Curso de Ingreso - Módulo Programación

Contents

Guía de ejercicios **1**

1 Introducción **1**
Ejercicios 1

2 Estructuras de control - Gráficos **5**
2.1 Condicionales 5
2.2 Ciclos 8
2.3 Gráficos 11

3 Funciones - DataFrames **19**
3.1 Parte A: Funciones 19
3.2 Parte B: DataFrames 24

Guía de ejercicios

1. Conceptos básicos de los programas imperativos
2. Estructuras de control - Gráficos
3. Funciones - DataFrames

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]

promedio <- suma / 3

promedio
```

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

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

```
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.

```
promedio <- mean(notas)

promedio
```

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

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

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

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

```
perro
```

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

```
gato
```

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

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

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

```
perro
```

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

```
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.

```r
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.

```r
max(imcs)
```

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

```r
min(imcs)
```

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

```r
mean(imcs)
```

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

```r
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.

```r
a_es_mas_grande <- COMPLETAR
```

```r
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.

```
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.

```
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.

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

```
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
}
```

```r
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. ;-)

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

```r
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.

```r
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.

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

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

```
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.

```
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
```

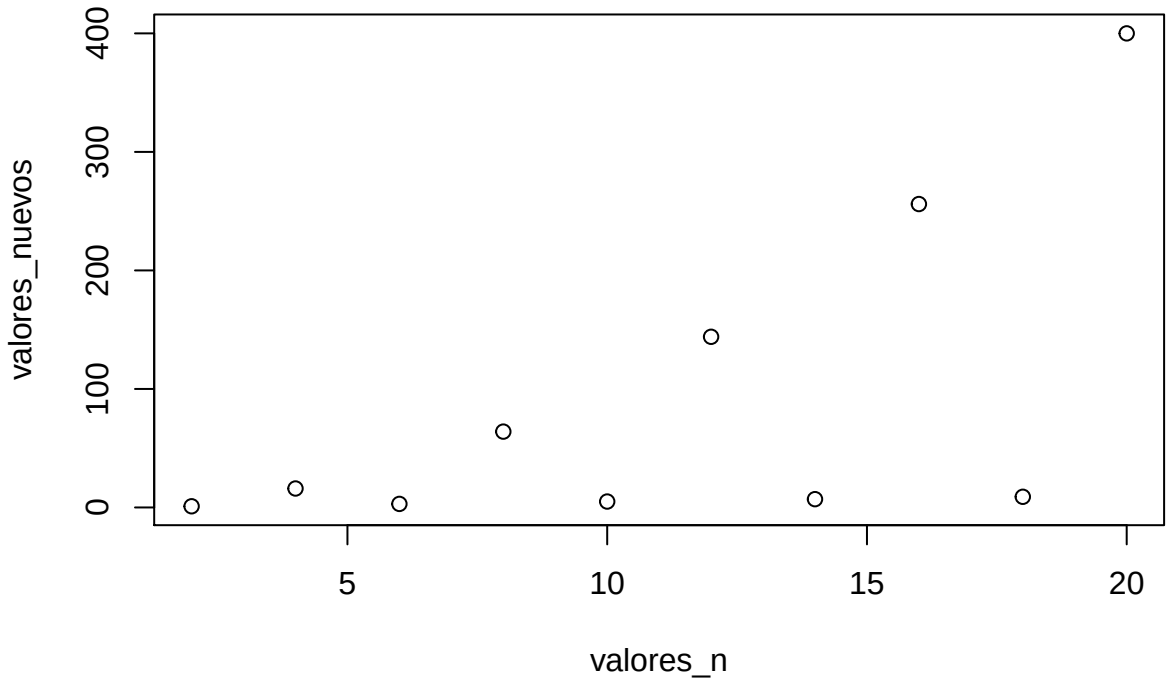
2.3 Gráficos

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

```
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)
```



