Data_types

September 25, 2025

1 Introducción a la Programación para Ciencia de Datos

1.1 Lenguaje de programación R

Rocío Romero Zaliz - rocio@decsai.ugr.es

2 Tipos de datos básicos

- numeric / integer
- logical
- complex
- character

[ ]:
```
1
```

[ ]:
```
print(1)
```

[ ]:
```
class(1)
```

[ ]:
```
class(1.5)
```

[ ]:
```
class(-3.78)
```

[ ]:
```
TRUE
```

[ ]:
```
class(TRUE)
```

[ ]:
```
class(FALSE)
```

[ ]:
```
class(TRUE)
T <- 9
class(T)
```

[ ]:
```
print(4 + 1.2i)
```

[ ]:
```
class(4 + 1.2i)
```

[ ]:
```r
print("a")
```

[ ]:
```r
class("a")
```

[ ]:
```r
class("hola")
```

[ ]:
```r
class("adios") # Recomendado
```

[ ]:
```r
class('adios') # No recomendado
```

[ ]:
```r
"hola" y "adios" # Error
```

[ ]:
```r
""hola" y "adios"" # Error
```

[ ]:
```r
'"hola" y "adios"' # Para este tipo de cosas se usan las comillas simples
```

[ ]:
```r
1 / 0
```

[ ]:
```r
1 / Inf
```

[ ]:
```r
0 / 0
```

[ ]:
```r
print(NULL)
```

[ ]:
```r
?NULL
```

[ ]:
```r
print(NA)
```

[ ]:
```r
?NA
```

[ ]:
```r
NaN
```

[ ]:
```r
class(NaN)
```

[ ]:
```r
?NaN
```

3 Operadores aritméticos

[ ]:
```r
1 + 5
```

[ ]:
```r
5 - 4
```

[ ]:
```r
3 * 2.1
```

[ ]:
```
7 / -4.3
```

[ ]:
```
2 ^ 3
```

[ ]:
```
2 ** -1/2 # Igual que ^
```

[ ]:
```
4 %% 3
```

[ ]:
```
3 %/% 2
```

4 Operadores lógicos

[ ]:
```
print(1 < 9.5)
```

[ ]:
```
2.7 <= -5.3
```

[ ]:
```
4.9 > 2
```

[ ]:
```
4.9 >= 3
```

[ ]:
```
5 == 5
```

[ ]:
```
5 != 5
```

[ ]:
```
!(5 == 5)
```

[ ]:
```
(5 == 5) | (5 != 5)
```

[ ]:
```
(5 == 5) & (5 != 5)
```

[ ]:
```
1 == TRUE
```

[ ]:
```
0 == FALSE
```

[ ]:
```
7 == TRUE
```

[ ]:
```
as.logical(7) # WTF?
```

5 Vectores

La funcion `c()` se usa para crear vectores

[ ]:
```
c(1,5,8)
```

```
[ ]: print(c(1,5,8))
```

```
[ ]: print(c(1,2,3,4,3,5,4,5,2,3,9,-2,5.9,28,44,3,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0))
```

```
[ ]: class(c(1,5,8))
```

```
[ ]: print(c(1.5,5,-8.5,5i))
```

```
[ ]: class(c(1.5,5,-8.5,5i))
```

```
[ ]: print(c("hola", "adios"))
```

```
[ ]: print(c("hola", 7))
```

```
[ ]: print(c(TRUE, FALSE, TRUE))
```

```
[ ]: print(c(T, F, T)) # Ojo cuidado!
```

```
[ ]: print(c(TRUE, 0, TRUE))
```

```
[ ]: print(as.logical(c(TRUE, 0, TRUE)))
```

```
[ ]: print(as.character(c(3,4,5)))
```

```
[ ]: is.character(as.character(c(3,4,5)))
```

```
[ ]: is.character(c(3,4,5))
```

Se pueden crear vectores dando un rango de valores usando :

```
[ ]: print(1:4)
```

```
[ ]: print(4:1)
```

```
[ ]: print(1.1:6.9)
```

```
[ ]: print(-1:5)
```

```
[ ]: vector("numeric", length = 10)
```

```
[ ]: print(vector("numeric", length = 10))
```

```
[ ]: print(vector("logical", length = 10))
```

