Tarea domiciliaria 1

STAT NT

28 de Marzo, 2019

Entrega

La fecha para entregar la tarea 1 es el Viernes 5 de Abril.

La tarea debe ser realizada en RMarkdown disponible en un repositorio de `GitHub` llamado "Tareas_STAT_NT" donde van a ir poniendo todas las tareas y actividades del curso en diferentes carpetas. La tarea es individual por lo que cada uno tiene que escribir su propia versión de la misma. El repositorio debe contener únicamente el archivo .Rmd con la solución de la tarea. Para que podamos ver sus tareas y corregir las mismas nos tienen que hacer colaboradoras de su repositorio de `GitHub` a Lucía (lcoudet) y a Natalia (natydasilva). Utilicen el archivo .Rmd de esta tarea como base para la solución, incorporando debajo de cada pregunta su respuesta, en caso que la respuesta sea un códico agregar el argumento `echo = TRUE` en el entorno de código de Rmarkdown para que se imprima en el documento.

Ejercicio 1

Parte 1: Coerción de vectores

Dado los siguientes vectores, indicá a qué tipo coercionan.

```
x <- c(TRUE, 15, 1L, "HOLA")
y <- c("ROJO", 25, NA)
z <- c(1, 2, 3, 4, 5L)
```

`x` coerciona a character, `y` coerciona character, y `z` coerciona a numeric.

Parte 2: Listas

1. Generá una lista que se llame `lista` que contenga:

- Un vector numérico de longitud 4 (llamalo `v`).
- Una matriz de dimensión 4×3 (llamala `m`).
- La palabra "hola" (llamala `palabra`).

```r
v <- 1:4
m <- matrix(data = rnorm(4*3), nrow = 4, ncol = 3)
palabra <- "hola"
lista <- list(v, m, palabra)
```

2. ¿Cuál es el tercer elemento del vector `v`?

```r
v[3]
```

```
[1] 3
```

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

```r
m[1,3]
```

```
[1] 1.142239
```

4. ¿Cuál es la última fila de la matriz `m`?

```r
m[dim(m)[1],]
```

```
[1] -1.1171676  0.1258163  1.1067628
```

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

`lista[[2]][] <- 0` asigna un 0 a todos los elementos de la matrix, mientras que `lista[[2]] <- 0` asigna un 0 a la segunda posición de la lista (es decir, ya no es una matrix sino un vector de largo 1 con el elemento 0).

Parte 3: Matrices

Generá una matriz A de dimensión 4×3 y una matriz B de dimensión 4×2.

```r
A <- matrix(data = rnorm(4*3), nrow = 4, ncol = 3)
B <- matrix(data = rnorm(4*2), nrow = 4, ncol = 2)
```

1. 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[,dim(B)[2]]
```

```
[1]  0.06160008 -0.01946405  0.86069498  0.24768696
```

2. Usando la función `t()` para lograr la conformidad de las matrices, calculá el producto matricial entre A y B.

```r
t(A) %*% B
```

```
           [,1]       [,2]
[1,]  0.5080291  1.1505180
[2,] -2.1405719  0.8824596
[3,] -2.1447760 -0.5977965
```

3. Usando `cbind()` usá las matrices A y B de forma tal de lograr una matriz C de dimensión 4×5.

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

4. Buscá una función en R que te permita cambiar los nombres de las columnas de una matriz. Para la matriz `B` que generaste, llamá a sus columnas `B1` y `B2`.

```r
colnames(B) <- c("B1", "B2")
```

5. Para la matriz C de dimensión 4×5 que generaste en el punto 3., seleccioná los elementos que pertenecen a la primera y la tercer fila de la cuarta columna de dicha matriz. Tenés que hacerlo todo en un solo paso (no vale subdividir varias veces).

```r
C[c(1,3),4]
```

```
[1] 0.0849002 1.2500423
```

Muy Bien!

Ejercicio 2

Parte 1: ifelse()

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

`Conditional element selection`.

2. Dado el vector x tal que: `x <- c(5, 6, 7, 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(5, 6, 7, 1, 0, -2, -45)
x <- ifelse(x > 0, 1, 0)
```

3. ¿Por qué no fue necesario usar un loop ?

`ifelse` está vectorizada.

Parte 2: `while` loops

1. ¿Qué es un `while` loop y cómo es la estructura para generar uno en R?

`while` es una estructura de control que repite una instrucción mientras una condición sea cierta. Estructura: `while (condition) instruction`

2. Dada la estructura siguiente, ¿por qué el objeto `suma` vale `NA` ?

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

Por que el resultado de las operaciones de subsetting del vector `x` es `NA` para todas las posiciones mayores al largo del vector. Sumar `NA` por default da como resultado `NA`.

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 función que se llame `sumar_si` y verificá que funcione utilizando los vectores `y <- c(1:3)`, `z <- c(1:15)`.

```r
# Estructura modificada
if (length(x) < 5) {
   suma <- 0
} else {
   suma <- sum(x[1:5])
}
# Veriificación para x
x
```

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

```r
# Función
sumar_si <- function(x, ...){
   if (length(x) < 5) {
   suma <- 0
   } else {
   suma <- sum(x[1:5])
   }
   return(suma)
}
sumar_si(y <- c(1:3))
```

```
[1] 0
```

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

```
[1] 15
```

4. Usando un while loop, generá una estructura que multiplique los números naturales (empezando por el 1) hasta que dicha multiplicación supere el valor `10000`. ¿Cuánto vale dicha productoria?

```r
start <- 1
prod <- 1
while (prod < 10000) {
   # print(start)
   start <- start + 1
   # print(start)
   prod <- prod * start
   # print(prod)
}
cat("La productoria vale", prod)
```

```
La productoria vale 40320
```

Muy Bien!

Ejercicio 3

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 testear tu función, generá dos vectores numéricos cualquiera y ponelos como argumento en `ordenar_x`.

```
x <- c(3, 4, 5, -2, 1)
ordenar_x <- function(x){
   n <- length(x)
   ordenado <- min(x)
   x <- x[-which.min(x)]
   while (length(ordenado) < n) {
      ordenado <- c(ordenado, min(x))
      x <- x[-which.min(x)]
   }
   return(ordenado)
}
ordenar_x(x)
```

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

```
#ordenar_x(rnorm(1000))
#ordenar_x(rnorm(1000))
```

Muy buen trabajo! 10/10. Sin comentarios adicionales