Curso de Ingreso - Módulo Programación

Contents

Guía de ejercicios

1. Conceptos básicos de los programas imperativos
2. Estructuras de control - Gráficos
3. Funciones - DataFrames
4. Modelado: Álbum de figuritas
5. Modelado: La ruina del jugador

1 Introducción

Esta es la guía de ejercicios correspondiente a la clase 01 (ver diapositivas). Deberá entregar todos los ejercicios resueltos en un archivo .R. Cada ejercicio debe estar resuelto entre comentarios que indique secciones dentro del archivo (en esta ocasión pueden usar el siguiente archivo como *template* para resolverlo: template-01.R).

Algunos de los temas necesarios para resolver esta guía no fueron incluidos en la teórica, muchos se encuentran en este documento, mientras que otros deberán ser investigados (por ejemplo, buscando en internet). El ejercicio de buscar cómo abordar/resolver problemas en internet es *casi tan* importante como poder resolverlos.

Además de escribir los programas pedidos deberán probarlos y dejar constancia de las pruebas realizadas, además de explicitar si anduvo como era esperado o no.

Ejercicios

1. Se desea tener un programa que dada la variable `grados`, que representa la temperatura en grados Farenheit, calcule en otra variable el valor en Celsius. Más info https://www.lmgtfy.es/?q=formula+formula+fahrenheit+a+celsius.

```
grados <- 90

enCelsius <- (grados - 32) * 5/9

enCelsius
```

```
## [1] 32.22222
```

2. Escribir otro programa que se comporte a la inversa, es decir, que dada una variable que represente la temperatura en Celsius, calcule su equivalente en Farenheit.

```
grados <- 25

enFarenheit <- grados * 9/5 + 32

enFarenheit
```

```
## [1] 77
```

3. Escribir un conversor de kilometros a millas.

```
unosKilometros <- 50

unasMillas <- unosKilometros/1.609

unasMillas
```

```
## [1] 31.0752
```

4. Dado un cuadrado, que el largo de su base se encuentra guardado en una variable llamada `base`, calcular:

   a. El perímetro

   ```
   base <- 30

   perimetro <- base * 4

   perimetro
   ```

   ```
   ## [1] 120
   ```

   b. El área

   ```
   base <- 30

   area <- base * base

   area
   ```

   ```
   ## [1] 900
   ```

5. Idem anterior pero para un triangulo equilatero.

```
base <- 30

perimetro <- base * 3
```

```
perimetro
```

```
## [1] 90
```

```
base <- 30

area <- sqrt(3)/4 * base^2

area
```

```
## [1] 389.7114
```

6. Asumiendo que los años tienen siempre 365 días, calcular:
   a. cuántos días vas a cumplir tu próximo cumpleaños,

   ```
   edad <- 27

   dias <- 27 * 365

   dias
   ```

   ```
   ## [1] 9855
   ```

   b. cuántas horas vas a haber vivido,

   ```
   horas <- dias * 60

   horas
   ```

   ```
   ## [1] 591300
   ```

   c. y cuántos segundos.

   ```
   segundos <- horas * 60

   segundos
   ```

   ```
   ## [1] 35478000
   ```

7. Se tienen las notas de 3 materias en sus respectivas variables: `matematica`, `lengua`, `dibujo`. Calcular el promedio de dichas notas.

```
matematica <- 6
lengua <- 7
dibujo <- 8

suma <- matematica + lengua + dibujo

promedio <- suma / 3

promedio
```

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

8. Repetir el ítem anterior, pero ahora con los valores guardados en un vector llamado `notas`. Hint: se puede acceder a los elementos de un vector con `[]`. Ej: `c(4, 6, 88)[2]` nos da el valor `6`.

```
notas <- c(6, 7, 8)

suma <- notas[1] + notas[2] + notas[3]
```

```r
promedio <- suma / 3

promedio
```

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

9. Si en el ítem anterior no usaste la función `lenght()` y `sum()`, volvé a resolverlo usandolas.

```r
notas <- c(6, 7, 8)

suma <- sum(notas)
cantidad <- length(notas)

promedio <- suma / cantidad

promedio
```

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

10. Si en el ítem anterio al anterior no usaste la función `mean()`, volvé a resolverlo usandola.

```r
promedio <- mean(notas)

promedio
```

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

11. Dadas dos variables, `perro` y `gato` escribir un programa que intercambie los valores de ambas variables.

```r
perro <- 70
gato <- -15

temporal <- perro
perro <- gato
gato <- temporal
```

```r
perro
```

```
## [1] -15
```

```r
gato
```

```
## [1] 70
```

```r
# opcion sin variables extras
perro <- 70
gato <- -15

perro <- perro + gato
gato <- perro - gato
perro <- perro - gato
```

```r
perro
```

```
## [1] -15
```

```r
gato
```

```
## [1] 70
```

12. Calcular el índice de masa corporal (IMC) de una persona cuya altura es `1.78`m y su peso es `80`kg. (`IMC = peso / altura^2`)

```
peso <- 80
altura <- 1.78

imc <- peso / altura^2

imc
```

```
## [1] 25.24934
```

13. Si tenemos los pesos y las alturas de personas en 2 vectores, calcular el IMC para cada uno.

```
pesos <- c(80, 70, 75, 94, 67)
alturas <- c(1.68, 1.75, 1.85, 1.90, 1.68)

imcs <- pesos / alturas^2

imcs
```

```
## [1] 28.34467 22.85714 21.91381 26.03878 23.73866
```

14. Sobre el cálculo del ejercicio anterior, encontrar el valor máximo, el mínimo, el promedio y la mediana de los IMCs.

```
max(imcs)
```

```
## [1] 28.34467
```

```
min(imcs)
```

```
## [1] 21.91381
```

```
mean(imcs)
```

```
## [1] 24.57861
```

```
median(imcs)
```

```
## [1] 23.73866
```

2 Estructuras de control - Gráficos

Esta es la guía de ejercicios correspondiente a la clase 02 (ver diapositivas). Deberá entregar al menos todos los ejercicios indicados con una estrella (★) en un archivo .R. Igualmente se recomienda realizar *todos* los ejercicios para ganar mayor habilidad en programación. Cada ejercicio debe estar resuelto entre comentarios que indique secciones dentro del archivo.

Algunos de los temas necesarios para resolver esta guía no fueron incluidos en la teórica, muchos se encuentran en este documento, mientras que otros deberán ser investigados (por ejemplo, buscando en internet). El ejercicio de buscar cómo abordar/resolver problemas en internet es *casi tan* importante como poder resolverlos.

Además de escribir los programas pedidos deberán probarlos y dejar constancia de las pruebas realizadas, además de explicitar si anduvo como era esperado o no.

2.1 Condicionales

(1) Completar el siguiente programa de manera tal que el valor de la variable `a_es_mas_grande` sea `TRUE` unicamente cuando la variable `a` es más grande que `b` y `FALSE` en caso contrario.

```
a_es_mas_grande <- COMPLETAR
```

```
a <- 3
b <- 2
a_es_mas_grande <- a > b
a_es_mas_grande
```

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

(2) ★ Completar el siguiente programa de manera tal que el valor de `nombre_mas_grande` sea el nombre de la variable cuyo valor es más grande entre `a` y `b`, en caso de ser iguales devolver cualquiera (puede ser siempre el mismo).

```
if (COMPLETAR) {
  nombre_mas_grande <- COMPLETAR
} else {
  COMPLETAR
}
```

```
a <- 3
b <- 2
if (a >= b) {
  nombre_mas_grande <- "a"
} else {
  nombre_mas_grande <- "b"
}
nombre_mas_grande
```

```
## [1] "a"
```

(3) Modificar el programa anterior para que en caso de que sean iguales devuelva `"iguales"`

```
if (COMPLETAR) {
  nombre_mas_grande <- COMPLETAR
} else {
  if (COMPLETAR) {
    COMPLETAR
  } else {
    COMPLETAR
  }
}
```

```
a <- 3
b <- 3
if (a > b) {
  nombre_mas_grande <- "a"
} else {
  if (a < b) {
    nombre_mas_grande <- "b"
  } else {
    nombre_mas_grande <- "iguales"
  }
}
nombre_mas_grande
```

```
## [1] "iguales"
```

(4) ★ Escribir un programa que dado el valor guardado en la variable `q`, si dicho valor no es múltiplo de 3,

entonces lo múltiplique por 3. En caso de serlo no debe modificarse.

```
q <- 5

if (q %% 3 != 0) {
  q <- q * 3
}
q
```

```
## [1] 15
```

(5) ★ Completar el programa para que calcule un paso de la función de Collatz. Recordar que dice:

$$collatz(n) = \begin{cases} n/2 & \text{si n es par} \\ (3n+1) & \text{si no} \end{cases}$$

```
if (COMPLETAR) {
  nuevo_n <- COMPLETAR
} else {
  COMPLETAR
}
```

```
n <- 15
if (n %% 2 == 0) {
  nuevo_n <- n / 2
} else {
  nuevo_n <- 3 * n + 1
}
nuevo_n
```

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

(6) ★ Dados 2 vectores de números `v1` y `v2` concatenarlos en una variable llamada `res` en el siguiente orden:

- si el primer elemento de `v1` es menor que el primero de `v2`, entonces `v1` y luego `v2`
- sino al revés

```
v1 <- c(15,14,2)
v2 <- c(30, -25)

if (v1[1] < v2[1]) {
  res <- c(v1, v2)
} else {
  res <- c(v2, v1)
}
res
```

```
## [1]  15  14   2  30 -25
```

(7) ★ Dados 2 vectores de números `v1` y `v2` y una variable llamada `cuenta` escribir un programa que calcula en la variable `res` los siguiente:

- si `cuenta` vale la cadena de caracteres “promedio”, el promedio de todos los valores de `v1` y `v2`,
- si `cuenta` vale la cadena de caracteres “minimo”, el mínimo entre todos los valores de `v1` y `v2`,
- si `cuenta` vale la cadena de caracteres “minimo1”, el mínimo entre todos los valores de `v1`,
- para cualquier otro valor devolver la suma de todos los valores.

```r
v1 <- c(15,14,2)
v2 <- c(30, -25)
cuenta <- "minimo"

if (cuenta == "promedio") {
  res <- mean(c(v1, v2))
} else {
  if (cuenta == "minimo") {
    res <- min(c(v1, v2))
  } else {
    if (cuenta == "minimo1") {
      res <- min(v1)
    } else {
      res <- sum(c(v1,v2))
    }
  }
}
res
```

```
## [1] -25
```

(8) Dados 2 vectores de números `v1` y `v2` concatenarlos en una variable llamada `res` de manera tal que el que primero que aparezca sea aquel cuya suma de elementos sea menor o igual que el otro.

```r
v1 <- c(15,14,2)
v2 <- c(30, -25)

if (sum(v1) <= sum(v2)) {
  res <- c(v1, v2)
} else {
  res <- c(v2, v1)
}
res
```

```
## [1]  30 -25  15  14   2
```

2.2 Ciclos

(9) ★ Completar el programa para que dado un vector de 10 posiciones llamado `v` cuente la cantidad de posiciones i cuyo valor es exactamente i.

```r
res <- 0
for (i in 1:10){
  if (v[i] == COMPLETAR) {
    res <- res + COMPLETAR
  }
}
```

```r
v <- c(1:5,1:5)
res <- 0
for (i in 1:10){
  if (v[i] == i) {
    res <- res + 1
  }
}
res
```

