

R: lo básico

El espacio de trabajo (Workspace)

Directorio de trabajo

El directorio de trabajo o *working directory* es el folder en tu computadora en el que estás trabajando en ese momento. Cuando se le pide a R que abra un archivo o guarde ciertos datos, R lo hará a partir del directorio de trabajo que le hayas fijado.

Para saber en qué directorio te encuentras, se usa el comando `getwd()`.

```
getwd()
```

```
## [1] "/home/animalito/study/aprendeR/lecture_01"
```

Para especificar el directorio de trabajo, se utiliza el comando `setwd()` en la consola. Y volvemos a

```
setwd("/home/animalito/study/")  
getwd()
```

Con lo que acabamos de hacer, R buscará archivos o guardará archivos en el folder `/home/animalito/study/`. En R también es posible navegar a partir de el directorio de trabajo. Como siempre,

- “../un_archivo.R” le indica a R que busque un folder arriba del actual directorio de trabajo por el archivo *un_archivo.R*.
- “datos/otro_archivo.R” hace que se busque en el directorio de trabajo, dentro del folder *datos* por el archivo *otro_archivo.R*

Ejemplos básicos

La consola permite hacer operaciones sobre números o caracteres (cuando tiene sentido).

```
# Potencias, sumas, multiplicaciones  
2^3 + 67 * 4 - (45 + 5)
```

```
## [1] 226
```

```
# Comparaciones  
56 > 78
```

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

```
34 <= 34
```

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

```
234 < 345
```

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

```
"hola" == "hola"
```

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

```
# modulo  
10 %% 4
```

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

Estas operaciones también pueden ser realizadas entre vectores

```
x <- -1:12  
x
```

```
## [1] -1 0 1 2 3 4 5 6 7 8 9 10 11 12
```

```
x + 1
```

```
## [1] 0 1 2 3 4 5 6 7 8 9 10 11 12 13
```

```
2 * x + 3
```

```
## [1] 1 3 5 7 9 11 13 15 17 19 21 23 25 27
```

```
x %% 5 ##-- is periodic
```

```
## [1] 4 0 1 2 3 4 0 1 2 3 4 0 1 2
```

```
x %% 5
```

```
## [1] -1 0 0 0 0 0 1 1 1 1 1 2 2 2
```

Comandos útiles

Para enlistar los objetos que están en el espacio de trabajo

```
ls()
```

```
## [1] "x"
```

Para eliminar todos los objetos en un workspace

```
rm(list = ls()) # se puede borrar solo uno, por ejemplo, nombrándolo
ls()
```

```
## character(0)
```

También se puede utilizar/guardar la historia de comandos utilizados

```
history()
history(max.show = 5)
history(max.show = Inf) # Muestra toda la historia

# Se puede salvar la historia de comandos a un archivo
savehistory(file = "mihistoria") # Por default, R ya hace esto
# en un archivo ".Rhistory"

# Cargar al current workspace una historia de comandos en particular
loadhistory(file = "mihistoria")
```

Es posible también guardar el workspace -en forma completa- en un archivo con el comando `save.image()` a un archivo con extensión `.RData`. Puedes guardar una lista de objetos específica a un archivo `.RData`. Por ejemplo:

```
x <- 1:12
y <- 3:45
save(x, y, file = "ejemplo.RData") #la extensión puede ser arbitraria.
```

Después puedo cargar ese archivo. Prueba hacer:

```
rm(list = ls()) # limpiamos workspace
load(file = "ejemplo.RData") #la extensión puede ser arbitraria.
ls()
```

Nota como los objetos preservan el nombre con el que fueron guardados.

Librerías

R puede hacer muchos análisis estadísticos y de datos. Las diferentes capacidades están organizadas en paquetes o librerías. Con la [instalación estándar](#) se instalan también las librerías más comunes. Para obtener una lista de todos los paquetes instalados se puede utilizar el comando `library()` en la consola.

Existen una gran cantidad de paquetes disponibles además de los incluidos por default.

CRAN

CRAN o *Comprehensive R Archive Network* es una colección de sitios que contienen exactamente el mismo material, es decir, las distribuciones de R, las extensiones, la documentación y los binarios. El master de CRAN está en Wirtschaftsuniversität Wien en Austria. Éste se “espeja” (*mirrors*) en forma diaria a muchos sitios alrededor del mundo. En la [lista de espejos](#) se puede ver que para México están disponibles el espejo del ITAM, del Colegio de Postgraduados (Texcoco) y Jellyfish Foundation.

