Repaso de R

Economía Laboral

Miriam Malament
UCEMA

Objetos y Operadores

Los **Objetos/Elementos** constituyen la categoría escencial del R. De hecho, todo en R es un objeto, y se almacena con un nombre específico que **no debe poseer espacios**. Un número, un vector, una función, la progresión de letras del abecedario, una base de datos, un gráfico, constituyen para R objetos de distinto tipo. Los objetos que vamos creando a medida que trabajamos pueden visualizarse en la panel derecho superior de la pantalla ("Environment").

Los **Operadores** son los símbolos que le indican a R que debe realizar una tarea. Mediante la combinación de datos y operadores es que logramos que R haga su trabajo.

Existen operadores específicos para cada tipo de tarea. Los tipos de operadores principales son los siguientes:

- De asignación (<- ó =)
- Aritméticos (+, -, *, /)
- Lógicos Relacionales (>, <, >=, <=, ==, !=)

Operadores de Asignación

Los operadores de asignación:

- $<-$
- $=$

Ambos operadores sirven para definir objetos, es decir, asignarle un valor. Sin embargo, en la práctica se suele utilizar el operador `←` para la definición de objetos, por más que el `=` sea equivalente. **A la izquierda** del `←` debe ubicarse el nombre que tomará el elemento a crear. **Del lado derecho** debe ir la definición del mismo.

Veamos un ejemplo:

```
A ← 1
A
```

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

`←` es un operador **Unidireccional**, es decir que: `A ← B` implica que **A** va tomar como valor el contenido del objeto **B**, y no al revés.

Operadores Aritméticos

- Suma: $+$
- Resta: $-$
- Cociente: $/$
- Multiplicación: $*$

```
#suma
A ← 5+6
A
#Resta
B ← 6-8
B
#cociente
C ← 6/2
C
#multiplicacion
D ← 6*2
D
```

Operadores Lógicos Relacionales

Los operadores lógicos son usados para describir relaciones, expresadas como verdadero (TRUE) o falso (FALSE).

- Mayor: $>$
- Mayor o igual: $>=$
- Menor: $<$
- Menor o igual: $<=$
- Igual: $==$
- Distinto: $!=$

Operadores Lógicos Relacionales

Ejemplos

```
#Redefinimos los valores A y B
A ← 10
B ← 20
#Realizamos comparaciones lógicas
A > B
```

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

```
A ⩾ B
```

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

```
A < B
```

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

```
A ⩽ B
```

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

Operadores Lógicos Relacionales

Otros operadores lógicos:

```
#Realizamos comparaciones lógicas
A == B
```

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

```
A != B
```

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

```
(A | B) > 30
```

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

```
(A & B) <= 30
```

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

Caracteres especiales

- R es sensible a mayúsculas y minúsculas, tanto para los nombres de las variables, como para las funciones y parámetros.
- Los **espacios en blanco** y los **carriage return** (*enter*) no son considerados por el lenguaje. Los podemos aprovechar para emprolijar el código y que la lectura sea más simple[^1].

- El **numeral** `#` se utiliza para hacer comentarios. Todo lo que se escribe después del # no es interpretado por R. Se debe utilizar un # por cada línea de código que se desea anular

- Los **corchetes** `[]` se utilizan para acceder a un objeto:

- el signo **\$** también es un método de acceso. Particularmente, en los dataframes, nos permitira acceder a una determinada columna de una tabla

- Los **paréntesis** `()` se utilizan en las funciones para definir los parámetros.

- Las **comas** `,` se utilizan para separar los parametros al interior de una función.