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

(10) Modificar el programa anterior para que acepte vectores de cualquier longitud. Pista: puede usar la función `length` para obtener la longitud del vector.

```
v <- c(1:5,1:5)
res <- 0
for (i in 1:length(v)){
  if (v[i] == i) {
    res <- res + 1
  }
}
res
```

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

(11) ★ Modificar el programa anterior para que además, en caso de que no cumpla con que la posición i valga i, reemplace dicho valor por un cero.

```
v <- c(1:5,1:5)
res <- 0
for (i in 1:length(v)){
  if (v[i] == i) {
    res <- res + 1
  } else {
    v[i] <- 0
  }
}
res
```

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

(12) ★ El siguiente programa recorre un vector `v` hasta encontrar un elemento que cumpla con tener como valor la posición. Dicho valor queda en la variable `i`. Experimentar con este programa teniendo en cuenta 2 casos: que existe y que no una posición que cumpla.

```
i <- 1
while (v[i] != i){
  i <- i + 1
}
```

(13) Arreglar el programa anterior para que en caso de que no exista, el programa termine sin dar error. Pista: antes de la condición diga `v[i]` debemos asegurarnos que `i` es una posición válida, es decir, es menor o igual que la longitud del vector.

```
i <- 1
while (COMPLETAR & v[i] != i){
  i <- i + 1
}
```

```
v <- c(0:4,1:4,11)
i <- 1
while (i <= length(v) & v[i] != i){
  i <- i + 1
}
i
```

```
## [1] 11
```

(14) Podemos verificar si existe o no dicho elemento en el ejercicio anterior mirando el valor de `i`. Si el

valor es una posición válida de `v` entonces sí existe. Completar el programa para que en una variable `existe_i` determine con `TRUE/FALSE` la existencia de dicho valor.

```
i <- 1
while (COMPLETAR & v[i] != i){
  i <- i + 1
}

existe <- FALSE
if (COMPLETAR) {
  existe <- TRUE
}
```

```
v <- c(0:4,1:4,11)
i <- 1
while (i <= length(v) & v[i] != i){
  i <- i + 1
}
existe <- i <= length(v)
existe
```

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

(15) ★ Dados dos dados (de 6 caras que valen de 1 a 6), uno azul y uno rojo, calcular con un programa cuantas combinaciones posibles de valores hay. Pista: debería dar 6*6=36. ;-)

```
contar <- COMPLETAR
for (dado_azul in 1:6){
  for (dado_rojo in COMPLETAR){
    contar <- contar + COMPLETAR
  }
}
```

```
contar <- 0
for (dado_azul in 1:6){
  for (dado_rojo in 1:6){
    contar <- contar + 1
  }
}
contar
```

```
## [1] 36
```

(16) ★ Con el mismo par de dados, calcular con un programa cuantas combinaciones tienen los 2 dados iguales.

```
contar <- 0
for (dado_azul in 1:6){
  for (dado_rojo in 1:6){
    if (dado_azul==dado_rojo) {
      contar <- contar + 1
    }
  }
}
contar
```

```
## [1] 6
```

(17) Con el mismo par de dados, calcular con un programa cuantas combinaciones suman 10.

```r
contar <- 0
for (dado_azul in 1:6){
  for (dado_rojo in 1:6){
    if (dado_azul + dado_rojo == 10) {
      contar <- contar + 1
    }
  }
}
contar
```

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

(18) ★ Dado un vector de los primeros n naturales, escribir un programa que calcule para cada elemento el cuadrado de dicho valor en un nuevo vector.

```r
valores_n <- COMPLETAR
valores_cuadrado <- c()

for (i in COMPLETAR) {
  valores_cuadrado <- c(valores_cuadrado,  COMPLETAR)
}
```

```r
n <- 15
valores_n <- 1:n
valores_cuadrado <- c()

for (i in valores_n) {
  valores_cuadrado <- c(valores_cuadrado,  i**2)
}
valores_cuadrado
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100 121 144 169 196 225
```

(19) Dado un vector de los primeros n números pares naturales, escribir un programa que calcule para cada elemento el cuadrado de dicho valor, si es múltiplo de 4, sino que lo divida por 2.

```r
n <- 10*2
valores_n <- seq(2, n, 2)
valores_nuevos <- c()

for (i in valores_n) {
  if (i %% 4 == 0 ){
    valores_nuevos <- c(valores_nuevos,  i**2)
  } else {
    valores_nuevos <- c(valores_nuevos,  i/2)
  }
}
valores_nuevos
```

```
##  [1]   1  16   3  64   5 144   7 256   9 400
```

2.3 Gráficos

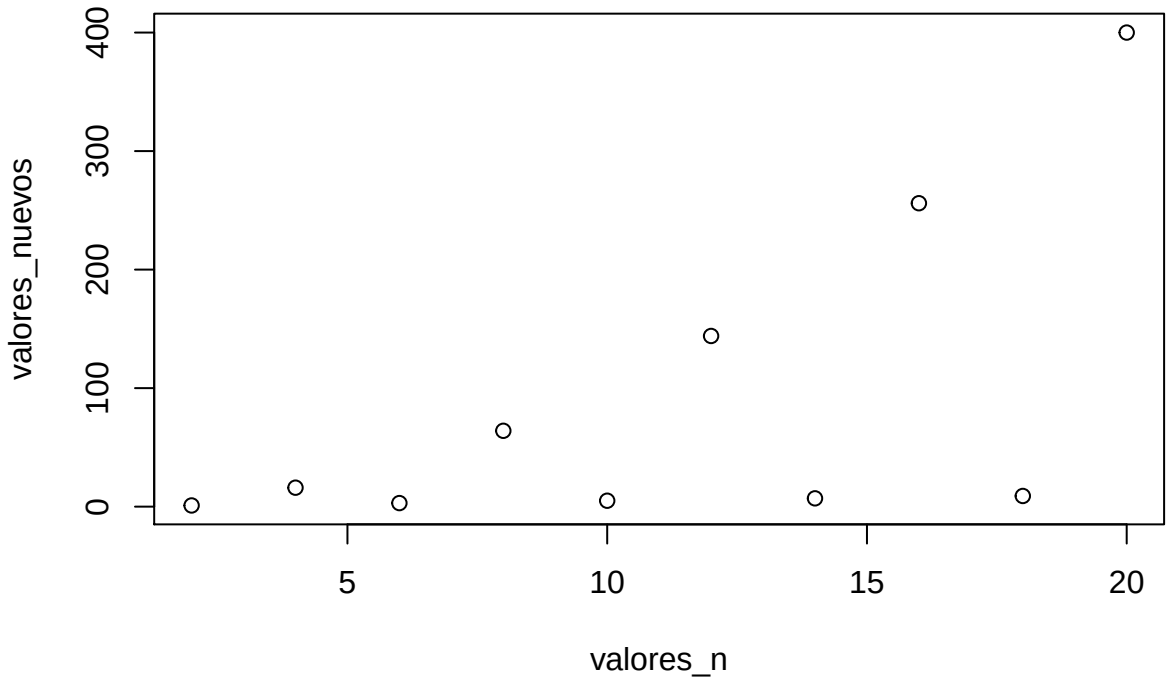
(20) Dados los valores calculados en el ejecicio anterior, graficarlos con el comando `plot`

```r
n <- 10*2
valores_n <- seq(2, n, 2)
valores_nuevos <- c()
```

```
for (i in valores_n) {
  if (i %% 4 == 0 ){
    valores_nuevos <- c(valores_nuevos,  i**2)
  } else {
    valores_nuevos <- c(valores_nuevos,  i/2)
  }
}
valores_nuevos
```

```
##  [1]   1  16   3  64   5 144   7 256   9 400
```

```
plot(valores_n, valores_nuevos)
```



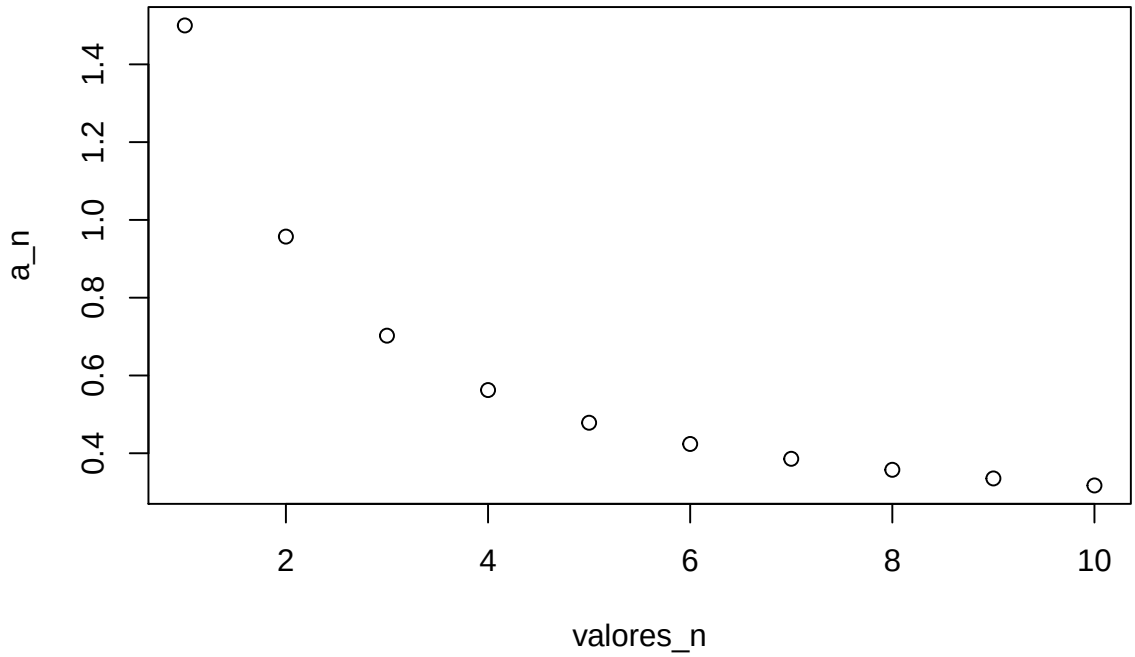
(21) ★ Escribir un programa que calcule y grafique los primeros 10 términos de las siguientes sucesiones:

a. $a_n = \frac{1}{\sqrt{n}} + \left(\frac{1}{2}\right)^n$

```
valores_n <- seq(1, 10)
a_n <- c()

for (i in valores_n) {
  nuevo_valor <- 1 / sqrt(i) + (1/2)**i
  a_n <- c(a_n,  nuevo_valor)
}
a_n
```

```
##  [1] 1.5000000 0.9571068 0.7023503 0.5625000 0.4784636 0.4238733 0.3857770
##  [8] 0.3574596 0.3352865 0.3172043
```

```r
plot(valores_n, a_n)
```

