Introducción a R

Lucía Babino

Introducción a R

Basado en la clase de Juan Barriola y Sofía Perini

Les dejo un curso online (mío) introductorio de R para quienes necesiten más detalles.

Modalidad

- Clases expositivas
- Ejercicios para el hogar

Material

- Presentaciones que uso durante las clases
- Apuntes de 2024
- Ejercicios para practicar (junto con los datos)
- Ejercicios resueltos

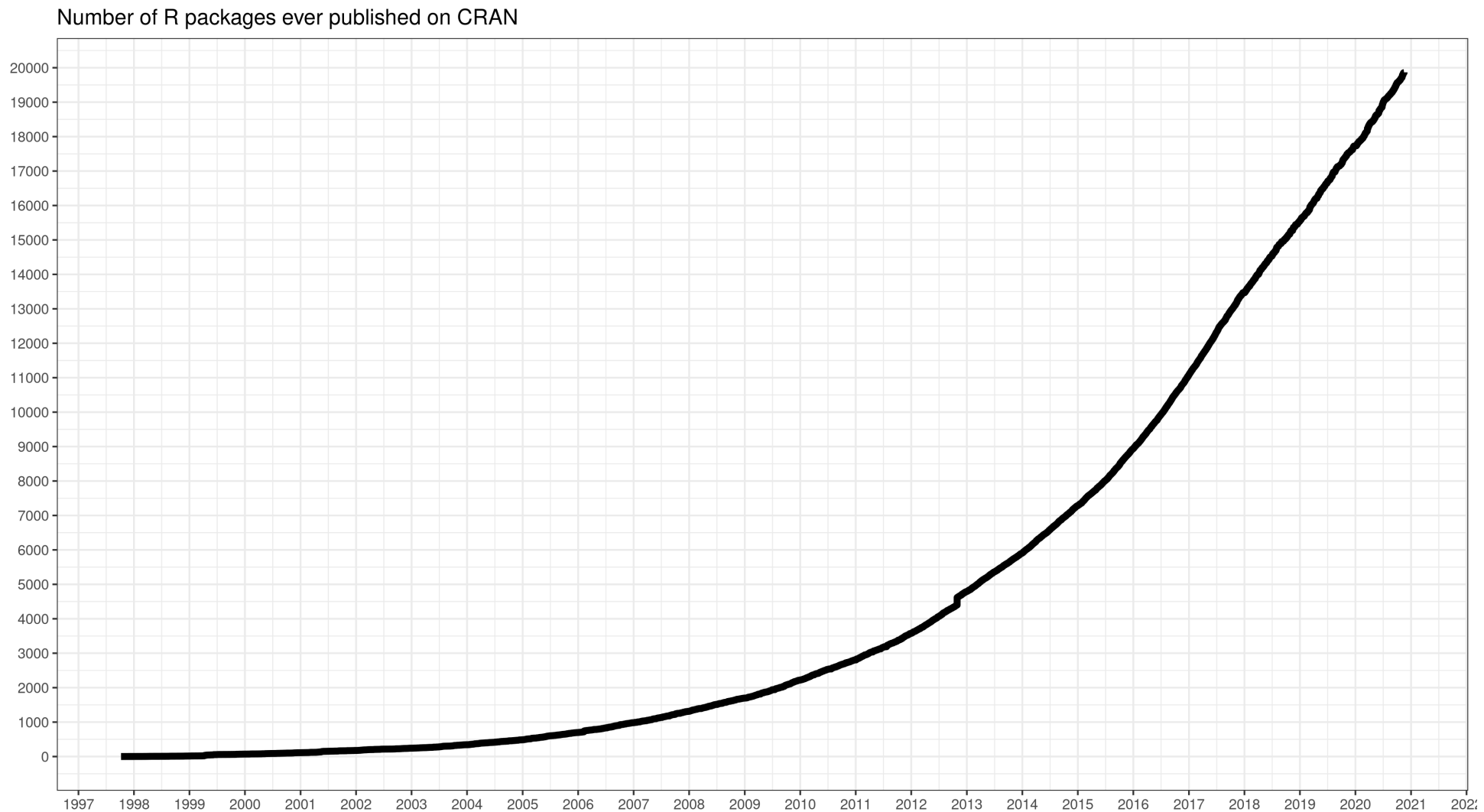
Temas de clases 1 y 2

- Clase 1:
  - Introducción a R y RStudio
  - Introducción a objetos
  - Manejo básico de bases de datos con R
  - Loops, estructuras condicionales y funciones
- Clase 2:
  - BLA

¿Qué es R?

