Guía 1

Laboratorio de datos 2023 (comisión: G. Solovey)

1. Objetos

El objetivo de estos ejercicios es reconocer, crear y hacer operaciones simples con los 5 tipos de objetos básicos de R:

- character
- numeric (números *reales*)
- integer
- complex
- logical (True/False)

También familiarizarse con la consola de R, con matrices y con operaciones lógicas.

1.1 Realizar las siguientes operaciones básicas en la consola

- `2+2`
- `a <- 2`
- `b <- 5`
- `a**3`
- `16 %% 5` (probar con otros números para entender qué significa)
- `2*a**2+0.5*b+(a+b)/2`
- asignar el resultado anterior a la variable `c`
- imprimir el contenido de `c` en la consola (corriendo `c` o `print(c)`). ¿Ven un `[1]` delante del valor de `c`? Esto indica que es un vector.

1.2 Interpretar las siguientes operaciones lógicas y predecir el resultado antes de probar en la consola.

- `a <- TRUE`
- `b <- FALSE`
- `a == b`
- `a || b`
- `a == !b`

1.3. Antes de probar en la consola, piense que van a dar estas operaciones:

- `a <- 3`
- `b <- 4`
- `a > b`
- `a <= b`
- `a != b`
- `x <- 2`
- `(x > a) && (10*x>b) || !(b/a>x)`

1.4 Los vectores en R se crean con la función `c()`. Por ejemplo `c(2,-1)` es un vector de clase `numeric` que tiene dos elementos, el 2 y el -1. Se puede comprobar usando la función `class()` así: `class(c(2,-1))`. Crear 5 vectores de 2 componentes de las siguiente clases básicas: character, numeric, integer, complex y logical (True/False).

1.5. ¿Qué sucede cuando se realizan las siguientes operaciones que mezclan variables de distinta clase? ¿Por qué les parece que pasa? (usar `class()` para investigar el tipo de dato de los objetos)

- `num_char <- c(1, 2, 3, "a")`
- `num_logical <- c(1, 2, 3, TRUE)`
- `char_logical <- c("a", "b", "c", TRUE)`
- `tramposo <- c(1, 2, 3, "4")`

1.6. Realizar las siguientes operaciones en la consola, ver el output de cada operación y escribir una frase explicando qué hace cada una.

- `alturas_cm <- c(180, 178, 154, 202)`
- `frutas <- c("banana", "pera", "durazno")`
- `length(alturas_cm)`
- `length(frutas)`
- `frutas <- c(frutas, "uva")`
- `frutas <- c("naranja", frutas)`
- `bajitos <- (alturas_cm < 165)`
- `class(bajitos)`
- `1:10`
- `21:30`

1.7. Explorar estas distintas formas de extraer información del vector `c("banana", "pera", "durazno", "anana")`. Primero crear el vector en la consola y asignarlo al vector `frutas`. Luego correr cada caso en la consola y explicar en pocas palabras qué es lo que sucede.

- `frutas[2]`
- `frutas[c(1,4)]`
- `frutas[c(1,4,1,1,2)]`
- `frutas %in% c("ciruela", "frutilla", "pera", "mandarina")`
- `frutas[ frutas %in% c("ciruela", "frutilla", "pera", "mandarina") ]`

1.8. ¿Por qué da error la línea 4 de esta secuencia?

```
datos <- c("banana", 2, "pera", 5, "durazno", 0)
datos[2]
datos[4]
datos[2] + datos[4]
```

1.9. Explorar distintas formas de extraer información del vector `alturas_cm` definido en el ejercicio 1.6. Escribir una frase explicando qué hace cada una. ¿Por qué la última retorna `numeric(0)`?

- `alturas_cm[c(TRUE, FALSE, TRUE, TRUE)]`
- `alturas_cm[alturas_cm >= 180]`
- `alturas_cm[alturas_cm >= 180 & alturas_cm < 200]`
- `alturas_cm[alturas_cm >= 160 & alturas_cm == 195]`

1.10. Las matrices en R son vectores con estructura en 2D. Por ejemplo, se pueden crear de esta forma

```
m <- matrix(1:6, nrow = 2, ncol = 3)
```

Usar `dim(m)` para averiguar el atributo “dimesión” (número de filas y columnas) de la matriz. Obtener el elemento que está en la fila 2 y columna 1 usando corchetes: `m[fila,columna]`. Obtener la primera fila completa dejando en blanco la columna: `m[fila,]`.