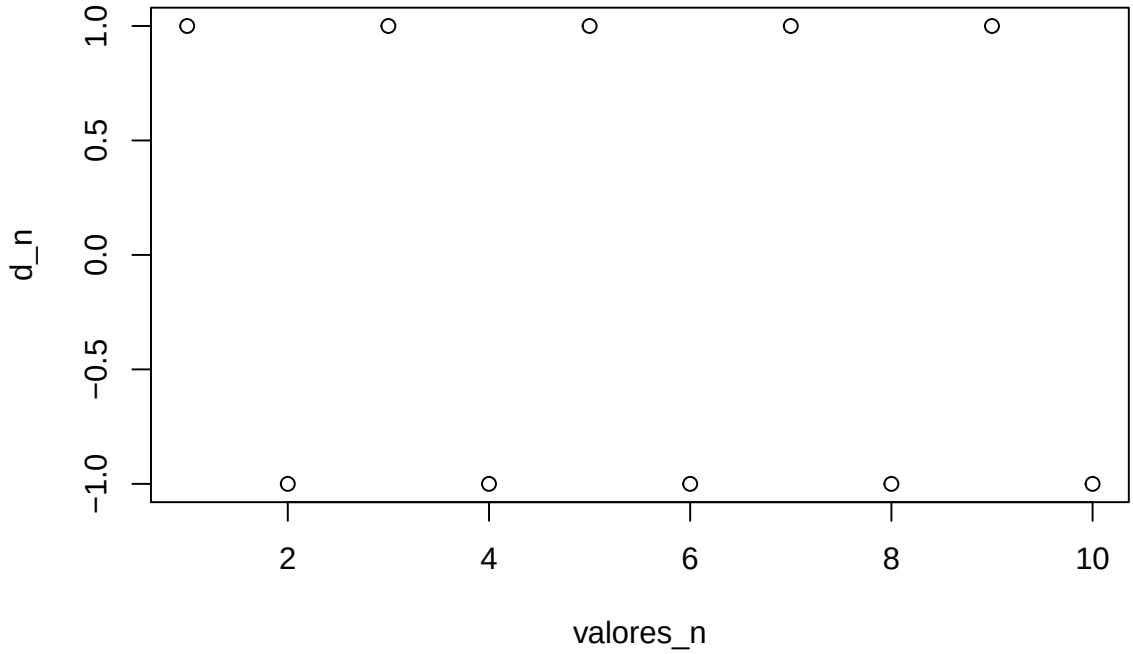


b. $d_n = (-1)^{n+5}$

```r
valores_n <- seq(1, 10)
d_n <- c()

for (i in valores_n) {
  nuevo_valor <- (-1)**(i+5)
  d_n <- c(d_n,  nuevo_valor)
}
d_n
```

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

```r
plot(valores_n, d_n)
```

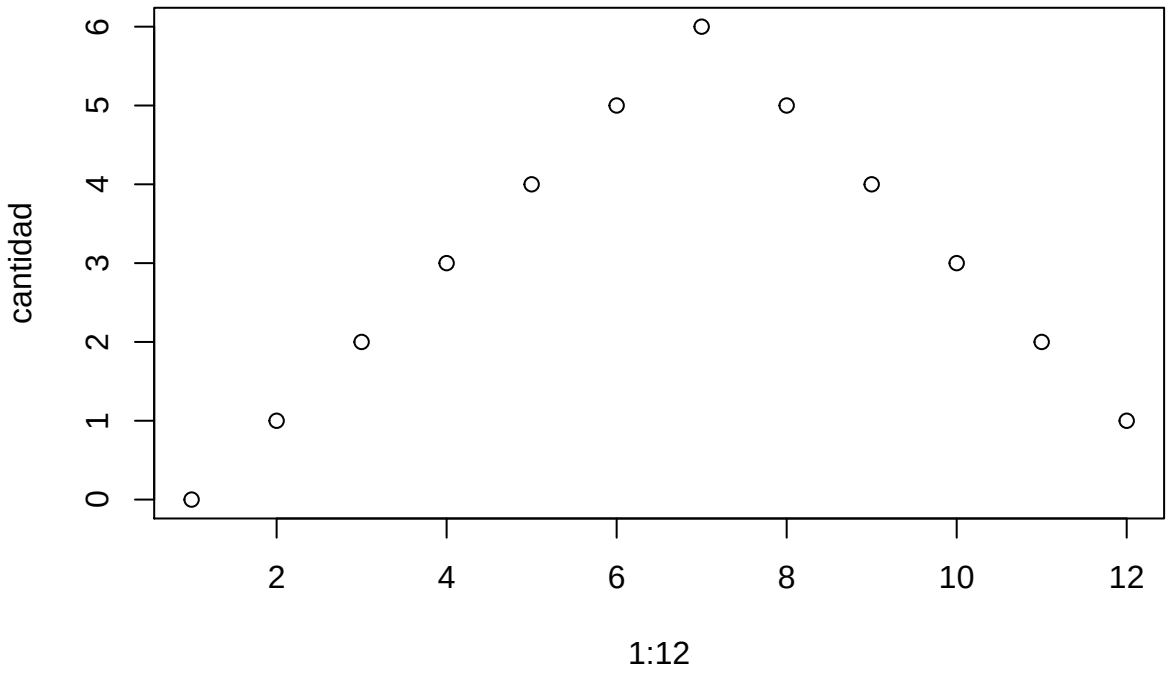


(22) ★ Retomando el ejercicio de las combinaciones de los dados, escribir un programa que inicie con un vector llamado `cantidad` de doce posiciones en cero (explorar el comando `rep`) y que para cada posición calcule cuántas combinaciones de los dados suman el valor indicado por la posición (ej. `cantidad[11]` vale 2, que son: (5,6) y (6,5)). Graficar los datos obtenidos.

```
cantidad <- rep(0,12)
for (dado_azul in 1:6){
  for (dado_rojo in 1:6){
    valor <- dado_azul + dado_rojo
    cantidad[valor] <- cantidad[valor] + 1
  }
}
cantidad
```

```
##  [1] 0 1 2 3 4 5 6 5 4 3 2 1
```

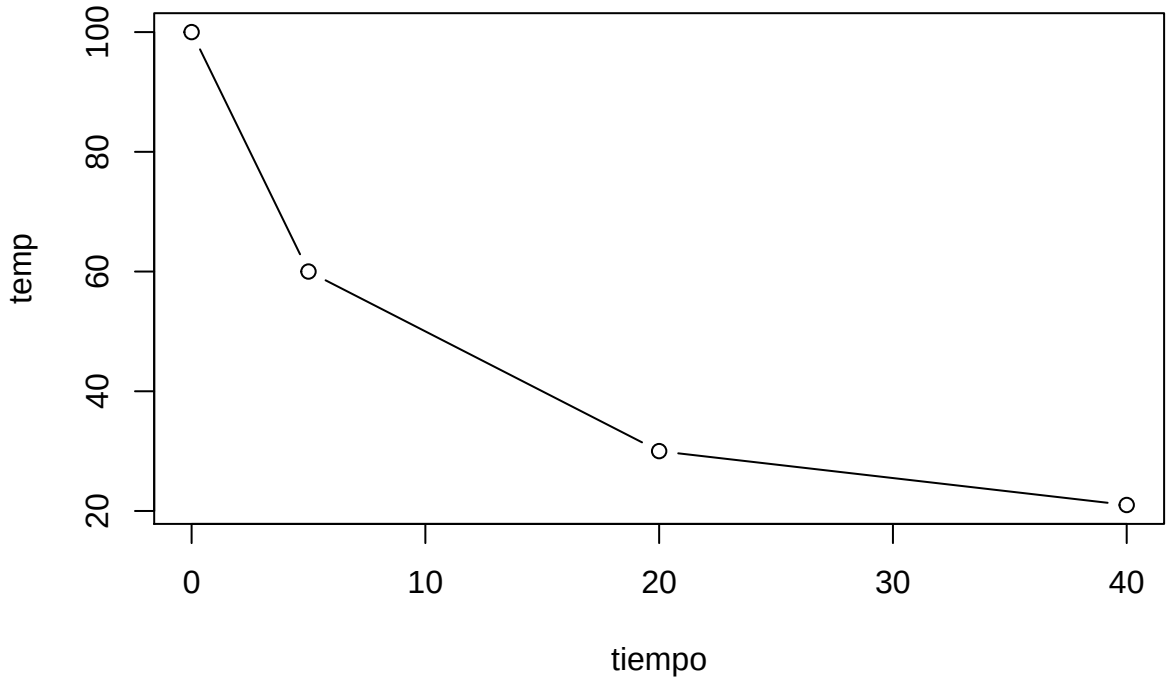
```
plot(1:12, cantidad)
```



(23) Hacé un gráfico que refleje la evolución de la temperatura del agua a lo largo del tiempo atendiendo a la siguiente descripción:

Saqué del fuego una cacerola con agua hirviendo. Al principio, la temperatura bajó con rapidez, de modo que a los 5 minutos estaba en 60 grados. Luego, fue enfriándose con más lentitud. A los 20 minutos de haberla sacado estaba en 30 grados y 20 minutos después seguía teniendo algo más de 20 grados, temperatura que se mantuvo, pues era la temperatura que había en la cocina.

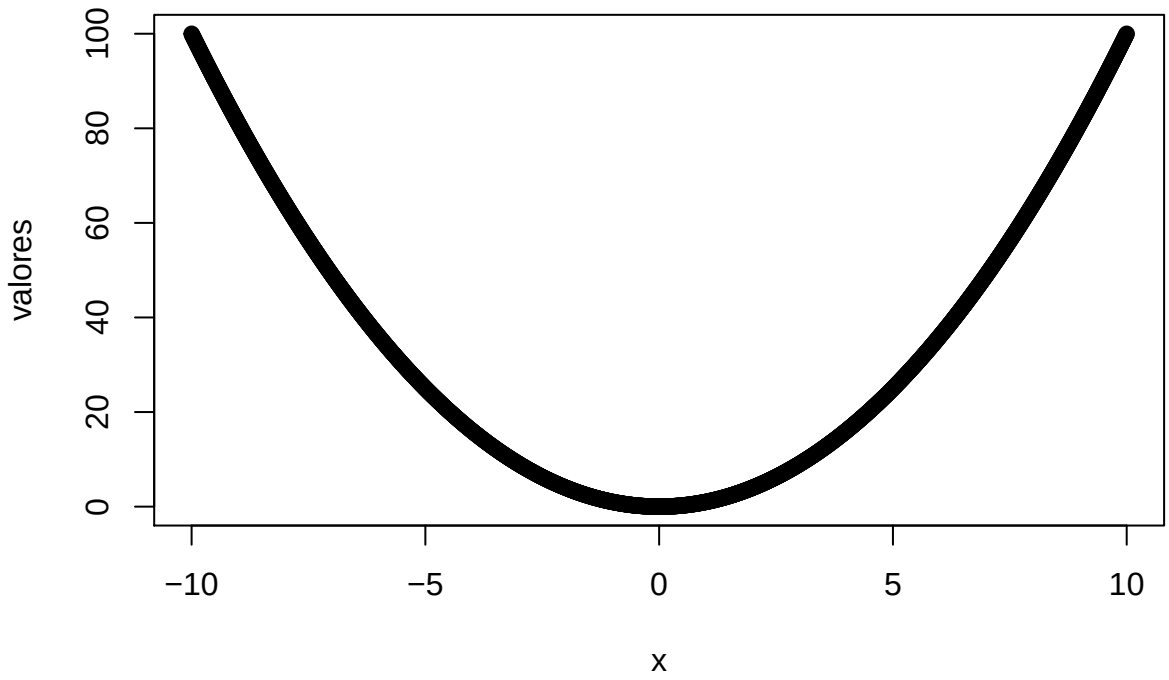
```
tiempo <- c(0,5,20,40)
temp <- c(100,60,30,21)
plot(tiempo, temp, type = "b")
```



(24) ★ Graficar la función $\rho(x) = x^2$, para $x \in [-10, 10]$.

