



Universidad
Carlos III de Madrid

Lenguaje R

José M. Valls Ferrán y Ricardo Aler Mur

José M^a Valls Ferrán y Ricardo Aler Mur



Presentación

- Contenido
 - Sesión 1: Introducción a R. Vectores
 - Sesión 2: E/S, Matrices, Funciones, Estructuras de control
 - Sesión 3: Data Frames, Listas.
 - Sesión 5: Sistemas de Plots, realización de informes en R.
 - Sesión 4: Procesamiento de datos avanzado



Universidad
Carlos III de Madrid

Sesión 1: Introducción a R. Vectores

José M. Valls Ferrán y Ricardo Aler Mur

José M^a Valls Ferrán y Ricardo Aler Mur



Introducción. ¿Qué es R?

- R es un lenguaje de programación, potente, funcional y orientado a objetos
- Está orientado al análisis de datos
 - Estructuras de datos especializadas (data.frames)
 - Por ejemplo, maneja directamente valores faltantes (NAs)
 - Las operaciones estadísticas (tests estadísticos, distribuciones, modelos lineales, ...) han sido realizadas y probados por especialistas
 - Muchas librerías / paquetes de buena calidad, listos para usar
 - Potentes gráficos

Introducción. Historia de R

- R es un dialecto del lenguaje S
- S fue desarrollado por John Chambers en los laboratorios Bell en 1976. El objetivo era facilitar el análisis estadístico. Inicialmente usaba librerías en Fortran, después fueron reescritas a C.
- Característica de S: análisis de datos interactivo y también posibilidad de escribir scripts (programas)
- En la actualidad es propiedad de TIBCO (25 millones de dolares)

Introducción. Historia de R

- R se crea en 1991 en Nueva Zelanda por Ihaka y Gentleman, con el objetivo de tener similares posibilidades a S. Sintaxis similar aunque los detalles internos son distintos
- En el 2000 se crea R 1.0.0 con licencia GNU GPL (software libre)
- La versión 3.2.1 salió en junio 2015. Desarrollo muy activo
- Ejecuta en cualquier plataforma

Introducción. Ventajas de R

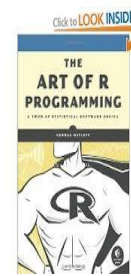
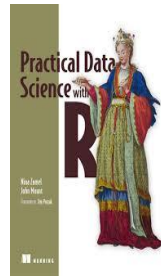
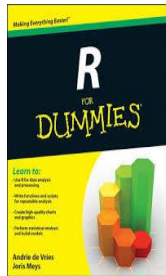
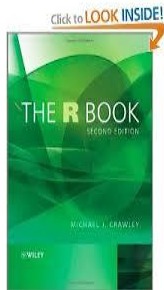
- Es libre
- Es bastante ligero (comparar el arranque de R con el arranque de Matlab)
- Orientado al proceso y análisis de datos (con data.frames). El acceso a matrices y data.frames es parecido al de Matlab
- Gran potencia de cálculo
- Gráficos potentes
- El más utilizado en análisis de datos (según encuestas)
- Comunidad muy activa. 4000 paquetes desarrollados, fácilmente instalables, y disponibles en CRAN: <http://cran.r-project.org/>
- Mucha documentación y libros sobre el lenguaje

Introducción. Bibliografía

Tutorial oficial en castellano:

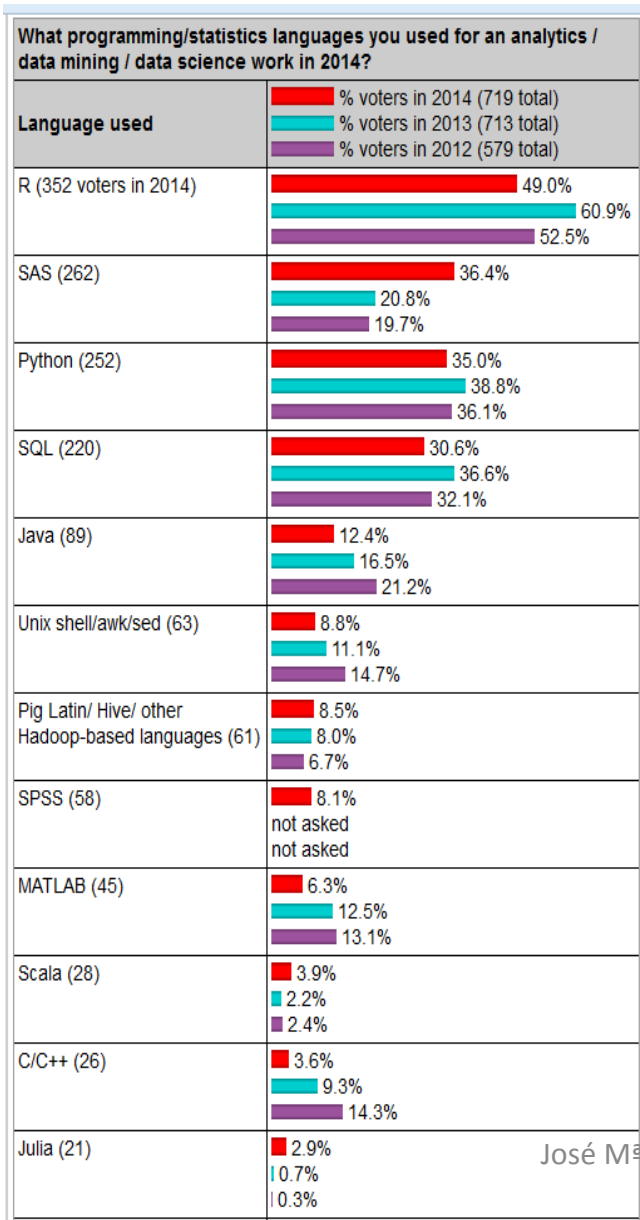
<https://cran.r-project.org/doc/contrib/R-intro-1.1.0-espanol.1.pdf>