Atención que se vienen curvas...

```
[ ]: print(1:3 + 4:6)
```

```
[ ]: print(1:3 + 4:7) # Warning! Recycling Rule
```

```
[ ]: print(1:3 * 4:6)
```

```
[ ]: print(1:3 - 4:6)
```

```
[ ]: print(1:3 / 4:6)
```

```
[ ]: 1:3 == 4:6
```

```
[ ]: 1:3 != 4:6
```

```
[ ]: 1:3 < 4:6
```

```
[ ]: 1:3 >= 4:6
```

```
[ ]: !(0:3)
```

5.0.1 WTF? (Falling into the Floating Point Trap)

The R Inferno. Patrick Burns. https://www.burns-stat.com/pages/Tutor/R_inferno.pdf

```
[ ]: .1 == (.3 / 3)
```

```
[ ]: c(.3, .4 - .1, .5 - .2, .6 - .3, .7 - .4)
```

```
[ ]: unique(c(.3, .4 - .1, .5 - .2, .6 - .3, .7 - .4))
```

R is good enough at hiding numerical error that it is easy to forget that it is there. Don't forget...

```
[ ]: 1:5-1
```

```
[ ]: 1:(5-1)
```

```
[ ]: 10^2:6
```

```
[ ]: ?Syntax
```

Paréntesis... usad paréntesis...

6 Variables

```
[ ]: x <- 1 # A la variable x le asigno 1
```

```
[ ]: print(x)
```

[ ]:
```r
x = 5 # A la variable x le asigno 1
```

[ ]:
```r
print(x)
```

Atención: <- no es lo mismo que =

[ ]:
```r
# Do not run!
foo(93, a = 47)
foo(93, a <- 47)
```

[ ]:
```r
# Do not run!
system.time(result <- Sys.sleep(5))
result <- system.time(Sys.sleep(5))
```

Seguimos...

[ ]:
```r
v <- c(2,6,8,10,-2,4.3)
print(v)
```

[ ]:
```r
print(v + 1) # Recycling Rule
```

[ ]:
```r
print(v * v)
```

[ ]:
```r
print(v == v)
```

[ ]:
```r
all(v == v) # Evalúa si todas las condiciones de un vector lógico son TRUE
```

[ ]:
```r
identical(v,v) # Comprueba si dos objetos son exactamente iguales en tipo,␣
↪longitud, atributos y valores
```

[ ]:
```r
x <- c(1,1)
y <- c(1,1,1)
all(x == y)
identical(x, y)
```

[ ]:
```r
?`%in%`
```

[ ]:
```r
1:2 %in% 1
```

[ ]:
```r
print(v)
print(c(3,2) %in% v)
print(v %in% c(3,2))
```

[ ]:
```r
intersect(c(3,2), v)
```

[ ]:
```r
print(union(c(3,2), v))
```

[ ]:
```
print(union(v, c(3,2)))
```

[ ]:
```
print(setdiff(c(3,2), v))
```

[ ]:
```
print(setdiff(v, c(3,2)))
```

[ ]:
```
print(setdiff(abs(v), c(3,2)))
```

[ ]:
```
print(v)
any(v > 2)
```

[ ]:
```
all(v > 2)
```

7 Más sobre vectores

[ ]:
```
print(v)
print(v[1])
```

[ ]:
```
print(v[0]) # Nada
```

[ ]:
```
print(v[-1])
```

[ ]:
```
print(v)
print(v[-1:-2])
```

[ ]:
```
print(v[c(1,4)])
```

[ ]:
```
length(v)
```

[ ]:
```
length(4)
```

[ ]:
```
length("hola")
```

[ ]:
```
nchar("hola") # Más sobre cadenas de caracteres más adelante...
```

[ ]:
```
print(v)
v >= 0
```

[ ]:
```
print(v[v >= 0])
```

[ ]:
```
print(v[v < 0])
```

[ ]:
```
print(v[(v >= 0) & (v <= 5)])
```

[ ]:
```
print(v)
v[c(TRUE,FALSE)]
```

[ ]:
```
v[c(-1,-4)]
```

[ ]:
```
sum(v)
```

[ ]:
```
prod(v)
```

[ ]:
```
min(v)
```

[ ]:
```
max(v)
```

[ ]:
```
w <- c(NULL, 3, NA, 7, NaN)
print(w)
print(class(NaN))
print(class(as.numeric(NA)))
```

