Intro

David García Callejas

¿Cómo funciona R?

Ventanas en Rstudio: consola, script, entorno, figuras/ayuda. Puedes escribir comandos de R directamente en la consola, y el resultado aparecerá también ahí.

```
3+3
```

```
sqrt(3)
```

Cuando tienes varios comandos en R, los puedes guardar en un *script*. En Rstudio, "File"->"New File"->"R Script".

Los comandos que se guardan en un script se pueden ejecutar de varias maneras:

- botón "Run", equivalente a ctrl+Intro, ejecuta la línea donde está el cursor
- botón "Source", ejecuta el script completo

En los scripts de R, se pueden incluir *comentarios*.

```
# líneas que empiezan con una almohadilla son consideradas como comentarios.
# los comentarios son tan importantes como el código de un script.
# Comentad todo lo que hagáis, siempre!

# el logaritmo de un número se calcula con log()
log(5)
```

```
# EJERCICIO
# Crea un script para almacenar los comandos que usaremos esta tarde.
```

Variables

En general, los resultados de nuestras operaciones nos interesa guardarlos, no sólo que aparezcan escritos en la consola. En R, y en cualquier lenguaje de programación, estos resultados se asignan a *variables*.

```
# la asignación se hace con "<-"
x <- 3+3
```

Podemos ver el valor de nuestra variable "llamándola" en la consola o el script:

```
x
```

```
# "=" también funciona, pero no es recomendable
x = 3+3

# ¿por qué <- y no =?
# probad esto:
# mean(y = 1:10)
```

```
#vs.
# mean(y <- 1:10)
```

Tras asignar un valor a una variable, podemos utilizarla como cualquier otro valor

```
y <- x + 5
```

Fijaos que, al asignar una variable, ésta aparecerá en la ventana "environment" de Rstudio.

Tipos de operadores

En R, tenemos a nuestra disposición todos los operadores aritméticos básicos

```
# suma, resta, multiplicación, división
x+y
x-y
x*y
x/y
```

así como un mundo de funciones matemáticas de serie

```
# exponencial
x^y

# logaritmo, del cual podemos especificar la base
log(x, base = 10)

# raiz cuadrada
sqrt(x)

# etc. Podemos combinar operaciones siguiendo las reglas
# matemáticas de precedencia
x + y*3
log(x*y)^2
```

También podemos aplicar operadores relacionales, que comparan una variable con otra

```
# ¿es x igual a y?
x == y

# ¿es x diferente de y?
x != y

# ¿es x menor que y?
x < y

# ¿es x mayor o igual que y?
x >= y
```

Estos operadores devuelven el valor "TRUE" o "FALSE" si se cumple, o no, la operación. Por último, existen operadores lógicos, que nos sirven para unir condiciones. Varios ejemplos:

```
# si queremos saber si una variable es mayor que 1 pero,
# a la vez, menor que 10...
# usamos el operador lógico AND
(x > 1) & (x < 10)
```

```r
# una variable menor de 1 o mayor de 10
# operador OR
(x < 1) | (x > 10)

# un signo de exclamación al principio de una condición
# se interpreta como la negación de la misma
# por ejemplo, ¿es x menor que 1?
(x < 1)
# y en este caso, literalmente, ¿es x NO menor que 1?
!(x < 1)
```

Tipos de datos básicos

Hasta ahora, hemos visto cómo R puede trabajar como si fuera una calculadora en esteroides. Pero hay muchas más posibilidades, empezando por los tipos de datos que podemos tratar en nuestros análisis.

Los tipos básicos de datos que puede contener una variable son cinco: número entero, número real, número complejo, carácter, y valor booleano (o lógico).

```r
# nuestra variable es un número entero,
# que es una variante del modo "numeric"
str(x)

# los números reales son otra variante del modo "numeric"
str(3.5)

# para crear un número complejo, tenemos que especificar
# su parte real y su parte imaginaria
z <- complex(real = 1, imaginary = 1)
str(z)

# y también podemos asignar variables a caracteres
a <- "mi variable"
str(a)

# los datos booleanos ya los hemos visto más arriba:
# son tipos especiales que pueden tomar valores "TRUE" o "FALSE"
# EJERCICIO: crea una condición usando la variable "a"
# y asígnala a una nueva variable
```

Vectores

Hasta ahora, hemos trabajado con variables que tienen un único elemento. Pero una variable también puede almacenar varios elementos de un mismo tipo, lo que llamamos un "vector".

```r
#4 formas de escribir el mismo vector
myvector <- c(1, 2, 3, 4, 5)
myvector <- c(1:5)
myvector <- 1:5
myvector #teclea my y tab para autocompletar
```

Igual que los tipos de datos básicos, un vector puede ser numérico, de caracteres, etc.

```
charvector <- c("caracter1","caracter2","caracter3")
```