Libros generales:



Introducción. Uso de R

<http://www.kdnuggets.com>



Introducción. Software Consola de R

- Se descarga el software desde

<https://www.r-project.org/>



[Home]

Download

CRAN

R Project

About R
Contributors
What's New?
Mailing Lists
Bug Tracking
Conferences
Search

R Foundation

Foundation
Board
Members
Donors

The R Project for Statistical Computing

Getting Started

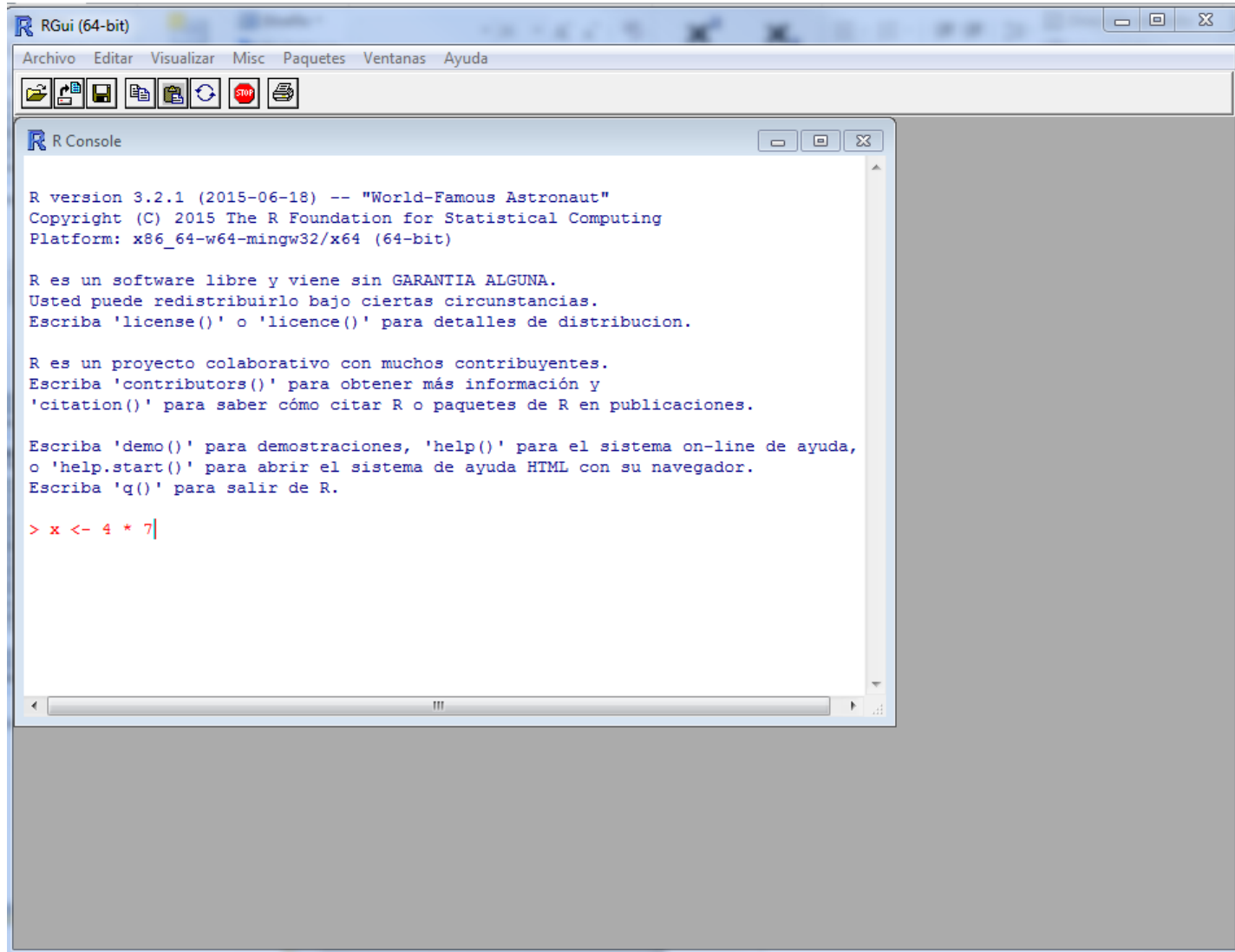
R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To [download R](#), please choose your preferred CRAN mirror.

If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

News

- **R version 3.2.2 (Fire Safety)** has been released on 2015-08-14.
- **The R Journal Volume 7/1** is available.
- **R version 3.1.3 (Smooth Sidewalk)** has been released on 2015-03-09.
- **useR! 2015**, will take place at the University of Aalborg, Denmark, June 30 - July 3, 2015.
- **useR! 2014**, took place at the University of California, Los Angeles, USA June 30 - July 3, 2014.