[ ]:
```
length(w)
```

[ ]:
```
class(w)
```

[ ]:
```
print(w)
print(sum(w))
```

[ ]:
```
sum(w, na.rm = TRUE)
```

[ ]:
```
prod(w)
```

[ ]:
```
prod(w, na.rm = TRUE)
```

[ ]:
```
min(w)
```

[ ]:
```
min(w, na.rm = TRUE)
```

[ ]:
```
max(w)
```

[ ]:
```
max(w, na.rm = TRUE)
```

[ ]:
```
print(w)
is.na(w)
```

[ ]:
```
any(is.na(c(1,NA,3)))
```

[ ]:
```
print(w)
is.nan(w)
```

Se pueden crear vectores tambien con `seq` o con `rep`

```
[ ]: print(1:6)
```

```
[ ]: ?seq
```

```
[ ]: seq(from=12, to=30, by=3)
```

```
[ ]: seq(12, 30, 3) # seq(from=12, to=30, by=3)
```

```
[ ]: seq(-12, 30, 3.4)
```

```
[ ]: seq(1.1, 7, 10)
```

```
[ ]: print(seq(1.1, 7, length=10))
```

```
[ ]: seq()
```

```
[ ]: ?rep
```

```
[ ]: rep(8,4)
```

```
[ ]: rep(8,4.1) # El entero
```

```
[ ]: print(rep("a",4))
```

```
[ ]: rep(1:3, 5)
```

```
[ ]: length(rep(1:3, 5))
```

```
[ ]: ?rep
```

```
[ ]: rep(c(5,12,13), 2)
```

```
[ ]: rep(c(5,12,13), each=2)
```

```
[ ]: letters
```

```
[ ]: LETTERS
```

```
[ ]: month.abb
```

```
[ ]: month.name
```

WTF?

[ ]:
```
print(x)
print(x == NA)
```

[ ]:
```
class(x == NA)
```

[ ]:
```
NA == NA
```

[ ]:
```
3 == c(3, 1, 3, NA)
```

[ ]:
```
is.na(c(3, 1, 3, NA))
```

[ ]:
```
xnotnull <- 42
print(xnotnull == NULL)
```

[ ]:
```
xnull <- NULL
print(xnull == NULL)
```

[ ]:
```
is.null(xnull)
```

[ ]:
```
x1 <- 10:1
print(x1)
x1 == c(4, 6)
```

[ ]:
```
print(x1)
print(x1 == 4)
print(x1 == 6)
print(x1 == 4 | x1 == 6)
print(x1 == 4 & x1 == 6)
```

[ ]:
```
print(x1 == 4 | 6)
as.logical(6)
```

[ ]:
```
print(x1)
x1 == (4 | 6)
```

[ ]:
```
4 | 6
```

Esos paréntesis... ¡que no falten!

[ ]:
```
50 < "7"
```

[ ]:
```
50 < as.numeric("7")
```

[ ]:
```
"50" < "7"
```

[ ]:
```
print(x != 1)
length(x != 1)
```

[ ]:
```r
length(x) != 1
```

No dejéis que R decida por vosotr@s…

[ ]:
```r
xna <- c(1, NA, 3, 2, 4, 2)
xna
```

[ ]:
```r
print(xna == 2)
xna[xna == 2]
```

[ ]:
```r
!is.na(xna)
```

[ ]:
```r
xna == 2
```

[ ]:
```r
xna[!is.na(xna) & xna == 2] # Analizar!
```

[ ]:
```r
print(xna)
print(!is.na(xna))
print(xna == 2)
```

[ ]:
```r
print(xna)
which(xna == 2)
```

[ ]:
```r
xna[which(xna == 2)]
```

[ ]:
```r
x4 <- c(a=1, b=2, a=3)
print(x4)
```

[ ]:
```r
x4["a"]
```

[ ]:
```r
names(x4)
```

[ ]:
```r
print(x4[names(x4) %in% "a"])
```

[ ]:
```r
print(c("missing value"=NA, "real string"="NA"))
```

Ejercicios vectores 1) Crea un vector de números impares entre el 1 y el 30 (ambos inclusive) 2) Crea los siguientes vectores: * un vector del 1 al 20 * un vector del 20 al 1 * un vector que tenga el siguiente patrón 1,2,3,…,19,20,19,18,…,2,1 3) Crea una secuencia de números del 1 al 30 con un incremento de 0.5 4) Crea una secuencia que contenga las cuatro primeras letras del abecedario en minúscula 3 veces cada una (a a a b b b c c c d d d) 5) Crea el vector numérico `x` con los valores 2.3, 3.3, 4.3 y accede al segundo elemento del vector 6) Crea un vector numérico `z` que contenga los números del 1 al 10. Cambia la clase del vector forzando que sea de tipo carácter. Después cambia el vector `z` a numerico otra vez 7) Crea un vector numérico con valores no ordenados usando la función `sample()`. Una vez creado ordena el vector de forma ascendente usando la función `sort()`. ¿Si quisieras invertir el orden de los elementos del vector que función utilizarías? 8) Crea un vector `x` que contenga los elementos -5,-1,0,1,2,3,4,5,6. Escribe un código del tipo `x[algo]`, para extraer:

* elementos de x menores que 0, * elementos de x menores o igual que 0, * elementos of x mayor o igual que 3, * elementos de x menor que 0 o mayor que 4, * elementos de x mayor que 0 y menor que 4, * elementos de x distintos de 0 9) Crea los siguientes vectores `x<-month.name[1:6]` y `z<-month.name[4:10]` a partir del vector original `month.name`. Recupera los valores idénticos entre los vectores x y z usando `%in%` 10) R permite extraer elementos de un vector que satisfacen determinadas condiciones usando la función `subset()`. Para el vector x <- c(6,1:3,NA,12) calcula los elementos mayores que 5 en x usando: * el filtrado normal, es decir, con el operador `>` * la función `subset()`

8 Listas

Son estructuras de datos que permiten combinar elementos de diferente tipo y/o tamaño.

[ ]:
```
list_data <- list("Red", "Green", c(21,32,11), TRUE, 51.23, 119.1i)
print(list_data)
```

[ ]:
```
class(list_data)
```

[ ]:
```
print(list_data[5:6])
```

[ ]:
```
print(list_data[6])
```

[ ]:
```
class(list_data[6])
```

[ ]:
```
print(list_data[[6]])
```

[ ]:
```
class(list_data[[6]])
```

[ ]:
```
names(list_data) <- c("Colors", "Age", "Time")
print(list_data)
```

[ ]:
```
names(list_data) <- c("A", "B", "C", "E", "F", "G")
print(list_data)
```

[ ]:
```
names(list_data)
attributes(list_data)
```

[ ]:
```
print(list_data[1])
print(list_data["A"])
```

[ ]:
```
class(list_data["B"])
```

[ ]:
```
list_data[[1]] <- 5
```

[ ]:
```
list_data[[1]]
```

[ ]:
```r
print(list_data)
```

[ ]:
```r
list_data[["C"]][2] <- "p"
```

[ ]:
```r
print(list_data)
```

[ ]:
```r
print(list_data[1:3])
```

[ ]:
```r
print(list_data[c("A","B")])
```

[ ]:
```r
print(list_data)
```

[ ]:
```r
print(unlist(list_data))
```

[ ]:
```r
class(unlist(list_data))
```

[ ]:
```r
num_list <- list(1,2,3,4,5)
day_list <- list("Mon","Tue","Wed", "Thurs", "Fri")
new_list <- c(num_list, day_list)
print(new_list)
```

[ ]:
```r
new_list[[5]] <- NULL # Borrar opcion 1
print(new_list)
```

[ ]:
```r
print(new_list[-5]) # Borrar opcion 2
```

[ ]:
```r
print(new_list)
```

[ ]:
```r
print(list(hola="Hola", 1, 3, adios = 5.6))
```

[ ]:
```r
a <- list(hola="Hola", 1, 3, adios = 5.6)
print(a)
print(a[1])
print(a["hola"])
print(a$hola)
```