Los espejos son importantes pues, cada vez que busquen instalar paquetes, se les preguntará qué espejo quieren utilizar para la sesión en cuestión. Del espejo que seleccionen, será del cuál R *bajará* el binario y la documentación.

Del CRAN es que se obtiene la última versión oficial de R. Diario se actualizan los espejos. Para más detalles consultar el [FAQ](#).

Para contribuir un paquete en CRAN se deben seguir las instrucciones [aquí](#).

Github

Git es un controlador de versiones muy popular para desarrollar software. Cuando se combina con [GitHub](#) se puede compartir el código con el resto de la comunidad. Éste controlador de versiones es el más popular entre los que contribuyen a R. Muchos problemas a los que uno se enfrenta alguien ya los desarrolló y no necesariamente publicó el paquete en CRAN. Para instalar algún paquete desde GitHub, se pueden seguir las instrucciones siguientes

```
install.packages("devtools")
devtools::install_github("username/packageName")
```

Donde **username** es el usuario de Github y **packageName** es el nombre del repositorio que contiene el paquete. Cuidado, no todo repositorio en GitHub es un paquete. Para más información ver el capítulo [Git and GitHub](#) en Wickham (2015).

Otras fuentes

Otros lugares en donde es común que se publiquen paquetes es en [Bioconductor](#) un proyecto de software para la comprensión de datos del genoma humano.

Paquetes recomendados

Hay muchísimas librerías y lo recomendable es, dado un problema y un modelo para resolverlo, revisar si alguien ya implementó el método en algunas de las fuentes de paquetes mencionadas antes. Para una lista de paquetes que son de mucha utilidad ver [estas recomendaciones](#).

Scripting

R es un intérprete. Utiliza un ambiente basado en línea de comandos. Por ende, es necesario escribir la secuencia de comandos que se desea realizar a diferencia de otras herramientas en donde es posible utilizar el mouse o menús.

Aunque los comandos pueden ser ejecutados directamente en consola una única vez, también es posible guardarlos en archivos conocidos como *scripts*. Típicamente, utilizamos la extensión **.R** o **.r**. En RStudio, CTRL + SHIFT + N abre inmediatamente un nuevo editor en el panel superior izquierdo.

Se puede *ir editando* el script y corriendo los comandos línea por línea con CTRL + ENTER. Esto también aplica para *correr* una selección del texto editable.

Es posible también correr todo el script

```
source("foo.R")
```

O con el atajo CTRL + SHIFT + S en RStudio.

Para enlistar algunos shortcuts comunes en RStudio presiona ALT + SHIFT + K. De la misma manera, si utilizas Emacs + ESS, existen múltiples atajos de teclado para realizar todo mucho más eficientemente. Estudiarlos no es tiempo perdido.

Ayuda & documentación

R tiene mucha documentación. Desde la consola se puede acceder a la misma.

Para ayuda general,

```
help.start()
```

Para la ayuda de una función en específico, por ejemplo, si se quiere graficar algo y sabemos que existe la función `plot` podemos consultar fácilmente la ayuda.

```
help(plot)
# o tecleando directamente
?plot
```

El segundo ejemplo se puede extender para buscar esa función en todos los paquetes que tengo instalados en mi ambiente al escribir `??plot`.

La documentación normalmente se acompaña de ejemplos. Para *correr* los ejemplos sin necesidad de copiar y pegar, prueba

```
example(plot)
```

Para búsquedas más comprensivas, se puede buscar de otras maneras:

```
apropos("foo") # Enlista todas las funciones que contengan la cadena "foo"
RSiteSearch("foo") # Busca por la cadena "foo" en todos los manuales de ayuda
# y listas de distribución.
```

Estructuras de datos

R tiene diferentes tipos y estructuras de datos que permiten al usuario aprovechar el lenguaje. La manipulación de estos objetos es algo que se hace diario y entender cómo operarlos o cómo convertir de una a otra es muy útil.



En R

- Todo lo que existe es un objeto.
- Todo lo que sucede es una llamada a una función.

