importantes y presentamos los distintos tipos de variables y operadores que existen. A continuación, profundizaremos en el uso y tratamiento de variables con el objetivo de adquirir fluidez y soltura en los procesos de transformación del dato.

El presente fastbook abordará dos áreas de desarrollo diferenciadas. La primera está compuesta por cinco secciones y muestra aquellos aspectos generales que nos permiten trabajar con variables estructurales: vectores, matrices, listas y dataframes. Tras ello, en el último apartado, y a través de un ejemplo práctico, nos centraremos en aquellas funciones y transformaciones que son específicas de cada tipo de dato: numérico, booleano, textual, temporal y especial.

Soy consciente de que hay muchísima información dentro de este fastbook. No os preocupéis por ello, lo importante es que os suenen los conceptos cuando os enfrentéis a la parte práctica de esta asignatura, que es lo realmente importante. De hecho, la idea es que este fastbook se convierta en un documento de consulta cuando queráis solventar ciertas dudas puntuales durante el estudio de esta materia y en vuestro futuro laboral.

¡Allá vamos!

Autor: Juan Jiménez García

Introducción a las variables estructurales

Trabajando con vectores

Trabajando con matrices

Trabajando con dataframes

Trabajando con listas

Trabajando con distintos tipos de datos

Introducción a las variables estructurales

X Edix Educación

Como ya sabemos, las variables son aquellos elementos de R en los que guardamos y transformamos nuestra materia prima, la información.

En concreto, aquellas que denominamos **estructurales** nos permiten **almacenaconjuntos de datos**.

En función de la tipología seleccionada, podremos usar una o más dimensiones, así como mezclar o no diferentes tipos de datos.

Aunque el anterior fastbook nos sirvió para definir las variables estructurales que existen, aún no hemos entrado en los detalles de su funcionamiento. En los siguientes apartados abordaremos este aspecto, y profundizaremos de forma práctica en el uso de este tipo de variables.

Como hemos visto, R contiene **4 tipos de variables estructurales** que presentan utilidad en el mundo de analytics. Son:

- 1 Vectores.
- 2 Matrices.
- 3 Dataframes.
- 4 Listas.

Para organizar las ideas, vamos a dividir el estudio de cada una de ellas en 5 apartados:

Definición —

Proceso por el cual somos capaces de crear y construir una variable.

Indexación —

Capacidad para obtener y modificar los diferentes datos y elementos que se encuentran dentro de la variable.

Nombrado —

Variación de los índices de la variable haciendo que podamos acceder a sus elementos con el uso de nombres o etiquetas y no solo de forma numérica.

Concatenación —

Procedimiento por el cual unimos o juntamos dos variables estructurales del mismo tipo.

Funciones de interés —

Presentación de las funciones R que resultan prácticas y útiles a la hora de trabajar con ese tipo de variables.

Trabajando con vectores

X Edix Educación

Los vectores son colecciones **unidimensionales** que almacenan un **mismo tipo de dato**.

Definición

Usamos la letra **c** seguida del paréntesis con los diferentes elementos separados por comas.

```
#Definimos un vector de numerics y lo asignamos a la variable vector_numerico
vector_numerico <- c(1, 2, 3, 4, 5)
#Definimos un vector de characters y lo asignamos a la variable vector_textual
vector_textual <- c("uno", "dos", "tres", "cuatro", "cinco")
```

Indexación

Usamos los **corchetes** indicando las posiciones que queremos leer o modificar.

```
#Leemos la primera posición del vector  
vector_numerico[1]  
  
#Leemos todas las posiciones del vector excepto la primera  
vector_numerico[-1]  
  
#Leemos las posiciones 2, 3 y 4 del vector  
vector_numerico[c(2,3,4)]  
  
#Leemos todas las posiciones del vector excepto la 2, 3 y 4  
vector_numerico[-c(2,3,4)]  
  
#Modificamos la primera posición del vector introduciendo el valor -2  
vector_numerico[1] <- -2  
  
#Modificamos las posiciones 2, 3 y 4 introduciendo los valores 7, 8 y 9  
vector_numerico[c(2,3,4)] <- c(7,8,9)
```