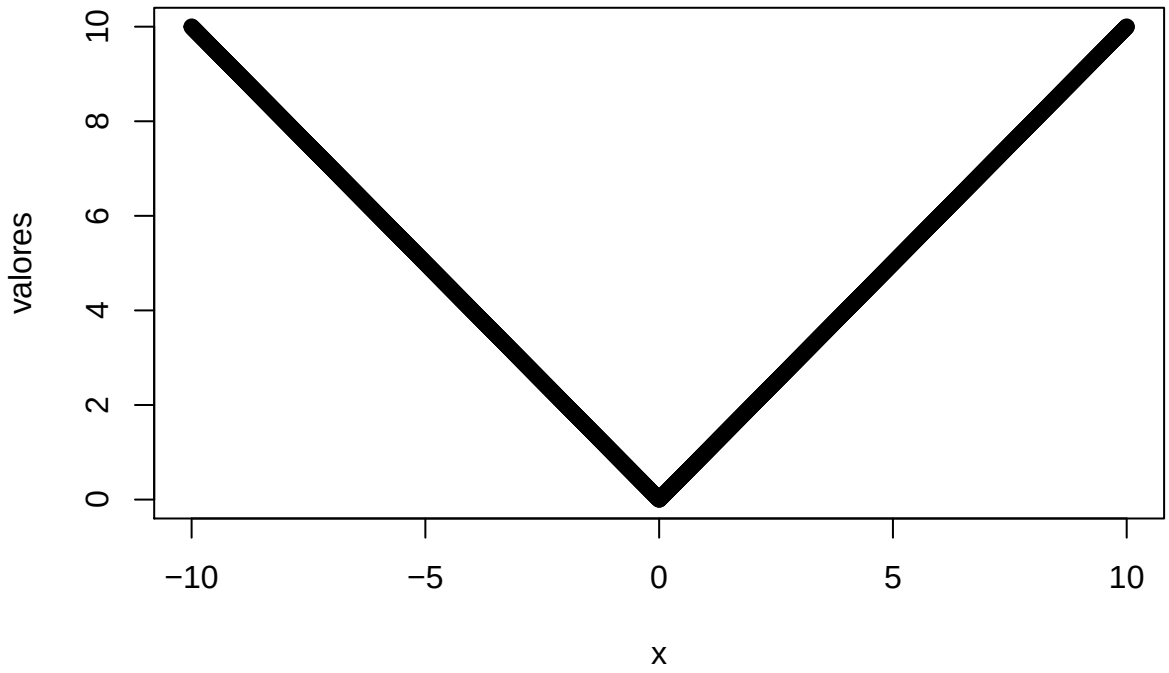
```
x <- seq(COMPLETAR, COMPLETAR, 0.01)
valores <- c()

for (i in x){
  nuevo_valor <- COMPLETAR
  valores <- COMPLETAR
}

plot(n, valores, type = "p")
```

```
x <- seq(-10, 10, 0.01)
valores <- c()

for (i in x){
  nuevo_valor <- i**2
  valores <- c(valores, nuevo_valor)
}

plot(x, valores, type = "p")
```



(25) Graficar la función $\rho(x) = |x|$, para $x \in [-10, 10]$.

```
x <- seq(-10, 10, 0.01)
valores <- c()

for (i in x){
  nuevo_valor <- abs(i)
  valores <- c(valores, nuevo_valor)
}

plot(x, valores, type = "p")
```

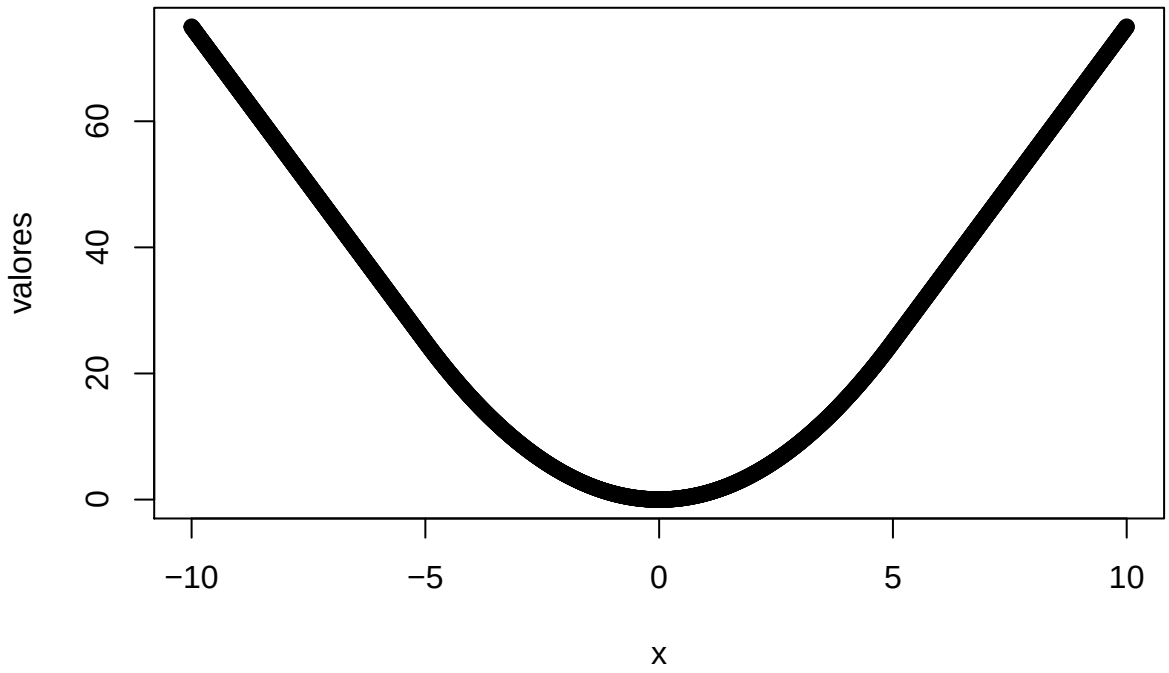


(26) ★ Considerar $\rho_k : \mathbb{R} \to \mathbb{R}$, definida de la siguiente manera

$$\rho_k(x) = \begin{cases} x^2 & \text{si} \quad |x| \leq k \\ 2k|x| - k^2 & \text{si} \quad |x| > k \end{cases}$$

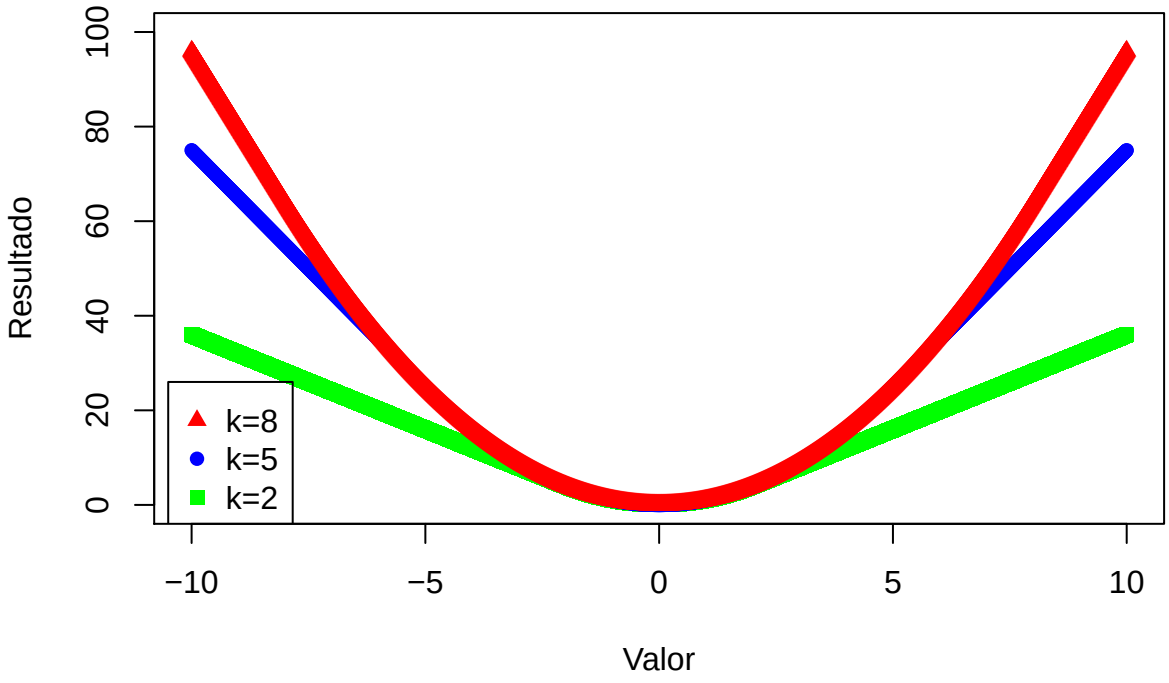
Calcular y graficar la la función ρ_k, con $k = 5$, para $x \in [-10, 10]$.

```r
x <- seq(-10, 10, 0.01)
valores <- c()
k <- 5

for (i in x){
  if (abs(i) <= k) {
    nuevo_valor <- i**2
  } else {
    nuevo_valor <- 2 * k * abs(i) - k**2
  }
  valores <- c(valores, nuevo_valor)
}

plot(x, valores, type = "p")
```



(27) En el mismo gráfico agregar el valor de la función ρ_k para $k = 2$ y $k = 8$ utilizando un color diferente para cada valor de k

```r
x <- seq(-10, 10, 0.01)
valores_2 <- c()
valores_5 <- c()
valores_8 <- c()

for (i in x){
  if (abs(i) <= 2) {
    valores_2 <- c(valores_2, i**2)
  } else {
    valores_2 <- c(valores_2, 2 * 2 * abs(i) - 2**2)
  }
  if (abs(i) <= 5) {
    valores_5 <- c(valores_5, i**2)
  } else {
    valores_5 <- c(valores_5, 2 * 5 * abs(i) - 5**2)
  }
  if (abs(i) <= 8) {
    valores_8 <- c(valores_8, i**2)
  } else {
    valores_8 <- c(valores_8, 2 * 8 * abs(i) - 8**2)
  }
}

plot(x, valores_2, type = "p", col="green",pch=15, ylim = c(0,100), ylab = "Resultado", xlab = "V
points(x, valores_5, col="blue",pch=16)
points(x, valores_8, col="red",pch=17)
legend(-10.5,26,legend=c("k=8","k=5","k=2"), col=c("red","blue","green"), pch = c(17,16,15))
```



3 Funciones - DataFrames

Esta es la guía de ejercicios correspondiente a la clase 03. En este caso la guía tiene 2 partes: Funciones y DataFrames. Cada parte se entregará por separado (con distintas fechas de entrega máxima, consultar en el campus). Deberá entregar todos los ejercicios en un archivo .R. Cada ejercicio debe estar resuelto entre comentarios que indique secciones dentro del archivo.

Algunos de los temas necesarios para resolver esta guía no fueron incluidos en la teórica, muchos se encuentran en este documento, mientras que otros deberán ser investigados (por ejemplo, buscando en internet). El ejercicio de buscar cómo abordar/resolver problemas en internet es *casi tan* importante como poder resolverlos.

Además de escribir los programas pedidos deberán probarlos y dejar constancia de las pruebas realizadas, además de explicitar si anduvo como era esperado o no.