- Lenguaje para el análisis estadístico de datos
- Software Libre
- Sintaxis incremental: paquete Base y paquetes que suben los distintos usuarios/as (universidades, empresas, etc)
- Es de código abierto
- Comunidad muy grande para realizar preguntas: RenBaires y R-Ladies Buenos Aires

Ecosistema de R

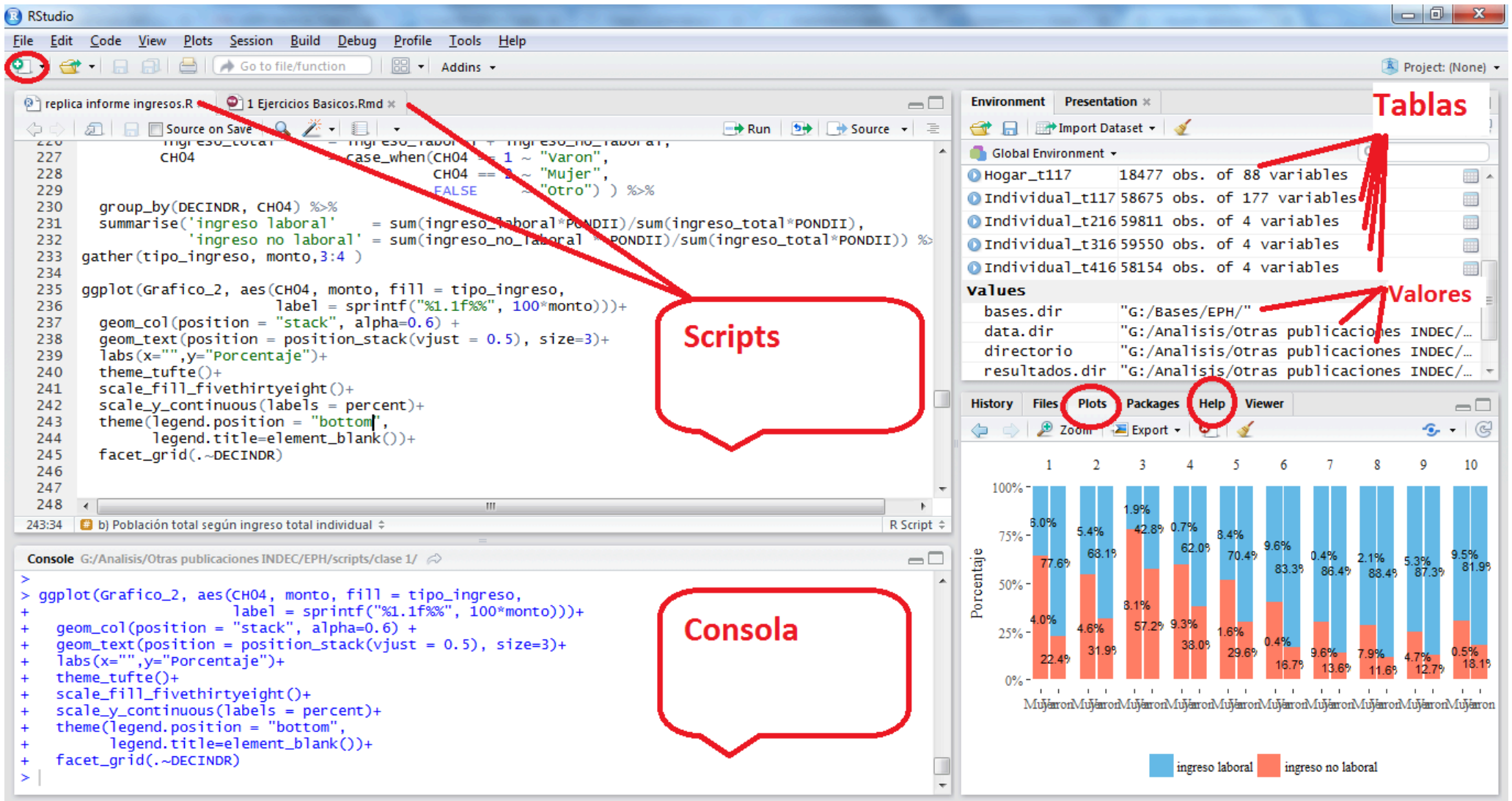


Fuente: https://gist.github.com/daroczig/3cf06d6db4be2bbe3368

¿Qué es RStudio?

- Entorno cómodo para trabajar con R
- Producto de la empresa Posit
- Gratuito
- Descargable (junto con R) desde https://posit.co/download/rstudio-desktop/

Interfaz de RStudio



Pantalla Rstudio

R como calculadora

```
1  3 + 5
```

```
[1] 8
```

Para ejecutar (correr) un comando, presionar…

- En consola: `Enter`
- En script: `Ctrl + Enter` o botón **Run** (con el cursor ubicado en el renglón)

Objetos

Objetos

- Ingrediente fundamental de R
- **“Caja”** que guarda algo adentro, por ejemplo:
  - número
  - lista de números
  - texto
  - tabla de datos
  - gráfico
  - función, etc.

