

Análisis de Datos

Tema 1 - Programación en R

Roi Naveiro

R y RStudio

¿Qué es R/RStudio?

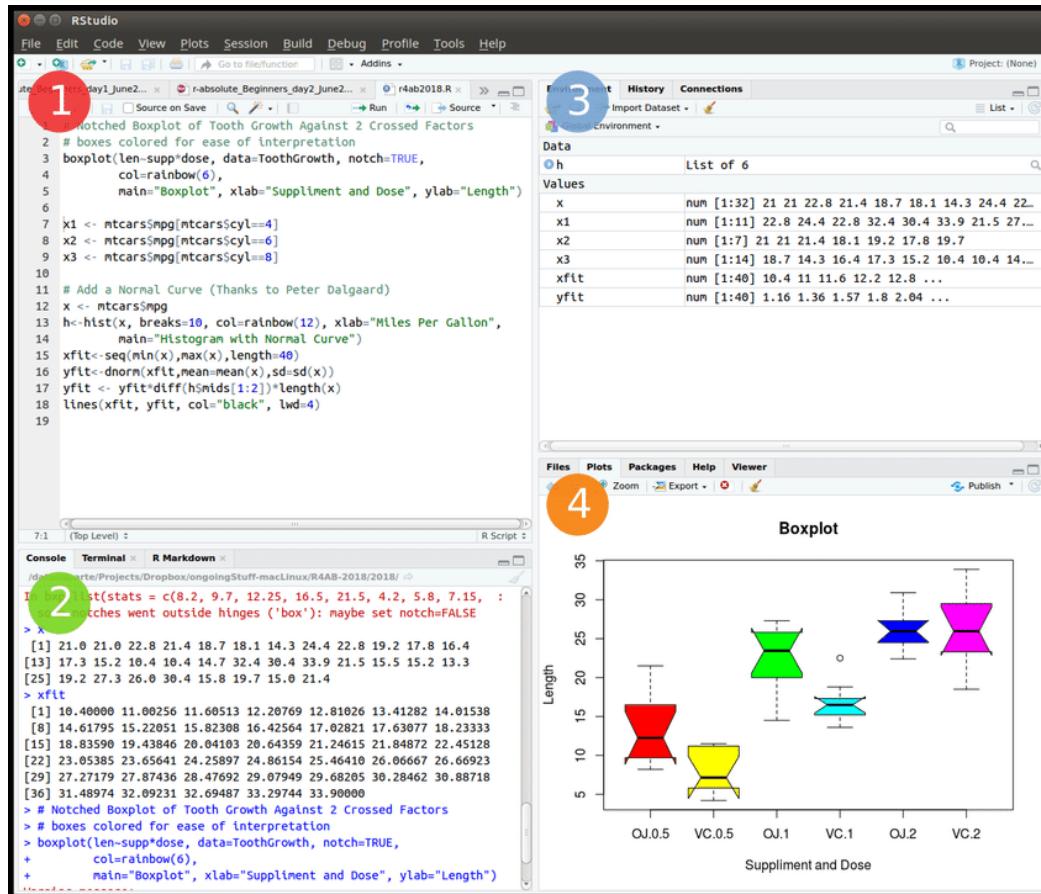
- R es un lenguaje de programación especializado en estadística
- RStudio es una interfaz para programar en R



Instalación de R y RStudio

Interfaz de usuario de R

- RStudio: forma de comunicarse con máquina
- R: lenguaje de comunicación



Introducción

Ejecutar comandos

```
123 * 65
```

```
## [1] 7995
```

En consola aparece como

```
> 123 * 65
[1] 7995
```

```
1000:1020
```

```
## [1] 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010
## [12] 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020
```

Uso como calculadora

```
(7 + 7) / 7 * 2
```

```
## [1] 4
```

Comentarios

- Comentarios con #

```
# Esto es un comentario
# Sumo los números del 1 al 100
sum(1:100)
```

```
## [1] 5050
```

- Salida de código con ##
- Comprueba que la siguiente expresión es cierta para $K = 100$ y $K = 1000$

$$\sum_{i=1}^K i = \frac{K \cdot (K + 1)}{2}$$

Objetos

- Si queremos un valor (o conjunto de valores) de forma recurrente hay que **guardarlos en memoria**, dándole un nombre.

```
# Esto crea un objeto de R con valor pi
a <- pi
```

```
# Sumo 3 al valor guardado
a + 3
```

```
## [1] 6.141593
```