1.11 ¿Qué sucede con `m` si corremos en la consola la operación `dim(m) <- c(1, 6)`?

1.12 Construir una matriz de 2 columnas que en cada columna tenga los vectores `x` e `y` de abajo usando la función `cbind()` y otro que haga lo mismo pero con 2 filas usando la función `rbind()`:

- `x <- c("fruta", "asado", "bebida")`
- `y <- c("si", "no", "si")`

2. Funciones

Las funciones son scripts de R "enlatados" que hacen una operación con un input (un número o una variable, por ejemplo) y devuelven un output. Los inputs van entre paréntedis y separados por una coma, si hay más de uno. Muchas funciones están disponibles en `R-base` cuando abrimos una sesión de `R`, `class()`, `length()` o `c()` que usamos en los ejercicios anteriores. El objetivo de estos ejercicios es familiarizarse con varias funciones básicas de R.

2.1. Ejecutar estas operaciones en la consola para entender qué hacen las funciones `sqrt()`, `round()`, `args()`, `ceiling()` y `floor()`:

- `sqrt(81)`
- `round(3.14159, digits = 3)`
- `round(3.14159)`
- `round(digits = 2, 3.14159)`
- `args(sample)`
- `?ceiling` y `?floor`

2.2 Explorar las funciones `max()`, `min()`, `mean()` y `sort()` aplicadas al vector `alturas_cm` del ejercicio 1.6. ¿Qué hace cada una?

2.3 Usar las funciones `which.min()` y `which.max()` con el mismo vector de alturas. Interpretar qué hace cada una.

2.4. Completar los blancos con distintos números para entender qué hacen las funciones `seq()` y `rep()`. Si hay dudas, `?rep` y `?seq` en la consola.

- `seq(from = ___, to = ___, lenght.out = ___)`
- `rep(x = ___, times = ___)`

2.5 Reproducir estos usos de la función `sample()` en la consola e interpretar qué hace esta función.

- `sample(x = c("frambuesa", "frutilla"), size = 7, replace = TRUE )`
- `sample(x = c(0,1), size = 7, replace = TRUE )`
- inventar otros 2 usos de la función `sample()`.

2.6 ¿Por qué da error esta operación?

- `sample(x = c(-1,1), size = 3)`

2.7 Usar la función `sum()` para sumar los números del 1 al 100 inclusive.

2.8 R está pensado para analizar datos, entonces tiene formas particulares de tratar datos faltantes. Explorar en la consola el output de estas líneas de código. ¿Qué significa `na.rm = TRUE`?

- `n <- c(2, 4, 4, NA, 6)`
- `mean(n)`
- `max(n)`
- `mean(n, na.rm = TRUE)`
- `max(n, na.rm = TRUE)`

2.9 A partir de este vector de pesos en Kg de distintas personas:

```
peso_kg <- c(63, 69, 60, 65, NA, 68, 61, 70, 61, 59, 64, 69, 63, 63, NA, 72, 65, 64, 70, 63, 6
```

- Construir otro vector que descarte los `NA`.
- Usar la función `mean()` para calcular el peso promedio.
- Escribir una línea código que calcule cuántas personas pesan más que 65 kg.
- Escribir una línea de código que calcule cuántas personas pesan 63 kg.

2.10 Usar la función `sample()` para simular el resultado de tirar un dado.

2.11 Simular el resultado de tirar 120 veces un dado y contar cuántas veces salió cada resultado. ¿Qué resultado esperás obtener? ¿Coincide el resultado con lo que esperabas? No hay una única forma de resolverlo. *Ayuda:* `sum(___ == x)` o averigue cómo se usa la función `table()`.

2.12 Usar la función `sample()` para simular el resultado de tirar una moneda cargada, cuya probabilidad de salir cara es 0.7. *Ayuda:* ver cómo el argumento `prob` en `?sample`.

3. Scripts

Trabajar en la consola no siempre es lo más conveniente. Lo ideal es escribir en el editor de RStudio un archivo .R (o bien un archivo RMarkdown como veremos más adelante).

3.1 Crear un script de `R` que haga la siguientes tres operaciones y guardarlo en un archivo llamado `TP1-01.R`. Correrlo línea por línea.

```
x1 <- 2  # defino la variable x1
x2 <- 3  # defino la variable x2
y <- x1 + 2*x2
y
```

3.2 Crear un script que contenga la resolución completa del ejercicio 2.9. Incluir el enunciado y comentarios en el código. Guardarlo en un archivo ".R".