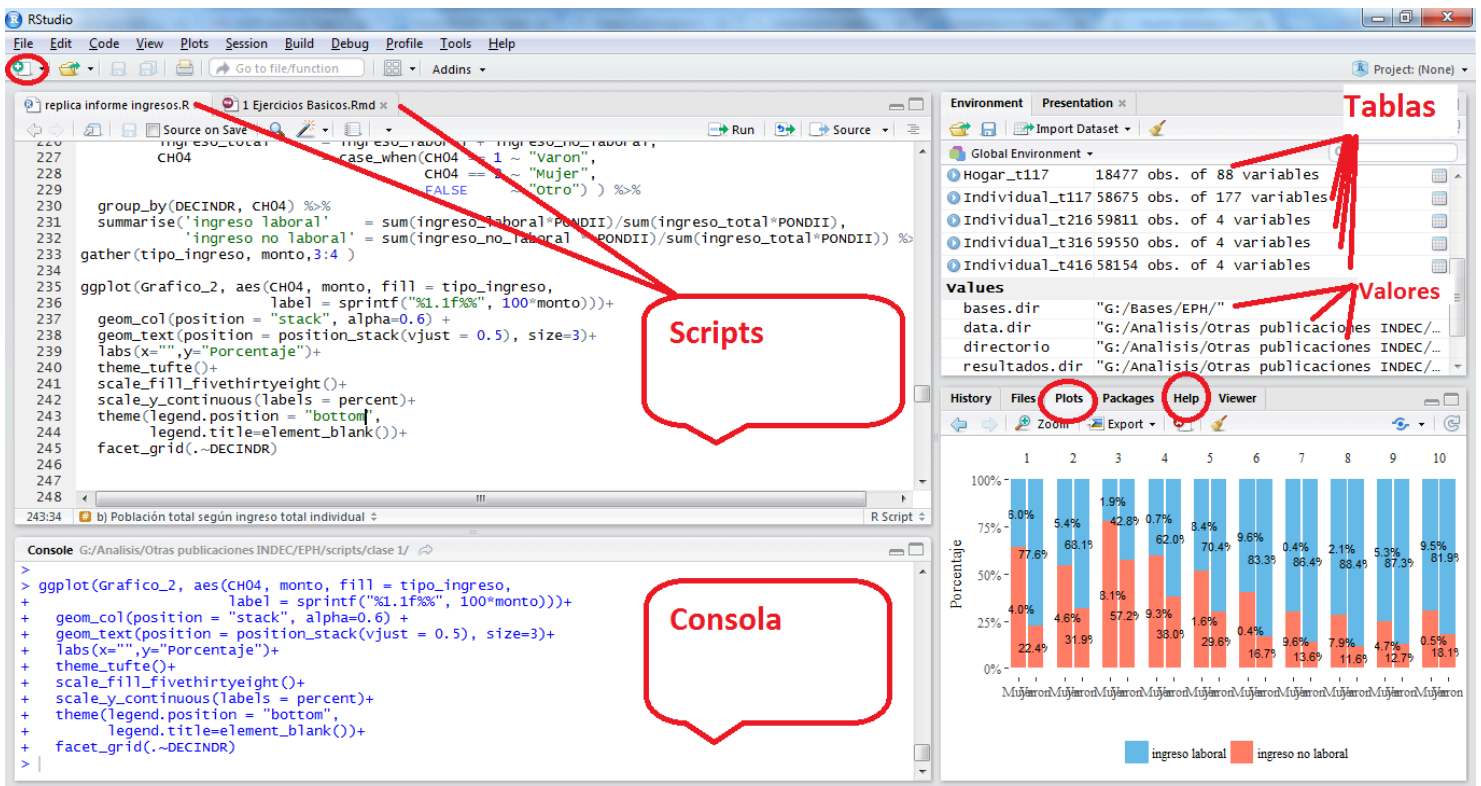
Objetos: ejemplo básico

```
1 a <- 3
2 b <- 5
3 suma <- a + b
4 suma
```

```
[1] 8
```

Environment (entorno de trabajo)

- lugar donde se guardan los objetos que creamos.
- está en el panel superior derecho de Rstudio.



Eliminar objetos del entorno de trabajo

- Con `rm()` eliminamos un objeto del entorno de trabajo (environment)

```
1  rm(suma)
2  suma
```

```
Error in eval(expr, envir, enclos): objeto 'suma' no encontrado
```

- Con `rm(list = ls())` eliminamos **todos los objetos** del entorno de trabajo.

Es común utilizar esta instrucción al comenzar una nueva sesión, para asegurarnos de no trabajar con objetos que hayan quedado de sesiones

Operadores

Operadores aritméticos

```
1 5+6 # suma
```

```
[1] 11
```

```
1 6-8 # resta
```

```
[1] -2
```

```
1 6/2.5 # división
```

```
[1] 2.4
```

```
1 6*2.5 # multiplicación
```

```
[1] 15
```

```
1 3^2 # potenciación
```

```
[1] 9
```

Operadores lógicos

Más adelante veremos que son muy útiles, por ejemplo, para filtrar datos de una base de datos.

```
1 A <- 10
2 B <- 20
3 A > B
```

```
[1] FALSE
```

```
1 A >= B
```

```
[1] FALSE
```

Operadores lógicos

```
1 A < B
```

```
[1] TRUE
```

¿A es igual a B?

```
1 A == B
```

```
[1] FALSE
```

¿A es distinto a B?

```
1 A != B
```

```
[1] TRUE
```

```
1 C <- A != B
2 C
```

```
[1] TRUE
```

Funciones

- Scripts “enlatados”
- Pueden estar
  - en paquete Base (como veremos ahora)
  - en paquetes que debemos instalar (clase que viene)
  - programadas por el usuario (en un rato)
- Toman **inputs** (*argumentos*) como entrada y devuelven un **output** (*resultado*)

Algunas funciones del paquete Base

```
1:5
```

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

```
sum(1:5)
```

```
[1] 15
```

```
mean(1:5)
```

```
[1] 3
```

```
paste("Hola", "mundo", sep = " ")
```

```
[1] "Hola mundo"
```

```
paste("Hola", "mundo", sep = "-")
```

```
[1] "Hola-mundo"
```

Caracteres especiales en R

Mayúsculas y minúsculas

R es sensible a mayúsculas y minúsculas.

```
1 a <- 3
2 A <- 5
```

```
1 a
```

```
[1] 3
```

```
1 A
```

```
[1] 5
```

a y A son objetos distintos para R.