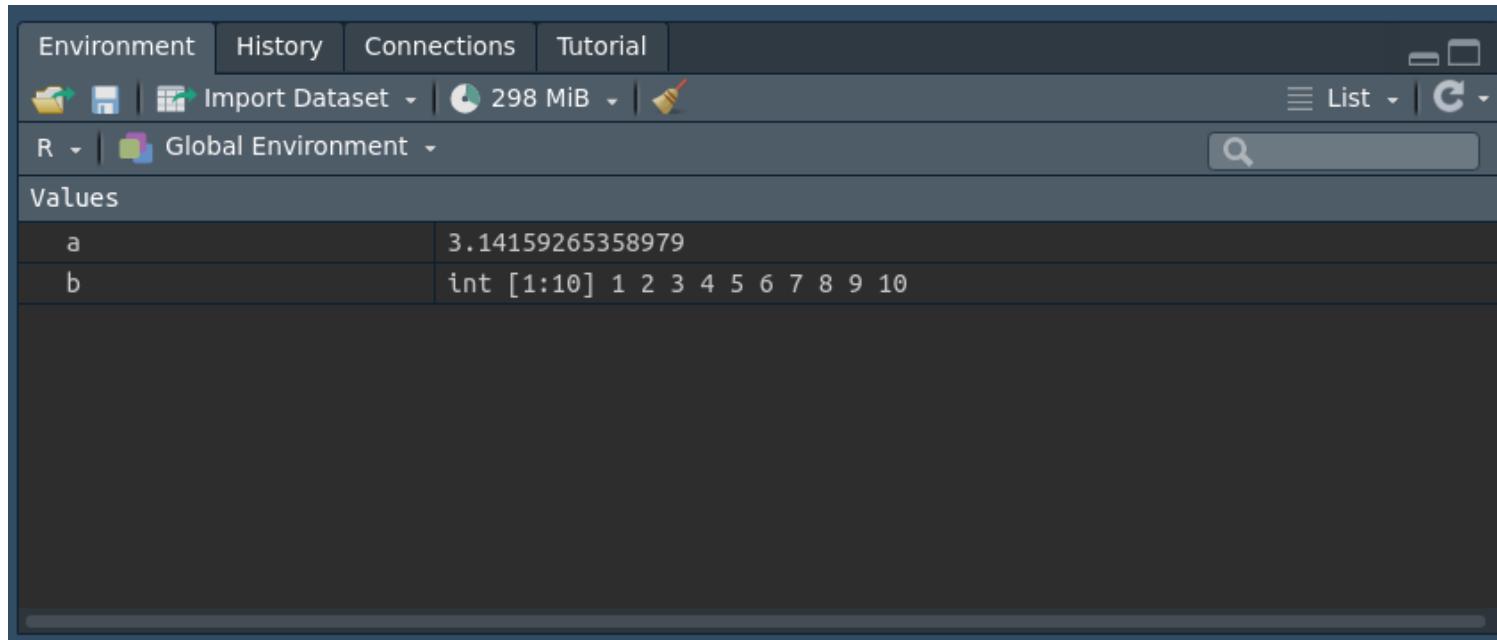
- Podemos guardar múltiples valores

```
b <- 1:10
b
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

Objetos

Los objetos guardados aparecen en la ventana de *Environment*



El nombre de un objeto no puede empezar con un número, ni tampoco incluir ciertos caracteres especiales como ^, \$, @, etc.

Objetos

R sobre-escribe

```
c <- 5  
c
```

```
## [1] 5
```

```
c <- 20  
c
```

```
## [1] 20
```

La función **ls()** muestra nombres usados

```
ls()
```

```
## [1] "a" "b" "c"
```

Operaciones sobre objetos

- R realiza operaciones *element-wise*
- Vector - Número

```
b <- 1:10  
b + 7
```

```
## [1] 8 9 10 11 12 13 14 15 16 17
```

- Vector - Vector (misma longitud)

```
b * b
```

```
## [1] 1 4 9 16 25 36 49 64 81 100
```

Operaciones sobre objetos

- Vector - Vector (diferente longitud)

```
# R repite el vector corto hasta alcanzar la longitud del largo
# Si la longitud del corto no es múltiplo de la del largo lanza un warning
d <- 1:2
e <- 1:4

d * e

## [1] 1 4 3 8
```

¡Pruébalo!

La función c()

Esta función nos ayuda a combinar objetos. Por ejemplo, podemos crear un vector de enteros de la siguiente forma

```
a <- c(1,2,3)  
a
```

```
## [1] 1 2 3
```

También podemos añadir elementos a un vector existente

```
a <- c(a, 4,5)  
a
```

```
## [1] 1 2 3 4 5
```

No solo podemos crear vectores de números!