4. Data frames

Un data frame es una representación de los datos en formato de tabla en la que cada columna son vectores del mismo tamaño. Como cada columna es un vector, cada columna puede contener datos de un único tipo (ejemplo: numeric, character, factor, logic, etc). Se pueden pensar como variables. Por ejemplo:

```
d <-  data.frame(x = 1:5, y = letters[1:5], z = c(T,T,F,T,F))
```

4.1. ¿Cuál es la clase del objeto `d`? ¿Cuál es la clase de cada uno de los vectores columna?

4.2. Extraer el vector columna `y` usando `$`. Comparar con usar `d[,2]`.

4.3. Cargar el paquete de R gapminder usando `library(gapminder)`. Si da error es posible que no esté instalado. En tal caso ejecutan `install.packages("gapminder")` en la consola y luego si `library(gapminder)`.

4.4. ¿Qué variables tiene el dataset gapminder y de qué clase son? *Ayuda: usar la función head(gapminder) y/o str(gapminder).* Este dataset tiene formato tibble que a los efectos prácticos en este punto es exactamente igual que un data frame.

4.5. ¿De cuántos países hay datos? *Ayuda: averiguar qué hace la función unique().*

4.6 Explorar el tamaño del dataset gapminder usando las funciones `dim()`, `nrow()` y `ncol()`.

4.7 ¿Cuáles son las variables? Usar la función `names()`.

4.8 Extraer la información de Argentina, Uruguay y Chile y guardarla en un nuevo data frame `gm.sur`. ¿Cuántas filas tiene? ¿Cuál es el primero y el último año para el cuál existen datos de Argentina en gapminder?

4.9 ¿Qué resulta de hacer `gm.sur["pop"]`? ¿Qué resulta de hacer `gm.sur[,-(1:3)]`? Explorar el contenido de este data frame en el visor de RStudio haciendo `View(gm.sur)`

4.10 Las variables de clase "factor" (factores, o *fct*) son una clase especial que tiene R para trabajar con variables categóricas. Una vez que se crean, los factores sólo pueden contener un conjunto pre-definido de

valores que se conocen como los niveles del factor. ¿Qué variables del dataset de gapminder son factores?

4.11 *Redefinir niveles*. Supongamos que queremos cambiar la denominación del continente de Argentina a "América" (sin la s final). Prueben lo siguiente. ¿Qué pasó? ¿Por qué no funciona?

```
class(gm.sur$continent)
gm.sur$continent <- "America"
class(gm.sur$continent)
```

Ahora prueben esto. ¿Entienden por qué funciona?

```
levels(gm.sur$continent) <- c("Africa", "America", "Asia", "Europe", "Oceania")
class(gm.sur$continent)
head(gm.sur)
```

4.12. Vamos a usar mucho "factores" a lo largo del curso, pero para que se den una idea, por ejemplo, los factores son muy útiles para codificar variables categóricas en gráficos. Vamos a ver esto bastante a lo largo de las clases, pero para que vean una aplicación simple, corran estas líneas usando el paquete (que vamos a ver en las próximas clases) `ggplot2`.

```
library(ggplot2)

ggplot(data = gm.sur,
       mapping = aes(x = year, y = pop, col = country)) +
  geom_point(size = 3) +
  theme_classic()
```

Ahora corran lo siguiente. ¿En qué difiere del anterior? ¿Pueden intuir por qué tenemos ese resultado?

```
ggplot(data = gm.sur,
       mapping = aes(x = year, y = pop, size = country)) +
  geom_point() +
  theme_classic()
```

¿Y si reemplazan `size` por `shape` dentro de `aes(...)`?

4.13. Cambien más cosas del código anterior y prueben el resultado. De hecho, **cambiar cosas y ver qué pasa** es una gran forma de aprender.

Referencias

- R Programming for Data Science. Roger D. Peng.
- R for Data Science (2e). Hadley Wickham, Mine Çetinkaya-Rundel, and Garrett Grolemund.
- R para Ciencia de Datos. Hadley Wickham y Garrett Grolemund.
- Learning statistics with R. Danielle Navarro.
- R para profesionales de los datos: una introducción. Carlos J. Gil Bellosta.
- Data Science. A First Introduction. Tiffany Timbers, Trevor Campbell, and Melissa Lee.