Espacios en blanco

No son considerados por R, se utilizan para emprolijar el código.

```
1  A<-1
```

Es lo mismo que…

```
1  A <- 1
```

Pero lo segundo es más prolijo.

Espacios en blanco

```
1 paste("Hola","mundo",sep=" ")
```

```
[1] "Hola mundo"
```

Es lo mismo que...

```
1 paste("Hola", "mundo", sep = " ")
```

```
[1] "Hola mundo"
```

Pero lo segundo es más prolijo.

Mas recomendaciones de estilo en
https://style.tidyverse.org/

Numeral

Se utiliza para hacer comentarios. Todo lo que se escribe después del # no es interpretado por R.

```
1 mean(1:5) # calcula el promedio
```

```
[1] 3
```

```
1 # calcula el promedio
2 mean(1:5)
```

```
[1] 3
```

->

Tipos de objetos

En R, hay distintos tipos de objetos (estructuras de datos) que permiten organizar la información de distintas maneras. Los más básicos son:

1. Valores
2. Vectores
3. Data Frames
4. Listas

1. Valores:

Están compuestos por un único elemento.

Ejemplos:

```
A <-  1
```

```
B <- "hola"
```

Clases de objetos

Los valores (y vectores) pueden ser a su vez de distintas **clases**

1. Numeric

```
1 A <-  1
2 class(A)
```

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

Clases de objetos

Los valores (y vectores) pueden ser a su vez de distintas **clases**

2. Character

```
1 B <-  "Hola mundo"
2 B
```

```
[1] "Hola mundo"
```

```
1 class(B)
```

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

Clases de objetos

Los valores (y vectores) pueden ser a su vez de distintas **clases**

3. Logical

```
1 C <-  3 < 5
2 C
```

```
[1] TRUE
```

```
1 class(C)
```

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

Tipos de objetos

1. Valores
2. Vectores
3. Data Frames
4. Listas

Vectores

- Un vector es una *concatenación* de valores de la misma clase.
- Se crean con el comando `c()`.

```
1  x <- c(1, 3, 4)
2  x
```

```
[1] 1 3 4
```

```
1  class(x)
```

Vector de caracteres

```
1 nombres <- c("Carlos", "Federico", "Pedr
2 nombres
```

```
[1] "Carlos"   "Federico" "Pedro"
```

```
1 class(nombres)
```

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

Vector lógico

```
1 menor_a_5 <- c(3 < 5, 4 < 5, 6 < 5)
2 menor_a_5
```

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

```
1 class(menor_a_5)
```

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

Vectores (más ejemplos)

```r
x <- c(1, "hola")
x
```

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

```r
class(x)
```

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

Vectores (más ejemplos)

```r
y <- c(1, TRUE)
y
```

```
[1] 1 1
```

```r
class(y)
```

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

Vectores (más ejemplos)

```r
z <- c(TRUE, "hola")
z
```

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

```r
class(z)
```

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

Vectores (más ejemplos)

```r
w <- c(1, TRUE, "hola")
w
```

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

```r
class(w)
```

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

Suma de vectores

Podemos sumar dos vectores numéricos.

```
1  x <- c(1, 3, 4)
2  y <- c(2, 5, 3)
3  x + y
```

```
[1] 3 8 7
```

Suma de vectores

Podemos sumar el mismo valor a todos los elementos de un vector

```
1  x <- c(1, 3, 4)
2  x + 2
```

```
[1] 3 5 6
```

¿Cómo ejecuta R internamente esta operación?

Suma de vectores

```
1  x <- c(1, 3, 4)
2  x + 2
```

```
[1] 3 5 6
```

es equivalente a

```
1  x <- c(1, 3, 4)
2  x + c(2, 2, 2)
```

```
[1] 3 5 6
```

Reciclado de vectores

Para sumar vectores de distinta longitud R *recicla* el vector más corto tantas veces como sea necesario para igualar la longitud del más largo.

Reciclado de vectores (ejemplo)

```r
# sumamos dos vectores de distinta longi
x <- 1:10
y <- 1:2
```

```r
x
```

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

```r
y
```

```
[1] 1 2
```

```r
x + y
```

```
 [1]  2  4  4  6  6  8  8 10 10 12
```

Elementos de un vector

Veamos cómo acceder a un elemento de un vector.

```
1 nombres <- c("Carlos", "Federico", "Pedr
```

Accedemos al segundo elemento

```
1 nombres[2]
```

```
[1] "Federico"
```

Podemos almacenarlos en otro objeto

```
1 elemento2 <- nombres[2]
2 elemento2
```

```
[1] "Federico"
```

Elementos de un vector

Podemos cambiar el contenido de un elemento de un vector

```
1 nombres <- c("Carlos", "Federico", "Pedr
2 nombres
```

```
[1] "Carlos"   "Federico" "Pedro"
```

```
1 nombres[2] <- "Pablo"
2 nombres
```

```
[1] "Carlos" "Pablo"  "Pedro"
```

Tipos de objetos

1. Valores
2. Vectores
3. Data Frames
4. Listas

Data Frames

Un **data frame** es una tabla de datos en R donde cada *columna* representa una *variable* y cada *fila* una observación.