```
a <- c('adiós', 'hola')  
a
```

```
## [1] "adiós" "hola"
```

Funciones

Funciones

R tiene muchas funciones predefinidas

```
# Calcula la media de los números del 1 al 10  
mean(1:10)
```

```
## [1] 5.5
```

```
# El argumento de una función puede ser el resultado de otra función  
round(mean(1:10))
```

```
## [1] 6
```

Funciones

Algunas funciones tienen múltiples argumentos. Cada uno tiene un nombre

```
# Muestra un número del 1 al 10
sample(x = 1:10, size = 1)

## [1] 2
```

```
# Nombre opcional
sample(1:10, 1)
```

```
## [1] 5
```

Funciones

La función **args()** muestra los argumentos de una función

```
args(sample)
```

```
## function (x, size, replace = FALSE, prob = NULL)
## NULL
```

¡Algunas tienen valores por defecto! Pregunta, ¿qué valores tiene por defecto la función **round()**?

Buena práctica: escribir nombre de argumentos (salvo quizás el primero).

Definiendo funciones propias

- Permiten definir operaciones de uso recurrente (no implementadas en R).

```
una_funcion <- function(){}
```

- Función que genera 10 números entre 1 y 100 y devuelva su suma

```
gen2_1_100 <- function(){
  nums <- sample(1:100, size = 2, replace = TRUE)
  print(nums)
  return(sum(nums))
}
```

Definiendo funciones propias

- La probamos

```
a <- gen2_1_100()
```

```
## [1] 49 88
```

```
a
```

```
## [1] 137
```

Definiendo funciones propias

- Argumentos

```
# Pasamos el tamaño como argumento
# 2 es el valor por defecto
gen2_1_100 <- function(n=2){
  nums <- sample(1:100, size = n, replace = TRUE)
  print(nums)
  return(sum(nums))
}
```

- La probamos sin argumento

```
gen2_1_100()
```

```
## [1] 25 15
## [1] 40
```

Definiendo funciones propias

- La probamos con argumento

```
gen2_1_100(10)
```

```
## [1] 30 65 53 43 64 89 54 58 88 61  
## [1] 605
```

Ejercicio 1

Escribe una función que devuelva k números, donde cada número es la suma de n números escogidos al azar entre 1 y 100 (con reemplazamiento), dividido entre n .

Pinta un histograma con los resultados de $k = 10000$ para $n = 1$, $n = 2$, $n = 10$ y $n = 100$. Pista: puedes usar la función **hist()**.

¿Qué observas?

Paquetes y ayudas

Paquetes y ayudas

- **Importante:** el primer paso en la creación (de cualquier cosa) consiste en comprobar si alguien ya la ha creado...
- ...lo mismo para funciones en R.
- Muchas funciones de R han sido creadas y encapsuladas en **paquetes**.
- Aquí podéis ver cómo instalar y actualizar paquetes en R.

Paquetes y ayudas

- Ejemplo, instalar **tidyverse**

```
install.packages("tidyverse")
```

- Para usar paquetes en R hay que cargarlos

```
library("tidyverse")
```

Paquetes y ayudas

- Existen infinidad de funciones en R y sus paquetes...
- ...memorizar todo es complicado (y absurdo).
- Las funciones cuentan con ayudas, que se invocan de la siguiente manera

```
?sample
```

Paquetes y ayudas

- La mejor fuente de ayuda es internet
- Por ejemplo, [Stack Overflow](#)
- Otro recurso interesante es [community.rstudio.com](#)

Ejercicio 2

Utiliza internet para conocer el funcionamiento de la función **replicate()** Resuelve el Ejercicio 1 sin usar un bucle (si es que lo habías usado).

Objetos de R

Objetos de R

Un objeto de R es una estructura de datos que tiene algunos métodos y atributos asociados.

Estudiaremos los siguientes objetos:

1. Vectores atómicos
2. Matrices
3. Arrays
4. Fecha y hora
5. Factores
6. Listas
7. Data frames

1. Vectores atómicos

No son más que vectores de datos. Ya hemos visto cómo crearlos.