Introducción. Consola de R



```
RGui (64-bit)
Archivo  Editar  Visualizar  Misc  Paquetes  Ventanas  Ayuda

R Console

R version 3.2.1 (2015-06-18) -- "World-Famous Astronaut"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R es un software libre y viene sin GARANTIA ALGUNA.
Usted puede redistribuirlo bajo ciertas circunstancias.
Escriba 'license()' o 'licence()' para detalles de distribucion.

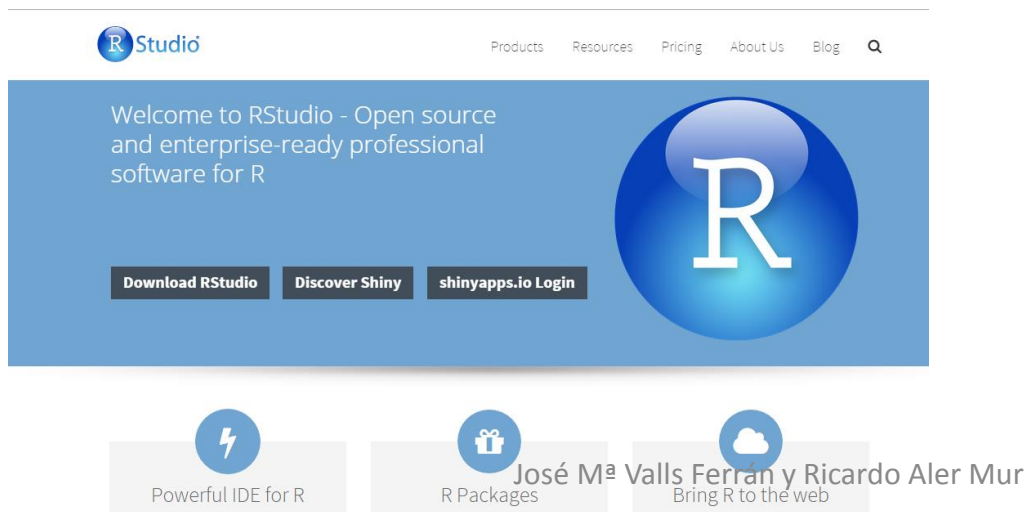
R es un proyecto colaborativo con muchos contribuyentes.
Escriba 'contributors()' para obtener más información y
'citation()' para saber cómo citar R o paquetes de R en publicaciones.

Escriba 'demo()' para demostraciones, 'help()' para el sistema on-line de ayuda,
o 'help.start()' para abrir el sistema de ayuda HTML con su navegador.
Escriba 'q()' para salir de R.

> x <- 4 * 7|
```

Introducción. R Studio

- Entorno de desarrollo integrado para R
- Más cómodo para trabajar con R
- Software libre
- Disponible para Windows, Mac, Linux
- Primero hay que instalar R y luego Rstudio
- Se descarga en: <http://rstudio.org/>



Introducción. R Studio

The screenshot displays the RStudio environment with the following components:

- Source Editor:** Shows a script named `scriptSintetico.R` with a single line of code: `1`.
- Console:** Displays the R startup message and workspace information:

```
R version 3.2.1 (2015-06-18) -- "world-Famous Astronaut"
Copyright (c) 2015 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

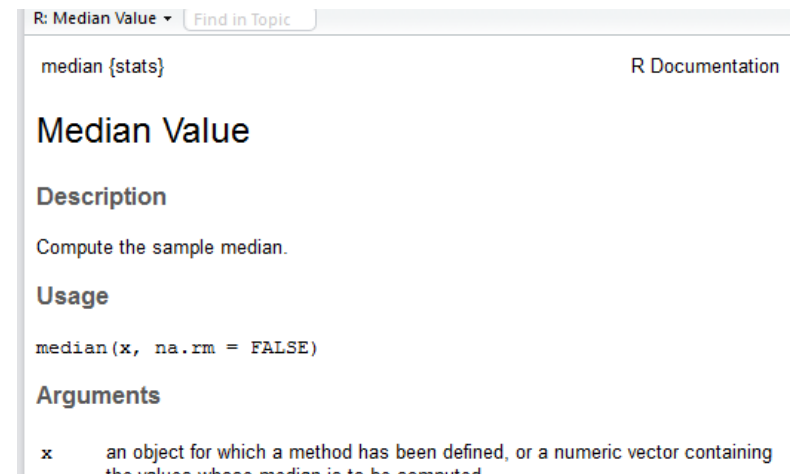
[workspace loaded from ~/.RData]
Loading required package: Matrix
>
```
- Environment:** Lists the objects in the global environment:

Object	Value
<code>datos</code>	Large matrix (2 elements, 1.7 Mb)
<code>entrada</code>	Large matrix (5256000 elements, 40.4 Mb)
<code>salida</code>	int [1:4380, 1] 12384900 11908500 12...
<code>test</code>	733 obs. of 1201 variables
<code>test1</code>	Large matrix (879600 elements, 6.8 Mb)
<code>train</code>	4380 obs. of 1201 variables
<code>a</code>	10
<code>agaricus.test</code>	List of 2
<code>agaricus.train</code>	Large list (2 elements, 1.7 Mb)
<code>bst</code>	
<code>D.sc</code>	$-(\cos(\cos(x + y^2)) * \sin(x + y^2))$
<code>h</code>	1
<code>lmat</code>	List of 2
<code>trig.exp</code>	$\text{expression}(\sin(\cos(x + y^2)))$
<code>x</code>	Large numeric (200001 elements, 1.5 Mb)
<code>ypred1</code>	num [1:733] 12241214 12241214 8085868...
- Files:** Shows the project files, including `scriptSintetico.R`, `SVMSt01NEW.R`, and `dominios.r`.