	FECHA	GRUPO	INDICE
1	Mar-20	Privado_Registrado	286.4
2	Mar-20	Público	262.1
3	Mar-20	Privado_No_Registrado	248.5
4	Abr-20	Privado_Registrado	285.7
5	Abr-20	Público	263.5
6	Abr-20	Privado_No_Registrado	250.2
7	May-20	Privado_Registrado	285.1
8	May-20	Público	264.9

Data frames

Los **data frames** son el tipo de objeto que usamos en R para importar y trabajar con bases de datos externas.

Leer datos en R

Funciones comunes para importar datos:

- `read.table()`
  - Paquete Base R
  - Para archivos de texto en general (`.txt`, `.csv`, `.tsv`)
- `read.csv()`
  - Paquete Base R
  - Para archivos separados por comas (`.csv`)
- `read_excel()`
  - Paquete `readxl`
  - Para archivos de Excel (`.xls`, `.xlsx`)

Importar datos con `read.table()`

Tenemos un dataset con algunos datos del índice de salarios del INDEC en un `salarios.txt` en la carpeta "./datos".

```
"FECHA" "GRUPO" "INDICE"
"Mar-20" "Privado_Registrado" 286.4
"Mar-20" "Público" 262.1
"Mar-20" "Privado_No_Registrado" 248.5
"Abr-20" "Privado_Registrado" 285.7
"Abr-20" "Público" 263.5
"Abr-20" "Privado_No_Registrado" 250.2
"May-20" "Privado_Registrado" 285.1
"May-20" "Público" 264.9
"May-20" "Privado_No_Registrado" NA
```

Importar datos con `read.table()`

```
1 Datos <- read.table("./datos/salarios.tx
```

```
1 head(Datos)
```

```
      V1                                V2      V3
1   FECHA                             GRUPO INDICE
2 Mar-20    Privado_Registrado  286.4
3 Mar-20                Público  262.1
4 Mar-20 Privado_No_Registrado  248.5
5 Abr-20    Privado_Registrado  285.7
6 Abr-20                Público  263.5
```

```
1 names(Datos)
```

```
[1] "V1" "V2" "V3"
```

Importar datos con `read.table()`

```
1 head(Datos)
```

```
      V1                                 V2      V3
1  FECHA                              GRUPO INDICE
2 Mar-20          Privado_Registrado  286.4
3 Mar-20                    Público  262.1
4 Mar-20 Privado_No_Registrado  248.5
5 Abr-20          Privado_Registrado  285.7
6 Abr-20                    Público  263.5
```

La primera fila del .txt tiene los nombres de las variables, pero R la interpretó como una observación.

Importar datos con `read.table()`

```
1 Datos <- read.table("./datos/salarios.tx
2                     header = TRUE)
```

```
1 head(Datos)
```

```
   FECHA                       GRUPO INDICE
1 Mar-20     Privado_Registrado  286.4
2 Mar-20                Público  262.1
3 Mar-20 Privado_No_Registrado  248.5
4 Abr-20     Privado_Registrado  285.7
5 Abr-20                Público  263.5
6 Abr-20 Privado_No_Registrado  250.2
```

```
1 names(Datos)
```

Elementos de un data frame

Para acceder a determinado elemento de un data frame podemos usar los corchetes `[]` como en los vectores pero especificando dos valores (`fila` y `columna`).

```
1 Datos[3, 1]
```

```
[1] "Mar-20"
```

```
1 Datos
```

```
   FECHA                      GRUPO INDICE
1 Mar-20     Privado_Registrado  286.4
2 Mar-20                Público  262.1
3 Mar-20 Privado_No_Registrado  248.5
4 Abr-20     Privado_Registrado  285.7
```

```
5 Abr-20                    Público  263.5
6 Abr-20 Privado_No_Registrado  250.2
7 May-20    Privado_Registrado  285.1
```

Fila de un data frame

Si queremos todos los elementos de una fila (ej. la fila 3)

```
1 Datos[3, ]
```

```
   FECHA                      GRUPO INDICE
3 Mar-20 Privado_No_Registrado  248.5
```

```
1 Datos
```

```
   FECHA                      GRUPO INDICE
1 Mar-20    Privado_Registrado  286.4
2 Mar-20              Público  262.1
3 Mar-20 Privado_No_Registrado  248.5
4 Abr-20    Privado_Registrado  285.7
5 Abr-20              Público  263.5
6 Abr-20 Privado_No_Registrado  250.2
```

```
7 May-20     Privado_Registrado  285.1
```

Columna de un data frame

Si queremos todos los elementos de una columna (ej. la columna 1)

```
1 Datos[, 1]
```

```
[1] "Mar-20" "Mar-20" "Mar-20" "Abr-20"
"Abr-20" "Abr-20" "May-20" "May-20"
[9] "May-20"
```

```
1 Datos
```

```
   FECHA                      GRUPO INDICE
1 Mar-20    Privado_Registrado  286.4
2 Mar-20              Público  262.1
3 Mar-20 Privado_No_Registrado  248.5
4 Abr-20    Privado_Registrado  285.7
```

```
5 Abr-20                    Público  263.5
6 Abr-20 Privado_No_Registrado  250.2
7 May-20    Privado_Registrado  285.1
8 May-20                    Público  264.0
```