También podemos hacer uso de operadores comparativos para acceder o modificar aquellos elementos que cumplen ciertas características.

```
#Leemos los elementos del vector que son mayores que 2  
vector_numerico[vector_numerico>2]  
  
#Modificamos los elementos del vector que son mayores que 2 introduciendo los valores  
10, 11, 12 y 13  
vector_numerico[vector_numerico>2] <- c(10, 11, 12, 13)
```

Nombrado

Se realiza con la función *names*.

```
#Leemos los nombres del vector  
names(vector_numerico)  
  
#Modificamos los nombres del vector  
names(vector_numerico) <- c("uno", "dos", "tres", "cuatro", "cinco")  
  
#Accedemos a los elementos 4 y 5 a través de sus nombres  
vector_numerico[c("cuatro","cinco")]
```

Concatenación

La realizamos introduciendo en la función *c* las diferentes variables que deseamos unir, obteniendo un nuevo vector que contiene todos los elementos.

```
#Unimos el vector_numerico con un nuevo vector  
c(vector_numerico, c(20, 21, 22, 23, 24))
```

Si a la hora de unir dos o más vectores queremos obtener una matriz, tenemos que usar las funciones *rbind* y *cbind*.

```
#Cada vector de entrada va a ser una fila de la matriz salida  
rbind(vector_numerico, c(20, 21, 22, 23, 24))  
  
#Cada vector de entrada va a ser una columna de la matriz salida  
cbind(vector_numerico, c(20, 21, 22, 23, 24))
```

Funciones de interés

Vectores	
Función	Descripción
length()	Devuelve la longitud del vector, el número de elementos que contiene.
sort()	Devuelve el vector ordenado.
unique()	Devuelve el conjunto de elementos distintos que están contenidos en el vector.
table()	Devuelve el conjunto de elementos distintos que están contenidos en el vector con el número de apariciones de cada uno.
which()	Devuelve el índice del vector en el que se cumple la condición introducida.
rep()	Genera un vector que repite la variable introducida tantas veces como se le indica.

Trabajando con matrices

X Edix Educación

Las matrices son colecciones **bidimensionales** que almacenan un **mismo tipo de dato**.

Definición

Para su creación usamos la función ***matrix***, introduciendo el número de filas y columnas, así como el vector a partir del cual vamos a realizar la construcción.

```
#Definimos una matriz numeric de 2 filas y 3 columnas con los valores 1, 2, 3, 4, 5 y 6
matriz_numerica <- matrix(c(1, 2, 3, 4, 5, 6), 2, 3)

#Definimos una matriz character de 3 filas y 2 columnas con los valores "uno", "dos",
#"tres", "cuatro", "cinco" y "seis"

matriz_textual <- matrix(c("uno", "dos", "tres", "cuatro", "cinco", "seis"), 3, 2)
```

Indexación

Usamos los **corchetes**, pero indicando la posición de ambas dimensiones, **primero filas y después columnas**.

```
#Leemos el elemento situado en la fila 1 y columna 2
matriz_numerica[1,2]

#Leemos los elementos de la columna 2 que no están situados en la fila 1
matriz_numerica[-1,2]

#Leemos todos los elementos que se encuentran en las columnas 1 y 2 (independientemente de su fila)
matriz_numerica[,c(1,2)]

#Modificamos el elemento situado en la fila 1 y columna 2 introduciendo el valor -2
matriz_numerica[1,2] <- -2
```

Al igual que en los vectores, podemos hacer uso de los operadores comparativos.

```
#Introducimos el valor 100 en aquellas posiciones en la que el valor actual es superior a 4
matriz_numerica[matriz_numerica>4] <- 100
```

Nombrado

Usamos dos funciones, ***rownames*** y ***colnames***, una para cada dimensión.

```
#Leemos los nombres de las filas
rownames(matriz_numerica)

#Leemos los nombres de las columnas
colnames(matriz_numerica)

#Modificamos los nombres de las filas
rownames(matriz_numerica) <- c("fila_uno", "fila_dos")

#Modificamos los nombres de las columnas
colnames(matriz_numerica) <- c("columna_uno", "columna_dos", "columna_tres")

#Accedemos al valor de la fila 1 columna 2 a través de sus nombres
matriz_numerica["fila_uno", "columna_dos"]
```

Concatenación

Se realiza con las funciones ***rbind*** y ***cbind***.

