Tarea 1

Ángela Vieyto 5.487.839-8

Entrega 7 de Mayo

1. Ejercicio 1

1.1. Parte 1: Vectores

1.1.1. Dado los siguientes vectores, indicá a qué tipo de vector coercionan.

```
w <- c(29, 1L, FALSE, "HOLA")
x <- c("Celeste pelela!", 33, NA)
y <- c(seq(3:25), 10L)
z <- paste(seq(3:25), 10L)

class(w)
```

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

```
class(x)
```

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

```
class(y)
```

```
## [1] "integer"
```

```
class(z)
```

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

Comentario: Correcto

1.1.2. ¿Cuál es la diferencia entre `c(4, 3, 2, 1)` y `4:1`?

La diferencia es el tipo de vector. Mientras que el primero es de tipo `numeric`, el último es de tipo `integer`.

Comentario: Correcto

1.2. Parte 2: `factor`

Dado el siguiente `factor` `x`:

```
x <-
   factor(
      c(
         "alto",
         "bajo",
         "medio",
         "alto",
         "muy alto",
         "bajo",
         "medio",
         "alto",
         "ALTO",
         "MEDIO",
         "BAJO",
         "MUY ALTO",
         "QUE LOCO",
         "QUE LOCO",
         "QUE LOCO",
         "A",
         "B",
         "C",
         "GUAU",
         "GOL",
         "MUY BAJO",
         "MUY BAJO",
         "MUY ALTO"
      )
   )
```

1.2.1. Generá un nuevo `factor` (llamalo `xx`) transformando el objeto `x` previamente generado de forma que quede como sigue:

```
xx
```

```
[1] A B M A A B M A A M B A B B A

Levels: B < M < A
```

Observación:

- El largo es de 23.
- Se deben corregir (y tomar en cuenta) todos los casos que contengan las palabras: bajo, medio, alto. Es decir, "MUY ALTO", "ALTO" deben transformarse a "alto" y así sucesivamente.

```
xx <- c()
for (i in seq_along(x)) {
  if (tolower(x[i]) == "alto" | tolower(x[i]) == "muy alto") {
    xx <- c(xx, "A")
  } else if (tolower(x[i]) == "medio") {
    xx <- c(xx, "M")
```

```
  } else if (tolower(x[i]) == "bajo" | tolower(x[i]) == "muy bajo") {
    xx <- c(xx, "B")
  }
}

xx <- factor(xx, levels = c("B", "M", "A"), ordered = TRUE)
xx
```

```
##  [1] A B M A A B M A A M B A B B A
## Levels: B < M < A
```

Comentario: Correcto

1.2.2. Generá el siguiente `data.frame()`

```
##   levels value
## 1      A     3
## 2      B     1
## 3      M     2
```

Para ello usá el vector `xx` que obtuviste en la parte anterior.

```
data.frame(levels = xx[1:3], value = c(3,1,2))
```

```
##   levels value
## 1      A     3
## 2      B     1
## 3      M     2
```

Comentario: Correcto

1.3. Parte 2: Listas

1.3.1. Generá una lista que se llame `lista_t1` que contenga:

- Un vector numérico de longitud 4 (`h`).
- Una matriz de dimensión 4*3 (`u`).
- La palabra "chau" (`palabra`).
- Una secuencia diaria de fechas (clase Date) desde 2021/01/01 hasta 2021/12/30 (`fecha`).

```
h <- c(6,8,1997,23)
u <- matrix(1:12, nrow = 4, ncol = 3)
palabra <- "chau"
fecha <- seq(as.Date("2021/01/01"), as.Date("2021/12/30"), by = 1)

lista_t1 <- list(h,u,palabra,fecha)
sapply(lista_t1, head)
```

```
## [[1]]
## [1]    6    8 1997   23
##
## [[2]]
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
##
## [[3]]
## [1] "chau"
##
## [[4]]
## [1] "2021-01-01" "2021-01-02" "2021-01-03" "2021-01-04" "2021-01-05"
## [6] "2021-01-06"
```

Comentario: Correcto

1.3.2. ¿Cuál es el tercer elemento de la primera fila de la matriz `u`? ¿Qué columna lo contiene?

```
lista_t1[[2]][1,3]
```

```
## [1] 9
```

La columna 3 contiene al elemento "9".

Comentario: Correcto

1.3.3. ¿Cuál es la diferencia entre hacer `lista_t1[[2]][] <- 0` y `lista_t1[[2]] <- 0`?

Corriendo la primera línea de código estamos remplazando todos los elementos de la matriz `u` por ceros, mientras que corriendo la segunda línea de código estamos remplazando la matriz `u` por un cero.