3.1 Parte A: Funciones

1. Se desea tener una función que tome un valor como parámetro que representa una temperatura en grados Farenheit, y retorne el valor equivalente en Celsius. Más info https://www.lmgtfy.es/?q=formula+formula+fahrenheit+a+celsius. Puede tomar como referencia el siguiente código incompleto:

```
de_celsius_a_farenheit <- function(COMPLETAR){
  res <- COMPLETAR
  return(res)
}
```

```
de_celsius_a_farenheit <- function(grados_c){
  res <- grados_c * 9/5 + 32
  return(res)
}

de_celsius_a_farenheit(30)
```

```
## [1] 86
```

```r
de_celsius_a_farenheit(10)
```

```
## [1] 50
```

2. Escribir otra que se comporte a la inversa.

```r
de_farenheit_a_celsius <- function(grados_f){
  res <- (grados_f  - 32) * 5/9
  return(res)
}

de_farenheit_a_celsius(90)
```

```
## [1] 32.22222
```

```r
de_farenheit_a_celsius(45)
```

```
## [1] 7.222222
```

```r
de_farenheit_a_celsius(de_celsius_a_farenheit(20))
```

```
## [1] 20
```

3. Escribir una función que junte las funcionalidades de las 2 anteriores. Para hacerlo agregue un parámetro extra que si es TRUE la entrada se asume en Celsius (y se espera la salida en Farenheit) y si es FALSE se asume la entrada en Farenheit.

```r
calcular_equivalencia <- function(grados, en_celsius){
  COMPLETAR

  COMPLETAR
  return(res)
}
```

```r
calcular_equivalencia <- function(grados, en_celsius){
  if(en_celsius){
    res <- de_celsius_a_farenheit(grados)
  } else {
    res <- de_farenheit_a_celsius(grados)
  }
  return(res)
}

calcular_equivalencia(10, TRUE)
```

```
## [1] 50
```

```r
calcular_equivalencia(50, TRUE)
```

```
## [1] 122
```

```r
calcular_equivalencia(50, FALSE)
```

```
## [1] 10
```

4. Escribir una función que tomes 2 parámetros, el tamaño de una lado y la cantidad de lados de un polígono regular y que devuelva el perímetro

```r
perimetro <- function(lado, cantidad){
    res <- lado * cantidad
    return(res)
```

```
}

perimetro(8,3)
```

```
## [1] 24
```

```
perimetro(10,3)
```

```
## [1] 30
```

```
perimetro(10,4)
```

```
## [1] 40
```

```
perimetro(9,15)
```

```
## [1] 135
```

5. Escribir una función que tome un vector (que se asume tiene al menos 1 elemento) y un segundo parámetro e indique si dicho vector lo contiene o no. Dar 2 implementaciones:

   a. Con ciclos

```
contiene <- function(vs, i){
  res <- FALSE
  for (v in vs) {
    if (v==i) {
      res <- TRUE
    }
  }
  return(res)
}

contiene(c(1,3,5,7), 2)
```

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

```
contiene(c(1,3,5,7), 3)
```

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

   b. Sin ciclos (con funciones de vectores)

```
contiene_v2 <- function(vs, i){
  res <- any(vs==i)
  return(res)
}

contiene_v2(c(1,3,5,7), 2)
```

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

```
contiene_v2(c(1,3,5,7), 3)
```

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

6. Escribir una función que tome la altura y el peso de una persona y devuelva el índice de masa corporal (`IMC = peso / altura^2`).

```
peso <- 80
altura <- 1.78
```

```r
imc <- peso / altura^2

imc
```

```
## [1] 25.24934
```

7. Escribir una función que tome un valor `a` y un vector `vec` (que asumimos contiene a `a`). La misma deberá recorrer el vector **con un ciclo** y retornar el número de posición en la se encuentra `a`.

```r
posicion <- function(a, vec){
  i <- 1
  while (vec[i]!=a) {
    i <- i + 1
  }
  return(i)
}

posicion(3, c(1,2,4,5,6,3))
```

```
## [1] 6
```

```r
posicion(4, c(4,4,4,4))
```

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

8. Escribir una función que tome un valor `a` y un vector `vec`. La misma deberá indicar cuántas veces el valor a aparece en el vector vec. Pensar 2 implementaciones: con `for` y con `while`.

```r
cantidad_apariciones <- function(a, vec){
  i <- 1
  contador <- 0
  while (i<=length(vec)) {
    if (vec[i]==a){
      contador <- contador + 1
    }
    i <- i + 1
  }
  return(contador)
}

cantidad_apariciones_2 <- function(a, vec){
  contador <- 0
  for (elem in vec) {
    if (elem==a){
      contador <- contador + 1
    }
  }
  return(contador)
}

cantidad_apariciones(3, c(1,2,4,5,6,3))
```

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

```r
cantidad_apariciones_2(3, c(1,2,4,5,6,3))
```

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

```r
cantidad_apariciones(4, c(4,4,4,4))
```

```
## [1] 4
```

```r
cantidad_apariciones_2(4, c(4,4,4,4))
```

```
## [1] 4
```

9. Escribir una función que calcule la suma de los primeros n naturales. No vale usar fórmulas cerradas, ni funciones pre-definidas que lo calcule, implementarla usando ciclos `while`.

```r
suma_primeros_n <- function(n){
  suma <- 0
  while (n > 0) {
    suma <- suma + n
    n <- n - 1
  }
  return(suma)
}

suma_primeros_n(3)
```

```
## [1] 6
```

```r
suma_primeros_n(10)
```

```
## [1] 55
```

10. Escribir una función que calcule el producto de los primeros n naturales. No vale usar fórmulas cerradas, ni funciones pre-definidas que lo calcule, implementarla usando ciclos `while`.

```r
prod_primeros_n <- function(n){
  suma <- 0
  while (n > 0) {
    suma <- suma * n
    n <- n - 1
  }
  return(suma)
}

prod_primeros_n(3)
```

```
## [1] 0
```

```r
prod_primeros_n(10)
```

```
## [1] 0
```

11. Escribir una función que dado un número natural calcule un paso de la función de Collatz. Recordar que dice:

$$collatz(n) = \begin{cases} n/2 & \text{si n es par} \\ (3n+1) & \text{si no} \end{cases}$$

```r
un_collatz <- function(n) {
  if (n %% 2 == 0) {
    n <- n / 2
  } else {
```

```
    n <-  3 * n + 1
  }
  return(n)
}

un_collatz(7)
```

```
## [1] 22
```

```
un_collatz(32)
```

```
## [1] 16
```

```
un_collatz(27)
```

```
## [1] 82
```

12. Escribir una función que dado un número natural calcula la secuencia generada por la función de Collatz hasta alcanzar el 1.

```
sec_collatz <- function(n) {
  res <- c(n)
  while (n!=COMPLETAR) {
    n <- COMPLETAR
    res <- COMPLETAR
  }
  return(res)
}
```

```
sec_collatz <- function(n) {
  res <- c(n)
  while (n!=1) {
    n <- un_collatz(n)
    res <- c(res, n)
  }
  return(res)
}

sec_collatz(7)
```

```
##  [1]  7 22 11 34 17 52 26 13 40 20 10  5 16  8  4  2  1
```

```
sec_collatz(32)
```

```
## [1] 32 16  8  4  2  1
```

```
sec_collatz(27)
```

```
##   [1]   27   82   41  124   62   31   94   47  142   71  214  107  322  161  484
##  [16]  242  121  364  182   91  274  137  412  206  103  310  155  466  233  700
##  [31]  350  175  526  263  790  395 1186  593 1780  890  445 1336  668  334  167
##  [46]  502  251  754  377 1132  566  283  850  425 1276  638  319  958  479 1438
##  [61]  719 2158 1079 3238 1619 4858 2429 7288 3644 1822  911 2734 1367 4102 2051
##  [76] 6154 3077 9232 4616 2308 1154  577 1732  866  433 1300  650  325  976  488
##  [91]  244  122   61  184   92   46   23   70   35  106   53  160   80   40   20
## [106]   10    5   16    8    4    2    1
```

13. Escribir una función que dado un número `n` calcule (y devuelva) los primeros `n` términos de las siguientes sucesiones:

a. $a_n = \frac{1}{\sqrt{n}} + \left(\frac{1}{2}\right)^n$

```
func_a <- function(n) {
  a_n <- c()
  for (i in 1:n) {
    nuevo_valor <- 1 / sqrt(i) + (1/2)**n
    a_n <- c(a_n,  nuevo_valor)
  }
  return(a_n)
}

func_a(2)
```

```
## [1] 1.2500000 0.9571068
```

```
func_a(20)
```

```
##  [1] 1.0000010 0.7071077 0.5773512 0.5000010 0.4472145 0.4082492 0.3779654
##  [8] 0.3535543 0.3333343 0.3162287 0.3015123 0.2886761 0.2773511 0.2672622
## [15] 0.2581998 0.2500010 0.2425366 0.2357032 0.2294167 0.2236078
```

b. $d_n = (-1)^{n+5}$

```
func_d <- function(n) {
  d_n <- c()
  for (i in 1:n) {
    nuevo_valor <- (-1)**(i+5)
    d_n <- c(d_n,  nuevo_valor)
  }
  return(d_n)
}

func_d(2)
```

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

```
func_d(20)
```

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

14. Recordar $\rho_k : \mathbb{R} \to \mathbb{R}$, definida de la siguiente manera

$$\rho_k(x) = \begin{cases} x^2 & \text{si} \quad |x| \leq k \\ 2k|x| - k^2 & \text{si} \quad |x| > k \end{cases}$$