Tipos de Objetos

Existen un gran cantidad de objetos distintos en R, en lo que respecta al curso trabajaremos principalmente con 4 de ellos:

- Valores
- Vectores
- Data Frames
- Listas

Valores

Los valores pueden ser de distintas *clases*:\ **Valores "Numeric" (numéricos)**

```
A ← 1
class(A)
```

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

```
A ← "Hola"
class(A)
```

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

```
A ← "Hola, ¿qué tal?"
class(A)
```

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

```
A ← paste('Hola,', '¿qué tal?', sep = " ")
class(A)
```

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

Valores

Valores de tipo 'factor'

En general, es un dato numérico representado por una etiqueta.

```
valor1 ← "A"
class(valor1)
```

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

```
valor2 ← factor("A")
valor2
```

```
## [1] A
## Levels: A
```

```
# Otra forma de definir al factor
valor2 ← factor(valor1)
valor2
```

```
## [1] A
## Levels: A
```

Vectores

Un vector es un conjunto de datos de un mismo tipo. En otras palabras, es un conjunto de valores de la misma clase. Puede haber vectores númericos, character, factores. Los vectores constituyen la estructura de datos más sencilla de R.

Para crear un **vector** utilizamos el comando `c()`, de combinar.

```
# Vector numérico
A ← c(1, 2, 2, 2, 1, 1, 1)
A
```

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

```
class(A)
```

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

Vectores

```
# Vector de caracteres
B ← c("Uno", "Dos", "Dos", "Dos", "Uno", "Uno", "Uno")
B
```

```
## [1] "Uno" "Dos" "Dos" "Dos" "Uno" "Uno" "Uno"
```

```
class(B)
```

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

```
# Vector de tipo factor
C ← as.factor(A)
C
```

```
## [1] 1 2 2 2 1 1 1
## Levels: 1 2
```

```
class(C)
```

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

Vectores

Con los vectores numéricos se pueden hacer operaciones como, por ejemplo: * sumarle 2 a cada elemento del **vector** anterior.

```
D ← c(1, 3, 4)
D ← D + 2
D
```

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

- sumarle 1 al primer elemento, 2 al segundo, y 3 al tercer elemento del **vector** anterior

```
E ← D + 1:3 #esto es equivalente a hacer 3+1, 5+2, 6+9
E
```

```
## [1] 4 7 9
```

`1:3` significa que queremos todos los números enteros desde 1 hasta 3.

Vectores

En R podemos quedarnos con algunos elemenos de los vectores para trabajar con o sobre ellos. Para acceder a un elemento del vector podemos buscarlo a través del número de orden, identificando al mismo utilizando los signos `[ ]`

```
# Si quiero al elemento 2 del objeto E:
E
```

```
## [1] 4 7 9
```

```
E[2]
```

```
## [1] 7
```

Si nos interesa quedarnos con dicho valor, al buscarlo lo asignamos a un nuevo objeto.

```
E_posicion2 ← E[2]
E_posicion2
```

```
## [1] 7
```

Para **borrar** un objeto del ambiente de trabajo, utilizamos la función `rm()`

Vectores

También podemos cambiar el texto del segundo elemento de E, por el texto "Pablo"

```
E
```

```
## [1] 4 7 9
```

```
E[2] ← "Pablo"
E
```

```
## [1] "4"     "Pablo" "9"
```

```
# Tener cuidado al modificar el tipo de uno de los valores y no el de todos los del ob
class(E)
```

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

Data Frames

Un Data Frame es una estructura de datos de 2 dimensiones o tabla, donde cada columna representa una variable, y cada fila una observación. Los data frames pueden contener datos de diferentes clases. Puede ser considerado como un conjunto de vectores de igual tamaño, donde cada vector (columna) tiene que tener datos del mismo tipo, pero las clases de vectores que conforman la tabla pueden ser distintas. Entonces, cada observación (fila) está compuesta por datos que pueden ser de distinto tipo.

Este objeto es central en el proceso de trabajo, ya que es la estructura más usada para realizar análisis de datos, y suele ser la forma en que se cargan datos externos para trabajar en el ambiente de R, y en que se exportan los resultados de nuestros trabajo. Veamos un ejemplo de data frame creado a partir de la combinación de vectores:

```
AGLOMERADO  ← c(32,33,33,33,32)

SEXO  ←  c("Varon","Mujer","Mujer","Varon","Mujer")

EDAD  ←  c(60,54,18,27,32)

Datos ← data.frame(AGLOMERADO, SEXO, EDAD)
```

Data Frames

```
Datos
```

```
##   AGLOMERADO  SEXO EDAD
## 1         32 Varon   60
## 2         33 Mujer   54
## 3         33 Mujer   18
## 4         33 Varon   27
## 5         32 Mujer   32
```

```
class(Datos)
```

```
## [1] "data.frame"
```

Tal como en un **vector** podemos acceder a los elementos a través de los `[ ]`, en un **dataframe** lo hacemos de la forma **`[fila, columna]`**.