Comentario: Correcto

```
# Vemos la matriz original
lista_t1 <- list(h,u,palabra,fecha)
lista_t1[[2]]
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```r
# Vemos cómo queda con la primera opción
lista_t1[[2]][] <- 0
lista_t1[[2]]
```

```
##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    0    0    0
## [3,]    0    0    0
## [4,]    0    0    0
```

```r
# Vemos cómo queda con la segunda opción
lista_t1 <- list(h,u,palabra,fecha)
lista_t1[[2]] <- 0
lista_t1[[2]]
```

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

Comentario: Correcto

1.3.4. Iteración

Iterá sobre el objeto `lista_t1` y obtené la clase de cada elemento teniendo el cuenta que si la longitud de la clase del elemento es mayor a uno nos quedamos con el último elemento. Es decir, si `class(x)` es igual a `c("matrix", "array")` el resultado debería ser "array". A su vez retorná el resultado como clase `list` y como `character`.

Pista: Revisá la familia de funciones `apply`.

```r
lista_t1 <- list(h,u,palabra,fecha)
clase <- sapply(lista_t1, class)

for (i in 1:length(clase)){
  clase[[i]] <- tail(clase[[i]],1)
}

clase # resultado tipo list
```

```
## [[1]]
## [1] "numeric"
##
## [[2]]
## [1] "array"
##
## [[3]]
## [1] "character"
##
## [[4]]
## [1] "Date"
```

```r
class(clase)
```

```
## [1] "list"
```

```r
as.character(clase) # resultado tipo character
```

```
## [1] "numeric"   "array"     "character" "Date"
```

```r
class(as.character(clase))
```

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

Comentario: Correcto

1.3.5. Iteración (2)

Utilizando las últimas 10 observaciones del elemento "fecha" del objeto "lista_t1" escriba para cada fecha "La fecha en este momento es ...." donde "..." debe contener la fecha para valor de lista$fecha. Ejemplo: "La fecha en este momento es '2021-04-28' ". Hacelo de al menos 2 formas y que una de ellas sea utilizando un for. Obs: En este ejercicio NO imprimas los resultados.

```r
# Forma 1
ult10 <- tail(lista_t1[[4]], 10)
f1 <- c()
for (i in seq_along(ult10)) {
  f1 <- c(f1,paste("La fecha en este momento es ", ult10[i], sep = ""))
}
```

```r
# Forma 2
f2 <- paste("La fecha en este momento es ", lista_t1[[4]][(length(fecha)-9):(length(fecha))], sep = "")
```

Comentario: Correcto

1.4. Parte 3: Matrices

1.4.1. Generá una matriz A de dimensión $4 * 3$ y una matriz B de dimensión $4 * 2$ con números aleatorios usando alguna función predefinda en R.

```r
A <- matrix(sample(0:50, 12), nrow = 4, ncol = 3)
A
```

```
##      [,1] [,2] [,3]
## [1,]   14   23   39
## [2,]   15   27   49
## [3,]   38   25   30
## [4,]   36   11    8
```

```r
B <- matrix(sample(0:50, 8), nrow = 4, ncol = 2)
B
```

```
##      [,1] [,2]
## [1,]   48   19
## [2,]   21    9
## [3,]   43   20
## [4,]   22   34
```

Comentario: Correcto

1.4.2. Calculá el producto elemento a elemento de la primera columna de la matriz A por la última columna de la matriz B.

```r
A[,1]*B[,2]
```

```
## [1]  266  135  760 1224
```

Comentario: Correcto

1.4.3. Calculá el producto matricial entre $D = A^T B$. Luego seleccioná los elementos de la primer y tercera fila de la segunda columna (en un paso).

```r
D <- t(A)%*%B
D[c(1,3),2]
```

```
## [1] 2385 2054
```

Comentario: Correcto

1.4.4. Usá las matrices A y B de forma tal de lograr una matriz C de dimensión $4 * 5$. Con la función `attributes` inspeccioná los atributos de C. Posteriormente renombrá filas y columnas como "fila_1", "fila_2"... "columna_1", "columna_2", vuelvé a inspeccionar los atributos. Finalmente, generalizá y escribí una función que reciba como argumento una matriz y devuelva como resultado la misma matriz con columnas y filas con nombres.

```r
C <- cbind(A, B)
C
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   14   23   39   48   19
## [2,]   15   27   49   21    9
## [3,]   38   25   30   43   20
## [4,]   36   11    8   22   34
```

```r
attributes(C)
```

```
## $dim
## [1] 4 5
```

```r
filas <- c("fila_1", "fila_2", "fila_3", "fila_4")
columnas <- c("columna_1", "columna_2", "columna_3", "columna_4", "columna_5")
dimnames(C) <- list(filas, columnas)
C
```

```
##        columna_1 columna_2 columna_3 columna_4 columna_5
## fila_1        14        23        39        48        19
## fila_2        15        27        49        21         9
## fila_3        38        25        30        43        20
## fila_4        36        11         8        22        34
```

```r
attributes(C)
```

```
## $dim
## [1] 4 5
##
## $dimnames
## $dimnames[[1]]
## [1] "fila_1" "fila_2" "fila_3" "fila_4"
##
## $dimnames[[2]]
## [1] "columna_1" "columna_2" "columna_3" "columna_4" "columna_5"
```