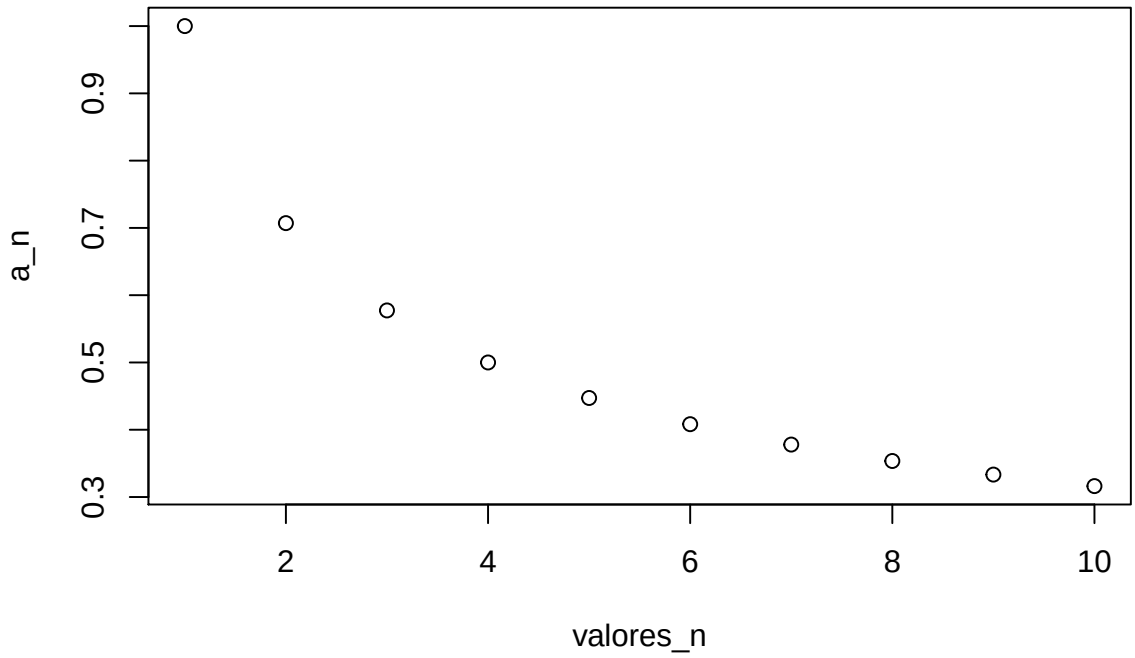
(21) ★ Escribir un programa que calcule y grafique los primeros 10 términos de las siguientes sucesiones:

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)**n
  a_n <- c(a_n,  nuevo_valor)
}
a_n
```

```
##  [1] 1.0000010 0.7071077 0.5773512 0.5000010 0.4472145 0.4082492 0.3779654
##  [8] 0.3535543 0.3333343 0.3162287
```

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

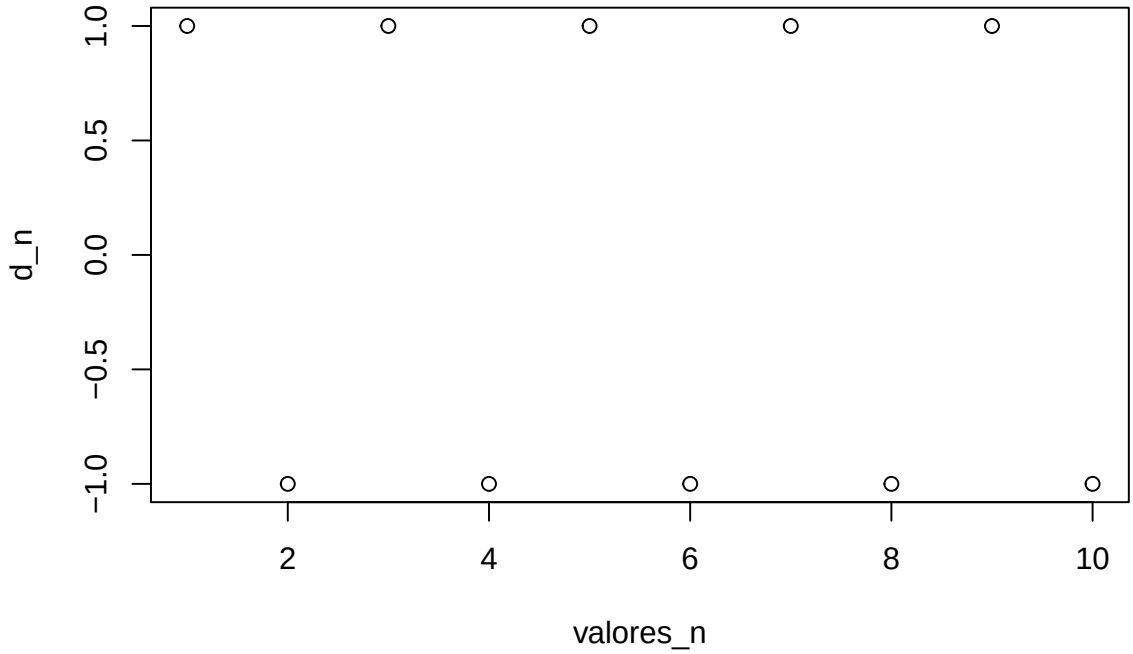


b. $d_n = (-1)^{n+5}$