[ ]:
```r
a <- list(hola="Hola", 1, 3, adios = 5.6)
a[[1]] <- NULL
print(a)
```

[ ]:
```r
a[1] <- NULL
print(a)
```

[ ]:
```r
a[[8]] <- 0
print(a)
```

[ ]:
```r
a[7] <- list(9)
length(a)
print(a)
```

[ ]:
```r
a[[6]] <- NULL # WTF?
print(a)
```

8.0.1 Ejercicios listas

1) Crea una lista llamada `mi_lista` que contenga los siguientes elementos: un vector numérico de 15 elementos, un vector de caracteres de 5 elementos y un vector de valores lógicos de 10 elementos todos TRUE
2) Dada la siguiente lista `my_list <- list(name="Fred", wife="Mary", no.children=3, child.ages=c(4,7,9))`:
   - Imprime los nombres de todos los componentes de la lista
   - Devuelve el segundo componente de la lista
   - Recupera el segundo elemento del cuarto componente de la lista
   - Imprime la longitud del cuarto componente de la lista
   - Reemplaza el cuarto componente de la lista por un vector de 12 numeros del 1 al 12
   - Elimina el componente `wife`
   - Añade un componente más a la lista llamado `pepe`
3) Convertir un vector de 30 números postivos y negativos en una lista con [1ra componente: los 2 primeros elementos, 2da componente: los 5 siguientes pero como caracteres, 3ra componente: los elementos restantes que sean valores positivos, 4ta componente: una lista de caracteres con tu nombre y apellidos (ej: "Rocío" "Romero" "Zaliz")]
   - Una vez creado ponles nombre (ej: "1ro", "2do", etc)
   - Accede al tercer elemento por su nombre
   - Fusiona el primer y cuarto componente en un quinto componente y borra los originales
4) Crea una nueva lista cuyos componentes sean las listas de los ejericios anteriores (OJO: Su longitude debería ser 3)
5) Crea una nueva lista que concatene las listas de los tres primeros ejericios. ¿Qué longitud tiene?

9 Matrices

- Las matrices son vectores con un atributo de dimensión. El atributo de dimensión es en sí mismo un vector entero de longitud 2 (nrow, ncol).
- Todas las columnas de una matriz deben tener el mismo tipo (numérico, carácter, etc.) y la misma longitud.

[ ]:
```r
x <- 1:20
print(x)
dim(x) <- c(5,4)
print(x)
attributes(x)
```

[ ]:
```r
x
```

[ ]:
```r
class(x)
```

[ ]:
```r
matrix(nrow = 2, ncol = 3)
```

[ ]:
```r
matrix(0, nrow = 2, ncol = 3)
```

[ ]:
```r
matrix("a", nrow = 2, ncol = 3)
```

[ ]:
```r
matrix(1:10, nrow = 3, ncol = 4)
```

[ ]:
```r
matrix(1:10, nrow=3, ncol = 4, byrow=TRUE)
```

[ ]:
```r
?matrix
```

[ ]:
```r
matrix(1:12, nrow = 3)
```

[ ]:
```r
matrix(1:12, ncol=3)
```

[ ]:
```r
m <- matrix(1.1:12.1, nrow=3, byrow=FALSE)
rownames(m) <- c("A","B","C")
colnames(m) <- c("1","2","x","y")
m
```

[ ]:
```r
print(dim(m))
```

[ ]:
```r
length(m)
```

[ ]:
```r
rbind(1:5,11:15)
```

[ ]:
```r
cbind(1:5,11:15)
```

[ ]:
```r
m
```

[ ]:
```r
m[1,1]
```

[ ]:
```r
print(m[1,])
```

[ ]:
```r
print(m[,1])
print(class(m[,1]))
```

[ ]:
```r
print(m[,])
```

[ ]:
```r
print(m[6])
```

[ ]:
```
print(m)
m[1:2,1:3]
```

[ ]:
```
m[,c("1", "x")]
```

[ ]:
```
m[,c(-1,-4)]
```

[ ]:
```
m[,-"x"] # Error!
```

[ ]:
```
print(colnames(m))
c("x") %in% colnames(m)
```

[ ]:
```
colnames(m) %in% c("x")
```

[ ]:
```
colnames(m)[4] <- "1"
print(colnames(m))
```

[ ]:
```
m <- m[,!(colnames(m) %in% c("x"))]
m
```

[ ]:
```
print(m[1,])
class(m[1,])
```

[ ]:
```
m[1,,drop=FALSE]
```

[ ]:
```
print(m[1,,drop=TRUE])
```

[ ]:
```
mx <- matrix(1:12, nrow=3, byrow=TRUE)
mx
```

[ ]:
```
mx[3, 2] <- 4
mx
```

[ ]:
```
mx[2, ] <- c(1,3) # Recycle!
mx
```

[ ]:
```
print(mx[1:2, 3:4])
mx[1:2, 3:4] <- c(8,4,2,1)
mx
```

[ ]:
```
print(dim(mx))
```

[ ]:
```
dim(mx)[1]
```

[ ]:
```
dim(mx)[2]
```

[ ]:
```
ncol(mx)
```

[ ]:
```r
nrow(mx)
```

[ ]:
```r
t(mx)
```

[ ]:
```r
diag(mx)
```

[ ]:
```r
m * m
```

[ ]:
```r
m %*% m # Multiplicación de matrices
```

[ ]:
```r
a <- attributes(m)
print(a)
```

[ ]:
```r
class(attributes(m))
```

[ ]:
```r
a$dimnames[[1]]
```

[ ]:
```r
print(list_data[[1]][1])
```

10 Arrays

Las matrices con más de dos dimensiones se llaman `array`

[ ]:
```r
x <- 1:20
dim(x) <- c(2,5,2)
print(x)
```

[ ]:
```r
a <- array(1:24, dim=c(3,4,2))
print(a) # Una matriz de 3 dimensiones
```