- Si queremos que las matrices se coloquen una encima de la otra, usamos ***rbind***. En ese caso, el número de filas aumenta mientras que el de columnas se mantiene.
- Por el contrario, si queremos que las matrices se coloquen una al lado de la otra, usamos ***cbind***. En ese caso, el número de filas se mantiene, pero el de columnas aumenta.

```
#Unimos la matriz_numerica con una matriz 3x3 generando nuevas filas (para que funcione
deben tener el mismo número de columnas)
rbind(matriz_numerica,matrix(c(10, 11, 12, 13, 14, 15, 16, 17, 18),3,3))
```

$$\text{rbind} \left(\begin{array}{c} \text{Matriz numérica} \\ \left[\begin{array}{ccc} 1 & -2 & 100 \\ 2 & 4 & 100 \end{array} \right] \end{array} \right. , \left. \begin{array}{c} \left[\begin{array}{ccc} 10 & 13 & 16 \\ 11 & 14 & 17 \\ 12 & 15 & 18 \end{array} \right] \end{array} \right) = \begin{array}{c} \text{Resultado} \\ \left[\begin{array}{ccc} 1 & -2 & 100 \\ 2 & 4 & 100 \\ 10 & 13 & 16 \\ 11 & 14 & 17 \\ 12 & 15 & 18 \end{array} \right] \end{array}$$

```
#Unimos la matriz_numerica con una matriz 2x2 generando nuevas columnas (para que
funcione deben tener el mismo número de filas)

cbind(matriz_numerica,matrix(c(10, 11, 12, 13),2,2))
```

$$\text{cbind} \left(\begin{array}{c} \text{Matriz numérica} \\ \left[\begin{array}{ccc} 1 & -2 & 100 \\ 2 & 4 & 100 \end{array} \right] \end{array} \right. , \left. \begin{array}{c} \left[\begin{array}{cc} 10 & 12 \\ 11 & 13 \end{array} \right] \end{array} \right) = \begin{array}{c} \text{Resultado} \\ \left[\begin{array}{ccccc} 1 & -2 & 100 & 10 & 12 \\ 2 & 4 & 100 & 11 & 13 \end{array} \right] \end{array}$$

Funciones de interés

MATRICES	
Función	Descripción
length()	Devuelve el número de elementos que contiene la matriz.
nrow()	Devuelve el número de filas de la matriz.

MATRICES

ncol()	Devuelve el número de columnas de la matriz.
dim()	Devuelve el número de elementos que contiene la matriz en cada dimensión (número de filas y columnas).
t()	Transpone la matriz.
table()	Devuelve el conjunto de elementos distintos que están contenidos en la matriz con su número de apariciones.

Trabajando con dataframes

X Edix Educación

Los dataframes son colecciones **bidimensionales** que almacenan **datos de distinta naturaleza**.

Cada columna viene definida por un vector de un determinado tipo (character, numeric, date, etc.). El número de filas del dataframe será igual a la longitud de dichos vectores.

Definición

Usamos la función *data.frame*, asignando a cada nombre de columna su vector correspondiente.

```
#Generamos un dataframe de 2 columnas y 4 filas. La primera columna contiene información
#textual, mientras que la segunda contiene información numérica.
df <- data.frame(texto=c("catorce", "doce", "siete", "dos"), numero=c(14,12,7,2))
```

Indexación

Utilizamos los **corchetes** referenciando ambas dimensiones, primero filas y después columnas. Si queremos acceder a una columna tenemos la posibilidad de usar el símbolo \$.

```
#Leemos el valor que se encuentra en la fila 1 columna texto
df[1,"texto"]

#Leemos todas las filas y todas las columnas excepto la 1
df[, -1]

#Modificamos todas las filas de la columna 1 introduciendo el valor 10
df[,1] <- 10

#Leemos los valores de la columna numero
df$numero
```

Podemos usar los operadores comparativos con los corchetes o con la función *subset*. Además, para eliminar una columna podemos asignarle el valor *NULL*.

```
#Modificamos aquellos elementos de la columna numero que tienen un valor mayor a 10
introduciendo el valor 9
df[df$numero>10,"numero"] <- 9

#Leemos aquellas muestras (filas) cuyo valor en la columna numero es menor a 9
subset(df, numero<9)

#Eliminamos la columna texto
df$texto <- NULL
```

Nombrado

Se realiza con las funciones ***rownames*** y ***colnames*** que ya hemos presentado en el apartado de matrices.