Escribir una función que calcule la función ρ_k tomando como parámetros `desde` y `hasta` para indicar el intervalo; `paso` para indicar cada cuanto se debe tomar un punto, y `k` que debe tener valor por defecto 5. Deberá poder usarse de la siguiente manera:

```
rho_func(-10, 10, 0.1)
rho_func(-5, 2, 0.03, 8)
```

```
rho_func <- function(desde, hasta, paso, k=5) {
  valores <- c()

  for (i in seq(desde, hasta, paso)){
    if (abs(i) <= k) {
      nuevo_valor <- i**2
```

```
    } else {
      nuevo_valor <- 2 * k * abs(i) - k**2
    }
    valores <- c(valores, nuevo_valor)
  }
  return(valores)
}

rho_func(-10, 10, 0.1)
```

```
##   [1] 75.00 74.00 73.00 72.00 71.00 70.00 69.00 68.00 67.00 66.00 65.00 64.00
##  [13] 63.00 62.00 61.00 60.00 59.00 58.00 57.00 56.00 55.00 54.00 53.00 52.00
##  [25] 51.00 50.00 49.00 48.00 47.00 46.00 45.00 44.00 43.00 42.00 41.00 40.00
##  [37] 39.00 38.00 37.00 36.00 35.00 34.00 33.00 32.00 31.00 30.00 29.00 28.00
##  [49] 27.00 26.00 25.00 24.01 23.04 22.09 21.16 20.25 19.36 18.49 17.64 16.81
##  [61] 16.00 15.21 14.44 13.69 12.96 12.25 11.56 10.89 10.24  9.61  9.00  8.41
##  [73]  7.84  7.29  6.76  6.25  5.76  5.29  4.84  4.41  4.00  3.61  3.24  2.89
##  [85]  2.56  2.25  1.96  1.69  1.44  1.21  1.00  0.81  0.64  0.49  0.36  0.25
##  [97]  0.16  0.09  0.04  0.01  0.00  0.01  0.04  0.09  0.16  0.25  0.36  0.49
## [109]  0.64  0.81  1.00  1.21  1.44  1.69  1.96  2.25  2.56  2.89  3.24  3.61
## [121]  4.00  4.41  4.84  5.29  5.76  6.25  6.76  7.29  7.84  8.41  9.00  9.61
## [133] 10.24 10.89 11.56 12.25 12.96 13.69 14.44 15.21 16.00 16.81 17.64 18.49
## [145] 19.36 20.25 21.16 22.09 23.04 24.01 25.00 26.00 27.00 28.00 29.00 30.00
## [157] 31.00 32.00 33.00 34.00 35.00 36.00 37.00 38.00 39.00 40.00 41.00 42.00
## [169] 43.00 44.00 45.00 46.00 47.00 48.00 49.00 50.00 51.00 52.00 53.00 54.00
## [181] 55.00 56.00 57.00 58.00 59.00 60.00 61.00 62.00 63.00 64.00 65.00 66.00
## [193] 67.00 68.00 69.00 70.00 71.00 72.00 73.00 74.00 75.00
```

```
rho_func(-5, 2, 0.03, 8)
```

```
##   [1] 25.0000 24.7009 24.4036 24.1081 23.8144 23.5225 23.2324 22.9441 22.6576
##  [10] 22.3729 22.0900 21.8089 21.5296 21.2521 20.9764 20.7025 20.4304 20.1601
##  [19] 19.8916 19.6249 19.3600 19.0969 18.8356 18.5761 18.3184 18.0625 17.8084
##  [28] 17.5561 17.3056 17.0569 16.8100 16.5649 16.3216 16.0801 15.8404 15.6025
##  [37] 15.3664 15.1321 14.8996 14.6689 14.4400 14.2129 13.9876 13.7641 13.5424
##  [46] 13.3225 13.1044 12.8881 12.6736 12.4609 12.2500 12.0409 11.8336 11.6281
##  [55] 11.4244 11.2225 11.0224 10.8241 10.6276 10.4329 10.2400 10.0489  9.8596
##  [64]  9.6721  9.4864  9.3025  9.1204  8.9401  8.7616  8.5849  8.4100  8.2369
##  [73]  8.0656  7.8961  7.7284  7.5625  7.3984  7.2361  7.0756  6.9169  6.7600
##  [82]  6.6049  6.4516  6.3001  6.1504  6.0025  5.8564  5.7121  5.5696  5.4289
##  [91]  5.2900  5.1529  5.0176  4.8841  4.7524  4.6225  4.4944  4.3681  4.2436
## [100]  4.1209  4.0000  3.8809  3.7636  3.6481  3.5344  3.4225  3.3124  3.2041
## [109]  3.0976  2.9929  2.8900  2.7889  2.6896  2.5921  2.4964  2.4025  2.3104
## [118]  2.2201  2.1316  2.0449  1.9600  1.8769  1.7956  1.7161  1.6384  1.5625
## [127]  1.4884  1.4161  1.3456  1.2769  1.2100  1.1449  1.0816  1.0201  0.9604
## [136]  0.9025  0.8464  0.7921  0.7396  0.6889  0.6400  0.5929  0.5476  0.5041
## [145]  0.4624  0.4225  0.3844  0.3481  0.3136  0.2809  0.2500  0.2209  0.1936
## [154]  0.1681  0.1444  0.1225  0.1024  0.0841  0.0676  0.0529  0.0400  0.0289
## [163]  0.0196  0.0121  0.0064  0.0025  0.0004  0.0001  0.0016  0.0049  0.0100
## [172]  0.0169  0.0256  0.0361  0.0484  0.0625  0.0784  0.0961  0.1156  0.1369
## [181]  0.1600  0.1849  0.2116  0.2401  0.2704  0.3025  0.3364  0.3721  0.4096
## [190]  0.4489  0.4900  0.5329  0.5776  0.6241  0.6724  0.7225  0.7744  0.8281
## [199]  0.8836  0.9409  1.0000  1.0609  1.1236  1.1881  1.2544  1.3225  1.3924
## [208]  1.4641  1.5376  1.6129  1.6900  1.7689  1.8496  1.9321  2.0164  2.1025
```

```
## [217]  2.1904  2.2801  2.3716  2.4649  2.5600  2.6569  2.7556  2.8561  2.9584
## [226]  3.0625  3.1684  3.2761  3.3856  3.4969  3.6100  3.7249  3.8416  3.9601
```

15. **El polinomio interpolador de Lagrange**. Dada una tabla de puntos $\{(x_i, y_i)\}_{i=0,\dots,n}$, con distintos valores para los x_i, existe un único polinomio $p_n(x)$ de grado menos o igual a n tal que

$$p_n(x_i) = y_i \ , \quad i = 0, \dots, n \ .$$

Observacion: los puntos de la tabla estan indexados empezando en $i = 0$; es decir, hay $n + 1$ puntos. Por ejemplo, si consideramos dos puntos, tenemos (x_0, y_0), (x_1, y_1) y el polinomio interpolador resulta la recta que pasa por los dos puntos, recordando que la recta es un polinomio de grado $n = 1$.

El polinomio se calcula cuando tenemos tres puntos según: (ver clase 1 del módulo de análisis).

Generalización:

- ¿Cómo generalizamos este procedimiento a mís puntos? Definimos

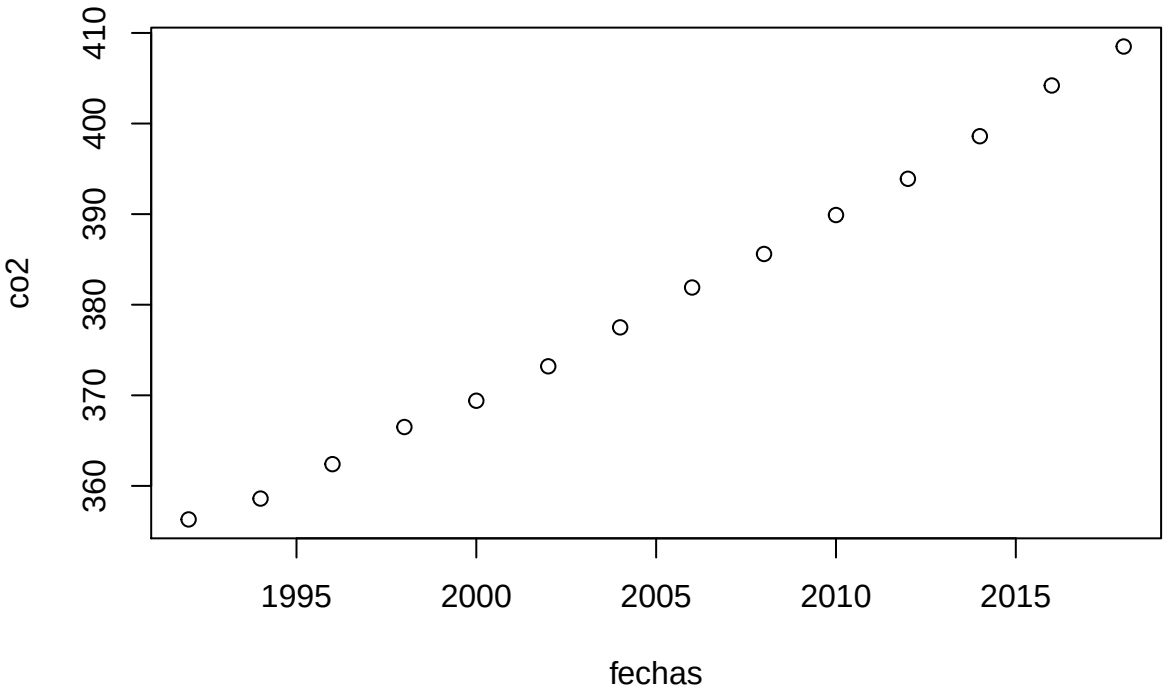
$$l_i(x) = \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)} \ , i = 0, \dots, n \ .$$

- ¿Cuál es la gracia? Simple, $l_i(x_i) = 1$ mientras que $l_i(x_j) = 0$ si $j \neq i$. Luego, el polonomio interpolador está definido por

$$p_n(x) = \sum_{i=0}^{n} y_i l_i(x) \ .$$

Miramos los datos de co2 que ya vimos en el módulo de análisis:

```
fechas <- seq(from=1992, to=2018, by=2)
co2 <- c(356.3, 358.6, 362.4, 366.5, 369.4, 373.2, 377.5, 381.9,
         385.6, 389.9, 393.9, 398.6, 404.2, 408.5)

plot(fechas, co2)
```



Definir:

a. Las funciones `sumar_todos` y `multiplicar_todos` que toman un vector y devuelven la suma y el producto de todos los elementos respectivamente.

```
sumar_todos <- function(vec) {
  suma <- 0
  for(i in vec) {
    suma <- suma + i
  }
  return(suma)
}
```

```
multiplicar_todos <- function(vec) {
  producto <- 1
  for(i in vec) {
    producto <- producto * i
  }
  return(producto)
}
```

b. Completar la siguiente función para que realice la interpolación según la fórmula previa. Recordar que para excluir un elemento de un vector se puede poner: `vec[-posicion]`. Más abajo ejemplo de con resulados para probar.

```
interpolador <- function(x_nuevo, x_datos, y_datos) {
  n_puntos <- length(COMPLETAR)
  l <- c()

  # Calculamos l_i(x)
  for(i in 1:n_puntos) {
    numerador <- multiplicar_todos(COMPLETAR - x_datos[-i])
    denominador <- COMPLETAR(x_datos[i] - COMPLETAR)
    nuevo_dato <- numerador/denominador
    l <- c(l, COMPLETAR)
  }

  # Calculamos la sumatoria
  res <- COMPLETAR(y_datos * l)
  return(COMPLETAR)
}
```

```
interpolador <- function(x_nuevo, x_datos, y_datos) {
  n_puntos <- length(x_datos)
  l <- c()
  for(i in 1:n_puntos) {
    numerador <- multiplicar_todos(x_nuevo-x_datos[-i])
    denominador <- multiplicar_todos(x_datos[i]-x_datos[-i])
    nuevo_dato <- numerador/denominador
    l <- c(l, nuevo_dato)
  }
  res <- sumar_todos(y_datos * l)
  return(res)
}
```

```
interpolador(2, c(2,3,5), c(4,6,10))
```

```
## [1] 4
```

```r
interpolador(4, c(2,3,5), c(4,6,10))
```

```
## [1] 8
```