[ ]:
```r
a <- array(1:24, dim=c(3,4,2),
           dimnames = list(c("ROW1","ROW2","ROW3"),
                           c("COL1","COL2","COL3","COL4"),
                           c("Matrix1","Matrix2")))
print(a)
```

[ ]:
```r
print(a[1,3,,drop=FALSE])
```

[ ]:
```r
a[1,3,1]
```

[ ]:
```r
print(a[3,,2])
```

[ ]:
```r
print(a[,,2])
```

Ejercicios matrices y arrays 1) Dada la siguiente matriz:

* Indique donde están las "X" usando la función `which`
* Indique donde están las "X" usando la función `which` pero usando el parámetro `arr.ind`
* Indique cuantos valores "X" hay en la matriz
* Reemplace las "X" por 1 y los "O" por 0 y convierta la matrix en una matrix numérica
* Indique si la matrix es simétrica
* Cree una nueva matrix con los valores en las filas y columnas pares

2) Crea una matriz que represente un Sudoku. Debe estar inicializada en NA. Luego rellenar solo un 10% de las casillas con un valor entre 1 y 9 (HINT: usa la función `sample`). No se preocupe de que sea un esquema válido (se pueden repetir valores en una fila por ejemplo… eso ya lo haremos más adelante… si acaso…)
3) Teniendo estas dos matrices:
   - Aplique la máscara a la matrix, en donde si la posicion en la máscara es FALSE entonces el resultado es 0 y sino se mantiene su valor
   - Calcule el producto de la matriz original por la matriz modificada traspuesta (NOTA: la matriz final será de 5x5)
4) Crea un array 5D llamado "mi_array5D" con las siguientes características, rellénalo con números enteros consecutivos comenzando desde 1:
   - Tamaño de la primera dimensión: 2 elementos
   - Tamaño de la segunda dimensión: 3 elementos
   - Tamaño de la tercera dimensión: 4 elementos
   - Tamaño de la cuarta dimensión: 5 elementos
   - Tamaño de la quinta dimensión: 6 elementos

   4.1) Accede a los siguientes elementos del array:
   - los elementos pares (analice la posición de esos valores en las 5 dimensiones)
   - estudiando el caso anterior muestre los elementos impares sin usar ningun tipo de condicional, igualdad o desigualdad

   4.2) Redimensiona el array para que tenga:
   - Tamaño de la primera dimensión: 6 elementos.
   - Tamaño de la segunda dimensión: 5 elementos.
   - Tamaño de la tercera dimensión: 4 elementos.
   - Tamaño de la cuarta dimensión: 3 elementos.
   - Tamaño de la quinta dimensión: 2 elementos.
5) Supongamos que tenemos un grafo no dirigido con 6 nodos etiquetados del 1 al 6, y las siguientes aristas:
   - Arista 1: (1, 2)
   - Arista 2: (1, 3)
   - Arista 3: (2, 4)
   - Arista 4: (2, 5)
   - Arista 5: (3, 6)
   - Arista 6: (4, 5)

   Crea una matriz de adyacencia "matriz_adyacencia" en R para representar este grafo. En una matriz de adyacencia, las filas y columnas representan los nodos, y un valor 1 indica que existe una arista entre los nodos correspondientes, mientras que un valor 0 indica la ausencia de una arista.
   - Calcula el grado del nodo 4 en el grafo. El grado de un nodo es la cantidad de aristas que inciden en él
   - Encuentra todos los nodos adyacentes del nodo 2

- Cuenta la cantidad de nodos que tiene una arista hacia si mismos (self-loop)

11 Factores (variables nominales)

Los factores son vectores para valores categóricos * Levels: refers to the input values * Labels: refers to the output values of the new factor

[ ]:
```
Pain <- c(0,3,2,2,1)
Pain
```

[ ]:
```
SevPain <- factor(c(0,3,2,2,1), levels=c(0,1,2,3),␣
 ↪labels=c("none","mild","medium","severe"))
class(SevPain)
```

