R LECTURES

10/04/2019

INFORME FINAL - DEADLINE 24/05

- Tema a elección de ustedes
  - Preferentemente relacionado a algún trabajo
- Tipo borrador de un paper
  - Con todos los elementos de un paper
  - Sin la terminación de un paper
- Entre 6 y 12 páginas, aprox.
- Pondremos distintos ejemplos en Classroom

REPASO DE LA CLASE ANTERIOR

VECTORES "AUMENTADOS"

- Data frames (y tibbles) sobre 'lists'
- Factores, construidos sobre vectores 'integer'
- Dates y date-times, sobre vectores 'numeric'

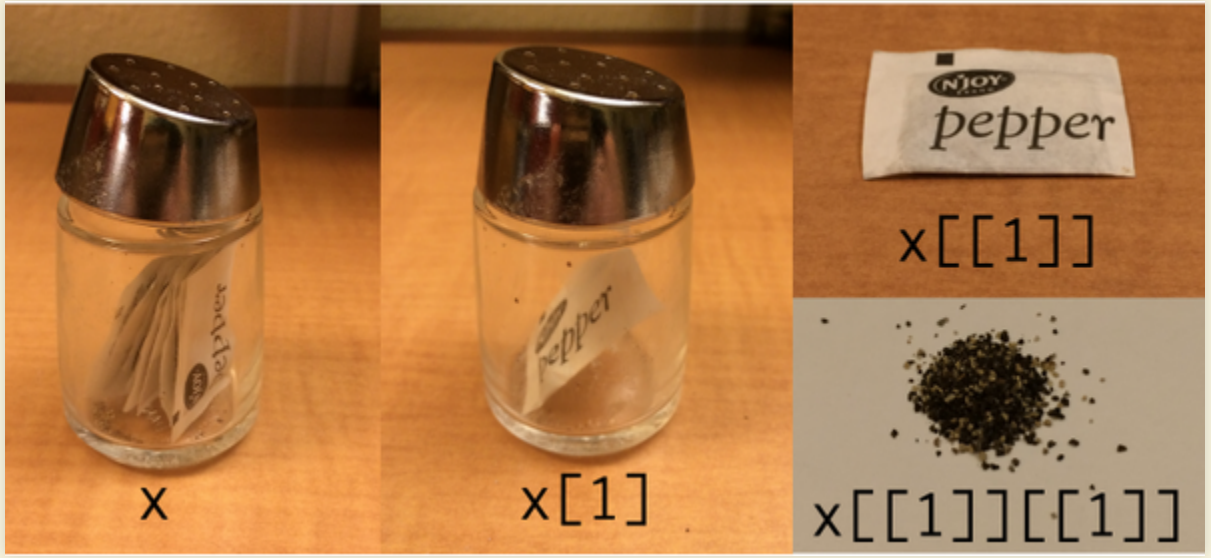
SUBSETTING

'[' es el operador para hacer *subsetting*, se usa poniendo algo del tipo x[a].
Hay distintas maneras de usarlo:

1. Usando vectores de enteros positivos o negativos
2. Usando vectores de lógicos
3. Usando vectores de caracteres para devolver elementos con los respectivos nombres
4. No usando nada

Otros operadores de subsetting: '[[' y '$'

INDEXADO DE LISTAS: '[[' VS. '['



crédito - Hadley Wickham: http://t.co/YQ6axb2w7t

CONTROL DE FLUJO

OPERADORES LÓGICOS

Son operadores que permiten comparar dos enunciados y evaluan a un resultado lógico

- > , >=
- < , <=
- != , ==

Más los operadores && (AND) y || (OR) para elaborar enunciados más complejos

```
10 == 10
[1] TRUE

x <- 10
x == 10
[1] TRUE

y <- NA
is.na(y)
[1] TRUE

## podemos combinar expresiones condicionales con || y &&
is.na(y) && x==10
[1] TRUE
```

EJECUCIÓN CONDICIONAL

```
if (condición) {
  # código que se ejecuta cuando la condición evalua a TRUE
} else {
  # código que se ejecuta cuando la condición evalua a FALSE
}
```

```
if (this) {
  # do that
} else if (that) {
  # do something else
} else {
  #
}
```