Introducción. Ayuda en línea

- Si queremos ayuda sobre alguna función, podemos teclear en la consola: **?** Junto con el nombre de la función. Ejemplo:

- **?median**
- También sirve **help("median")**
- Si queremos ayuda sobre un operador **help("^")**



Arithmetic Operators

Description

These unary and binary operators perform arithmetic on numeric or complex vectors (or objects which can be coerced to them).

Usage

```
+ x
- x
x + y
x - y
x * y
x / y
x ^ y
x %>% y
```

Objetos en R

Todo lo que utilizamos en R son “objetos”

Los objetos primitivos de R son:

Variables

- Corresponden a un espacio de memoria donde se almacena un dato y están identificadas por un **nombre**
- Pueden ser de distinta clase o tipo: **numéricas, lógicas, de caracteres**
- Las variables atómicas (escalares) pueden agruparse en objetos más complejos: **vectores, matrices, listas**

Funciones

- Objetos que reciben argumentos y devuelven un resultado

Variables

- Expresiones

- Podemos usar R como calculadora

```
> 3+4 *2
```

```
[1] 11
```

```
> 21 %% 2    (módulo)
```

```
[1] 1
```

```
> 5/2        (división real)
```

```
[1] 2.5
```

```
> sqrt(2)    (raíz cuadrada)
```

```
[1] 1.414214
```

```
> 5%/%2      (división entera)
```

```
[1] 2
```

```
> exp(1)     (e elevado a 1)
```

```
[1] 2.718282
```

```
> 4^3        (potencia)
```

```
[1] 64
```

```
> 2*pi       (número pi)
```

```
[1] 6.283185
```


Variables

- **Asignaciones a variables:** El resultado de la expresión se almacena en una variable
 - se usa el operador `<-` o `=`
 - `> x <- 945`
 - `> y <- 56 * 5 / 3`
 - Se evalúa la expresión a la derecha del `<-` y luego se asigna el valor resultante
 - `> y <- x + 10`
 - `> y <- y + 20`
 - Ojo! R es case-sensitive: `x` y `X` son variables distintas
- **Visualizar el valor de una variable**
 - `> x`
 - `[1] 945`
 - `> y`
 - `[1] 975`

Listar y borrar variables

- `ls()`: listar objetos
- `rm()`: borrar objetos
- `rm(list=ls())`: borrar todos los objetos

```
> ls()  
[1] "x" "y"
```

```
> rm(x)  
> ls()  
[1] "y"
```

- *Ejercicio: Crear 5 variables diferentes, asignando resultados de expresiones numéricas donde aparezcan variables anteriores. Listarlas. Borrar una de ellas. Borrar todas*

Clases o Tipos de datos escalares (“ATOMIC”)

- **character:**

```
b <- "cadena de caracteres"
```

```
b <- 'cadena de caracteres'
```

```
class(b)
```

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

- **logical:**

```
q <- TRUE
```

```
q <- T
```

```
q <- FALSE
```

```
q <- F
```

```
class(q)
```

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

- **numeric:**

```
x <- 3
```

```
class(x)
```

```
[1] "numeric"
```

- **integer**

```
n <- 8L
```

```
class(n)
```

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

```
class(z)
```

```
[1] "complex"
```

- **complex:**

```
z <- -2 + 0i
```

```
class(z)
```

```
[1] "complex"
```

```
sqrt(z)
```

```
[1] 0+1.414214i
```

Operadores aritméticos y lógicos

Aritméticos

Se aplican a valores numéricos, dan como resultado un valor numérico

+	suma
-	resta
*	multiplicación
/	división
^	potenciación
%%	división entera
%%	módulo

De comparación

Comparan valores numéricos dando como resultado un valor lógico

>	mayor
<	menor
>=	mayor o igual
<=	menor o igual
==	igual
!=	distinto

Se aplican a valores lógicos dando como resultado un valor lógico.

(son operadores vectorizados, si se aplican a vectores dan como resultado un vector de valores lógicos)

&	AND
	OR
!	NOT

Lógicos

Operadores lógicos

Variante de operadores lógicos:

&& AND

|| OR

Son operadores no vectorizados, dan como resultado un valor lógico escalar.

Si se utilizan con vectores, se aplican sólo al primer elemento de cada vector.

Normalmente se utilizan en las condiciones de las estructuras de control (if, while..) para garantizar un valor lógico único

Expresiones con operadores

- Ejercicio. Calcular los valores de las variables c, d y e

```
> a <- 5
```

```
> b <- 10
```

```
> c <- a > (b-a)
```

```
> d <- (b == (2*a))
```

```
> e <- !(c | d)
```

Ojo: el primer símbolo de cada línea es el prompt de la consola, no el operador mayor

```
> c  
[1] FALSE  
> d  
[1] TRUE  
> e  
[1] FALSE
```

Objetos en R

Los objetos que normalmente se utilizan en R son estructuras o agrupaciones de datos

- **Vectores**: colección de datos del mismo tipo. Los escalares son realmente vectores con un solo elemento
 - **Factores**: vectores que representan datos categorizados
- **Matrices**: todos los elementos del mismo tipo, se accede a los elementos por fila y columna
- **Data frames**: son matrices con columnas de distintos tipos
- **Listas**: permite combinar elementos de tipos distintos