Concatenación

Se lleva a cabo con las funciones ***rbind*** y ***cbind***, al igual que con las matrices.

- Si queremos que los dataframes se unan generando nuevas filas, usamos ***rbind*** (deben tener las mismas columnas).
- Por el contrario, si queremos que los dataframes se unan generando nuevas columnas, usamos ***cbind*** (deben tener las mismas filas).

Funciones de interés

DATAFRAMES	
Función	Descripción
nrow()	Devuelve el número de filas de la matriz.
ncol()	Devuelve el número de columnas de la matriz.
head()	Devuelve las primeras muestras (filas) del dataframe para que podamos visualizarlas.

DATAFRAMES

tail()	Devuelve las ultimas muestras (filas) del dataframe para que podamos visualizarlas.
str()	Devuelve un descriptivo del dataframe. Numero de muestras (filas) y variables (columnas), así como el tipo de dato.
order()	Realmente aplica sobre vectores, pero la usamos para ordenar dataframes en base a sus columnas. Recibe un vector y devuelve los índices que ordenan dicho vector.

Trabajando con listas

X Edix Educación

Las listas son colecciones **bidimensionales** que admiten **datos y variables de distinto tipo**.

Su gran utilidad se debe a que pueden almacenar cualquier tipo de elemento. Por ejemplo, podemos generar una lista que contenga un dataframe, una matriz, un vector, un character y una lista.

Definición

Se realiza con la función *list*, introduciendo en el paréntesis los distintos elementos a incluir.

```
#Creamos una lista que contiene el dataframe df, un vector numérico y una variable básica tipo character  
lista <- list(df, c(10,24.6,3,62), "RStudio")
```

Indexación

Seguimos trabajando con corchetes, aunque las listas tienen una peculiaridad que debemos comentar:

- Si queremos acceder a un elemento concreto de la lista debemos usar **doble corchete**.
- Sin embargo, si lo que queremos obtener sigue siendo una lista de elementos, debemos usar **corchete simple**.

```
#Obtenemos el elemento 2 (vector numérico)
lista[[2]]

#Obtenemos una lista con los elementos 1 y 2 (dataframe y vector numérico)
lista[c(1,2)]

#Obtenemos una lista que contiene únicamente el vector numérico
lista[2]
```

Podemos modificar y eliminar elementos.

```
#Modificamos el tercer elemento de la lista
lista[[3]] <- "R y RStudio"

#Eliminamos el tercer elemento de la lista
lista[[3]] <- NULL
```

Nombrado

Se realiza con la función *names*. Al igual que ocurre con las columnas de un dataframe, con el símbolo del dólar accedemos a los elementos de la lista.

```
#Asignamos nombres a los elementos de la lista
names(lista) <- c("dataframe","vector","character")

#Accedemos al elemento dataframe
lista[["dataframe"]]

#Accedemos al elemento dataframe
lista$dataframe
```

Concatenación

Usamos la función *c*, obteniendo una única lista con todos los elementos.

```
#Concatenamos la lista creada anteriormente con una nueva lista
c(lista, list(14, "texto", c(1,2,3)))
```

Funciones de interés

LISTAS	
Función	Descripción
length()	Devuelve la longitud de la lista: el número de elementos que contiene.
union()	Devuelve la unión lógica de dos listas. Es decir, todos aquellos elementos que están definidos en alguna de las dos listas.
intersect()	Devuelve la intersección lógica de dos listas. Es decir, todos aquellos elementos que están definidos en ambas listas.
unlist()	Devuelve un vector con todas las variables básicas que existen dentro de la lista.

Trabajando con distintos tipos de datos

X Edix Educación

Llegados a este punto del fastbook, ya hemos aprendido a manipular los cuatro tipos de **variables estructurales** que existen en R, por lo que antes de continuar con la última sección, vamos a reflexionar sobre la **utilidad** de cada una de ellas en la práctica de analytics.

Los **vectores** están orientados a la **transformación del dato**, por lo que su presencia en el código de un data scientist está más que garantizada. Debido a que todos sus elementos son de la misma clase, podemos aplicar funciones y modificaciones sobre cada uno de ellos de una forma sencilla y rápida.

El uso de **matrices** es menos frecuente debido a su naturaleza (2 dimensiones y un solo tipo de dato), aunque por supuesto debemos conocerlas por si en algún caso su uso nos resulta necesario.