EJEMPLO

```
y <- NULL
l <- length(y)
if(is.null(x)) print("el vector tiene cero elementos") else print(paste("el vector tien

[1] "el vector tiene cero elementos"

x <- rep("relleno", 5)
l <- length(x)
if(is.null(x)) print("el vector tiene cero elementos") else print(paste("el vector tien

[1] "el vector tiene longitud 5"
```

FUNCIONES LÓGICAS ACCESORIAS

- `any()` # devuelve `TRUE` si alguno `TRUE`
- `all()` # devuelve `FALSE` si alguno `FALSE`
- `is.na()`, `is.null()` y el resto de la familia `is./algo/()`
- `%in%` # está `x` en este vector?`
- `which()` # devuelve posiciones de elementos `TRUE`
- `identical()` # por ej., numeric vs. integer
- muchas otras

```
"relleno" %in% x
[1] TRUE

which( 5 > c(3, 4, 5, 6))
[1] 1 2

identical(0L, 0)
[1] FALSE
```

LOOPS

Loops son bucles y se usan para repetir código.

```
for (variable dentro de /secuencia/){
# - código que se repite tantas veces como el largo de la secuencia
# - la variable va cambiando en cada pasada
}
```

EJEMPLO

```
df <- tibble(
  a = rnorm(10),
  b = rnorm(10),
  c = rnorm(10),
  d = rnorm(10)
)

output <- vector("double", ncol(df))
for (i in seq_along(df)) {            # seq_along(df) es parecida a 1:length(df)
  output[[i]] <- median(df[[i]])
}
output
# [1] -0.2458 -0.2873 -0.0567  0.1443
```

COMPONENTES DE UN FOR

- Output: siempre es buena idea crear el objeto antes de calcularlo
- Secuencia: variable sobre la que funciona el bucle

```
for (x in xs)
for(i in seq_along(df))
for (nm in names(xs))
```

- Cuerpo: código que se ejecuta las veces que la secuencia indique

CON SECUENCIA CONDICIONAL

Útil en el caso en donde no hay un número fijo de 'vueltas'.

```
while (condición) {
  # cuerpo
}
```

FUNCIONES

ANATOMÍA DE F()

```
# mi función se llama alta_funcion, con dos argumentos
alta_funcion <- function(arg1 = 10, arg2 = TRUE, ...){

  # acá empieza mi código
  library(paquete_externo)
  x <- funcion_externa(arg_ext = arg1)
  ...
  alto código
  código y más código
  ...
  alto_resultado <- mansa_funcion(arg2) # genero alto_resultado

  return(alto_resultado) # devuelvo alto_resultado
}
```

```
# llamo a mi función de distintas maneras
x_default <- alta_funcion() # uso arg1 = 10 y arg2 = TRUE
x_100_F   <- alta_funcion(100, FALSE)
x_200_T   <- alta_funcion(200, TRUE)
mi_var    <- alta_funcion(arg2 = FALSE, arg_ext = 10.2) # uso arg1 = 10
```

EJEMPLO

```r
# función que normaliza de dos maneras distintas un vector de valores
normaliza_vector <- function(vector = c(10, 9, 8, 7, 6), square_root = FALSE){

    if(square_root == TRUE) N <- sum(vector^2)^(1/2)
    else                    N <- sum(vector)/length(vector) # ojo, da error si vector e

    if(N != 0) return(vector/N)
    else print("error: Norma igual a cero!")
}

# distintas maneras de llamar la función
normaliza_vector() # con los argumentos por default
# [1] 1.250 1.125 1.000 0.875 0.750
x <- c(10, 9, 8, 7 , 6) # c() es una función que toma N argumentos y devuelve un vector
normaliza_vector(x) # haciendo explícito el vector
# [1] 1.250 1.125 1.000 0.875 0.750
y <- c(1, 2, 3)
normaliza_vector(y) # un vector diferente
# [1] 0.5 1.0 1.5
normaliza_vector(y, TRUE) # usando la norma euclidiana
# [1] 0.2672612 0.5345225 0.8017837
```

PRÁCTICA 7

Descargar práctica 7.