13Funciones_cheatsheet.R

moka

2023-04-20

```
# Autor: Monika Avila Marquez, Ph.D.
# Fecha: 12.04.2023
# Objetivo: Funciones
# Referencia: Basado en R Programming Fundamentals, StanfordOnline XDFS112
# Definicion: Programacion funcional divide las tareas en funciones.
```

```
# Limpiar el espacio de trabajo
rm(list=ls())
```

```
# Configurar el directorio
```

```
midirectorio<-setwd("~/Dropbox/0.POST-PHD/GOALS/2.CODE/R/Ecomienza/13Funciones")
midirectorio
```

```
## [1] "/Users/moka/Dropbox/0.POST-PHD/GOALS/2.CODE/R/Ecomienza/13Funciones"
```

```
# Ejemplos de funcion
library(readxl)
# Podemos escribir la funcion para ver que es lo que pasa en esta funcion
read_excel
```

```
## function (path, sheet = NULL, range = NULL, col_names = TRUE,
##     col_types = NULL, na = "", trim_ws = TRUE, skip = 0, n_max = Inf,
##     guess_max = min(1000, n_max), progress = readxl_progress(),
##     .name_repair = "unique")
## {
##     path <- check_file(path)
##     format <- check_format(path)
##     read_excel_(path = path, sheet = sheet, range = range, col_names = col_names,
##         col_types = col_types, na = na, trim_ws = trim_ws, skip = skip,
##         n_max = n_max, guess_max = guess_max, progress = progress,
##         .name_repair = .name_repair, format = format)
## }
## <bytecode: 0x10bf41290>
## <environment: namespace:readxl>
```

```
apply
```

```
## function (X, MARGIN, FUN, ..., simplify = TRUE)
## {
##     FUN <- match.fun(FUN)
##     simplify <- isTRUE(simplify)
##     dl <- length(dim(X))
##     if (!dl)
##         stop("dim(X) must have a positive length")
##     if (is.object(X))
```