Los **dataframes** son el estándar de trabajo de los analistas de datos, dado que en ellos podemos **almacenar información** de distinta clase. Es muy importante no olvidar que cada una de sus columnas es un vector, por lo que todas aquellas modificaciones que seamos capaces de aplicar sobre vectores de un determinado tipo, podremos realizarlas sobre las columnas de un dataframe.

En último lugar se encuentran las **listas**, cuya principal aplicación es el **almacenamiento**, ya que en ellas podemos guardar cualquier tipo de variable u objeto. Si necesitamos guardar varios dataframes, debemos usar una lista. Si tenemos que almacenar un conjunto de vectores con distintas longitudes o que no responden a la estructura tabular de un dataframe, la mejor opción es una lista.

En un futuro aprenderéis a desarrollar modelos y algoritmos matemáticos dentro de R. Estos se almacenan en objetos que guardan sus parámetros y coeficientes. Para guardar un conjunto de este tipo de elementos también usamos listas.

Tras este inciso, vamos a realizar un recorrido por los **distintos tipos de datos**, presentando las transformaciones y funciones de mayor interés para cada caso. Con la ayuda de un ejemplo desarrollado en vídeo, presentaremos las transformaciones y funciones que resultan de interés en cada caso. Como ya hemos comentado, los vectores son los elementos principales sobre los que realizamos dichas modificaciones. Por esta razón, el contenido que presentamos a continuación aplica mayoritariamente sobre ellos.

1

Dato numérico

Tal y como vimos en el fastbook 01, haciendo uso de los operadores podemos realizar operaciones sobre las variables básicas de tipo numérico. De igual forma, podemos hacerlo sobre vectores.

```
#Multiplicamos todos los elementos de un vector por 3 (el resultado es c(3, 6, 9, 12, 15, 18))
c(1, 2, 3, 4, 5, 6) * 3
```

Es importante entender que, cuando realizamos operaciones entre vectores, se realizan elemento por elemento.

```
#Sumamos dos vectores (el resultado es c(2, 4, 6, 8, 10, 12))
c(1, 2, 3, 4, 5, 6) + c(1, 2, 3, 4, 5, 6)
```

Además, existen algunas funciones que nos interesa aplicar cuando trabajamos con valores numéricos.

DATO NUMÉRICO	
Función	Descripción
sum()	Realiza la suma aritmética de todos los elementos.
max()	Devuelve el número de mayor valor.
min()	Devuelve el número de menor valor.
mean()	Devuelve la media aritmética de todos los elementos.
median()	Devuelve la mediana de todos los elementos.
summary()	Devuelve información completa sobre la distribución de los valores (valor mínimo y máximo, media, 1º, 2º, 3º y 4º cuartil).
round()	Redondea con el número de decimales que se le indique.
seq()	Genera una secuencia de valores en base al valor mínimo, máximo e intervalo que indiquemos.

2

Dato booleano

Como ya hemos observado, principalmente el dato booleano nos resulta útil cuando queremos comprobar si ciertas condiciones se están cumpliendo.

```
#Cuando realizo comparaciones sobre una variable básica obtengo como resultado una
variable básica booleana (en este caso FALSE)

14>16

#Si realizo comparaciones sobre un vector, obtengo un vector de booleanos (en este caso
c(FALSE, TRUE, FALSE))

c(14,18,9) > 16
```

Bajo esta perspectiva, hay cuatro funciones de R que nos interesa aplicar cuando trabajamos con vectores booleanos.

DATO BOOLEANO	
Función	Descripción
which()	Devuelve las posiciones numéricas en las que el valor es TRUE.
any()	Devuelve TRUE si hay al menos un TRUE en el vector.
all()	Devuelve un TRUE si todos los elementos del vector son TRUE.
sum()	Devuelve el número de elementos que son TRUE.

3

Dato textual

En primer lugar, debemos recordar que, aparte de la clase character, R tiene la clase factor, diseñada para trabajar con variables categóricas.

Por otro lado, nos interesa conocer las siguientes funciones R que trabajan con variables character, tanto en su formato básico como en su formato vectorial.