Columna de un data frame

Otra forma de accedera a la primera columna

```
1 Datos$FECHA
```

```
[1] "Mar-20" "Mar-20" "Mar-20" "Abr-20"
"Abr-20" "Abr-20" "May-20" "May-20"
[9] "May-20"
```

```
1 Datos
```

```
   FECHA                      GRUPO INDICE
1 Mar-20    Privado_Registrado  286.4
2 Mar-20              Público  262.1
3 Mar-20 Privado_No_Registrado  248.5
4 Abr-20    Privado_Registrado  285.7
5 Abr-20              Público  263.5
```

```
6 Abr-20 Privado_No_Registrado  250.2
7 May-20    Privado_Registrado  285.1
```

Elemento de un data frame

Otra forma de accedera a un elemento de un data frame (ej. observación 3 de la variable FECHA)

```
1 Datos$FECHA[3]
```

```
[1] "Mar-20"
```

```
1 Datos
```

```
   FECHA                       GRUPO INDICE
1 Mar-20     Privado_Registrado  286.4
2 Mar-20                Público  262.1
3 Mar-20 Privado_No_Registrado  248.5
4 Abr-20     Privado_Registrado  285.7
5 Abr-20                Público  263.5
6 Abr-20 Privado_No_Registrado  250.2
```

```
7 May-20     Privado_Registrado  285.1
8 May-20                Público  264.0
```

Filtros en data frames

Supongamos que queremos quedarnos con todas las observaciones de abril de 2020.

```
1 Datos
```

```
   FECHA                    GRUPO INDICE
1 Mar-20     Privado_Registrado  286.4
2 Mar-20                Público  262.1
3 Mar-20 Privado_No_Registrado  248.5
4 Abr-20     Privado_Registrado  285.7
5 Abr-20                Público  263.5
6 Abr-20 Privado_No_Registrado  250.2
7 May-20     Privado_Registrado  285.1
8 May-20                Público  264.9
```

Filtros en data frames

```
1 Datos[4:6, ]
```

```
   FECHA                      GRUPO INDICE
4 Abr-20     Privado_Registrado  285.7
5 Abr-20               Público  263.5
6 Abr-20 Privado_No_Registrado  250.2
```

Filtros con condiciones lógicas

```
1 Datos[Datos$FECHA == "Abr-20", ]
```

```
   FECHA                          GRUPO INDICE
4 Abr-20     Privado_Registrado   285.7
5 Abr-20                Público   263.5
6 Abr-20 Privado_No_Registrado   250.2
```

Veamos por qué funciona

Filtros con condiciones lógicas

Veamos qué realiza este comando

```
1 Datos$FECHA == "Abr-20"
```

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

```
1 Datos
```

```
   FECHA                    GRUPO INDICE
1 Mar-20     Privado_Registrado  286.4
2 Mar-20                Público  262.1
3 Mar-20 Privado_No_Registrado  248.5
4 Abr-20     Privado_Registrado  285.7
5 Abr-20                Público  263.5
6 Abr-20 Privado_No_Registrado  250.2
```

```
7 May-20     Privado_Registrado  285.1
```

Datos faltantes

```
1 tail(Datos)
```

```
   FECHA                      GRUPO INDICE
4 Abr-20     Privado_Registrado  285.7
5 Abr-20               Público  263.5
6 Abr-20 Privado_No_Registrado  250.2
7 May-20     Privado_Registrado  285.1
8 May-20               Público  264.9
9 May-20 Privado_No_Registrado     NA
```

"NA" (Not Available) es la forma en que R representa los datos faltantes

Eliminar datos faltantes

Queremos quedarnos con un data frame que tenga únicamente los registros sin datos faltantes en `INIDCE`.

Para ello debemos usar la función `is.na()`

```
1 is.na(Datos$INDICE)
```

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

Para quedarnos con los datos distintos de NA, necesitamos un vector que tenga `TRUE` si el valor de INDICE **No** es NA.

Para eso, usamos el operador `!` al principio para negar is.na() y obtener TRUE cuando el valor **No** es faltante.

```
1 !is.na(Datos$INDICE)
```

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

Eliminar datos faltantes

```
1 Datos[!is.na(Datos$INDICE), ]
```

```
   FECHA                        GRUPO INDICE
1 Mar-20     Privado_Registrado  286.4
2 Mar-20                Público  262.1
3 Mar-20 Privado_No_Registrado  248.5
4 Abr-20     Privado_Registrado  285.7
5 Abr-20                Público  263.5
6 Abr-20 Privado_No_Registrado  250.2
7 May-20     Privado_Registrado  285.1
8 May-20                Público  264.9
```

Manejo de datos faltantes

Podemos aplicar el filtro sobre una columna

```
1 Datos$INDICE[Datos$FECHA == "Abr-20"]
```

```
[1] 285.7 263.5 250.2
```

```
1 Datos
```

```
   FECHA                      GRUPO INDICE
1 Mar-20    Privado_Registrado  286.4
2 Mar-20               Público  262.1
3 Mar-20 Privado_No_Registrado  248.5
4 Abr-20    Privado_Registrado  285.7
5 Abr-20               Público  263.5
6 Abr-20 Privado_No_Registrado  250.2
7 May-20    Privado_Registrado  285.1
```