Clases atómicas (atomic classes)

R tiene 6 clases atómicas.

- character (*character*)
- numeric (números reales o decimales)
- integer (números enteros)
- logical (booleanos, i.e. falsos-verdaderos)
- complex (números complejos)

Tipo	Ejemplo
Caracter	"hola", "x"
Numérico	67, 45.5
Integer	2L, 67L
Lógico	TRUE, FALSE, T, F
Complejo	1 + 4i

Cuadro 1: Clases atómicas.

Algunos comandos importantes para las clases atómicas son su tipo `typeof()`, su tamaño `length()` y sus atributos `attributes()`, es decir, sus metadatos.

```
##### Ejemplo 1
```

```
x <- "una cadena"  
typeof(x)
```

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

```
length(x) # tamaño: cuántas cadenas son?
```

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

```
nchar(x) # Número de caracteres
```

```
## [1] 10
```

```
attributes(x) # Le pusimos metadatos?
```

```
## NULL
```

```
##### Ejemplo 2
```

```
y <- 1:10  
typeof(y)
```

```
## [1] "integer"
```

```
length(y)
```

```
## [1] 10
```

```
attributes(y)
```

```
## NULL
```

```
##### Ejemplo 3
```

```
z <- c(1L, 2L, 3L) # Nota como para denotar enteros debes incluir una L al final  
typeof(z)
```

```
## [1] "integer"
```

```
length(z)
```

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

Vectores

Los vectores son la estructura de datos más común y básica de R. Hay dos tipos de vectores: vectores atómicos y listas.

Típicamente -en libros, blogs, manuales, cuando se mencionan vectores se refieren a los atómicos y no a las listas.

Vectores atómicos

Un vector es un conjunto de elementos con alguna de las clases atómicas, es decir, `character`, `logical`, `integer`, `numeric`. Se puede crear un vector vacío con el comando `vector()` así como especificar su tamaño y su clase.

```
v <- vector()  
v
```

```
## logical(0)
```

```
## Especifico clase y longitud  
vector("character", length = 10)
```

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

```
## Lo mismo pero usando un wrapper  
character(10)
```

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

```
## Numerico de tamaño 5
numeric(5)
```

```
## [1] 0 0 0 0 0
```

```
## Lógico tamaño 5
logical(5)
```

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

Realiza los siguientes ejemplos en la consola de R.

```
x <- rep(1, 5)
x
typeof(x)

xi <- c(1L, 3L, 56L, 4L)
xi
typeof(xi)

y <- c(T, F, T, F, F, T)

z <- c("a", "aba", "andrea", "b", "bueno")
class(z)
str(z)
```

Operaciones con vectores

Accesar partes del vector.

```
a <- c(1:5)
a
```

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

```
a[1]
```

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

```
a[2]
```

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

```
a[4:5]
```

```
## [1] 4 5
```

Aritmética: por default, se realizan componente a componente.


```
b <- a + 10
b
```

```
## [1] 11 12 13 14 15
```

```
c <- sqrt(b) # square root = raíz
c
```

```
## [1] 3.316625 3.464102 3.605551 3.741657 3.872983
```

```
a + c
```

```
## [1] 4.316625 5.464102 6.605551 7.741657 8.872983
```

```
10 * (a + c)
```

```
## [1] 43.16625 54.64102 66.05551 77.41657 88.72983
```

```
a^2
```

```
## [1] 1 4 9 16 25
```

```
a * c
```

```
## [1] 3.316625 6.928203 10.816654 14.966630 19.364917
```

Agregar elementos aun vector ya creado

```
a <- c(a, 7)
a
```

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

Para construir datos rápido, podemos usar comandos como `rep`, `seq` o distintas distribuciones, e.g., la normal `rnorm`, uniformes `runif` o cualquiera en [esta lista](#).