```
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
```

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

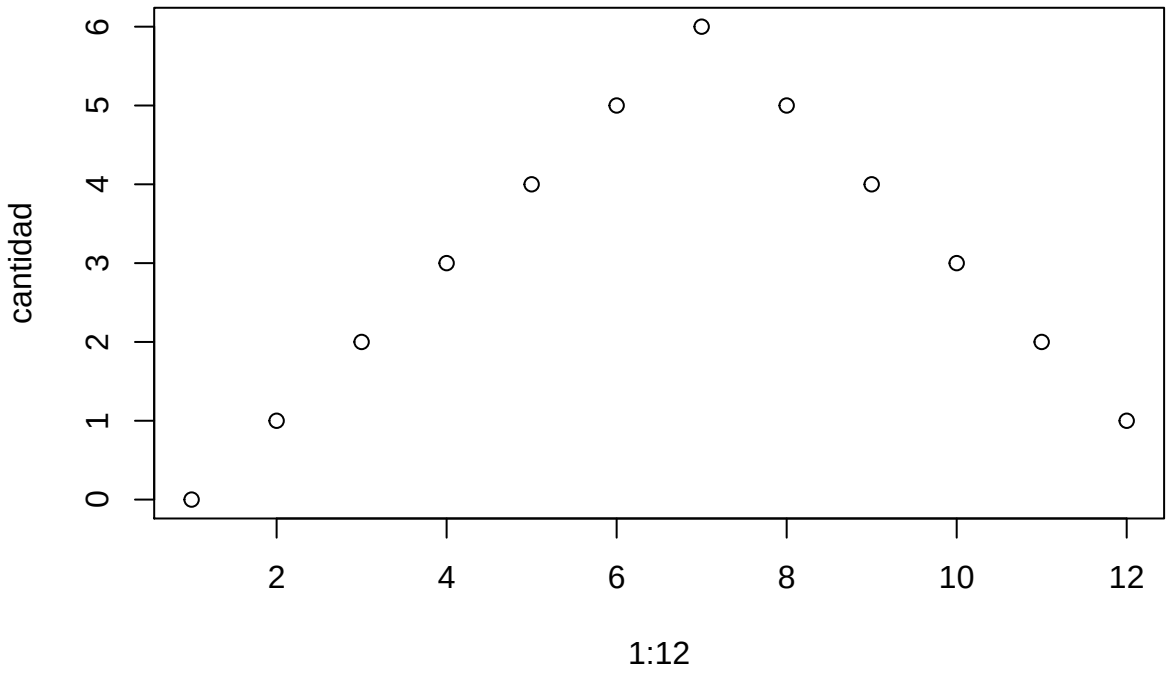


(22) ★ Retomando el ejercicio de las combinaciones de los dados, escribir un programa que inicie con un vector llamado `cantidad` de doce posiciones en