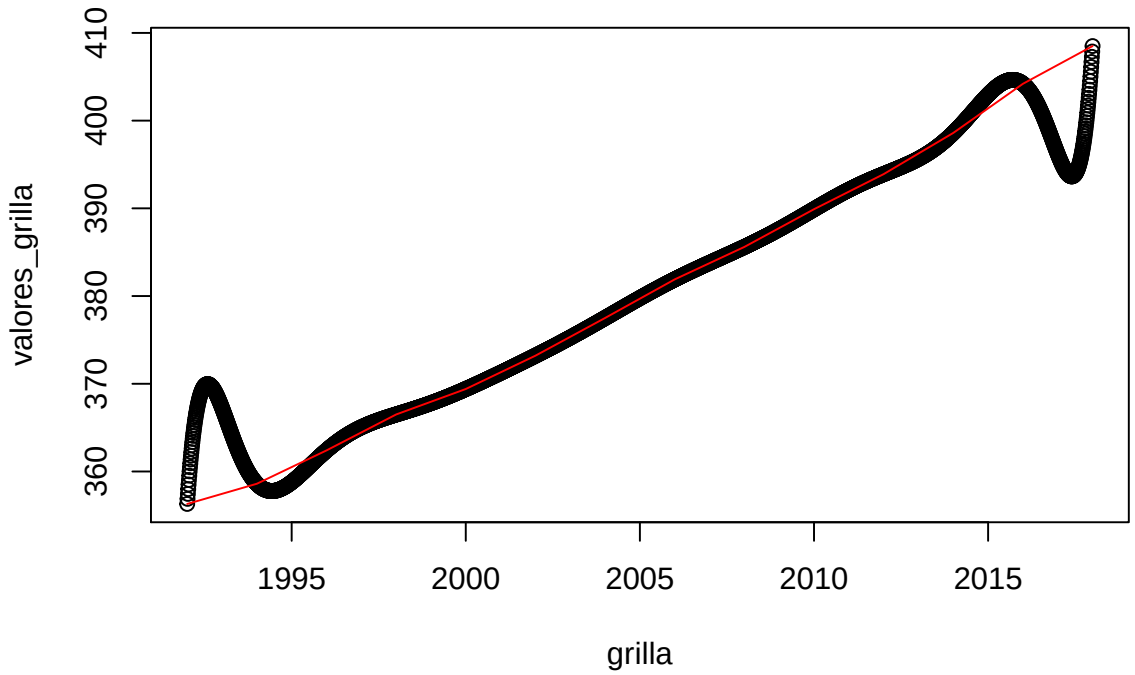
```r
interpolador(4.25, c(2,3,5), c(4,6,10))
```

```
## [1] 8.5
```

c. Completar el siguiente código para poder interpolar sobre un vector de puntos, probarla graficando los puntos interpolados y una recta con los puntos originales.

```r
interpolar_grilla <- function(datos_nuevos, x_datos, y_datos) {
 res <- c()
  for (d in datos_nuevos){
    nuevo <- interpolador(COMPLETAR)
    res <- COMPLETAR
  }
 return(res)
}

# hago una grilla con puntos intermedios
grilla <- seq(fechas[1], fechas[length(fechas)], 0.01)
valores_grilla <- interpolar_grilla(grilla, fechas, co2)
```

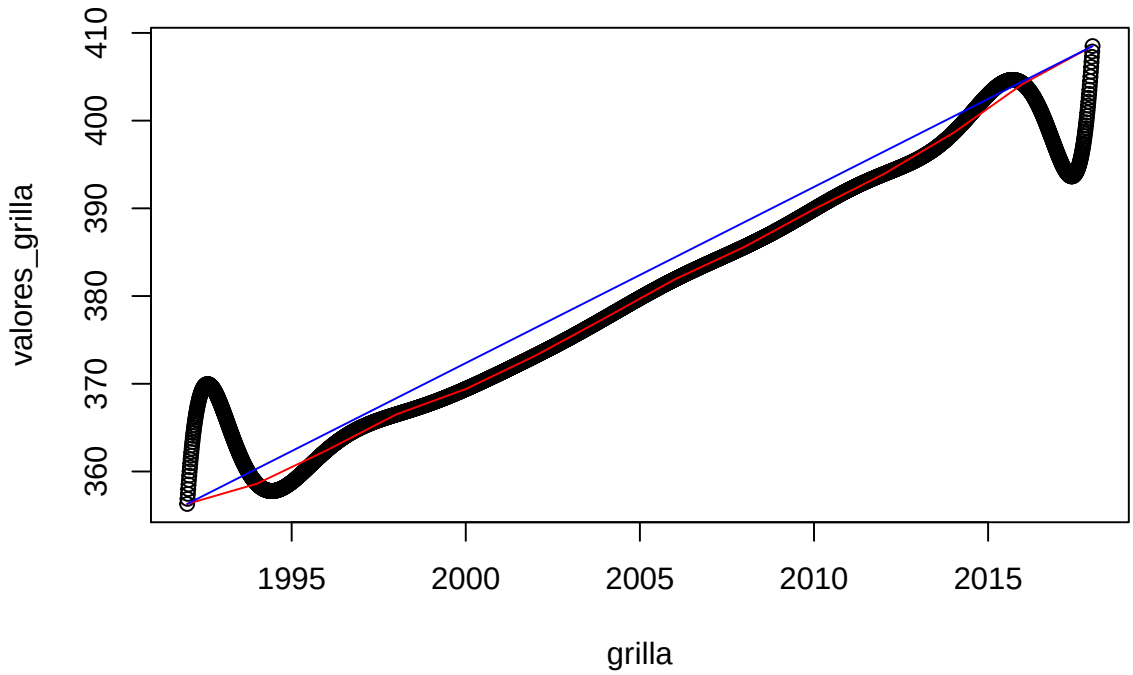
```r
interpolar_grilla <- function(datos_nuevos, x_datos, y_datos) {
 res <- c()
  for (d in datos_nuevos){
    nuevo <- interpolador(d, x_datos, y_datos)
    res <- c(res, nuevo)
  }
 return(res)
}

grilla <- seq(fechas[1], fechas[length(fechas)], 0.01)
valores_grilla <- interpolar_grilla(grilla, fechas, co2)

plot(grilla, valores_grilla)
lines(fechas, co2, col="red")
```



d. Agregar una recta interpolando únicamente por el primer y último punto de los datos y agregarla al gráfico anterior con otro color

```
grilla <- seq(fechas[1], fechas[length(fechas)], 0.01)
valores_grilla_nueva <- interpolar_grilla(COMPLETAR,
                          fechas[c(1,length(fechas))], co2[COMPLETAR])
```

```
grilla <- seq(fechas[1], fechas[length(fechas)], 0.01)
valores_grilla_nueva <- interpolar_grilla(grilla, fechas[c(1,length(fechas))], co2[c(1,length(

plot(grilla, valores_grilla)
lines(fechas, co2, col="red")
lines(grilla, valores_grilla_nueva, col="blue")
```



16. Para cada uno de los conjuntos de datos dados, calcular y grafocar el polinomio $p(x)$ interpolador de grado menor o igual que 3 .

    a. Primer conjunto:

    x

    y

    -1

    -1

    0

    3

    2

    11

    3

    27

    b. Segundo conjunto:

    x

    y

    -1

    -3

    0

    1

    2

    1

3

3

3.2 Parte B: DataFrames

Vamos a trabajar con la base de datos *Iris*. Esta es una base de datos muy conocida y utilizada en cursos introductorios. Para más información mirá acá. Por defecto, la base de datos *Iris* viene con la instalación de R. Si queres ver como se ve hace:

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

1. Describir a la base de datos. ¿Cuantas filas tiene?. ¿Y columnas?. ¿Cuantas variables tiene?
2. Seleccionar las filas que pertenezcan a la especie *versicolor*.¿Cuantos individuos fueron seleccionados?.

   ```
   nrow(iris[iris$Species=="versicolor",])
   ```

   ```
   ## [1] 50
   ```

3. Seleccionar solo la columna *Petal.Length*.¿Cual es la media de la longitud de los petalos?.

   ```
   mean(iris$Petal.Length)
   ```

   ```
   ## [1] 3.758
   ```

4. Si suponemos que el area del pétalo se puede estimar como (largo * ancho * π), ¿Cuanto es el área promedio del pétalo de la especie *setosa*?.

   ```
   mean(iris[iris$Species=="setosa",3]*iris[iris$Species=="setosa",4]*pi)
   ```

   ```
   ## [1] 1.148566
   ```

5. Calcular la frecuencia de individuos de la especie *virginica* que poseen un `Sepal.Width` mayor a 3.

   ```
   nrow(iris[iris$Species=="virginica" & iris$Sepal.Width>3,])
   ```

   ```
   ## [1] 17
   ```

6. Calcular la frecuencia de individuos de la especie *setosa* que poseen un `Sepal.Length` mayor a 5 y un `Petal.Width` menor a 0.3.

   ```
   nrow(iris[iris$Species=="setosa" & iris$Sepal.Length>5 & iris$Petal.Width<0.3,])
   ```

   ```
   ## [1] 12
   ```

7. Ahora agregale una columna al `df` que sea la suma de los cuatro atributos para cada individuo.

   ```
   iris$suma <- iris[,1]+iris[,2]+iris[,3]+iris[,4]
   ```

8. Asociación entre rasgos paternos y tamaño de la bola de cría en el escarabajo estercolero *Sulcophanaeus* sp.

   Los rasgos de los padres suelen afectar el desarrollo de rasgos de su cría. Un mecanismo por el cual los padres pueden influir en el fenotipo de la descendencia es a través del nivel de cuidado que proporcionan. *Sulcophanaeus* sp (Coleoptera, Scarabaeidae) es un escarabajo estercolero con cuidados biparentales.

El macho construye una bola de cría formada por estiércol, en la que la hembra deposita un huevo, que al eclosionar se alimenta de esa masa. Se sabe que cuanto mayor es el tamaño de la bola de cría, mayor es la disponibilidad de alimento y mayor el desarrollo de la cría, pero se desconocen los factores que regulan el tamaño de las bolas de cría. Los machos presentan además variaciones en el tamaño de sus cuernos, que utilizan durante las peleas por las hembras permitiendoles incrementar su éxito reproductivo. Se desea probar la hipótesis de que ambas características morfológicas de los machos (tamaño corporal y la longitud de los cuernos) afectan en forma sinérgica el tamaño de las bolas de cría. Para demostrarlo se capturaron ejemplares adultos en la provincia de Buenos Aires y se les midió el largo corporal total (LT) y del cuerno (LC), ambos en mm. Se seleccionaron machos de manera de cubrir el rango de combinaciones de tamaños y se los cruzó con una hembra de tamaño promedio. Cada pareja fue mantenida en una cámara de cría individual con estiércol de vaca. Se obtuvieron 75 bolas de cría, a las que se les determinó el peso seco (PS), en gramos. Los resultados se encuentran en el archivo escarab.csv (click para descargar)

Lea el archivo de datos y calcule los valores medios para cada una de las mediciones `LT`, `LC` y `PS`.

```
escarab <- read.csv("escarab.csv",header=T)
```

9. Hacer una función que dada una *variable de interés* devuelva la especie que posea la mayor media para ese atributo.

```
mean(escarab$LT)
```

```
## [1] 16.95747
```

```
mean(escarab$LC)
```

```
## [1] 9.528867
```

```
mean(escarab$PS)
```

```
## [1] 5.668827
```

10. Ahora hacer una función que dada una *variable de interés* devuelva la especie que posea la menor varianza para ese atributo.

3.2.1 Análisis exploratorio a partir del manejo de salidas gráficas

A continuación, vamos a aprender a realizar, manipular y almacenar salidas gráficas de R usando las funciones existentes. Hasta ahora habíamos usado `plot` que realizaba varias cosas automáticamente, como por ejemplo decidir dónde se iba a generar un gráfico (a esto se lo llama dispositivo). Hay diferentes formas de crear dispositivos donde realizar gráficos, por ejemplo un archivo con extensión pdf, jpg, o bien la pantalla de nuestra computadora. Si queremos que el gráfico aparezca en la pantalla, al ejecutar la función windows() (para el Sistema Operativo Windows), x11() (para Liunx) o quartz() (para Mac), nos devolverá una nueva ventana donde se empezará a generar el gráfico que queramos realizar.

En términos generales, la ventana de dibujo en R puede dividirse en tres partes: un área de dibujo, un margen interno y un margen externo. La función *par()* permite ajustar el tamaño de los márgenes (abajo, izquierda, arriba, derecho), a partir del ajuste de los parámetros *mar* (margin size) y *oma* (outer margin area).