Prueba lo siguiente:

```
# Dame un vector donde el minimo sea 0, maximo 1 en intervalos de 0.25
seq(0, 1, 0.25)
# Vector con 10 unos
rep(1, 10)
# 5 realizaciones de una normal(0,1)
rnorm(5)
# De una normal(10, 5)
rnorm(5, mean = 10, sd = sqrt(5))
# De una uniforme(0,1)
runif(5)
# De una uniforme(5, 15)
runif(5, min = 5, max = 15)
```

Otros objetos importantes

Inf es como R denomina al infinito. En el mundo de R se permite también positivo o negativo.

```
1/0
```

```
## [1] Inf
```

```
1/Inf
```

```
## [1] 0
```

NaN es como R denota a algo que no es un número (literal: *not a number*).

```
0/0
```

```
## [1] NaN
```

Cada objeto tiene atributos. Hay atributos específicos para vectores que, sin importar su clase, tienen en común. Ya revisamos algunos: tamaño (`length`), clase (`class`). También son importantes atributos como los nombres

```
calificaciones <- c(6, 5, 8, 9, 10)
names(calificaciones) <- c("Maria", "Jorge", "Miguel", "Raúl", "Carla")
attributes(calificaciones)
```

```
## $names
## [1] "Maria" "Jorge" "Miguel" "Raúl" "Carla"
```

```
# O llamamos directo a los nombres
names(calificaciones)
```

```
## [1] "Maria" "Jorge" "Miguel" "Raúl" "Carla"
```

Mezclar tipos no es una buena idea

```
c(1.7, "a")
```

```
## [1] "1.7" "a"
```

```
c(TRUE, 2)
```

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

```
c("a", TRUE)
```

```
## [1] "a" "TRUE"
```

R realiza una coerción implícita entre los objetos y “decide” cuál es la clase del vector. También hay coerción explícita (*explicit coercion*) utilizando `as.<nombre_clase>`.

```
as.numeric()
as.character()
as.integer()
as.logical()
```

Muchos problemas suceden cuando le permites a R decidir por ti (o cuando no sabes cuál decisión tomará R por *default*).

```
x <- 0:5

identical(x, as.numeric(x))
```

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

En este ejemplo, cuando declaramos *x* no especificamos su clase y R decidió que era entero. Al coercionar al objeto para que fuese numérico, R no considera a los dos objetos iguales. R te protege -no te permite hacer o te advierte- de algunas cosas

```
1 < "2"
```

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

Pero en otras, hace lo mejor que puede con lo que le das (cosa que a veces no tiene sentido)

```
x <- c("a", "b", "c")
as.numeric(x)
```

```
## Warning: NAs introduced by coercion
```

```
## [1] NA NA NA
```

```
as.logical(x)
```

```
## [1] NA NA NA
```

Matrices

Las matrices son un tipo especial de vectores. Son un vector atómico con dimensión pues tienen filas y columnas.

```
m <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2)
m
```

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

Como puedes notar, las matrices se llenan siguiendo las columnas. Podemos simplemente “agregarle” una dimensión a un vector para construir una matriz.

```
m <- 1:10
dim(m) <- c(2, 5)
m
```

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

También podemos pegar vectores de la misma longitud como si fueran columnas de una matriz `cbind` o como si fueran filas `rbind` (`r = row`, `c = column`).

```
x <- runif(4)
y <- rnorm(4)
cbind(x, y)
```

```
##           x           y
## [1,] 0.7175439 1.5110723
## [2,] 0.9804935 0.8484441
## [3,] 0.6155117 -1.6513531
## [4,] 0.7500755 -1.3797025
```

```
rbind(x, y)
```

```
##      [,1]      [,2]      [,3]      [,4]
## x 0.7175439 0.9804935 0.6155117 0.7500755
## y 1.5110723 0.8484441 -1.6513531 -1.3797025
```

Le agregamos atributos para acceder más fácilmente a los objetos.

```
m <- matrix(c(x, y), nrow = 4, ncol = 2, byrow = T,
            dimnames = list(paste0("row", 1:4),
                             paste0("col", 1:2)))
m
```

```
##           col1      col2
## row1 0.7175439 0.9804935
## row2 0.6155117 0.7500755
## row3 1.5110723 0.8484441
## row4 -1.6513531 -1.3797025
```

```
m[1, 1] == m["row1", "col1"]
```

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

Listas

Factores

Data frames

Leer y escribir archivos de datos

Lectura

Desde archivo

Datos de muestra

Escritura

Estructuras de programación

IFs

Fors

Whiles

Funciones propias

Referencias

[Wic15] Hadley Wickham. *R packages*. "O'Reilly Media, Inc.", 2015.