Tarea 1 - Martín Olivera - CI. 4845488-3

STAT_NT

7/5/2021

Ejercicio 1

Parte 1: Vectores

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"
```

A partir de la función class (así como de la función str) se obtiene que el vector w, x y z son vectores characters, mientras que el vector y es integer. w, x contiene palabras, expresiones lógicas y números. y contiene números mientras que z concatena la secuencia con el número 10.
Comentario: Correcto

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

```
c(4, 3, 2, 1)
```

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

```
4:1
```

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

La diferencia que se observa entre ambas opciones es que aunque el resultado final devuelve en ambos casos la secuencia de números de 4 a 1 decrecientes de a 1, c() devuelve la secuencia de carácter numéric, mientras que la secuencia 4:1 devuelve la secuencia siendo integer.
Comentario: Correcto

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"
      )
   )
```

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.

```
unique(x)
```

```
##  [1] alto     bajo     medio    muy alto ALTO     MEDIO    BAJO     MUY ALTO
##  [9] QUE LOCO A        B        C        GUAU     GOL      MUY BAJO
## 15 Levels: A alto ALTO B bajo BAJO C GOL GUAU medio MEDIO muy alto ... QUE LOCO
```

```
aux<-c("alto", "ALTO","bajo", "BAJO", "medio", "MEDIO","muy alto","MUY ALTO", "MUY BAJO")
xx<-x[which(x%in%aux)]
xx<-factor(xx,levels=aux)
xxx<-c()
```

```
for (i in 1:length(xx)){
  if (xx[i]%in%c("ALTO","MUY ALTO", "muy alto")){
    xxx[i]="alto"
  } else if (xx[i]%in%c("BAJO","MUY BAJO")){
    xxx[i]="bajo"
  } else if (xx[i]=="MEDIO"){
    xxx[i]="medio"
  }
}

xx<-factor(xxx,levels=c("bajo","medio","alto"),labels=c("B","M","A"))
xx
```

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

```
xxx
```

```
##  [1] NA      NA      NA      NA      "alto"  NA      NA      NA      "alto"
## [10] "medio" "bajo"  "alto"  "bajo"  "bajo"  "alto"
```

Comentario: Resultado incorrecto, aunque muy buen intento. Revisar la solución

Generá el siguiente `data.frame()`

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

```
data<- as.data.frame(xx)
data
```

```
##      xx
## 1  <NA>
## 2  <NA>
## 3  <NA>
## 4  <NA>
## 5     A
## 6  <NA>
## 7  <NA>
## 8  <NA>
## 9     A
## 10    M
## 11    B
## 12    A
## 13    B
## 14    B
## 15    A
```

Comentario: Incorrecto

Parte 2: Listas

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(1, 2, 3, 4)
u <- matrix(1:12, nrow = 4, ncol = 3)
palabra <- "chau"
s <- as.Date("2021-01-01")
e <- as.Date("2021-12-30")
fecha <- seq(from = s, to = e, by = 1)
lista_t1 <- list(h, u, palabra, fecha)
```

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

```
lista_t1[[2]][3]
```

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

El tercer elemento de la matriz u es el número 3 y al haber completado la matriz de 4 filas, por columnas, el tercer elemento, es decir, el número 3, pertenece a la columna 1 de la matriz u.
Comentario: Correcto

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

lista_t1[[2]][] devuelve el segundo objeto de la lista en su totalidad, en este caso, la matriz u de dimensión 4x3 en su totalidad. Al asignarle el número cero, sustituye los elementos de la matriz por ceros, generando la matriz nula. La segunda alternativa también devuelve en su totalidad el segundo elemento de la lista (la matriz u), pero al asignarle el valor cero, no retorna la matriz nula, sino el número cero.
Comentario: Correcto

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`.

```
aux<- matrix(0, nrow=4, ncol=2)
for (i in 1:length(lista_t1)) {
  aux[i,]<-class(lista_t1[[i]])

}
 clases <- print(aux[,2])
```

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

Comentario: Resultado correcto. Aunque no es necesario (ni eficiente) definir una matriz en este caso.

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.