Filtros en data frames

Calculamos la media de los índices de abril de 2020

```
1 mean(Datos$INDICE[Datos$FECHA == "Abr-20
```

```
[1] 266.4667
```

o la de los índices de mayo de 2020

```
1 mean(Datos$INDICE[Datos$FECHA == "May-20
```

```
[1] NA
```

Devuelve `NA` porque uno de los valores del vector que está promediando es `NA`. Si queremos calcular el promedio de los valores no faltantes...

```
1 mean(Datos$INDICE[Datos$FECHA == "May-20
2       na.rm = TRUE)
```

Tipos de objetos

1. Valores
2. Vectores
3. Data Frames
4. Listas

Listas

1. Valores
2. Vectores: concatenación de valores de la misma clase
3. Data Frames: concatenación de vectores
4. Listas: concatenación de objetos de cualquier tipo

Listas (ejemplo)

```
1 mi_lista <- list(nombre = "Lucía",
2                   numeros = 1:10,
3                   indices_df = Datos)
4 mi_lista
```

```
$nombre
[1] "Lucía"

$numeros
 [1]  1  2  3  4  5  6  7  8  9 10

$indices_df
   FECHA                      GRUPO INDICE
1 Mar 20     Privado Registrado  286.4
```

Elemento de una lista

Accedemos al 2do elemento

```
1 mi_lista[2]
```

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

¿Qué tipo de objeto devuelve?

```
1 class(mi_lista[2])
```

```
[1] "list"
```

Elemento de una lista

Accedemos al 2do elemento

```
1 mi_lista[[2]]
```

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

¿Qué tipo de objeto devuelve?

```
1 class(mi_lista[[2]])
```

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

Elemento de una lista

Accedemos al 2do elemento

```
1 mi_lista$numeros
```

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

¿Qué tipo de objeto devuelve?

```
1 class(mi_lista$numeros)
```

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

Elemento de una lista

Accedemos al 2do elemento

```
1 mi_lista[["numeros"]]
```

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

¿Qué tipo de objeto devuelve?

```
1 class(mi_lista[["numeros"]])
```

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

Loops, estructuras condicionales y funciones

¿Qué es un loop?

Loop (o bucle): estructura que nos permite **repetir una misma acción varias veces**, variando el valor de una variable en cada repetición.

Ejemplo: imprimimos el cuadrado de los números del 1 al 5.

```
1  for(i in 1:5){
2    print(i^2)
3  }
```

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

Otro ejemplo de loop y buenas prácticas

```
1 for(valor in 1:5){
2   print(valor^2)
3 }
```

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

Podemos usar el nombre que querramos para la variable del `for`.

Estructuras condicionales

- permiten ejecutar diferentes instrucciones según si se cumple o no una condición lógica.

- Comandos principales: `if`, `else` y `else if`.

Comando `if`

Sintaxis básica:

```
if (condición) {
  # código a ejecutar si la cond. es ver
}
```

Ejemplo:

```
x <- 5

if (x > 0) {
  print("x es positivo")
}
```

```
[1] "x es positivo"
```

Comando `if` (condición no cumplida)

```r
x <- -3

if (x > 0) {
  print("x es positivo")
}
```

Comando `else`

```r
x <- 5

if (x > 0) {
  print("x es positivo")
} else {
  print("x no es positivo")
}
```

```
[1] "x es positivo"
```

Comandos `if` y `else`

```
1 x <- 0
2
3 if (x > 0) {
4   print("x es positivo")
5 } else {
6   print("x no es positivo")
7 }
```

```
[1] "x no es positivo"
```

Comando `else if`

```r
x <- 0

if (x > 0) {
  print("x es positivo")
} else if (x == 0) {
  print("x es cero")
} else {
  print("x es negativo")
}
```

```
[1] "x es cero"
```

Comando `else if`

```r
x <- -3

if (x > 0) {
  print("x es positivo")
} else if (x == 0) {
  print("x es cero")
} else {
  print("x es negativo")
}
```

```
[1] "x es negativo"
```

Funciones

- Función: script “enlatado”
  - Inputs: argumentos
  - Output: resultado

- Pueden estar
  - en paquete Base (como ya vimos)
  - en paquetes que debemos instalar (clase que viene)
  - programadas por el usuario (en un rato)

Funciones en paquete Base (repaso)

```r
x <- c(1, 3, 2)
sum(x)
```

```
[1] 6
```

```r
sum(c(1, 3, 2))
```

```
[1] 6
```

```r
x <- c(1, 3, 2)
mean(x)
```

```
[1] 2
```

```r
paste("Hola", "mundo", sep = " ")
```