```
a = c(1,2,3)  
# Comprueba si es vector  
is.vector(a)
```

```
## [1] TRUE
```

```
# Devuelve la longitud del vector  
length(a)
```

```
## [1] 3
```

Cada vector sólo guarda un tipo de dato

1. Vectores atómicos

Tipos de datos - Doubles

Los vectores tipo double almacenan números reales

```
a = c(1.7, 2, 3.5)  
# Devuelve el tipo de vector  
typeof(a)  
  
## [1] "double"
```

1. Vectores atómicos

Tipos de datos - Enteros

Los vectores tipo entero almacenan números enteros

```
a = c(1L,-2L,3L)  
# Devuelve el tipo de vector  
typeof(a)  
  
## [1] "integer"
```

1. Vectores atómicos

Tipos de datos - Caracteres

Los vectores tipo carácter (string) almacenan texto

```
a = c("Hola", "Adiós")  
  
# Devuelve el tipo de vector  
typeof(a)  
  
## [1] "character"
```

OJO: ¿Qué está sucediendo?

```
a = c(1,2,3,"Hola")  
a  
  
## [1] "1"     "2"     "3"     "Hola"
```

Este fenómeno se llama **coerción**. Es importante que leas sobre el mismo en <https://rstudio-education.github.io/hopr/r-objects.html#coercion>.

1. Vectores atómicos

Tipos de datos - Lógicos

Los vectores tipo lógico almacenan **TRUE** o **FALSE** (datos Booleanos).

```
a = c(5>4, 4>5)
typeof(a)
```

```
## [1] "logical"
```

```
a
```

```
## [1] TRUE FALSE
```

Inciso: Atributos

Son piezas de información que se pueden adjuntar a los objetos de R.

```
a = 1:10  
attributes(a)
```

```
## NULL
```

Algunos atributos importantes: nombres y dimensión

Atributos - Nombres

```
a = 1:3
# Asigno nombres al vector a
names(a) <- c("uno", "dos", "tres")
names(a)
```

```
## [1] "uno"  "dos"  "tres"
```

```
#Compruebo que se ha guardado el atributo
attributes(a)
```

```
## $names
## [1] "uno"  "dos"  "tres"
```

Atributos - Dimensión

```
a = 1:10
# Asigno dimensión, esto convierte a en matriz
dim(a) = c(2,5)

#Compruebo la dimensión
dim(a)

## [1] 2 5
```

2. Matrices

Se utilizan para almacenar matrices bi-dimensionales

```
m <- matrix(1:10, nrow=2)  
m
```

```
##      [,1] [,2] [,3] [,4] [,5]  
## [1,]     1     3     5     7     9  
## [2,]     2     4     6     8    10
```

```
m <- matrix(1:10, nrow=2, byrow=TRUE)  
m
```

```
##      [,1] [,2] [,3] [,4] [,5]  
## [1,]     1     2     3     4     5  
## [2,]     6     7     8     9    10
```

3. Arrays

Se utilizan para almacenar matrices n-dimensionales

```
m <- array(1:12, dim = c(2,2,3))  
m
```

```
## , , 1  
##  
##      [,1] [,2]  
## [1,]    1    3  
## [2,]    2    4  
##  
## , , 2  
##  
##      [,1] [,2]  
## [1,]    5    7  
## [2,]    6    8  
##  
## , , 3  
##  
##      [,1] [,2]  
## [1,]    9   11  
## [2,]   10   12
```

Inciso: Clases

Una clase es simplemente la huella de un objeto de R. Representa el conjunto de propiedades o métodos que son comunes a todos los objetos de un tipo. La función 'class' describe el tipo de clase correspondiente al objeto.

```
m <- array(1:12, dim = c(2,2,3))
class(m)

## [1] "array"
```

4. Fechas y Horas

```
hora <- Sys.time()  
hora
```

```
## [1] "2022-09-02 11:35:05 CEST"
```

```
typeof(hora)
```

```
## [1] "double"
```

```
class(hora)
```

```
## [1] "POSIXct" "POSIXt"
```

5. Factores

Se utilizan para almacenar variables categóricas, con un nivel de categorías fijo.