Para generar distintos sub-gráficos en una misma figura pueden usarse los parámetros *mfrow* o *mfcol* de la función *par()*.

1. Realizar los siguientes gráficos y analizá qué significan los parámetros *main*, *xlab*, *ylab*, *pch* y *col*.

```
plot(iris$Sepal.Length, iris$Sepal.Width, col=iris$Species)
```

```
pairs(iris[,2:5], pch=as.numeric(iris$Species))
```

```
hist(iris$Sepal.Length, ylab="Frecuencia", xlab="Longitud del Sépalo")
```

```r
plot(iris$Sepal.Length~iris$Species, main="Longitud del sepalo por especie")
```

```r
barplot(tapply(iris$Sepal.Length,iris$Species,mean), main="Longitud del sepalo por especie")
```

3.2.2 Gráficos básicos: Probando modificar parámetros

1. Modificar el número de divisiones en el histograma utilizando el parámetro *breaks*. Una vez hecho esto, ingresar:

   ```r
   plot(density(iris$Sepal.Length), main="Densidad de LongSepalo")
   ```

2. Hacer en una ventana aparte un gráfico subdividido en cuatro donde se muestre el histograma de cada una de las variables medidas.
3. Hacer un gráfico adecuado para las 4 variables registradas en las 150 flores. Analizar qué se está representando e investigar la presencia de datos atípicos en los datos.
4. Hacer un boxplot para cada variable pero discriminando por especie.
5. Al boxplot del punto anterior rotularle los ejes con titulos adecuados.
6. Explorar como modificar los colores con los que se grafica cada especie.
7. Mover la posición de la leyenda del gráfico a otra zona.

4 Modelado: Álbum de figuritas

Esta es la guía de ejercicios correspondiente a la clase 04. Deberá entregar todos los ejercicios en un archivo .R. Cada ejercicio debe estar resuelto entre comentarios que indique secciones dentro del archivo.

Además de escribir los programas pedidos deberán probarlos y dejar constancia de las pruebas realizadas, además de explicitar si anduvo como era esperado o no.

4.1 Algunas herramientas útiles de R

Para la presente actividad tenga presente los siguientes comandos:

- `sample(x, size, replace=FALSE)`
- `sample(x, size, replace=TRUE)`
- `seq(from =, to =  by = )`
- `rep (x =, times = )`
- `sum(x)`
- `mean(x)`

Explorarlos y revisar su documentación antes de empezar.

4.2 El álbum de figuritas

El llenado de un álbum de figuritas es un fenómeno mundial de larga data que ha dejado recuerdos en mucha gente:

> *(. . .) A todos los que ahorraron el dinero de su bolsillo para completar el álbum Panini de México 86 con su figurita (. . .)*
>
> Emmanuel Macron (Presidente de Francia) ante la muerte de Diego Armando Maradona. Ver nota

También ha sido abordado desde las matemáticas:

> *Las figuritas del mundial: ¿cuántas figuritas hay que comprar para completar el álbum del Mundial?*
>
> Adrián Paenza; Página 12 - Contratapa 06/07/2014 Ver nota

4.2.1 Datos

Para este problema utilizaremos los siguientes datos:

- Álbum con 640 figuritas.
- Cada figurita se imprime en cantidades iguales y se distribuye aleatoriamente.
- Cada paquete trae cinco figuritas.

Los usaremos considerando distintos escenarios simplificados hasta alcanzar el modelo completo.

4.2.2 Primer modelo: Las figuritas se compran de a una

1. Simular el número de una figurita elegida al azar si el álbum es de 6 figuritas. ¿Qué comando sirve?
2. Simular el llenado de un álbum de 6 figuritas e indicar cuántas figuritas se debieron comprar para completarlo.
3. Implementar la función `cuantas_figus(figus_total)` que, dado el tamaño del álbum (`figus_total`), simule su llenado y devuelva la cantidad de figuritas que se debieron adquirir para completarlo.
4. Calcular `n_rep=1000` veces la función anterior utilizando `figus_total=6` y guardar los resultados obtenidos en cada repetición en un vector. Con los resultados obtenidos para un álbum de seis figuritas, estimar:
   a. El número de figuritas que hay que comprar, en promedio, para completar el álbum.
   b. La probabilidad de completar el álbum comprando 16 figuritas.
   c. El número de figuritas que hay que comprar para tener probabilidad del 90% de completar el álbum.

   Sobre variabilidad y convergencia - Pensar para discutir durante la próxima clase
5. Repetir el ítem a del punto anterior para los siguientes valores de `n_rep`: 200, 500, 1000, 5000, 10000. Gráficar `n_rep` (en el eje x) vs. los promedios calculados para cada valor.
6. Repetir el ítem anterior utilizando otro color para graficar los puntos. Vuelva a repetir. Comentar los resultados obtenidos.

   Extender el álbum considerando que tiene `figus_total=640` figuritas, manteniendo que las figuritas se compran individualmente. ‘
7. Calcular `n_rep=100` veces la función `cuantas_figus` para la nueva cantidad de `figus_total` y guardar los resultados obtenidos en cada repetición. Con los resultados obtenidos estimar cuántas figuritas hay que comprar, en promedio, para completar el álbum original (640 figuritas).

4.2.3 Segundo modelo: Las figuritas se compran por paquetes

1. Simular la generación de un paquete con 5 figuritas, sabiendo que el álbum es de 640. *Nota:* al igual que en la realidad, puede haber figuritas repetidas en un paquete.
2. Implementar la función `generar_paquete(figus_total, figus_paquete)` que dado el tamaño del álbum (`figus_total`) y la cantidad de figuritas por paquete (`figus_paquete`) genere un paquete de figuritas l azar. *Nota:* al igual que en la realidad, puede haber figuritas repetidas en un paquete.

3. Implementar la función `cuantos_paquetes(figus_total, figus_paquete)` que dado el tamaño del álbum (`figus_total`) y la cantidad de figuritas por paquete (`figus_paquete`) simule el llenado del álbum y retorne cuántos paquetes se debieron adquirir para completarlo.
4. Ejecutar `n_rep=100` veces la función `cuantos_paquetes`, utilizando `figus_total=640`, `figus_paquete=5` y guardando los resultados obtenidos en un vector y calcular el promedio.
5. *Optativo*: Utilizando lo implementado estimar la probabilidad de completar el álbum con 850 paquetes o menos.
6. *Optativo*: Utilizando lo implementado en estimar cuántos paquetes se deben comprar para tener una chance del 90% de completar el álbum. ‘

5 Modelado: La ruina del jugador

Esta es la guía de ejercicios correspondiente a la clase 05. Deberá entregar un archivo .R con las funciones implementadas y un informe sobre la exploración del problema en formato .Rmd y .pdf.

En esta oportunidad presentamos la última actividad del curso. Es un nuevo problema para seguir programando y aprovechamos la oportunidad para invitar a explorar *algunas bondades* de R, a la hora de escribir un informe. Por un lado, queremos que resuelvas la guía armando, en un archivo .R, todas las funciones que vas a utilizar a la hora de escribir un informe sobre el problema, pensando en un estudiante de la escuela media. Incluímos algunas pautas para ayudar en la elaboración del informe:

- Contar el problema.
- Mostrar resultados correspondientes a diferentes experimentaciones incluyendo gráficos y especificando que se representa en cada caso.
- Explorar en detalle el caso $p = 0.5$
- Conjeturar una fórmula en función de las pruebas realizadas.
- Incluir referencias.

5.1 Comandos útiles

- `sample(x, size, replace = FALSE, prob = NULL)`
- `runif(1)`
- `rep(x, ...)`

5.2 Recursos para elaborar el informe

Algunos recursos que pueden explorar:

- Tutorial interactivo online
- Introducción
- QuickTour
- Hoja de trucos

5.3 La ruina del jugador

Se trata de un desafío entre un jugador y el casino que consta de un indeterminado número de jugadas. En cada jugada, se lanza una moneda y quién pierde la jugada entrega una moneda al otro jugador. La partida concluye cuando uno de los dos participantes ha perdido todas sus monedas (De Santos et al. 2008).

A modo de ejemplo, supongamos que inicialmente Juan dispone de $j = 3$ monedas, mientras que el Casino tiene $c = 2$ monedas. Es decir, hay un total de $m = j + c$ monedas en juego. Cada jugada consiste en lanzar un dado; si sale un cuatro, gana Juan; caso contrario, gana el Casino. Es decir, en cada jugada, con probabilidad $p = 1/6$ Juan gana una moneda del Casino, mientras que con probabilidad $q = 1 - p = 5/6$ el

Casino gana una moneda de Juan. La partida continúa hasta que uno de los actores se queda sin monedas (esa es su ruina), mientras que su contrincante es quien gana la partida.

Proponemos este problema para explorar diferentes aspectos de la programación, como se sugiere a lo largo de la siguiente lista (tenga presente la diferencia entre **partida** y **jugada**):

1. Implemente una función `una_jugada` que dado p simule una jugada y devuelva `TRUE` si ganó Juan o `FALSE` en caso contrario.
2. Simule una jugada con $p = 1/6$. Repita `n_rep=1000` veces. ¿Qué proporción de veces gana Juan la jugada?
3. Calcule `n_rep=1000` veces la función `una_jugada` para los siguientes valores de p: `0.2, 0.5, 0.8`. ¿Qué proporción de veces gana Juan en cada caso?
4. Implemente una función `juan_se_arruina` que dados j (cantidad de monedas de Juan), m (cantidad total de monedas) y p (probabilidad de que Juan gane una jugada) simule una partida completa y devuelva `FALSE` si Juan gana y `TRUE`, caso contrario.
5. Simule $n\_rep = 1000$ partidas para $j = 3$, $m = 5$ y $p = 1/6$. Calcule la proporción de veces en las que Juan gana.
6. Implemente una función `estimacion_juan_gana` que dados j, m, p y `n_rep`, simule `n_rep` partidas con los parámetros que se le pasen y devuelva la proporción de veces en que gana Juan.
7. Fije $m = 5$ y $p = 1/6$. Para $j \in \{0, 1, \ldots, m\}$, simule `n_rep=1000` partidas y, en cada caso, calcule la proporción de veces en que gana Juan.
8. Grafique la proporción de veces en que Juan gana (eje y) cuando $p = 1/6$ y $m = 5$ en función de j (eje x), para $0 \leq j \leq m = 5$. Repita para $p = 0.5$ y $p = 0.8$.
9. Fije $p = 0.5$. Grafique la probabilidad estimada de que Juan gane en función de j cuando $m = 10$, para todos los posibles valores $0 \leq j \leq m$. Repita para $m = 20$, $m = 30$, $m = 50$. ¿Qué forma observa en cada uno de los gráficos?
10. Conjeturar una fórmula para la probabilidad de que Juan gane cuando $p = 0.5$, en función de j y de m, la cantidad inicial y total de monedas.
11. Quiero más: para los valores de $p =$ `1/6, 1/2, 4/5} y los valores de $m=$`$10, 20, 30, 50\}$, combine estos valores y grafique la probabilidad estimada de que Juan gane en función de j (para todos los j que cumplan con $0 \leq j \leq m$).

5.3.1 Referencias

Santos, J. B., Ruiz, J. A. C., & Hidalgo, M. D. P. (2008). El problema de la ruina del jugador. Suma: Revista sobre Enseñanza y Aprendizaje de las Matemáticas, (59), 23-30. ‘