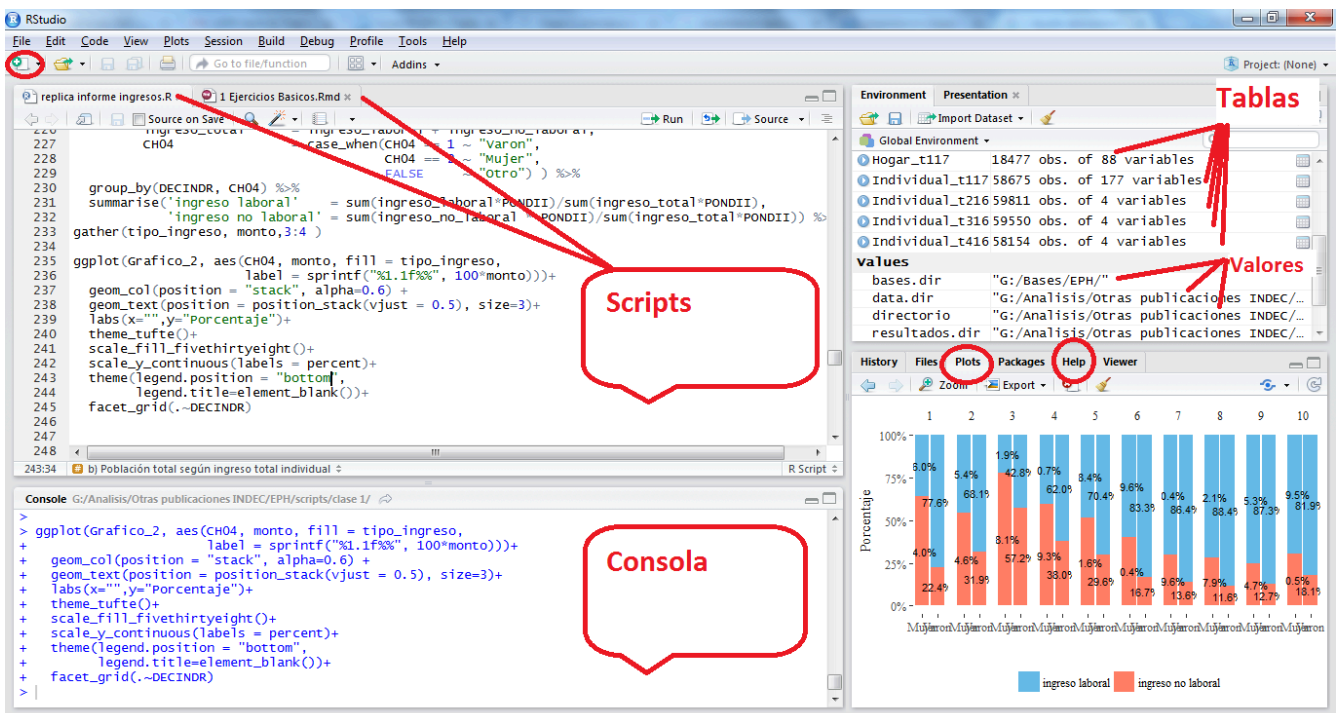
```
[1] "Hola mundo"
```

Help

Podemos ver la **ayuda** de una función con

```
?nombre_función
```

o en la pestaña “Help” del panel inferior derecho.



Pantalla Rstudio

Crear nuestra propia función

Podemos **definir nuestras propias funciones** con la función `function()`:

```
1 cuadrado <- function(x) {
2   return(x^2)
3 }
4
5 cuadrado(5)
```

```
[1] 25
```

`return()` no es obligatorio, pero hace el código más claro.

Función sin return

```
1 cuadrado <- function(x) {
2   x^2
3 }
4
5 cuadrado(5)
```

```
[1] 25
```

Otra función sin return

```
1 polinomio <- function(x) {
2    y <- x + 2
3    y^2
4 }
5
6 polinomio(5)
```

```
[1] 49
```

Otra función sin return

```
1 polinomio <- function(x) {
2    y <- x + 2
3    return(y^2)
4 }
5
6 polinomio(5)
```

```
[1] 49
```

`return()` hace el código más claro especialmente cuando hay más de una línea de código en la función.

Funciones en Environment

Las funciones que creamos nosotros permanecen en el entorno de trabajo de R (Environment) temporariamente.

Para poder usar una funcion primero debemos ejecutarla, para que quede guardada en el entorno de trabajo y R la pueda usar.

Funciones con múltiples argumentos

Podemos definir funciones que tomen más de un argumento.

```
1 potencia <- function(base, exponente) {
2   return(base ^ exponente)
3 }
4
5 potencia(3, 4)
```

```
[1] 81
```

Funciones con múltiples argumentos

Podemos aclarar explícitamente qué valor toma cada argumento

```
1 potencia <- function(base, exponente) {
2   return(base ^ exponente)
3 }
4
5 potencia(base = 3, exponente = 4)
```

```
[1] 81
```

Funciones con múltiples argumentos

En ese caso, podemos cambiar el orden de los argumentos

```
1 potencia <- function(base, exponente) {
2   return(base ^ exponente)
3 }
4
5 potencia(exponente = 4, base = 3)
```

```
[1] 81
```

Funciones con múltiples argumentos

pero, ¿qué pasa si cambiamos el orden de los argumentos sin nombrarlos?

```
1 potencia <- function(base, exponente) {
2   return(base ^ exponente)
3 }
4
5 potencia(4, 3)
```

```
[1] 64
```

Valores por defecto

También podemos establecer **valores por defecto** para que no siempre sea necesario especificarlos.

```
potencia <- function(base, exponente = 2
  return(base ^ exponente)
}

potencia(3, 4)      # Usa exponente = 4
```

```
[1] 81
```

Valores por defecto

```
1 potencia <- function(base, exponente = 2
2   return(base ^ exponente)
3 }
4
5 potencia(3) # Usa exponente = 2 por defe
```

```
[1] 9
```