Tarea 1

Curso NTAED

Entrega Domingo 27 de marzo

COMENTARIO: De la letra siguiente OMITAN (por ahora) la parte que menciona que deben subir el archivo a su repositorio de github. No hemos borrado la letra puesto que posteriormente (cuando veamos git y github) SI van a tener que subir la tarea a su repositorio. Mientras tanto DEBEN subir la tarea a EVA en la sección Entrega Tarea 1. DEBEN subir el archivo .Rmd y el PDF

La fecha para entregar la Tarea 1 es el domingo 27 de marzo (**NO** se aceptan retrasos). La tarea es individual por lo que cada uno tiene que escribir su propia versión de la misma aunque se incentiva la consulta de dudas con estudiantes del curso así como en foros de EVA.

La tarea debe ser realizada en RMarkdown disponible en tu repositorio de GitHub llamado **Tareas_STAT_NT** (generado en la Actividad 5) donde van a ir poniendo todas las tareas y actividades del curso en diferentes carpetas y cada uno con su correspondiente proyecto de RStudio.

En el **YAML** del .Rmd incluí tu nombre y CI (cambiar donde dice author: "STAT NT").

MUY IMPORTANTE Cuando generen el proyecto de RStudio siempre revisar que están utilizando la codificación de texto UTF-8 (text encoding UTF8). Para ello, debes ir a:

Tools -> Project Options -> Code Editing -> Text encoding seleccionar UTF-8.

El repositorio de GitHub para esta tarea debe contener el únicamente archivo .Rmd con la solución de la tarea 1 (en la carpeta correspondiente).

Para que podamos ver sus tareas y corregir las mismas nos tienen que hacer colaboradores de su repositorio de GitHub a Federico (fedemolina) y a Natalia (natydasilva).

IMPORTANTE utilicen el archivo .Rmd proporcionado en eva ("Tarea1_estructura.Rmd") compilen a PDF y escriban su código donde dice "SOLUCIÓN AQUÍ" mostrando el código del chunk (lo cual se hace por defecto puesto que echo = TRUE). Cuando vean "SOLUCIÓN AQUÍ: NO ESCRIBIR EN UN CHUNK" quiere decir que NO deben escribir en el chunk, escriban debajo de ese comentario. **Entrega que no siga esta instrucción NO se corrige y tiene 0 puntos**.

Verán que tal vez algunos ejercicios tienen alguna dificultad adicional que las actividades, se espera que revisando el material sugerido en el curso y leyendo la ayuda en R deberían ser capaces de resolver los problemas. Si las preguntas no son suficientemente claras, pregunten en el foro de EVA. Si las dudas no son de comprensión de la letra se aconseja primero **buscar por su cuenta inicialmente** ya que es parte del aprendizaje.

Si un ejercicio no lo pudiste realizar pero intentaste diferentes formas **no** dejes en blanco el ejercicio, mantené el código junto al razonamiento que utilizaste (si **no** te compila porque ese chunk te genera error agrega en el chunk la opción `eval = FALSE`).

Comentario:
NO MODIFICAR LA ESTRUCTURA DEL ARCHIVO

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): character class(x): character class(y): integrer class(z): character

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

y `4:1`?

Comentario:
NO modificar la estructura del archivo, usar utf-8.

```
# SOLUCIÓN AQUÍ
 x<-c(4,3,2,1)
y<-4:1
class(x)
```

```
## [1] "numeric"
```

```
class(y)
```

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

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:

Pistas:

- B es bajo, M es medio, A es alto.
- Conviertan todos los valores a minúsculas mediante una función de R, NO lo hagan a mano.
- Borren "muy" para que "bajo" y "muy bajo" sean iguales.
- Todo lo que no sea "bajo", "medio", "alto" debe ser excluido.
- Recuerden (como vieron en las actividades) que pueden usar *labels*.

```
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.

```
# SOLUCIÓN AQUÍ
```

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

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

```
# SOLUCIÓN AQUÍ
```

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`).

```
# SOLUCIÓN AQUÍ
h<-c(1,2,3,4)
u<-matrix(1:12,nrow=4)
palabra<-"chau"
fecha<-seq(as.Date("2021/01/01"), by = "days", length.out = 364)
lista_t1<-c(h,u,palabra,fecha)
```

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

```
# SOLUCIÓN AQUÍ
u[1,3]
```

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

```
which(u[1,]==u[1,3])
```

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

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

1.3.4. Iteración

Iterá sobre la 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`.

```
# SOLUCIÓN AQUÍ
```

1.3.5. Iteración (2)

Utilizando las últimas 10 observaciones de el 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.

```
# SOLUCIÓN AQUÍ
```

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.

```
# SOLUCIÓN AQUÍ
 A<-matrix(data = rexp(12, rate = 1), nrow = 4, ncol = 3)
 B<-matrix(data = rexp(8, rate = 1), nrow = 4, ncol = 2)
```

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.

```
# SOLUCIÓN AQUÍ
A[ ,1]*B[ ,2]
```

```
## [1] 0.72422423 0.06064502 0.27939586 2.51825159
```

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).

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

```
## [1] 3.582517 3.197648
```

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.

```
# SOLUCIÓN AQUÍ
C<-cbind(A,B)
attributes(C)
```

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

```
 row.names(C)<-c("fila_1","fila_2","fila_3","fila_4")
 colnames(C)<-c("columna_1","columna_2","columna_3","columna_4","columna_5")
 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"
```

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

2. Ejercicio 2

2.1. Parte 1: `ifelse()`

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

Es la union entre la funcion if y la funcion else. Se escribe de la siguiente manera ifelse(Prueba logica, resultado si verdadero, resultado si falso)

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`.

```
# SOLUCIÓN AQUÍ
x <- c(8, 6, 22, 1, 0, -2, -45)
ifelse(x>0, 1 , 0)
```

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

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

Porque se tomaron todos los posibles valores dentro del vector, y se cambiaron por otro, no habia necesidad de irse fijando uno por uno ## Parte 2: `while()` loops

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

2.1.5. 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
}
```

21

2.1.6. 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)`.

```
# SOLUCIÓN AQUÍ
```

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

```
# SOLUCIÓN AQUÍ
```

2.2. Parte 3: Ordenar

2.2.1. 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.

```
# SOLUCIÓN AQUÍ
```

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

```
# SOLUCIÓN AQUÍ
```

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

```
# SOLUCIÓN AQUÍ
```

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`.

```
# SOLUCIÓN AQUÍ
```

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.

```
# SOLUCIÓN AQUÍ
```

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.

```
# SOLUCIÓN AQUÍ
```

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))
```