En los Data.Frames tenemos, por definición, más de una columna (variable). Para acceder a alguna de ellas podemos utilizar el operador **`$`**.

Data Frames

```
Datos[3,2]
```

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

```
Datos[4,3]
```

```
## [1] 27
```

```
Datos$AGLOMERADO
```

```
## [1] 32 33 33 33 32
```

```
class(Datos$AGLOMERADO)
```

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

```
Datos$AGLOMERADO[2]
```

```
## [1] 33
```

Data Frames

Acorde a lo visto anteriormente, el acceso a los **dataframes** mediante `[ ]` puede utilizarse también para realizar filtros (devolver el o los valores en función de otro valor definido). Por ejemplo, puedo utilizar los `[ ]` para obtener del **dataframe** `Datos` unicamente los registros del AGLOMERADO 32:

```
Datos[Datos$AGLOMERADO==32, ]
```

```
##   AGLOMERADO  SEXO EDAD
## 1         32 Varon   60
## 5         32 Mujer   32
```

La lógica del paso anterior sería: Accedo al dataframe `Datos`, pidiendo únicamente conservar las filas (por eso la condición se ubica a la *izquierda* de la `,`) que cumplan el requisito de pertenecer a la categoría **32** de la variable **AGLOMERADO**.

Aún más, podría aplicar el filtro y al mismo tiempo identificar una variable de interés para luego realizar un cálculo sobre aquella. Por ejemplo, podría calcular la media de la edad para aquellas personas que residen en el aglomerado 32.

Data Frames

```
###Por separado
Edad_Aglo32 ← Datos$EDAD[Datos$AGLOMERADO==32]
Edad_Aglo32
```

```
## [1] 60 32
```

```
mean(Edad_Aglo32)
```

```
## [1] 46
```

```
# Otra forma de lograr el mismo resultado
Edad_Aglo32 ← mean(Datos$EDAD[Datos$AGLOMERADO==32])
```

La lógica de esta sintaxis sería: "Me quedó con la variable **EDAD**, cuando la variable AGLOMERADO sea igual a **32**, luego calculo la media de dichos valores"

Listas

Contienen una concatenación de objetos de cualquier tipo. Así como un vector contiene valores, un dataframe contiene vectores, una lista puede contener dataframes, pero también vectores, o valores, y *todo ello a la vez*

```
LISTA ← list(A,B,C,D,E,Datos$AGLOMERADO, DF = Datos)
LISTA
```

```
## [[1]]
## [1] 1 2 2 2 1 1 1
##
## [[2]]
## [1] "Uno" "Dos" "Dos" "Dos" "Uno" "Uno" "Uno"
##
## [[3]]
## [1] 1 2 2 2 1 1 1
## Levels: 1 2
##
## [[4]]
## [1] 3 5 6
##
## [[5]]
## [1] "4"     "Pablo" "9"
##
```

Listas

Tal como para con los Vectores y los Data.Frames, podemos acceder a un elemento de una lista, utilizando el operador `$`:

```
LISTA$DF
```

```
##   AGLOMERADO  SEXO EDAD
## 1         32 Varon   60
## 2         33 Mujer   54
## 3         33 Mujer   18
## 4         33 Varon   27
## 5         32 Mujer   32
```

```
LISTA$DF$EDAD
```

```
## [1] 60 54 18 27 32
```

```
LISTA$DF$EDAD[2]
```

```
## [1] 54
```

Listas

También se pueden usar corchetes dobles `[[]]` para acceder a los distintos elementos de una lista.

```
LISTA[[6]]
```

```
## [1] 32 33 33 33 32
```