cero (explorar el comando `rep`) y que para cada posición calcule cuántas combinaciones de los dados suman el valor indicado por la posición (ej. `cantidad[11]`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

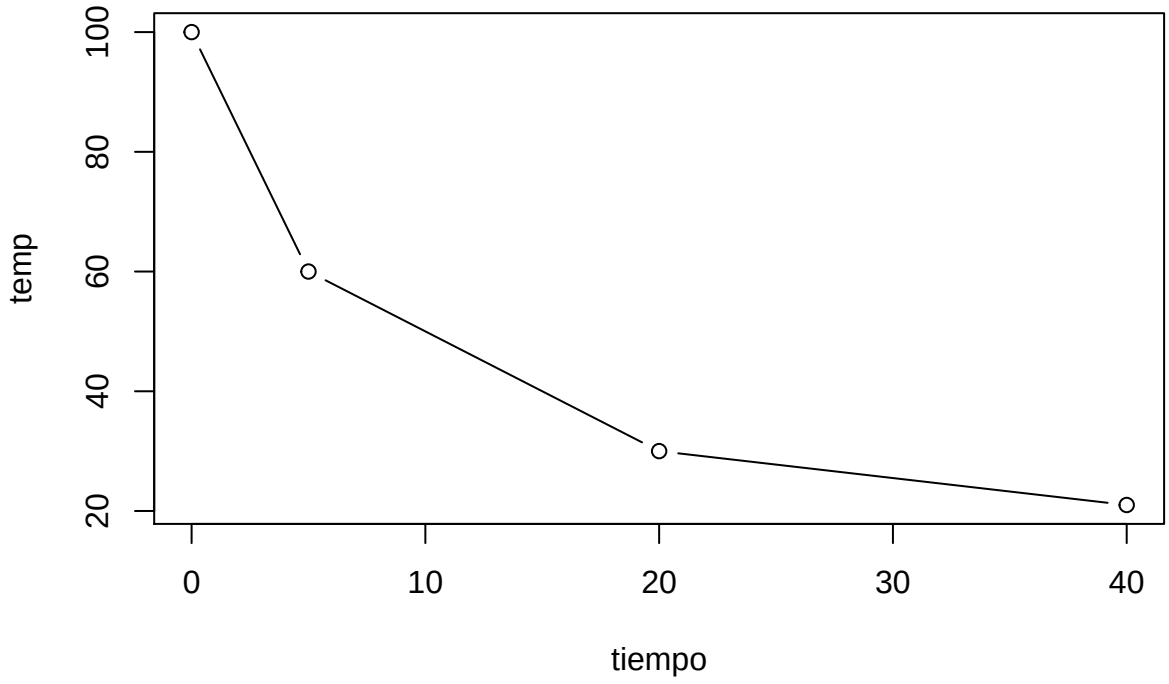
```
plot(1:12, cantidad)
```