DATO TEXTUAL	
Función	Descripción
paste()	Concatena las variables que se le han introducido con un espacio entre ellas.
pasteo()	Concatena las variables que se le han introducido sin un espacio entre ellas.
tolower()	Transforma a minúsculas.
toupper()	Transforma a mayúsculas.
nchar()	Devuelve el número de caracteres.
grepl()	Comprueba si un determinado conjunto o patrón de caracteres se incluye dentro del texto.
gsub()	Además de comprobar si un determinado conjunto o patrón de caracteres se incluye dentro del texto, lo sustituye por el que le indiquemos.

DATO TEXTUAL

strsplit()	Rompe el texto en trozos en base al separador que le indiquemos.
------------	--

4

Dato temporal

El tratamiento de fechas y horas presenta una gran importancia en el análisis de datos debido a la temporalidad con la que cuentan la mayoría de los fenómenos que estudiamos.

En primer lugar, debemos tener claro que en R podemos almacenar fechas con tres principales clases: **character**, **date** y **POSIX**, cuyas diferencias ya conocéis.

Para movernos entre ellas usamos las funciones **as.character()**, **as.Date()** y **as.POSIXct()**, respectivamente.

Debido a que date y POSIX son clases 100% orientadas al tratamiento de información temporal, R ya tiene definido el formato de su almacenamiento y presentación. Sin embargo, cuando trabajamos con la clase character dicha información puede ser presentada de formas distintas. Es lo que llamamos formato de fecha y hora.

```
#Cambio la fecha de clase character a clase date indicando el formato de entrada
fecha_date <- as.Date("24-12-2020",format="%d-%m-%Y")

#Cambio la fecha de clase date a clase character indicando el formato de salida
fecha_character <- as.character(fecha_date, format="%Y-%m-%d")

#Cambio la fecha de clase character a clase POSIX indicando el formato de entrada
fecha_posix <- as.POSIXct("24-12-2020 22:45:12", format="%Y-%m-%d %H:%M:%S")
```

Símbolo	Descripción
%S	Segundo.
%M	Minuto.
%H	Hora.
%d	Día en forma numérica.
%a	Día de la semana abreviado.
%A	Día de la semana completo.
%m	Mes numérico.
%b	Nombre de mes abreviado.
%B	Nombre de mes completo.
%y	Año con dos dígitos.
%Y	Año con cuatro dígitos.

Una vez entendido este aspecto, pasemos a las funciones. Algunas de ellas se encuentran en R base, mientras que otras vienen incorporadas en el paquete *lubridate* que tendremos que instalar e importar.

DATO TEMPORAL	
Función	Descripción
seq()	Genera una secuencia de fechas en base a la fecha inicial, final y el intervalo que indiquemos.
difftime()	Calcula la diferencia temporal entre dos fechas.

DATO TEMPORAL (paquete <i>lubridate</i>)	
Función	Descripción
month()	Dada una fecha, devuelve su número de mes (de 1 a 12).
week()	Dada una fecha, devuelve su número de semana del año (de 1 a 52).
wday()	Dada una fecha, devuelve su número de día de la semana (de 1 a 7).
floor_date()	Redondea hacia abajo la fecha al nivel indicado (semanal, mensual, anual).
today()	Devuelve la fecha actual.
now()	Devuelve la fecha y hora actual.

5

Dato especial

En muchas ocasiones, cuando realizamos análisis nos encontramos con situaciones en las que necesitamos comprobar si un elemento es NA o NULL para que, en ese caso, podamos desarrollar las acciones pertinentes.

Para llevar a cabo dicho cometido, debemos conocer las funciones `is.na()` e `is.null()`.

```
#La comparativa directa entre NULLs no funciona
NULL == NULL

#Debemos usar la función is.null() que arroja TRUE o FALSE
is.null(NULL)

#Lo mismo ocurre con los NA. La comparativa no funciona.
NA == NA

#Tenemos que usar la función is.na() que arroja TRUE o FALSE
is.na(NA)

#También funciona de forma vectorial. En el siguiente ejemplo modificamos el valor
#perdido del vector introduciendo el valor 3.
vector <- c(1, 2, NA, 4, 5, 6)
vector[is.na(vector)] <- 3
```

Enhorabuena, has alcanzado el final del que sin duda es uno de los fastbooks más complejos de la asignatura. Hemos abordado una gran cantidad de contenido que necesita ser asimilado. Para ello, y como ya os he comentado en otras ocasiones, necesitamos practicar. Lo que tanta gente dice: a programar se aprende programando.

¡Enhorabuena! Fastbook superado

edix

Creamos Digital Workers