Vectores. Función c()

- Todos los elementos son del mismo tipo
- Puede utilizarse la función **c()** para crear vectores (c() concatena elementos del mismo tipo)

```
> x<-c("hoola","adios","a")
```

```
> x           # x es un vector de character con tres elementos
```

```
[1] "hoola" "adios" "a"
```

```
> y<-c(3,6,4.5,7)
```

```
# vector de 4 numeric
```

```
> y
```

```
[1] 3.0 6.0 4.5 7.0
```

```
> v <- c(T,F,T,F,FALSE)
```

```
# vector de 5 logical
```

```
> v
```

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

- Pueden **concatenarse vectores**

```
> x <- c(1,2,3)
```

```
> y <- c(4,5)
```

```
> z <- c(y,x,c(20,30))
```

```
> z
```

```
[1] 4 5 1 2 3 20 30
```


Vectores

- Para ver tipo y longitud: `class()`, `length()`

```
> x <- c(3,5,7,2,8)
> class(x)
[1] "numeric"
> length(x)
[1] 5
```

- Si se mezclan tipos, se fuerzan para que todos sean de la misma clase
 - Comprobar qué ocurre si mezclamos los siguientes tipos

```
v <- c(4, 6.5, 3, "hola")
U <- c(T, F, 5, 7)
```

```
> v
[1] "4" "6.5" "3" "hola"

> u
[1] 1 0 5 7
```

Vectores. Valores especiales

- **NA** significa “sin valor” o valor faltante (missing value, not available)

```
> y<-c(3,6,2,NA,1)
> y
[1] 3 6 2 NA 1
> class(y)
[1] "numeric"
```

```
> y<-c("jose","luis",NA,"andres")
> y
[1] "jose" "luis" NA "andres"
> class(y)
[1] "character"
```

- Con valores numéricos: **Inf**, **NaN**
 - **Inf** representa infinito y NaN un valor numérico indeterminado

- 3/0 Inf

- 0/0 NaN

```
> x<-c(2,4,6,0,10)
> y<-c(0,2,2,0,2)
> x/y
[1] Inf 2 3 NaN 5
```

Probar a hacer operaciones como divisiones por 0, logaritmos de 0, etc..

Vectores. Valores especiales

- **Comprobar si un valor es NA, Inf, NaN**

```
is.na()  
is.nan()  
is.infinite()
```

Estas funciones se pueden aplicar a vectores. Devuelven un vector de valores lógicos (T,F)

```
x <- c(4,6,8,NA,9, Inf,10,20)  
> is.na(x)  
[1] FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE  
> is.infinite(x)  
[1] FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE
```

- **¿Cuántos NA hay en un vector? ¿dónde están?**
 - Aplicar `is.na()` y guardar el resultado
 - Contar los valores TRUE, utilizar la función `sum()`
 - Utilizar la función `which()` para saber dónde están los TRUE

Vectores. Valores especiales

- **Ejercicio:** Generar un vector con 10 valores, algunos de los cuales son NA. Comprobar la **longitud** del vector. Guardar en una variable **cuantos** el número de valores NA y en una variable **donde** las posiciones donde están los NA. Guardar en las variables **cuantosNo** y **dondeNo** el número y posiciones de los valores que NO son NA

```
x <- c(3,6,2,NA,12,5,NA,NA,9,0)
> length(x)
[1] 10
```

```
faltan <- is.na(x)
> cuantos <- sum(faltan)
> donde <- which(faltan)
> cuantos
[1] 3
> donde
[1] 4 7 8
```

```
noFaltan <- !is.na(x)
# el contrario is.na(), cambia los T por F y los F por T
> cuantosNo <- sum(noFaltan)
> dondeNo <- which(noFaltan)
> cuantosNo
[1] 7
> dondeNo
[1] 1 2 3 5 6 9 10
```

Vectores. Valores especiales

- Ciertas funciones pueden manejar NA's
- Por ejemplo, la función ***mean()*** (media aritmética) devuelve NA si el vector contiene algún NA, pero mediante un argumento le podemos indicar a la función que ignore los NA

```
> x  
> [1] 4 5 NA 3 6 NA 8 1 2  
> mean(x)  
> [1] NA
```

```
> mean(x, na.rm=T)  
> [1] 4.142857
```

Le pedimos a la función `mean()` que ignore los NA (`na.rm` significa `remove na`)

Vectores. Conversión de tipos

- Crear un vector de un tipo y longitud determinado: **vector()**
 - Crea un vector nulo de longitud y tipo determinados

```
> x <- vector("integer",10)
> x
[1] 0 0 0 0 0 0 0 0 0 0
```

```
> c <- vector("character",10)
> c
[1] "" "" "" "" "" "" "" "" "" ""
```

- Para forzar la **conversión de tipos**

as.numeric()

as.integer()

as.logical()

as.character()

as.complex()

Si queremos que el propio vector se transforme:

```
x <- as.character(x)
```

```
> x
[1] 0 1 2 3 4 5
> class(x)
[1] "integer"
> as.character(x)
[1] "0" "1" "2" "3" "4" "5"
> as.logical(x)
[1] FALSE TRUE TRUE TRUE TRUE TRUE
```

```
> car
[1] "0" "1899" "2" "3" "4h" "5"
> as.numeric(car)
[1] 0 1899 2 3 NA 5
Warning message:
NAs introduced by coercion
```