(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.

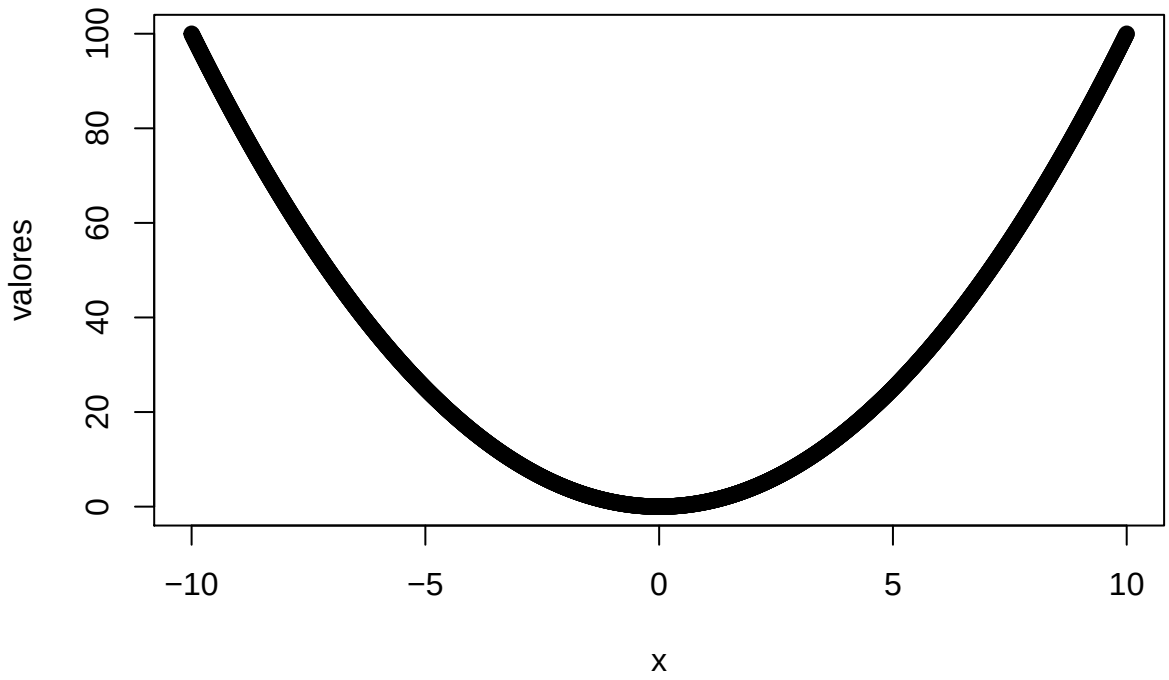
```
tiempo <- c(0,5,20,40)
temp <- c(100,60,30,21)
plot(tiempo, temp, type = "b")
```



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

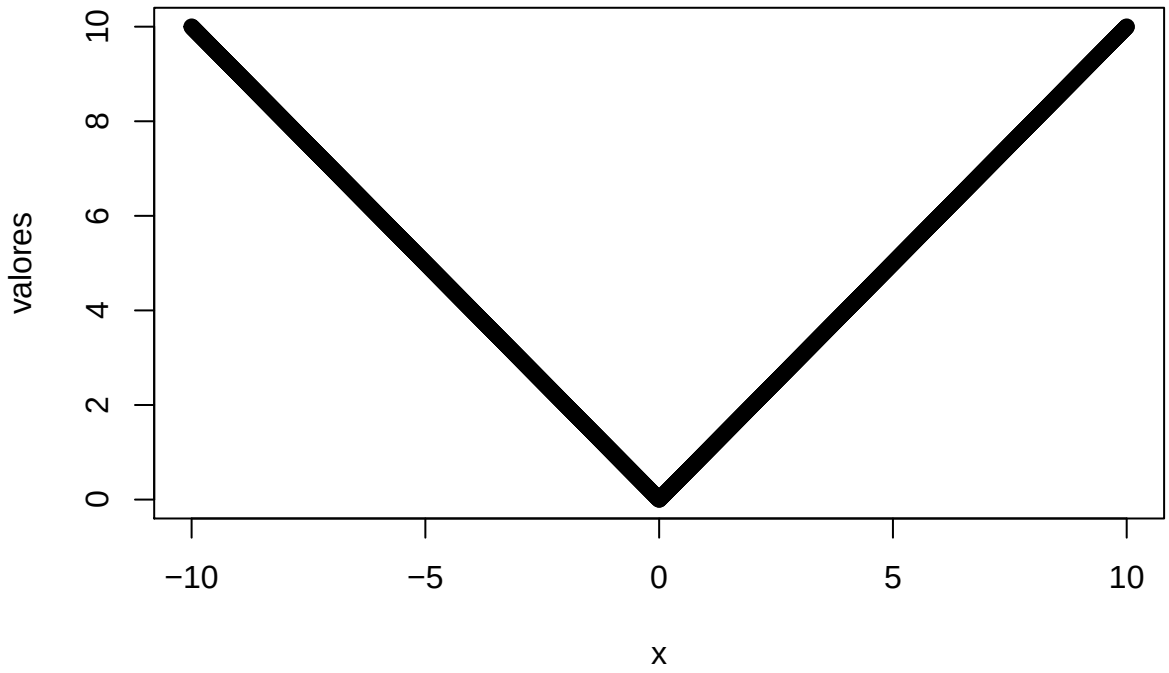
```
x <- seq(COMPLETAR, COMPLETAR, 0.01)
valores <- c()