O para acceder a un valor/vector (dependiendo del tipo de elemento de la lista) de un objeto de la lista:

```
LISTA[[6]][1]
```

```
## [1] 32
```

```
LISTA[[7]][2]
```

```
##    SEXO
## 1 Varon
## 2 Mujer
## 3 Mujer
## 4 Varon
## 5 Mujer
```

Funciones básicas

Las funciones son series de procedimientos estandarizados, que toman como imput determinados argumentos a fijar por el usuario (llamados parámetros), y devuelven un resultado acorde a la aplicación de dichos procedimientos. Su lógica de funcionamiento es:\

```
funcion(argumento1 = arg1, argumento2 = arg2)
```

A lo largo del curso iremos viendo numerosas funciones, según lo requieran los distintos ejercicios. Sin embargo, veamos ahora algunos ejemplos para comprender su funcionamiento:

- paste() : concatena una serie de caracteres, indicando por última instancia como separar a cada uno de ellos\
- paste0(): concatena una serie de caracteres sin separar
- sum(): suma de todos los elementos de un vector\
- mean() promedio aritmético de todos los elementos de un vector\

Funciones básicas

```
paste("Pega","estas", 4, "palabras", sep = " ")
```

```
## [1] "Pega estas 4 palabras"
```

```
#Puedo concatenar caracteres almacenados en objetos
a ← c(1, 2, 3)
b ← "con"
c ← c(4, 5, 6)

paste(a,b,c,sep = "-")
```

```
## [1] "1-con-4" "2-con-5" "3-con-6"
```

```
# Paste0 pega los caracteres sin separador
paste0(a,b,c)
```

```
## [1] "1con4" "2con5" "3con6"
```

```
# Para calcular medias
mean(1:5)
```

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

Instalación de paquetes

Hasta aquí hemos visto múltiples funciones que están contenidas dentro del lenguaje básico de R. Ahora bien, al tratarse de un software libre, distintos usuarios de R contribuyen sistemáticamente a expandir este lenguaje mediante la creación y actualización de **paquetes** complementarios. Lógicamente, los mismos no están incluidos en la instalación inicial del programa, pero podemos descargarlos e instalarlos con el siguiente comando: **install.packages("nombre_del_paquete")**

Al ejecutar el comando se descargarán de la pagina de CRAN los archivos correspondientes al paquete hacia el directorio en donde hayamos instalado el programa.

Los paquetes sólo se instalan una vez en la computadora (si cambias de computadora, tenés que volver a instalarlo). Una vez instalado el paquete, cada vez que abramos una nueva sesión de R y querramos utilizar el mismo debemos **cargarlo al ambiente de trabajo** mediante la siguiente función: **library(nombre_del_paquete)**

Ambientes de trabajo

Hay algunas cosas que tenemos que tener en cuenta respecto del orden del ambiente en el que trabajamos:

- **Working Directory**: El directorio de trabajo, pueden ver el suyo con `getwd()`, es *hacia donde apunta el código*, por ejemplo, si quieren leer un archivo, la ruta del archivo tiene que estar explicitada como el recorrido desde el Working Directory.

- **Environment**: Esto engloba tanto la información que tenemos cargada en *Data* y *Values*, como las librerías que hemos "convocado" para trabajar.

- **Proyectos**: Rstudio tiene una herramienta muy útil de trabajo que son los proyectos. Estos permiten mantener un ambiente de trabajo delimitado por cada uno de nuestros trabajos. Un proyecto no es un sólo script, sino toda una carpeta de trabajo.

Para crearlo, vamos al logo de nuevo projecto (Arriba a la izquierda de la panatalla), y elegimos la carpeta de trabajo.

Nota: también recuerden que es importante ir actualizando R, Rstudio y los paquetes.

Paquetes que utilizaremos

Para quienes esten trabajando con sus computadoras personales, dejamos a continuación un listado de los paquetes complementarios del R base que más utilizaremos a lo largo del curso.

- EPH (sí, hay un paquete específicamente diseñado para trabajar la EPH)
- Tidyverse
- Ggplot2
- Gtables
- data.table
- readxl
- writexl

A medida que avancemos con el curso, iremos explorando nuevos paquetes.