Vectorización

- Normalmente, una **función aplicada a un vector**, es aplicada a cada uno de los elementos

```
v <- c(2,3,4,5,6,7)
> sqrt(v)
[1] 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751
```

- **Operaciones con dos vectores iguales**

- Se efectúa la operación elemento a elemento

```
> v1 <- c(3,5,7)
> v2 <- c(10,20,30)
> v1+v2
[1] 13 25 37
```

```
> v1<v2
[1] TRUE TRUE TRUE
```

Vectores. Regla del reciclado

- Si se realiza una operación con dos vectores de distinto tamaño, **el más pequeño se replica** hasta que tiene el mismo tamaño que el grande (esto es cierto incluso con valores individuales)

```
> v1 <-c(4,6,8,24)
> v2 <- c(10,2)
> v1+v2
[1] 14 8 18 26
```

El vector v2 se replica a (10,2,10,2) y ahora se hace la operación elemento a elemento

```
> v1 <-c(4,6,8,24)
> 2 * v1
[1] 8 12 16 48
```

El vector [2] se replica a (2,2,2,2) y ahora se hace la operación elemento a elemento

Vectores. Regla del reciclado

- También se aplica en operaciones lógicas

```
> v <- c(3, 4, 5, 6, 8)
> v > 4
[1] FALSE FALSE TRUE TRUE TRUE
```

El vector [4] se replica a (4,4,4,4,4) y ahora se hace la comparación elemento a elemento

- Si la longitud del vector mayor **no es múltiplo** de la del menor, se produce un **warning** pero se realiza la operación

```
v1 <- c(4,6,8,24)
v2 <- c(10,2,4)
v1+v2
```

El vector [4] se replica a (10,2,4,10) y ahora se hace la comparación elemento a elemento

```
[1] 14 8 12 34
```

Warning message: In v1 + v2 : longer object length is not a multiple of shorter object length

Vectores. Regla del reciclado

- **Ejercicio:** hacer sobre el papel las siguientes operaciones. Luego, comprobarlo en R Studio

- $x \leftarrow c(1,2,3,4,5)$

- $x + 1$

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

Se suma elem. a elem. (1,2,3,4,5) con (1,1,1,1,1)

- $x + c(1,2)$

```
[1] 2 4 4 6 6      con warning
```

Se suma elemento a elemento (1,2,3,4,5) con (1,2,1,2,1)

- $x \geq 3$

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

Se compara elemento a elemento (1,2,3,4,5) con (3,3,3,3,3)

- $x ^ c(1,2)$

```
[1] 1 4 3 16 5      con warning
```

Eleva (1,2,3,4,5) a los exponentes (1,2,1,2,1)

Generación de vectores mediante secuencias

- **Secuencias de enteros** **num1 : num2**

- Crear un vector con los enteros de 1 a 100

```
> x <- 1:100
```

```
> x
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
```

```
[19] 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
```

```
[37] 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
```

```
[55] 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
```

```
[73] 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
```

```
[91] 91 92 93 94 95 96 97 98 99 100
```

- Secuencia invertida si num1 es mayor que num2

```
> x <- 10:4
```

```
> x
```

```
[1] 10 9 8 7 6 5 4
```

Generación de vectores mediante secuencias. seq()

- La función **seq()** nos permite muchas más opciones para definir secuencias (desde, hasta, incremento, longitud)

from to
> **seq(5,10)**
[1] 5 6 7 8 9 10

Secuencia desde 5 hasta 10 (incremento de 1 en 1)

from to by
> **seq(5, 10, 0.5)**

Secuencia desde 5 hasta 10, incrementando de 0.5 en 0.5

[1] 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5 9.0 9.5 10.0

from to length
> **seq(3,5,length=10)**

[1] 3.000000 3.222222 3.444444 3.666667 3.888889 4.111111 4.333333 4.555556
[9] 4.777778 5.000000

Secuencia desde 3 hasta 5, con un total de 10 elementos. La función calcula los incrementos

Generación de vectores mediante secuencias. seq()

- Se pueden hacer todas las combinaciones con los argumentos de la función

```
> seq(from=5, by=0.4, length=10)
```

```
[1] 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8 8.2 8.6
```

```
> seq(to=5, by=0.4, length=10)
```

```
[1] 1.4 1.8 2.2 2.6 3.0 3.4 3.8 4.2 4.6 5.0
```

- **Ejercicios:**

- Generar una secuencia de 100 números de 10 a 20. Guardarla en un vector x
- Generar una secuencia de números de 10 a 20 con incrementos de 0.015. Guardarla en un vector x. ¿Cuántos valores hay?
- Generar una secuencia de 500 números desde 0 con incrementos de 0.01. ¿Dónde termina?