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")
```

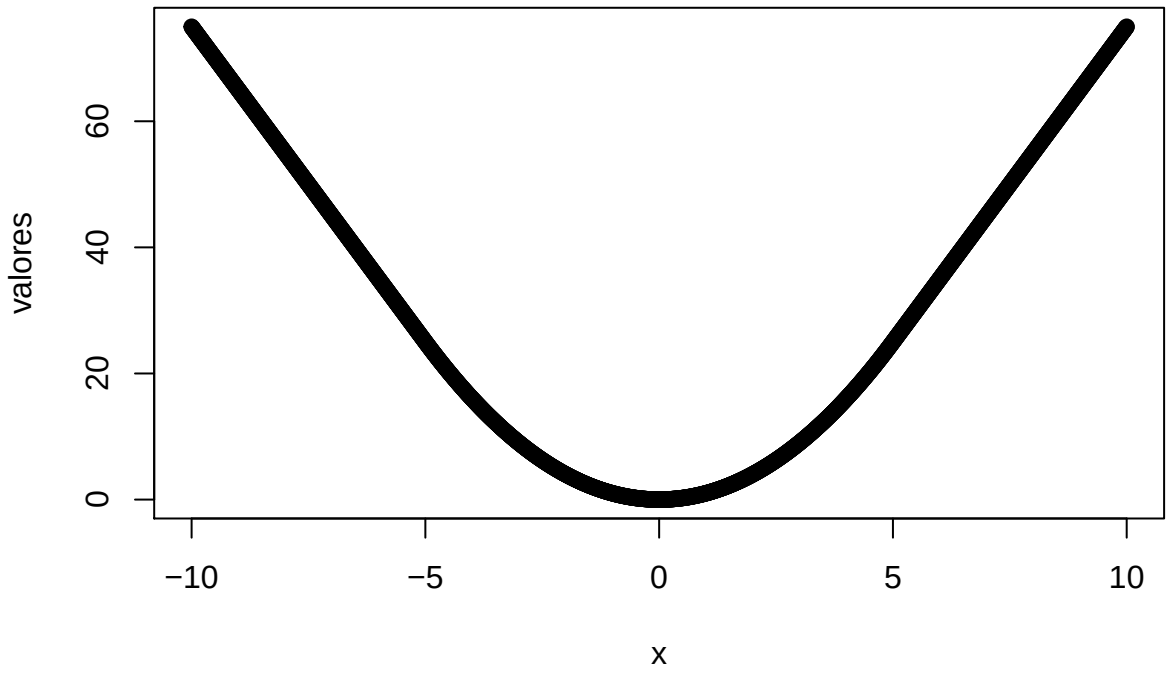


(26) ★ Considerar $\rho_k : \mathbb{R} \to \mathbb{R}$, definida de la siguiente manera

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

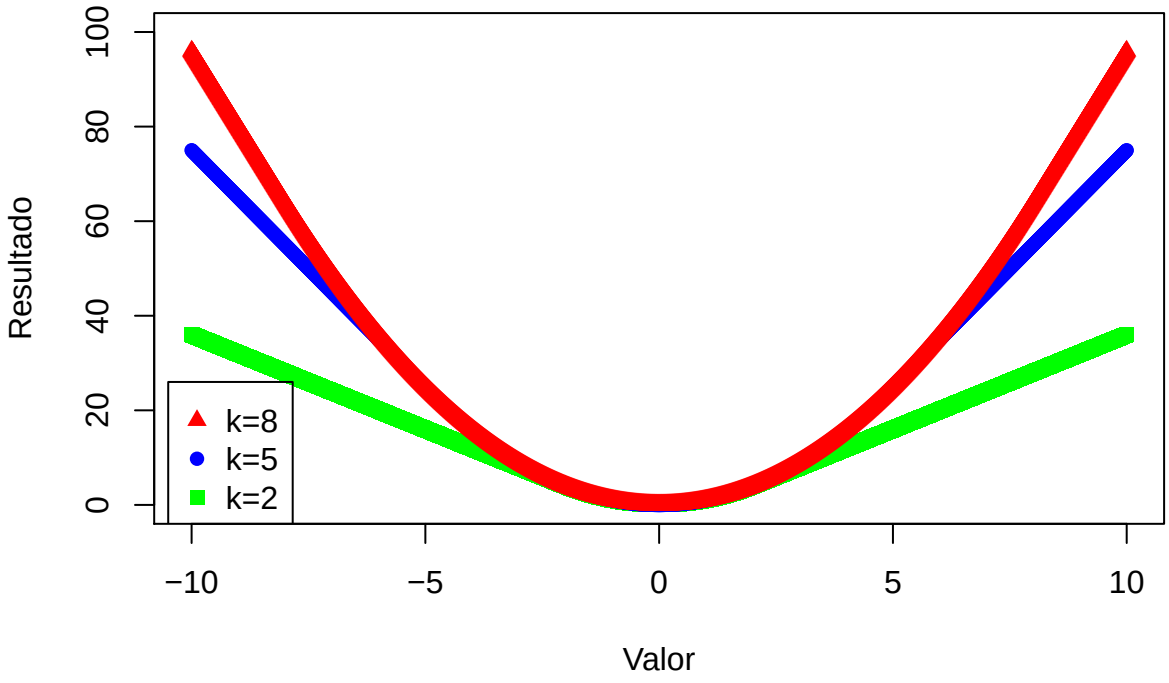
Calcular y graficar la la función ρ_k, con $k = 5$, para $x \in [-10, 10]$.