[ ]:
```
SevPain
```

[ ]:
```
directions <- c("North", "East", "South", "South")
factor(directions)
```

[ ]:
```
factor(directions, levels=c("North", "East", "South", "West"),␣
 ↪labels=c("N","E","S","W"))
```

[ ]:
```
fdir <- factor(directions, levels=c("North", "East", "South", "West"),␣
 ↪labels=c("N","E","S","W"))
table(fdir)
```

[ ]:
```
table(factor(directions))
```

[ ]:
```
print(fdir)
fdir[1] <- "E"
print(fdir)
fdir[1] <- "pepe"
print(fdir)
```

[ ]:
```
levels(fdir) <- c(levels(fdir), "pepe")
fdir[1] <- "pepe"
print(fdir)
```

12 Factores ordenados (variables ordinales)

[ ]:
```
status <- c("Lo", "Hi", "Med", "Med", "Hi")
ordered.status <- factor(status, levels=c("Lo", "Med", "Hi"), ordered=TRUE)
print(ordered.status)
```

[ ]:
```r
table(status)
```

[ ]:
```r
table(ordered.status)
```

WTF?

[ ]:
```r
f <- factor(c(101,6,8,9,6,103))
print(f)
is.numeric(f)
```

[ ]:
```r
as.numeric(f)
```

[ ]:
```r
as.numeric(levels(f))
```

[ ]:
```r
as.numeric(levels(f))[as.numeric(f)]
```

[ ]:
```r
print(as.character(f))
as.numeric(as.character(f))
```

[ ]:
```r
ff <- factor(c("AA", "BA", "CA"))
print(ff)
ff[1:2]
```

[ ]:
```r
ff[1:2, drop=TRUE] # Borrar opcion 1
```

[ ]:
```r
factor(ff[1:2]) # "Borrar" opcion 2
```

[ ]:
```r
f1 <- factor(c("AA", "BA", NA, "NA"))
f1
```

[ ]:
```r
f1 <- factor(c("AA", "BA", NA, "NA"), exclude=NULL)
f1
```

12.0.1 Ejercicios factores

1) Crea un factor llamado "fact_puntuacion" que contenga puntuaciones de 1 al 5 para 1000 productos. Utiliza la función `summary()` para obtener un resumen de la distribución de puntuaciones. ¿Qué puedes deducir del tipo de aleatoriedad usada en `sample`? ¿Es normal, uniforme, ...?
2) Crea un factor ordenado para edades entre 0 y 100 de 100 personas. Utiliza la función `cut` para generar 3 nieveles separados. Ahora reetiqueta el factor como "joven", "adulto" o "anciano". Muestra la frecuencia de cada nivel.
3) Crea un factor para el siguiente vector teniendo en cuenta que los niveles posibles son todas las letras de la "A" a la "Z":
   - Modifica una de las "I" por una "F"
   - Agrega una nueva "A" en última posición

- Indica las letras que aparecen en el factor ordenadas por su frecuencia, empezando con la menos frecuente

Data Frames

Un marco de datos o data frame es una matriz generalizada, donde las diferentes columnas pueden tener diferentes tipos de datos (numérico, carácter, factor, etc.).

Por ejemplo, vectores y/o factores de la misma longitud que están relacionados "transversalmente", de forma que los datos en la misma posición proceden de la misma unidad experimental (sujeto, animal, etc.).

[ ]:
```
S <- as.factor(c("F","M","M","F"))
Patients <- data.frame(age=c(31,32,40,50), sex=S)
Patients
```

[ ]:
```
employee <- c("John Doe","Peter Gynn","Jolie Hope")
salary <- c(21000, 23400, 26800)
startdate <- as.Date(c("2010-11-1","2008-3-25","2007-3-14"))

employ.data <- data.frame(employee, salary, startdate)
employ.data
```

[ ]:
```
str(employ.data)
```

[ ]:
```
employ.data <- data.frame(employee, salary, startdate, stringsAsFactors=TRUE)
employ.data
```