```
fechas<-rep(0, 10)
for (i in (length(lista_t1[[4]])-9):length(lista_t1[[4]])) {
```

```
  fechas[i] <- paste("La fecha en este momento es", lista_t1[[4]][i], sep = " ")
}
# print(fechas)
```

Comentario: A medias, es incorrecto el uso de fechas y los índices. Termina quedando un vector de largo 364 en vez de 10

```
lista_t1[[4]][length(lista_t1[[4]]-9)] <- "La fecha en este moemnto es 2021-12-21"
lista_t1[[4]][length(lista_t1[[4]]-8)] <- "La fecha en este moemnto es 2021-12-22"
lista_t1[[4]][length(lista_t1[[4]]-7)] <- "La fecha en este moemnto es 2021-12-23"
lista_t1[[4]][length(lista_t1[[4]]-6)] <- "La fecha en este moemnto es 2021-12-24"
lista_t1[[4]][length(lista_t1[[4]]-5)] <- "La fecha en este moemnto es 2021-12-25"
lista_t1[[4]][length(lista_t1[[4]]-4)] <- "La fecha en este moemnto es 2021-12-26"
lista_t1[[4]][length(lista_t1[[4]]-3)] <- "La fecha en este moemnto es 2021-12-27"
lista_t1[[4]][length(lista_t1[[4]]-2)] <- "La fecha en este moemnto es 2021-12-28"
lista_t1[[4]][length(lista_t1[[4]]-1)] <- "La fecha en este moemnto es 2021-12-29"
lista_t1[[4]][length(lista_t1[[4]]-0)] <- "La fecha en este moemnto es 2021-12-30"
```

Comentario: Incorrecto, de hecho no corre. Se esta queriando asignar un chacter a un elemento de clase Date

Parte 3: Matrices

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.

```
datos1<-runif(12, min=0, max=1)
datos2<-runif(8, min=0, max=1)
A <- matrix(datos1, nrow=4, ncol=3)
B <- matrix(datos2, nrow=4, ncol=2)
print(A)
```

```
##            [,1]      [,2]       [,3]
## [1,] 0.36282054 0.5333188 0.51567617
## [2,] 0.45997351 0.6848579 0.45817808
## [3,] 0.10830474 0.5067977 0.05132353
## [4,] 0.06799454 0.9055017 0.20316184
```

```
print(B)
```

```
##            [,1]      [,2]
## [1,] 0.54623446 0.7602144
## [2,] 0.07402834 0.5981380
## [3,] 0.45999045 0.6392969
## [4,] 0.33834678 0.5782579
```

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

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

```
## [1] 0.27582141 0.27512761 0.06923889 0.03931838
```

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).

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

```
## [1] 0.6595063 0.8163691
```

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.

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

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

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

```
##         columna_1 columna_2  columna_3  columna_4 columna_5
## fila_1 0.36282054 0.5333188 0.51567617 0.54623446 0.7602144
## fila_2 0.45997351 0.6848579 0.45817808 0.07402834 0.5981380
## fila_3 0.10830474 0.5067977 0.05132353 0.45999045 0.6392969
## fila_4 0.06799454 0.9055017 0.20316184 0.33834678 0.5782579
```

```
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"
```

```
renombrar <- function(x) {
  filas <- factor()
  columnas <- factor()
  for (i in 1:dim(x)[1]) {
    filas <- cbind(filas, paste("fila", i, sep = "_"))
  }
  for (i in 1:dim(x)[2]) {
    columnas <- cbind(columnas, paste("columna", i, sep = "_"))
  }
  dimnames(x) <- list(filas, columnas)
  return(x)
}
C <- cbind(A, B)
renombrar(C)
```