```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)
}
```

2. Escribir otra que se comporte a la inversa.

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.

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

```
  COMPLETAR
  return(res)
}
```

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
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
   b. Sin ciclos (con funciones de vectores)
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`).
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`.
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`.
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`.
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`.
11. Escribir una función que dado un número natural calcule un paso de la función de Collatz. Recordar que dice:

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

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)
}
```

13. Escribir una función que dado un número `n` calcule (y devuelva) los primeros `n` términos de las siguientes sucesiones:
   a. $a_n = \frac{1}{\sqrt{n}} + \left(\frac{1}{2}\right)^n$
   b. $d_n = (-1)^{n+5}$
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)
```

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

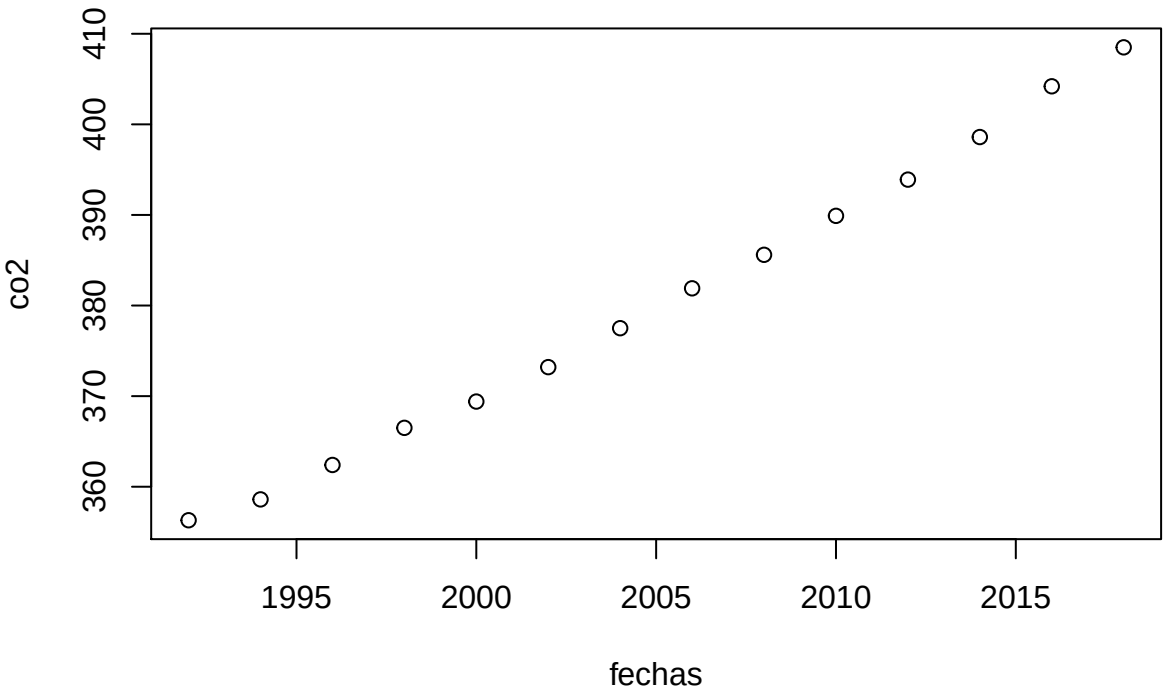
$$l_i(x) = \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)} \ , i = 0, \dots, n \ .$$

- ¿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.

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

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

```
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)
```

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

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

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?.
3. Seleccionar solo la columna *Petal.Length*.¿Cual es la media de la longitud de los petalos?.
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*?.
5. Calcular la frecuencia de individuos de la especie *virginica* que poseen un `Sepal.Width` mayor a 3.
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.
7. Ahora agregale una columna al `df` que sea la suma de los cuatro atributos para cada individuo.
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`.

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

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

```
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:

```
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.