R permite automatizar la creación de vectores con reglas básicas

```
# crear un vector con cien repeticiones del número 1
vec1 <- rep(1,100) # literalmente: REPite 1, 100 veces

# crear una secuencia 10, 20, 30,... hasta 100
vec2 <- seq(from = 10, to = 100, by = 10)
# literalmente: desde 10, hasta 100, de 10 en 10
```

Podemos operar con un vector exactamente igual que si fuera un único elemento, y las operaciones se aplicarán a todos los elementos del vector

```
vec2 + 1
vec2 * vec2
log(vec2)

# podemos sumar todos los elementos de un vector
sum(vec2)

# también podemos crear nuevos vectores a partir de otros
vec3 <- c(vec2, vec2)
```

Es sencillo saber cuántos elementos tiene un vector (su "longitud", en inglés "length")

```
length(vec3)
```

Una particularidad importante es que R recicla valores. Si llega al final de un vector y necesita más valores para terminar una operación, vuelve al principio.

```
# los dos vectores son del mismo tamaño,
# así que la multiplicación es elemento a elemento
1:5 * 1:5
# ¿qué pasa si multiplicamos vectores de tamaños diferentes?
1:5 * 1:6
```

¿Cómo accedemos a los elementos concretos de un vector?

```
vec2

# quiero recuperar el quinto elemento de este vector
vec2[5]

# igual que lo puedo recuperar, puedo darle otro valor
vec2[5] <- 55
vec2

# entre los corchetes también pueden ir varios elementos,
# por ejemplo si quiero poner el primer y el tercer elemento a cero
vec2[c(1,3)] <- 1
# ¿y por qué no esto?
# vec2[1,3] <- 1
# c(1,3) es un vector, mientras que 1,3 no lo es.
# Probad a escribir ambos en la consola.


# si las posiciones que nos interesan están en otro vector,
```

```r
# también se puede utilizar
posiciones <- c(1,3)
vec2[posiciones]

# podemos "preguntar" qué elementos de un vector
# cumplen una cierta condición, usando la función "which"
# esta función nos devuelve las posiciones que cumplen la condición
which(vec2 < 30)
```

Funciones

Hemos visto que hay dos tipos de operaciones que podemos realizar con variables. Por un lado están las operaciones básicas aritméticas, como suma o resta. Pero hay otro tipo, como el logaritmo, que tienen una sintaxis muy diferente:

```r
# compara cómo se escribe una suma estándar
x + 1

# con cómo se escribe un logaritmo
log(x)
```

El segundo tipo de operaciones se puede leer así, en el caso del logaritmo: Aplica la función "logaritmo" al elemento que hay entre paréntesis. De hecho, el logaritmo no es ni más ni menos que eso, una *función*. De manera general, podemos definir una función como una serie de instrucciones que ejecutan una tarea específica, a partir de una serie de argumentos. Las funciones son uno de los pilares más importantes de R, y de cualquier lenguaje de programación. Muchos de los cálculos y el manejo de datos que haremos se organizan a partir de funciones.

Para familiarizarnos con ellas, repasemos algunas funciones básicas que ya hemos visto

```r
# raiz cuadrada, etc
sqrt(x)

# función para generar secuencias
seq(from = 1, to = 10, by = 0.1)

# suma todos los elementos de un vector numérico
sum(1:10)
```

Algunas funciones aceptan un solo argumento, otras más de uno, y algunas, incluso ninguno. Al aplicar una función, ésta devuelve un valor del tipo correspondiente.

```r
# función para calcular la media de un vector
media <- mean(vec2)

# o su desviación típica
desv <- sd(vec2)
```

Ocurre a menudo que funciones más avanzadas tienen varios argumentos, como el caso de "seq". Cuando no hemos utilizado nunca una función, podemos usar la ayuda de R para tener una idea de su uso. Esto se consigue escribiendo un signo de interrogación delante del nombre de la función, y nos abrirá una página en la ventana de ayuda de Rstudio.

```r
?seq
```

En la ayuda podemos leer una descripción de la función, su sintaxis, los argumentos que acepta, el tipo de

valor que devuelve, y en muchas ocasiones, ejemplos prácticos de su uso.

Ejercicios

1

a) crea un vector numérico que vaya de 1 a 100 en intervalos de 0.1.
b) usa la función "which" para saber qué elementos del vector que has creado son mayores de 90, y almacena ese resultado.
c) crea un nuevo vector que almacene los valores del primer vector que son mayores de 90.

2

a) la función "runif" genera vectores de números aleatorios entre un número mínimo y un máximo. Comprueba la ayuda de R para esta función (verás que hay otras funciones en su misma página de ayuda, mira las secciones "usage" para ver su sintaxis, y "arguments" para ver sus argumentos).
b) usa la función runif para almacenar un vector con 50 elementos aleatorios entre 0 y 10.
c) obtén la media y la desviación típica de esos elementos.
d) crea un nuevo vector que sólo contenga los elementos del primero que sean menores o iguales a 5 (usa la función "which" como en el primer ejercicio).