```
##          columna_1 columna_2  columna_3  columna_4 columna_5
## fila_1 0.36282054 0.5333188 0.51567617 0.54623446 0.7602144
## fila_2 0.45997351 0.6848579 0.45817808 0.07402834 0.5981380
## fila_3 0.10830474 0.5067977 0.05132353 0.45999045 0.6392969
## fila_4 0.06799454 0.9055017 0.20316184 0.33834678 0.5782579
```

```
#
```

Comentario: La primera parte esta bien, la segunda también pero falta retornar el objeto. No se si viste que de hecho la matriz C no se modifica por la función por tanto tendrías que retornar un objeto. Revisar la solución

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

```
renombrar_array <- function(x) {
  lapply(1:3, function(idx) {
    dimnames(x)[[idx]] <<-
      paste0(ifelse(idx == 1, "fila_", ifelse(idx == 2, "columna_", "matriz_")), 1:dim(x)[[idx]])
  })
  return(x)
}
```

Comentario: Correcto!

Ejercicio 2

Parte 1: `ifelse()`

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

La función ifelse funciona como un condicional booleano. Es decir plantea una condición que ejecuta cierta orden si se cumple. Si no se cumple 'else' ejecuta otra sentencia dada. La estructura de la función es: una expresión booleana seguida de la sentencia a ejecutar si el booleano se cumple y por último, la sentencia a ejecutar si el booleano no se cumple.

Comentario: Correcto

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)
for (i in 1:length(x)) {
  x[i] <- ifelse(x[i] > 0, 1, 0)
}
print(x)
```

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

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

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

Comentario: Faltó definir el vector x!! Pero esta correcto el código.

¿Por qué no fué necesario usar un loop ?

Porque es una única condición a aplicarse a todos los elementos del vector x. Además, la función ifelse permite no solo modificar cuando se cumple la condición, sino tambien cuando no se cumple.
Comentario: Correcto

Parte 2: `while()` loops

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

La función while se diferencia del for en que no es necesario hacer un recorrido iterativo estricto, sino que permite que la iteración se lleve adelante sólo si se cumple una condición dada.
Comentario: Correcto

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 razonamiento (sin previo cálculo en R) es el siguiente: dado x, defino una variable suma en cero. Luego inicializo un contador i a partir de 1. La interación comienza en i=1 y se detiene cuando i toma valores a partir de 6. Así, iniciando en 1, se realiza la iteración sumando la variable suma que originalmente vale cero por el primer elemeno de x, es decir 0+1. Luego le ordeno al contador que tome el valor 2 (i=1+1). Como el contador toma el valor 2 vuelvo a iterar sumando la variable suma que vale 1 por el segundo elemento de x, 2, lo que da 3. Luego el contador parte de i=2+1=3 y como es menor que 6 nuevamente sumo el tercer elemento de x, lo que resulta, 3+3 =6. El contador vale 4 pero no hay más elementos en x para sumar. Así el valor de suma dará un elemento en blanco porque no se logró completar con éxito el algoritmo.
Comentario: Correcto

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)`.

```
x <- c(1,2,3)

if (length(x)>=5) {
  for (i in 1:5) {
      suma = suma + x[i]
      }
} else {
  suma=0
}

sumar_si <- function(v) {
  suma=0
  if (length(v)>=5) {
    for (i in 1:5) {
```

```
      suma = suma + v[i]
      }
  }
  print(suma)
}

y <- c(1:3)
z <- c(1:15)
sumar_si(y)
```

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

```
sumar_si(z)
```

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

Comentario: Excelente

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

```
producto <- 1
i <- 1
while (producto <= 10000) {
  producto = producto*(i+1)
  i=i+1
  print(producto)
}
```

```
## [1] 2
## [1] 6
## [1] 24
## [1] 120
## [1] 720
## [1] 5040
## [1] 40320
```

El resultado de la productoria es de 40320: el primer resultado luego de superados los 10000. Como 40320 no es inferior a 10000 el algoritmo se detiene y se dejan de realizar productos.
Comentario: Excelente