Comentario: Correcto

```r
C <- cbind(A, B)
renombrar <- function(x) {

  filas <- vector()
  columnas <- vector()

  for(i in 1:dim(x)[1]) {
    filas = c(filas, paste("fila",i, sep = "_"))
  }

  for (i in 1:dim(x)[2]) {
    columnas = c(columnas, paste("columna",i, sep = "_"))
  }

  dimnames(x) <- list(filas, columnas)
  return(x)
}
renombrar(C)
```

```
##        columna_1 columna_2 columna_3 columna_4 columna_5
```

```
## fila_1        14        23        39        48        19
## fila_2        15        27        49        21         9
## fila_3        38        25        30        43        20
## fila_4        36        11         8        22        34
```

Comentario: Correcto, aunque faltó retornar el objeto

1.4.5. Puntos Extra: genelarizá la función para que funcione con arrays de forma que renombre filas, columnas y matrices.

```
renombrar <- function(x) {

  filas <- vector()
  columnas <- vector()
  matrices <- vector()


  for(i in 1:dim(x)[1]) {
    filas = c(filas, paste("fila",i, sep = "_"))
  }

  for (i in 1:dim(x)[2]) {
    columnas = c(columnas, paste("columna",i, sep = "_"))
  }

  for (i in 1:dim(x)[3]) {
    matrices = c(matrices, paste("matriz",i, sep = "_"))
  }

  dimnames(x) <- list(filas, columnas, matrices)
  return(x)
}

renombrar(array(1:10, c(2, 2, 2)))
```

```
## , , matriz_1
##
##        columna_1 columna_2
## fila_1         1         3
## fila_2         2         4
##
## , , matriz_2
##
##        columna_1 columna_2
## fila_1         5         7
## fila_2         6         8
```

Comentario: Correcto, pero retornar el objeto o la otra opción es cambiar globalmente los atributos esto se debe a una forma fácil sería con ó assign

2. Ejercicio 2

2.1. Parte 1: `ifelse()`

2.1.1. ¿Qué hace la función `ifelse()` del paquete `base` de R?

La función recibe una expresión lógica y por cada elemento devuelve un valor si se verifica la expresión y otro valor en caso contrario. El largo del objeto resultante es igual al largo del objeto recibido por la función.

Comentario: Correcto y es vectorizada.

2.1.2. Dado el vector x tal que: `x <- c(8, 6, 22, 1, 0, -2, -45)`, utilizando la función `ifelse()` del paquete `base`, reemplazá todos los elementos mayores estrictos a `0` por `1`, y todos los elementos menores o iguales a `0` por `0`.

```
x <- c(8, 6, 22, 1, 0, -2, -45)
x
```

```
## [1]   8   6  22   1   0  -2 -45
```

```
x <- ifelse(x > 0, 1, 0)
x
```

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

Comentario: Correcto

2.1.3. ¿Por qué no fué necesario usar un loop?

Porque al utilizar la función `ifelse()` estamos verificando la expresión lógica en cada elemento del vector, de modo que un loop no resulta necesario.
Comentario: Correcto esta vectorizada

2.2. Parte 2: `while()` loops

2.2.1. ¿Qué es un while loop y cómo es la estructura para generar uno en R? ¿En qué se diferencia de un for loop?

Un while loop implica repetir parte del código en tanto una cierta expresión lógica se verifique. El while loop se rompe cuando la expresión lógica deja de ser verdadera.

2.2.2. Dada la estructura siguiente, ¿Cuál es el valor del objeto `suma`? Responda sin realizar el cálculo en `R`.

```
x <- c(1,2,3)
suma <- 0
i <- 1
while(i < 6){
 suma = suma + x[i]
 i <- i + 1
}
```

El resultado no será numérico ya que el largo del vector `x` es de 3 elementos, mientras que el while loop requiere que el largo del vector sea de al menos 5.