```
fumador <- factor( c("SI", "NO", "NO", "SI") )  
attributes(fumador)
```

```
## $levels  
## [1] "NO" "SI"  
##  
## $class  
## [1] "factor"
```

```
# Convertimos factor a caracter  
fumador <- as.character(fumador)  
class(fumador)
```

```
## [1] "character"
```

6. Listas

Hemos visto que los vectores atómicos agrupan valores individuales. Las listas almacenan objetos de R.

```
milista <- list(c("A", "B"), 1:10, Sys.time())
milista
```

```
## [[1]]
## [1] "A" "B"
##
## [[2]]
## [1] 1 2 3 4 5 6 7 8 9 10
##
## [[3]]
## [1] "2022-09-02 11:35:05 CEST"
```

7. Data Frames

Versión bi-dimensional de una lista. Cada columna del data frame puede contener tipos de datos diferentes. Un dataframe se crea especificando las columnas (que son vectores atómicos)

```
df <- data.frame(paciente = c("MARIANO", "IRENE", "INES"), fumador = c("SI",  
df
```

```
## paciente fumador edad  
## 1 MARIANO      SI    64  
## 2 IRENE        NO    32  
## 3 INES         SI    38
```

7. Data Frames

```
# Comprobamos la clase
class(df)

## [1] "data.frame"

# Resumimos la información
str(df)

## 'data.frame':   3 obs. of  3 variables:
## $ paciente: Factor w/ 3 levels "INES","IRENE",...: 3 2 1
## $ fumador : Factor w/ 2 levels "NO","SI": 2 1 2
## $ edad    : num  64 32 38
```

Podemos acceder al dataframe desde el *Environment*

Selección y modificación de valores

Seleccionar valores

En un dataframe

```
# Selección con números enteros
df[1,2]
```

```
## [1] SI
## Levels: NO SI
```

```
# Más de un elemento
df[1:2, 2:3]
```

```
##     fumador edad
## 1      SI    64
## 2      NO    32
```

Seleccionar valores

En un dataframe

```
# Selección con enteros negativos
df[-c(1,2),1:3]
```

```
## paciente fumador edad
## 3      INES      SI   38
```

```
# Selección con espacios
df[,-2]
```

```
## paciente edad
## 1  MARIANO  64
## 2  IRENE    32
## 3  INES     38
```

```
# Selección con variables lógicas
df[1,c(TRUE, TRUE, FALSE)]
```

```
## paciente fumador
## 1  MARIANO      SI
```

Seleccionar valores

En un dataframe

```
# Selección con nombres  
df[c(1,3), "fumador"]
```

```
## [1] SI SI  
## Levels: NO SI
```

```
# Selección de columnas con $  
df$paciente
```

```
## [1] MARIANO IRENE INES  
## Levels: INES IRENE MARIANO
```

Seleccionar valores

¡Con expresiones lógicas!

```
# Seleccionar los fumadores
df[df$fumador == "SI", ]
```

```
## paciente fumador edad
## 1 MARIANO SI 64
## 3 INES SI 38
```

```
# Seleccionar los menores de 33
df[df$edad < 33, ]
```

```
## paciente fumador edad
## 2 IRENE NO 32
```

Seleccionar valores

Se pueden combinar expresiones lógicas con operadores Booleanos

```
exp1 <- 5>3
exp2 <- "A" != "A"
exp3 <- 3 %in% c(1,2,3)

# AND
exp1 & exp2
```

```
## [1] FALSE
```

```
# OR
exp1 | exp2
```

```
## [1] TRUE
```

```
# Negación
!exp1
```

```
## [1] FALSE
```

Seleccionar valores

Se pueden combinar expresiones lógicas con operadores Booleanos

```
# Any  
any(exp1, exp2, exp3)
```

```
## [1] TRUE
```

```
# All  
all(exp1, exp2, exp3)
```

```
## [1] FALSE
```

Seleccionar valores

Se pueden combinar expresiones lógicas con operadores Booleanos

```
# Seleccionar los fumadores mayores de 60
df[df$fumador == "SI" & df$edad < 60,]

## paciente fumador edad
## 3      INES      SI    38
```

Seleccionar valores

En una lista

```
milista <- list(letras = c("A", "B"), numeros = 1:10, tiempo = Sys.time())
milista[1]
```