[ ]:
```
ncol(employ.data)
```

[ ]:
```
nrow(employ.data)
```

[ ]:
```
length(employ.data) # ¡Cuidado! -> Nunmero de filas
```

[ ]:
```
summary(employ.data)
```

[ ]:
```
print(employ.data[,1])
employ.data[1,1]
```

[ ]:
```
employ.data[1,]
```

[ ]:
```
employ.data[,1]
```

[ ]:
```
employ.data$employee
```

[ ]:
```
class(c("John Doe", 50000, "1990-11-1"))
```

[ ]:
```
rbind(employ.data, c("John Doe", 50000, "1990-11-1"))
```

[ ]:
```r
rbind(employ.data, c("Rocio Romero", 50000, "1990-11-1")) # Error
```

[ ]:
```r
rbind(employ.data, c("John Doe", 50000)) # Error
```

[ ]:
```r
# Crear los datos para el dataframe
year <- c(2000, 2020, 2021, 2021, 2022, 2022, 2023, 2023, 2024, 2024)
country <- c("Argentina", "Brasil", "Chile", "Colombia", "Ecuador", "Perú",␣
 ↪"México", "Uruguay", "Paraguay", "Bolivia")
valor1 <- c(100, 120, 90, 110, 80, 105, 115, 95, 125, 75)

# Crear el dataframe
data1 <- data.frame(year, country, valor1)
data1
```

[ ]:
```r
# Crear los datos para el dataframe
country <- rep(c("Argentina", "Brasil", "Chile"), each = 1, times = 5)
year <- c(rep(seq(2000, 2006, by = 1), each = 2), 2020)
valor2 <- runif(15, min = 1, max = 100) # Valores aleatorios entre 1 y 100

# Crear el dataframe
data2 <- data.frame(country, year, valor2)
data2
```

[ ]:
```r
data1$country <- as.factor(data1$country)
data1
```

[ ]:
```r
length(data1)
```

[ ]:
```r
levels(data[,2]) <- c(levels(data[,2]), "Corea del Sur")
data[1,2] <- "Corea del Sur"
```

[ ]:
```r
levels(data[,2])[length(levels(data[,2])) + 1] <- "Corea del Sur"
data[1,2] <- "Corea del Sur"
```

[ ]:
```r
data1[,2] <- list(1:10)
data1
```

[ ]:
```r
as.data.frame(as.list(data1))
```

[ ]:
```r
?merge
```

[ ]:
```r
data1
```

[ ]:
```r
data2
```

[ ]:
```r
merge(data1, data2, by = c("country", "year")) # conserva sólo las␣
 ↪observaciones coincidentes
```

[ ]:
```r
merge(data1, data2, by = c("country", "year"), all.x = TRUE)
```

[ ]:
```r
merge(data1, data2, by = c("country", "year"), all.y = TRUE)
```

[ ]:
```r
merge(data1, data2, by = c("country", "year"), all = TRUE)
```

12.0.2 Ejercicios data frames

1) Usando el dataset `mtcars`:
   - Muestra las 5 primeras filas
   - Presenta un resumen del datasest
   - Haz una copia del dataset y:
     - Convierte la variables "cyl", "gear" y "carb" en factores
     - Convierte las variables "vs" y "am" en lógicas
     - Muestra solo los coches con una potencia ("hp") mayor a 100
     - Crea un nuevo data frame llamado coches_seleccionados que incluya solo las columnas "mpg", "cyl", "hp" y "qsec" del dataset
     - Calcula el resumen estadístico (media, mediana, desviación estándar, etc.) de la columna "mpg" (millas por galón) para todos los coches en el dataset
     - Idem para "cyl" (número de cilindros)
     - Calcula la cantidad total de coches para cada valor único en la columna "cyl" (número de cilindros)
     - Encuentra el modelo de coche con la mayor potencia ("hp") y muestra su información completa
     - Calcula el promedio de potencia de los coches con 8 cilindros
2) Usando el dataset `Cars93` del paquete `MASS` y el dataset `mtcars`:
   - Agrega una columna al dataset de `mtcars` que contenga la marca y modelo de cada coche
   - Combina los dos dataset por modelo y marca del coche solo para aquellos modelos presentes en ambos datasets. ¿Coinciden los demás valores?
3) Supongamos que tienes dos dataframes con información sobre estudiantes de dos cursos diferentes. Cada data frame contiene tres columnas: "Nombre", "Edad" y "Promedio".

- Combina los dos dataframes en uno solo llamado todos_los_estudiantes
- Agrega una nueva columna llamada "Curso" al dataframe todos_los_estudiantes que indicar a qué curso pertenece cada estudiante ("A" o "B")

[ ]:
```r
?mtcars
```

[ ]:
```r
?Cars93
```

[ ]:
```r
mtcars
```

[ ]:
```r
library(MASS)
Cars93
```