```
##         X <- if (dl == 2L)
##             as.matrix(X)
##         else as.array(X)
##     d <- dim(X)
##     dn <- dimnames(X)
##     ds <- seq_len(dl)
##     if (is.character(MARGIN)) {
##         if (is.null(dnn <- names(dn)))
##             stop("'X' must have named dimnames")
##         MARGIN <- match(MARGIN, dnn)
##         if (anyNA(MARGIN))
##             stop("not all elements of 'MARGIN' are names of dimensions")
##     }
##     d.call <- d[-MARGIN]
##     d.ans <- d[MARGIN]
##     if (anyNA(d.call) || anyNA(d.ans))
##         stop("'MARGIN' does not match dim(X)")
##     s.call <- ds[-MARGIN]
##     s.ans <- ds[MARGIN]
##     dn.call <- dn[-MARGIN]
##     dn.ans <- dn[MARGIN]
##     d2 <- prod(d.ans)
##     if (d2 == 0L) {
##         newX <- array(vector(typeof(X), 1L), dim = c(prod(d.call),
##             1L))
##         ans <- forceAndCall(1, FUN, if (length(d.call) < 2L) newX[,
##             1] else array(newX[, 1L], d.call, dn.call), ...)
##         return(if (is.null(ans)) ans else if (length(d.ans) <
##             2L) ans[1L][-1L] else array(ans, d.ans, dn.ans))
##     }
##     newX <- aperm(X, c(s.call, s.ans))
##     dim(newX) <- c(prod(d.call), d2)
##     ans <- vector("list", d2)
##     if (length(d.call) < 2L) {
##         if (length(dn.call))
##             dimnames(newX) <- c(dn.call, list(NULL))
##         for (i in 1L:d2) {
##             tmp <- forceAndCall(1, FUN, newX[, i], ...)
##             if (!is.null(tmp))
##                 ans[[i]] <- tmp
##         }
##     }
##     else for (i in 1L:d2) {
##         tmp <- forceAndCall(1, FUN, array(newX[, i], d.call,
##             dn.call), ...)
##         if (!is.null(tmp))
##             ans[[i]] <- tmp
##     }
##     ans.list <- !simplify || is.recursive(ans[[1L]])
##     l.ans <- length(ans[[1L]])
##     ans.names <- names(ans[[1L]])
##     if (!ans.list)
##         ans.list <- any(lengths(ans) != l.ans)
##     if (!ans.list && length(ans.names)) {
```

```
##         all.same <- vapply(ans, function(x) identical(names(x),
##             ans.names), NA)
##         if (!all(all.same))
##             ans.names <- NULL
##     }
##     len.a <- if (ans.list)
##         d2
##     else length(ans <- unlist(ans, recursive = FALSE))
##     if (length(MARGIN) == 1L && len.a == d2) {
##         names(ans) <- if (length(dn.ans[[1L]]))
##             dn.ans[[1L]]
##         ans
##     }
##     else if (len.a == d2)
##         array(ans, d.ans, dn.ans)
##     else if (len.a && len.a%%d2 == 0L) {
##         if (is.null(dn.ans))
##             dn.ans <- vector(mode = "list", length(d.ans))
##         dn1 <- list(ans.names)
##         if (length(dn.call) && !is.null(n1 <- names(dn <- dn.call[1])) &&
##             nzchar(n1) && length(ans.names) == length(dn[[1]]))
##             names(dn1) <- n1
##         dn.ans <- c(dn1, dn.ans)
##         array(ans, c(len.a%/%d2, d.ans), if (!is.null(names(dn.ans)) ||
##             !all(vapply(dn.ans, is.null, NA)))
##             dn.ans)
##     }
##     else ans
## }
## <bytecode: 0x158b00ee8>
## <environment: namespace:base>
```

```
  # Se ve cuales son los argumentos de la funcion entre parentesis, tambien se puede ver el codigo de l
# Esto puede ser util si queremos usar esa funcion para construir una propia.
# Funciones internas no muestran mucho
q
```

```
## function (save = "default", status = 0, runLast = TRUE)
## .Internal(quit(save, status, runLast))
## <bytecode: 0x10bf30f80>
## <environment: namespace:base>
```

```
# Crear funciones
# Ejemplo: crear una funcion que exponencia los elementos a la potencia 2 y se suma 1
ExpAnd1<-function(vec,exponent,addition){
vec^exponent+addition
}
# Esta funcion se la puede:
# 1. Correr el codigo de la funcion, y sera cargada.
# 1. guardar en un script separado, y despues llamarla desde source.
source("ExpAnd.R")

# Ahora, tratamos de correr la funcion
#ExpAnd()
# me da error porque no declare valores default en la funcion
```

```r
source("ExpAnd2.R")
z=ExpAnd2(exponent=3,addition=1)

# Mejores practicas para escribir funciones
# 1. Describir la funcion
# 2. Describir argumentos and salidas
# 3. Siempre utilizar return, especialmente si queremos generar varias cosas.
# Si no usamos return, la funcion va a retornar el ultimo item.
# 4. Debugging(depuracion) usando condiciones if.  Se puede poner limites y mensajes que expliquen el e
m=matrix(c(1,2,3,4,5,6),ncol=2)
z=ExpAnd2(vec=m,exponent=3,addition=0)
#z=ExpAnd2(vec=m,exponent=3,addition="0")
# 5. Podemos pasar argumentos que no esten especificados utilizando ... en la lista de argumentos
# Esto es comun en funciones de graficos
source("ExpAnd3.R")
z=ExpAnd3(vec=m,exponent=3,addition=10,1001)
z
```

```
##      [,1] [,2]
## [1,]   11   74
## [2,]   18  135
## [3,]   37  226
```

```r
# Base R cheat sheet

# No quitar el comentario de la linea inferior. Solamente copiar en la consola para que ejecute

# rmarkdown::render("13Funciones_cheatsheet.R",c("pdf_document","html_document"))
```