```
## $letras
## [1] "A" "B"
```

```
milista[[1]]
```

```
## [1] "A" "B"
```

```
milista["letras"]
```

```
## $letras
## [1] "A" "B"
```

```
milista[["letras"]]
```

```
## [1] "A" "B"
```

Ejercicio 3

La base de datos **mtcars** de R contiene información extraída de la *1974 Motor Trend US magazine* acerca de 10 aspectos de diseño de rendimiento de 32 vehículos. Usando esta base de datos, calcula

Pregunta 1. La diferencia entre el consumo medio de los vehículos de transmisión automática y los de transmisión manual.

Pregunta 2. La misma diferencia pero únicamente para vehículos de más de 3000 lbs.

Pista: la ayuda ?mtcars te puede ser útil

Modificar valores

```
vec <- 1:10

# Modificar valores con enteros
vec[3] <- 300
vec
```

```
## [1] 1 2 300 4 5 6 7 8 9 10
```

```
# Modificar valores con selección
vec[c(1,4)] <- c(100, 400)
vec
```

```
## [1] 100 2 300 400 5 6 7 8 9 10
```

```
# Modificar valores con selección
vec[c(5:6)] <- vec[c(5:6)] + 100
vec
```

```
## [1] 100 2 300 400 105 106 7 8 9 10
```

Modificar valores

```
# Añadimos variable a dataframe
df$hipertenso <- c("SI", "NO", "NO")
df
```

```
## paciente fumador edad hipertenso
## 1 MARIANO      SI    64      SI
## 2 IRENE        NO    32      NO
## 3 INES         SI    38      NO
```

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

# Modificamos un valor específico
df[1, "edad"] <- 57
df
```

```
## paciente fumador edad
## 1 MARIANO      SI    57
## 2 IRENE        NO    32
## 3 INES         SI    38
```

Bucles y condicionales

Condicionales y bucles

Las condicionales y los bucles son las dos estructuras de programación más importantes cualquiera sea el lenguajes que utilicemos.

1. Las condicionales permiten ejecutar cierto código en función de si se cumple una condición lógica.
2. Los bucles permiten repetir cierto código un número específico de veces.

1. Condicionales

Mandan a R realizar una tarea si se cumple una condición

```
if (se cumple esto) {  
    haz esto  
}
```

Los **if** reciben una expresión lógica (**TRUE** o **FALSE**).

Pueden anidarse

```
x <- 7  
if(x <= 8){  
    if(x>=5){  
        print("Número en [5,8]")  
    }  
}
```

```
## [1] "Número en [5,8]"
```

Haz lo anterior con un único **if**.

1. Condicionales

if se complementa con **else**

```
if (se cumple esto) {  
    haz esto  
} else{  
    Sino haz esto otro  
}
```

Si hay más de dos condiciones, se puede usar **else if**

```
if (se cumple esto) {  
    haz esto  
} else if(si se cumple esto){  
    haz esto otro  
} else{  
    sino haz esto  
}
```

1. Condicionales

Ejemplo

```
x <- 7
if (x>7) {
  print(x + 3)
} else if(x<7){
  print(x-3)
} else{
  print(x)
}
```

2. Bucles

Permiten repetir operaciones un cierto número de veces. Estudiaremos los bucles **for** y **while**.

2. Bucles - For

Repite un trozo de código muchas veces, una por cada elemento en un conjunto de entrada

```
for(valor in conjunto){  
    haz esto  
}
```

2. Bucles - For

Ejemplo

```
for(valor in c("A", "B", "C")){
  print("Iterando")
}
```

```
## [1] "Iterando"
## [1] "Iterando"
## [1] "Iterando"
```

```
for(valor in c("A", "B", "C")){
  print(valor)
}
```

```
## [1] "A"
## [1] "B"
## [1] "C"
```

2. Bucles - While

Ejecuta un trozo de código mientras se cumple la condición de entrada

```
while(condicion){  
    haz esto  
}
```

Ejemplo

```
x <- 1  
while(x < 8){  
    print(x)  
    x <- x + 1  
}
```

```
## [1] 1  
## [1] 2  
## [1] 3  
## [1] 4  
## [1] 5  
## [1] 6  
## [1] 7
```

Ejercicio 4

Utilizando un bucle **for** y un **if**, suma los números pares entre 1 y 100 (ambos incluidos).

Bibliografía

Este tema está fundamentalmente basado en [Hands-On Programming with R](#), Grolemund (2014)