Generación de vectores mediante secuencias. seq()

- Soluciones de los ejercicios anteriores:

```
> x <- seq(10,20, length=100)
```

```
> x<- seq(10,20,by=0.015)
```

```
> length(x)
```

```
[1] 667
```

```
x<- seq(0,length=500,by=0.01)
```

```
Termina en 4.99
```

Generación de vectores mediante secuencias. rep()

- **Función rep()**. Sirve para replicar elementos

```
> rep(5,10)
```

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

Repite 10 veces el elemento 5

```
> x <- rep("hola", 4)
```

```
> x
```

```
[1] "hola" "hola" "hola" "hola"
```

Repite 4 veces la cadena "hola" y la guarda en el vector x

```
> rep(1:3,4)
```

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

Repite 4 veces la secuencia 1, 2, 3

```
> rep(1:3, each=4)
```

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

Repite 4 veces cada elemento de la secuencia 1, 2, 3

Secuencias aleatorias

- **rnorm ()**. Generación de secuencia aleatorias con **distribución normal**

```
> rnorm(10)
```

Genera 10 valores con media 0 y desviación 1

```
[1] 1.2348094 2.5280030 -0.4916411 -0.1921868 0.6248684 -0.6549314  
[7] -1.8158976 -1.3319541 2.3614266 -0.2032376
```

```
> rnorm (10,5,0.1)
```

Genera 10 valores con media 5 y desviación 0.1

```
[1] 4.819050 5.124391 4.902603 4.931901 4.862523 4.849432 5.000581 5.013229  
[9] 5.191178 4.986566
```

```
> rnorm (10,mean=5,sd=2)
```

Genera 10 valores con media 5 y desviación 2

```
[1] 5.984334 3.833287 5.252370 3.030158 3.735222 7.134420 2.115884 2.574890  
[9] 7.663965 4.650143
```


Secuencias aleatorias

- Ejercicio:
 - Generar 1000 valores aleatorios con dist. normal (0,1) y hallar su media (guardarlos en un vector y usar la función mean)

```
> x<-rnorm(1000)
> mean(x)
[1] 0.02069629
```

- Hacer lo mismo con 1 millón de valores

```
> x<-rnorm(1E6)
> mean(x)
[1] -0.0006940407
```

Secuencias aleatorias

- **runif ()**. Generación de valores aleatorios con **distribución uniforme**

> runif (10)

Genera 10 valores con dist. uniforme entre 0 y 1

```
[1] 0.4351369 0.9763666 0.1054699 0.3690474 0.1330247 0.2676920 0.6860112  
[8] 0.5195046 0.1763224 0.3553553
```

> runif(10,5,9)

Genera 10 valores con dist. uniforme entre 5 y 9

```
[1] 8.651682 7.936000 8.375836 7.025929 6.381778 8.134046 5.597491 7.338969  
[9] 8.210275 5.689115
```

> runif(10,min=5,max=10)

Genera 10 valores con dist. uniforme entre 5 y 10

```
[1] 7.530510 8.131322 5.734138 6.593115 9.020961 8.210324 9.331195 7.031099  
[9] 6.844149 8.791955
```

Acceso a vectores (subsetting)

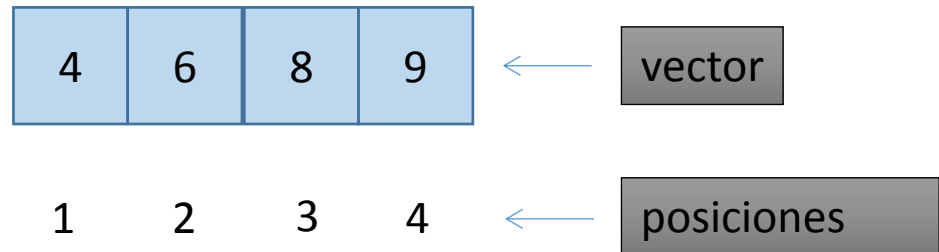
- **Acceso simple**

- **x[n]**: para acceder al elemento número n del vector x

```
> x<-c(4,6,8,9)
```

```
> x[3]
```

```
[1] 8
```



- **Acceso mediante índices. Tres tipos**

- Índices numéricos (o de valores)
- Índices lógicos (booleanos)
- Índices por nombres

Acceso a vectores mediante índices numéricos

- Se indica en [] la posición o posiciones de los elementos (realmente es un vector de índices)

- Elemento individual (acceso simple): $x[3]$

- Varios elementos

$x[1:4]$

Los elementos en las posiciones 1, 2, 3 y 4

$x[c(3,5,1)]$

Los elementos en las posiciones 3, 5 y 1

- Se puede usar – para excluir elementos

$x[-2]$

Todos los elementos excepto el que está en la pos. 2

$x[c(-2,-4)]$ o $x[-c(2,4)]$

Todos los elementos excepto los que están en las posiciones 2 y 4

Acceso a vectores mediante índices numéricos

- **Ejercicio**

- Generar un vector con los números 10, 20, 30, ... 100. Obtener:
- El elemento de la posición 5
- Los elementos de las pos. 8,5 y 9
- Los 4 primeros elementos
- El último elemento (suponer que no sabemos qué posición ocupa)
- Todos menos el 6 y el 8

```
> x<-seq(10,100,10)
> x[5]
[1] 50
> x[c(8,5,9)]
[1] 80 50 90
> x[1:4]
[1] 10 20 30 40
```

```
> x[length(x)]
[1] 100
> x[-c(6,8)]
[1] 10 20 30 40 50 70 90 100
```

Acceso a vectores mediante índices lógicos

- **Sirve para seleccionar valores que cumplan alguna condición**

- Ejemplo. Dado el vector x, seleccionar los mayores de 40

```
> x
```

```
[1] 10 20 30 40 50 60 70 80 90 100
```

```
> x[x>40]
```

```
[1] 50 60 70 80 90 100
```

- Realmente $x > 40$ es un vector lógico:

```
> x>40
```

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

- Estamos indexando con un vector lógico y sólo se va a acceder a las posiciones TRUE