Comentario: No es númerico ok, pero eso es muy amplio. Es NA

2.2.3. Modificá la estructura anterior para que `suma` valga 0 si el vector tiene largo menor a 5, o que sume los primeros 5 elementos si el vector tiene largo mayor a 5. A partir de ella generá una fución que se llame `sumar_si` y verificá que funcione utilizando los vectores `y <- c(1:3)`, `z <- c(1:15)`.

```
sumar_si <- function (x) {
  suma <- 0

  if (length(x) > 5) {
    i <- 1
    while (i <= 5) {
      suma <- suma + x[i]
      i <- i + 1
    }
  }
  suma
}

y <- c(1:3)
y
```

```
## [1] 1 2 3
```

```
sumar_si(y)
```

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

```
z <- c(1:15)
z
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

```
sumar_si(z)
```

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

Comentario: Correcto

2.2.4. Generá una estructura que multiplique los números naturales (empezando por el 1) hasta que dicha multiplicación supere el valor `10000`. Cuánto vale dicha productoria?

```
i <- 1
productoria <- 1

while (productoria <= 10000) {
  productoria <- productoria*i
  i <- i + 1
}

productoria
```

```
## [1] 40320
```

Comentario: Correcto

2.3. Parte 3: Ordenar

2.3.1. Generá una función `ordenar_x()`que para cualquier vector numérico, ordene sus elementos de menor a mayor. Por ejemplo:

Sea `x <- c(3,4,5,-2,1)`, `ordenar_x(x)` devuelve `c(-2,1,3,4,5)`.

Para controlar, generá dos vectores numéricos cualquiera y pasalos como argumentos en `ordenar_x()`.

Observación: Si usa la función `base::order()` entonces debe escribir 2 funciones. Una usando `base::order()` y otra sin usarla.

```
ordenar_x <- function (x) {

}

# Control
x <- c(3,4,5,-2,1)
ordenar_x(x)
```

```
## NULL
```

```
y <- c(5,-1,3,4,2,6,0,-2)
ordenar_x(y)
```

```
## NULL
```

Comentario: Quedó vacía

FALTA

2.3.2. ¿Qué devuelve `order(order(x))`?

FALTA

Comentario: Esta era fácil..

3. Ejercicios Extra

Esta parte es opcional pero de hacerla tendrán puntos extra.

3.1. Extra 1

3.1.1. ¿Qué función del paquete base es la que tiene mayor cantidad de argumentos?

Pistas: Posible solución:

0. Argumentos = `formals()`
1. Para comenzar use `ls("package:base")` y luego revise la función `get()` y `mget()` (use esta última, necesita modificar un parámetro ó formals).
2. Revise la funcion Filter
3. Itere
4. Obtenga el índice de valor máximo

3.2. Extra 2

Dado el siguiente vector:

```
valores <- 1:20
```

3.2.1. Obtené la suma acumulada, es decir 1, 3, 6, 10... de dos formas y que una de ellas sea utilizando la función `Reduce`.

Dados los siguientes data.frame

```
a = data.frame(a1 = 1:10,
               b1 = 1:10,
               c1 = 1:10,
               key = 1:10)
b = data.frame(d1 = 1:10,
               e1 = 1:10,
               f1 = 1:10,
               key = 1:10)
c = data.frame(g1 = 1:10,
               h1 = 1:10,
               i1 = 1:10,
               key = 1:10)
```

Uní en un solo data.frame usando la función `Reduce()`. Pista: Revisá la ayuda de la función `merge()` y buscá en material adicional si es necesario que es un join/merge.

3.3. Extra 3

3.3.1. Escribí una función que reciba como input un vector númerico y devuelva los índices donde un número se repite al menos k veces. Los parámetros deben ser el vector, el número a buscar y la cantidad mínima de veces que se debe repetir. Si el número no se encuentra, retorne un `warning` y el valor `NULL`.

A modo de ejemplo, pruebe con el vector `c(3, 1, 2, 3, 3, 3, 5, 5, 3, 3, 0, 0, 9, 3, 3, 3)`, buscando el número 3 al menos 3 veces. Los índices que debería obtener son 4 y 14.

3.4. Extra 4

Dado el siguiente `factor`

```
f1 <- factor(letters)
```

3.4.1. ¿Qué hace el siguiente código? Explicá las diferencias o semejanzas.

```
levels(f1) <- rev(levels(f1))
f2 <- rev(factor(letters))
f3 <- factor(letters, levels = rev(letters))
```

Comentario: Casi todo bien!! faltaron 2 ejercicios y un par de funciones que no devolvían el resultado esperado porque faltó agregar un return o un * ó assign. Se realizó un ejercicio extra. El código y archivo como siempre re ordenado, da gusto. PERO los nombres de los chunks no deben tener espacios, de hecho se recomienda que sean solo letras, números y guiones (-), ni siquiera guión bajo

- «-