Parte 3: Ordenar

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(v) {
  a <- c()
  j <- 1
  while (j < length(v)) {
```

```
    for (i in 1:(length(v) - 1)) {
      if (v[i] > v[i + 1]) {
        a[i] <- v[i]
        v[i] <- v[i + 1]
        v[i + 1] <- a[i]  #Cambio de lugar los elementos del vector ordenandolos de menor a mayor
      }
    }
    j <- j + 1
  }
  print(v)
}
x <- c(3, 4, 5, -2, 1)
ordenar_x(x)
```

```
## [1] -2  1  3  4  5
```

Comentario: Correcto

¿Qué devuelve `order(order(x))`?

La función order devuelve en las posiciones de los elementos del vector x, estos elementos ordenados de menor a mayor. Esta función a su vez devuelve un nuevo vector con las posiciones. Al aplicar nuevamente la función order a la ya aplicada función order del vector x, se repite la lógica: del nuevo vector de posiciones de x devuelve otro vector con las posiciones de los elementos ordenados de menor a mayor del vecto dado. order(x) devuelve un vector de posiciones de los elementos de x ordenados de menor a mayor. order(order(x)) develve un vector de posiciones de los elementos de (order(x)) ordenados de menor a mayor.

Comentario: Correcto

Ejercicios Extra

Esta parte es opcional pero de hacerla tendrán puntos extra.

Extra 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

Extra 2

Dado el siguiente vector:

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

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

```
valores <- 1:20
# Forma 1
suma_acum<-1
for (i in 2:20) {
  suma_acum = suma_acum + valores[i]
  print(suma_acum)
}
```

```
## [1] 3
## [1] 6
## [1] 10
## [1] 15
## [1] 21
## [1] 28
## [1] 36
## [1] 45
## [1] 55
## [1] 66
## [1] 78
## [1] 91
## [1] 105
## [1] 120
## [1] 136
## [1] 153
## [1] 171
## [1] 190
## [1] 210
```

La suma acumulada es 210
Comentario: Excelente

```
valores <- 1:20
# Forma 2
suma_acum2 <- function(x) {
  if (length(x)==1) {
    y<-x[1]   }
  else {
    y <- x[1] + suma_acum2(x[2:length(x)])
  }
  print(y)
}
suma_acum2(valores)
```

```
## [1] 20
## [1] 39
## [1] 57
## [1] 74
## [1] 90
## [1] 105
## [1] 119
## [1] 132
## [1] 144
## [1] 155
```

```
## [1] 165
## [1] 174
## [1] 182
## [1] 189
## [1] 195
## [1] 200
## [1] 204
## [1] 207
## [1] 209
## [1] 210
```

Nuevamente se verifica que el resultado de la suma acumulada es de 210.
Comentario: Excelente

```
valores <- 1:20
# Forma 1
Reduce(function(x,y) x + y,  valores)
```

```
## [1] 210
```

Nuevamente, el resultado es 210
Comentario: Faltó definir el parámetro accumulate

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.

```
d <- merge(a, b, by.x= a$key, by.y=b$key)
unificada <- merge(d, c, by.x=a$key, by.y= c$key)
```

Comentario: Había que usar Reduce

Extra 3

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.

Extra 4

Dado el siguiente `factor`

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

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

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

levels(f1) devuelve los niveles del factor f1 en órden invertido. f2 es igual a f1 en cuanto a devolver el orden reveso. Sin embargo, mientras en el primer caso lo que se devuelve en orden inverso son los niveles del factor, en f2 lo que se devuelve en orden inverso son los factores propiamente dicho. Los niveles se devuelven en orden alfabético. f3 hace un factor con las letras del abecedario, al igual que f1, pero devuelve los niveles de los factores en orden inverso.

Comentario: Muy bien!! En algunos casos te marque unos errores que estaría bueno lo compares con la solución especialmente los del principio. No es recomendable generar vectores que se vayan actualizando en un loop, es muy ineficiente. Los nombres de los archivos van sin acento