Acceso a vectores mediante índices lógicos

- Recordad los operadores de comparación y lógicos

Comparan valores numéricos dando como resultado un valor lógico

>	mayor
<	menor
>=	mayor o igual
<=	menor o igual
==	igual
!=	distinto

Se aplican a valores lógicos dando como resultado un valor lógico

(Son operaciones vectorizadas)

&	AND
---	-----

	OR
--	----

!	NOT
---	-----

(no vectorizadas)

&&	AND
----	-----

	OR
--	----

- Y las funciones que devuelven valores lógicos como
 - is.na()
 - is.nan()

Acceso a vectores mediante índices lógicos

- Del vector `x` seleccionar los valores comprendidos entre 40 y 70 (incluyéndolos)

```
> x[x>=40 & x<=70]  
[1] 40 50 60 70
```

Es importante entender lo que ocurre:

`x>=40` es el vector `FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE`

`x<=70` es el vector `TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE`

`x >=40 & x<=70` es `FALSE FALSE FALSE TRUE TRUE TRUE TRUE FALSE FALSE FALSE`

Acceso a vectores mediante índices lógicos

- **Ejercicios**

1. Crear el vector x (4,-5,7,8, NA, 2, 6, 80, 3, -6)
2. Mostrar los valores negativos (¿qué pasa con NA?)
3. Mostrar los valores negativos que no sean NA
4. Mostrar los valores pares positivos
5. Mostrar los valores mayores que 2 y menores que 10
6. Hacer una copia en el vector y
7. En y, poner a 0 los valores negativos
8. Poner a 0 los NA

Acceso a vectores mediante índices lógicos

Soluciones

1. `x <- c(4,-5,7,8, NA, 2, 6, 80, 3, -6)`
2. `> x[x<0]`
`[1] -5 NA -6` # muestra también los NA
3. `> x[x<0 & !(is.na(x))]`
`[1] -5 -6`
4. `> x[(x%%2==0)&(x>=0)]`
`[1] 4 8 NA 2 6 80`
5. `> x[x>2 & x<10]`
`[1] 4 7 8 NA 6 3`
6. `y <- x`
7. `y[y<0] <- 0`
`> y`
`[1] 4 0 7 8 NA 2 6 80 3 0`
8. `> y[is.na(y)] <- 0`
`> y`
`[1] 4 0 7 8 0 2 6 80 3 0`

Acceso a vectores por nombre

- Las posiciones de un vector pueden tener nombre

```
pH <- c(4.5,7,7.3,8.2,6.3)
names(pH) <- c('area1','area2','mud','dam','middle')
pH
```

```
## area1 area2 mud dam middle
## 4.5 7.0 7.3 8.2 6.3
```

```
pH <- c(area1=4.5,area2=7,mud=7.3,dam=8.2,middle=6.3)
```

- Acceso por nombre:

```
pH['mud']
```

```
## mud
## 7.3
```

```
pH[c('area1','dam')]
```

```
## area1 dam
## 4.5 8.2
```

Acceso al vector completo

- Por ejemplo, para borrar todos los elementos de un vector x :
 - $x[] = 0$ es lo mismo que $x[1:\text{length}(x)] = 0$ (se aplica la “regla de reciclado”). Asigna 0 a todas las posiciones del vector
 - pero es distinto de $x = 0$, el cual convierte x en un único valor

```
> x
[1] 1 2 3 4
> x[]=0
> x
[1] 0 0 0 0
> x = 0
> x
[1] 0
```

Ejercicios con vectores

- **Ejercicios**

1. Generar un vector x con una secuencia del 1 al 50; añadir 10 NA y desordenarlos (utilizar `sample()`)
2. Extraer los valores comprendidos entre 20 y 30 (inclusive). No incluir los NA
3. Contar los NA
4. Hallar la media y la suma de los elementos de x
Funciones `sum()`, `mean()`, uso del argumento `na.rm=T`
5. Sustituir los NA por 0
6. Hallar la suma y la media de los elementos de x

Ejercicios con vectores

- **Soluciones a los ejercicios anteriores**

1. Generar un vector x con una secuencia del 1 al 50; añadir 10 NA y desordenarlos (utilizar sample())

```
x<-c(1:50, rep(NA, 10))  
x <- sample(x)
```

2. Extraer los valores comprendidos entre 20 y 30 (inclusive). No incluir los NA

```
x[x>=20 & x <= 30 & !is.na(x)]
```

3. Contar los NA

```
sum(is.na(x))
```

4. Hallar la media y la suma de los elementos de x
Funciones sum(), mean(), uso del argumento na.rm=T

```
sum(x, na.rm=T)          mean(x, na.rm=T)
```

5. Sustituir los NA por 0

```
x[is.na(x)]<- 0
```

6. Hallar la suma y la media de los elementos de x

```
sum(x)          mean(x)
```

Ejercicios con vectores

Más ejercicios

1. Crear un vector de 10 elementos aleatorios entre 1 y 100:
`x=sample(1:100, 10)`
2. Poner a cero los valores pares.
3. Poner a cero las **posiciones** pares.

Ejercicios con vectores

Soluciones

- Crear un vector de 10 elementos aleatorios entre 1 y 100
 - `x<-sample(1:100, 10)`
 - `x<-round(runif(10,min=1,max=100))`
- Poner a cero los valores pares.
 - `x[x%%2==0] <- 0`
- Poner a cero las **posiciones** pares.
 - `x[seq(2,length(x),by=2)] <- 0`